Imperial College London

MENG INDIVIDUAL PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

On Dynamics of Interpretable Neural Models

Author:
Rishabh Jain

Supervisor:
Dr Pranava Madhyastha

June 17, 2019

Abstract

Neural networks have a reputation for behaving as *black-boxes*. To motivate their usage in real world systems their interpretability is essential. Evidently, there has been a lot of discussion in the community regarding this aspect of neural networks.

While most of the recent interest has focused on providing local explanations, there has been a much lower emphasis on studying the effects of model dynamics and its impact on explanation. We address the challenges of both over-confident and under-confident predictions with interpretability using attention distribution. Our results indicate that the means of using attention distributions for interpretability are highly unstable for un-calibrated models.

We also conduct a comprehensive study on the behavior of deep learning models under different random seed initializations. We try to quantify the model in-stability as a function of random seeds by investigating the effects of the induced randomness on model performance, attention mechanisms, and the robustness of the model in general.

Our experiments indicate that deep learning models can behave in-consistently, providing counter-factual explanations, under the impression of different random seeds. We propose a novel technique called *Aggressive Stochastic Weight Averaging (ASWA)* and an extension called *Norm-filtered Aggressive Stochastic Weight Averaging (NASWA)* which improves the stability of model over random-seeds. With our ASWA and NASWA implementations, we are able to improve the robustness of the original model, on an average, reducing the standard deviation of the model's performance by 72%.

Acknowledgements

I would like to thank my supervisor, Dr. Pranava Madhyastha for his guidance. His willingness to help, incredible dedication, and invaluable advice have assisted me significantly throughout the project.

I would also like to thank my friends and family for their continued support and belief in me.

Contents

Chapter 1

Introduction

Nowadays, artificial intelligence plays a crucial role in different aspects of our day to day life. Deep learning[GBC16] (and neural networks in general) has been a major driving force for this progress. It has proven its excellence in a variety of tasks ranging from image analysis [KSH12], understanding natural languages [CW08], speech recognition [HDY+12], movie recommendation systems [Xue18] among others. A lot of these machine learning models (especially, neural networks) are opaque, non-intuitive, and difficult for people to understand. Therefore, explainable artificial intelligence is essential for users to trust and understand this technology.

1.1 Problem statement

With the recent advancements in neural networks, an increasingly important aspect for them is their interpretability. For their practical usage in production systems, it is important that neural networks (including Deep neural networks – DNNs) are inferable by their users. For instance, in a medical diagnosis system, it is not possible to rely on a *black-box* model. Each decision in those cases should be open to validation from an expert. Similarly, in autonomous cars, given the sensitivity of each decision, the models have to be reliant and transparent. Unfortunately, in terms of explaining their output, neural networks have a reputation for behaving as a *black-box*. Their complex structure with hundreds of parameters and variables makes it hard to understand what the model has learnt and how it behaves.

As expected, there has been a lot of development towards demystifying these models and the research continues till this date[EBCV09, MV15, LTB+13, SVZ13, BBM+15].

In this project, we aim to analyze various dynamics of neural models that report its interpretability. We primarily focus on models that use *attention mechanism* (explained in Section 2.3) to explain their outputs.

1.2 Motivation

Some of the key motivations for explainable/interpretable neural networks are mentioned below.

- To build user trust and transparency. Users can be aware of what the system has learnt.

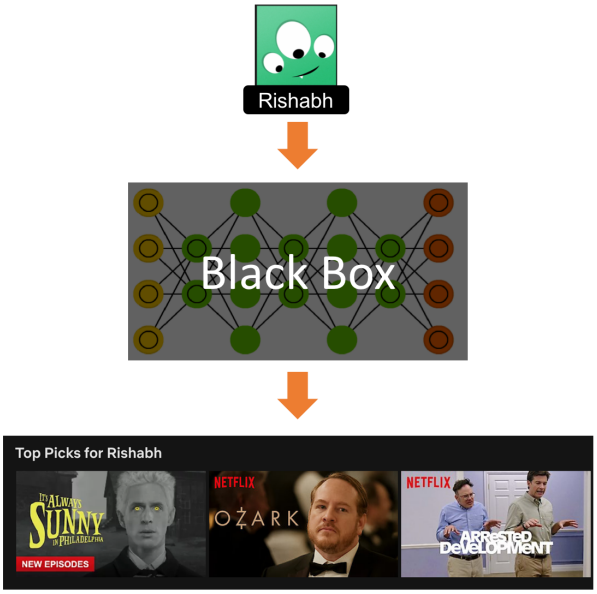


Figure 1.1: Figure highlighting the *black-box* nature of a neural recommendation system.

- They can helps us to understand the model and check if it works the intended way.
- Sometimes, interpretability also helps us in improving the accuracy of the model (e.g. attention based neural recommendation systems[Xue18]).
- Recently, there have been legislative enforcements (by the European Union) on the "right to explanation" which gives users the right to ask for an explanation of an algorithmic decision that was made about them. This suggests that in the future it might be necessary for any (user centered) AI system to be explainable.[Lip17]

1.3 Contributions

- In the first part of the project (Chapter 3) we focus on neural-networks based recommendation systems, especially the ones that claim interpretability using attention (discussed in Section 2.3).

  We analyze various model dynamics like calibration (measure of over-confidence or under-confidence in the model) (Section 2.4), reliability of attention weights and the stability of those models. We perform a focused study on the impact of model interpretability in the context of calibration (discussed in Section 2.4). Specifically, we address the challenges of both over-confident and under-confident predictions with interpretability using attention distributions. We try to relate these model dynamics with one another. Our results indicate that the means of using attention distributions for interpretability are highly unstable for un-calibrated models. We also propose a solution (Section 3.4.3) to fix the problem of un-reliable explanations (from attention weights) highlighted from our experiments.

- In the second part of this project (Chapter 4), we focus on the stability of DNNs, used in Natural Language Processing (NLP) systems[CW08], again in the context

of their interpretability and reliability. We measure the impact of random initializations on different aspects of DNNs and try to quantify the in-stability in them with various experiments.

Our analysis suggests that random-seeds can adversely affect the consistency in models resulting in counter-factual explanations and interpretations. We propose a novel technique called *Aggressive Stochastic Weight Averaging (ASWA)*(Section 4.4) and an extension called *Norm-filtered Aggressive Stochastic Weight Averaging (NASWA)*(Section 4.4.1) which improve the stability of model over random-seeds by tweaking the training procedure. With our ASWA and NASWA implementations, we are able to improve the robustness of the original model, on an average, reducing the standard deviation of the model's performance by 72%.

1.3.1 Publications

- **Model Explanations under Calibration**: Our paper[RJ] (to be published in July) based on the research done in Chapter 3 has been accepted by *The International Workshop on ExplainAble Recommendation and Search (EARS 2019)*[1] (a part of the *International ACM SIGIR Conference on Research and Development in Information Retrieval* [2]), to be held in Paris in July.

- **On Model Stability as a Function of Random Seed**: We have written another paper based on the work from Chapter 4, submitted to *The SIGNLL Conference on Computational Natural Language Learning*[3] to be held in Hong Kong in November 2019 (results for acceptance to be announced in July).

Both the submitted papers have been added to the Appendix of this report.

[1] https://ears2019.github.io/
[2] https://sigir.org/sigir2019/
[3] https://www.conll.org/2019

Chapter 2

Background and Related work

In this section, we look at the related works on the range of topics that we will be looking into during the course of this project. The key topics include neural attention (Section 2.3), calibration (Section 2.4), stability of neural networks (Section 2.5) among others.

In the first part of this project (Chapter 3), we focus on Recommendation systems (Section 2.1), however, for the second part of the project (Chapter 4), we perform our experiments on Natural language processing (Text-classification and Question-Answering) based (deep) neural networks (Section 2.6).

2.1 Recommendation systems

Given the growth of content available on the web, users are often greeted with an overflow of choices. Thus, it is important to advertise content that matches the user's interests. This is where recommendation systems have started playing an important role. These systems incorporate user preferences (implicit or explicit) when picking the items to be served to a particular user. At this point, recommendation systems are critical instruments to enhance user experience in many online websites and applications. For instance, 80 percent of the movies watched on Netflix were suggested to the users from their recommendation system[Zha18a].

Recommendation systems are used for item filtering based on user preferences in a variety of areas including movies, news, books, social recommendations and products in general. There are mainly three approaches to recommendation systems:

- Collaborative Filtering: These methods are based on capturing similarity between users (or items) based on the multiple user-item interactions and recommending them likewise. Figure 2.1 shows a user based collaborative filtering model where we recommend item C to the active user because of the preferences of users that are similar to them.

- Content based filtering: These models make recommendations based on the user-item preferences. They try to capture item (content) features that a user prefers. As we can see in Figure 2.2, in these recommendation systems, the items are recommended based on similarly-featured items that the user has liked in the past.

- Hybrid systems: These models combine multiple techniques together to achieve better accuracy by capturing better user-item preferences.

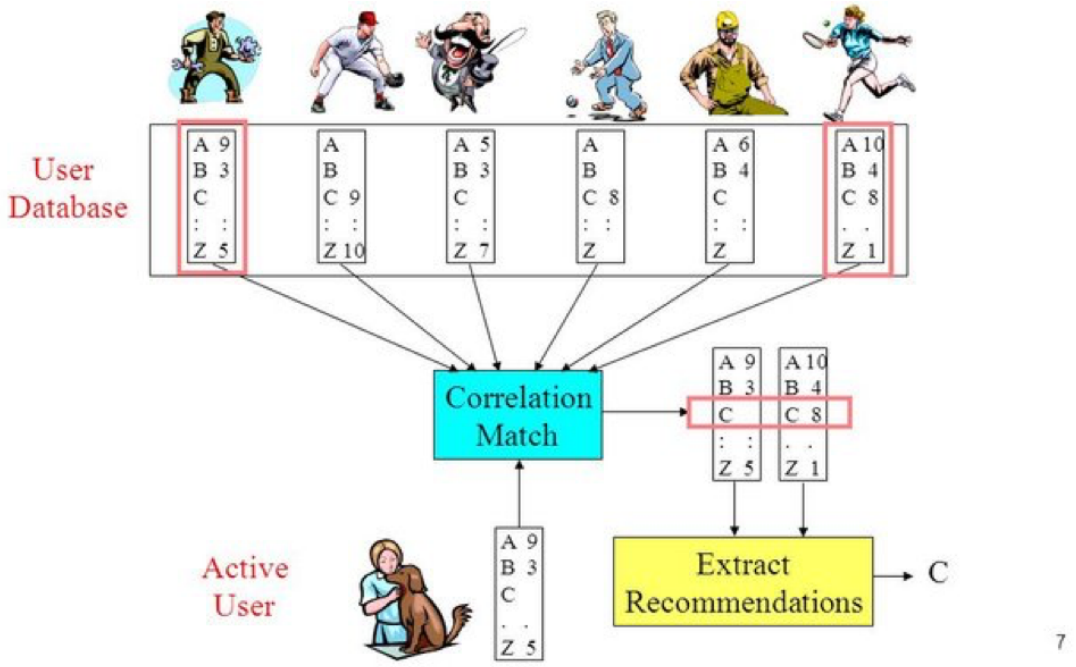


Figure 2.1: Figure showing the working of a Collaborative filtering algorithm. Image source: [Cha17]

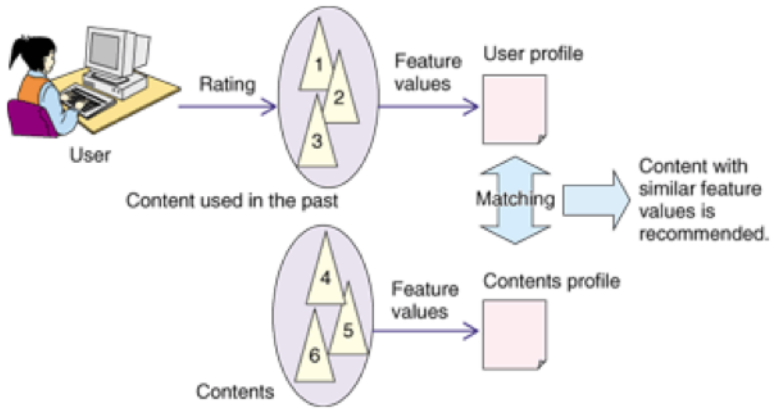


Figure 2.2: Figure showing the working of a Content based filtering algorithm. Image source: [Cha17]

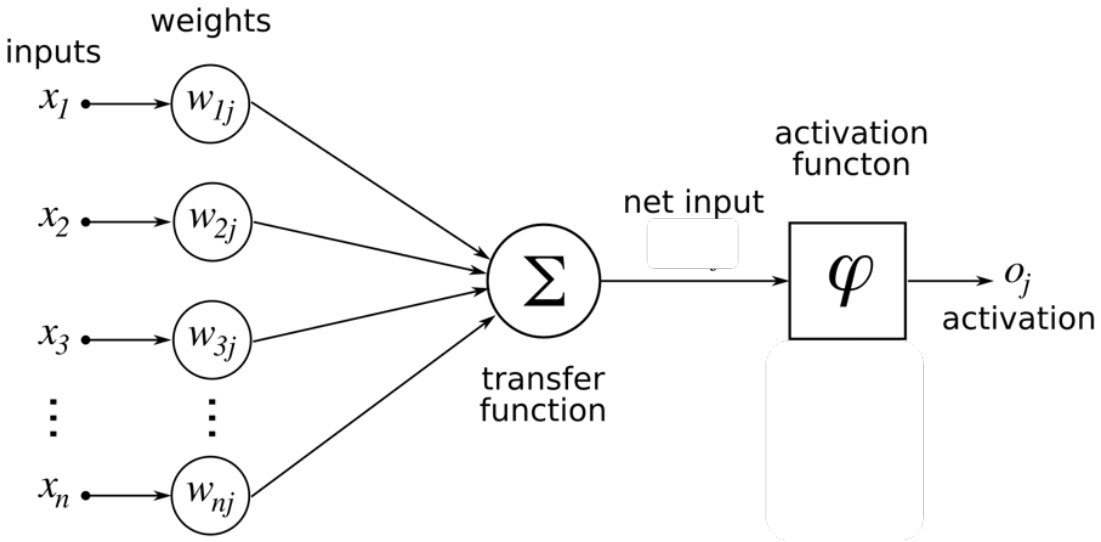


Figure 2.3: Basic architecture of an artificial neuron in a neural network. Image source: [Cas]

2.1.1 Neural networks in recommendation systems

Neural networks

Neural networks (Artificial neural networks) are a framework for machine learning models where the basic architecture is a collection of artificial neurons connected together (aggregated in a layered structure). These neurons receive and process signals and then signal additional artificial neurons connected to them (basic architecture shown in Figure 2.3).

Deep-learning neural-networks

Deep-learning networks are representation-learning models with multiple (neural) layers of representation. These non-linear modules transform the representation (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations (layers), it is able to learn complex functions. Figure 2.4 shows how a deep neural network learns and classifies animals, given their image as the input.

Motivation for neural networks in recommendation systems

The conventional models used in recommendation systems have limitations on the complexity of the model, the types of inputs etc. A lot of these problems have been overcome by using deep learning/neural models. The main advantages of using neural-networks in recommendation systems are[Zha18a]:

- Unlike other algorithms, neural models are capable of capturing complex dependencies and non linearities in large sets of data.
- They can incorporate additional data when training and can learn from large amounts of auxiliary information, which is usually available to recommendation systems. For example, in content based filtering, convolution neural network models can be used to learn intricate user-item relationships.

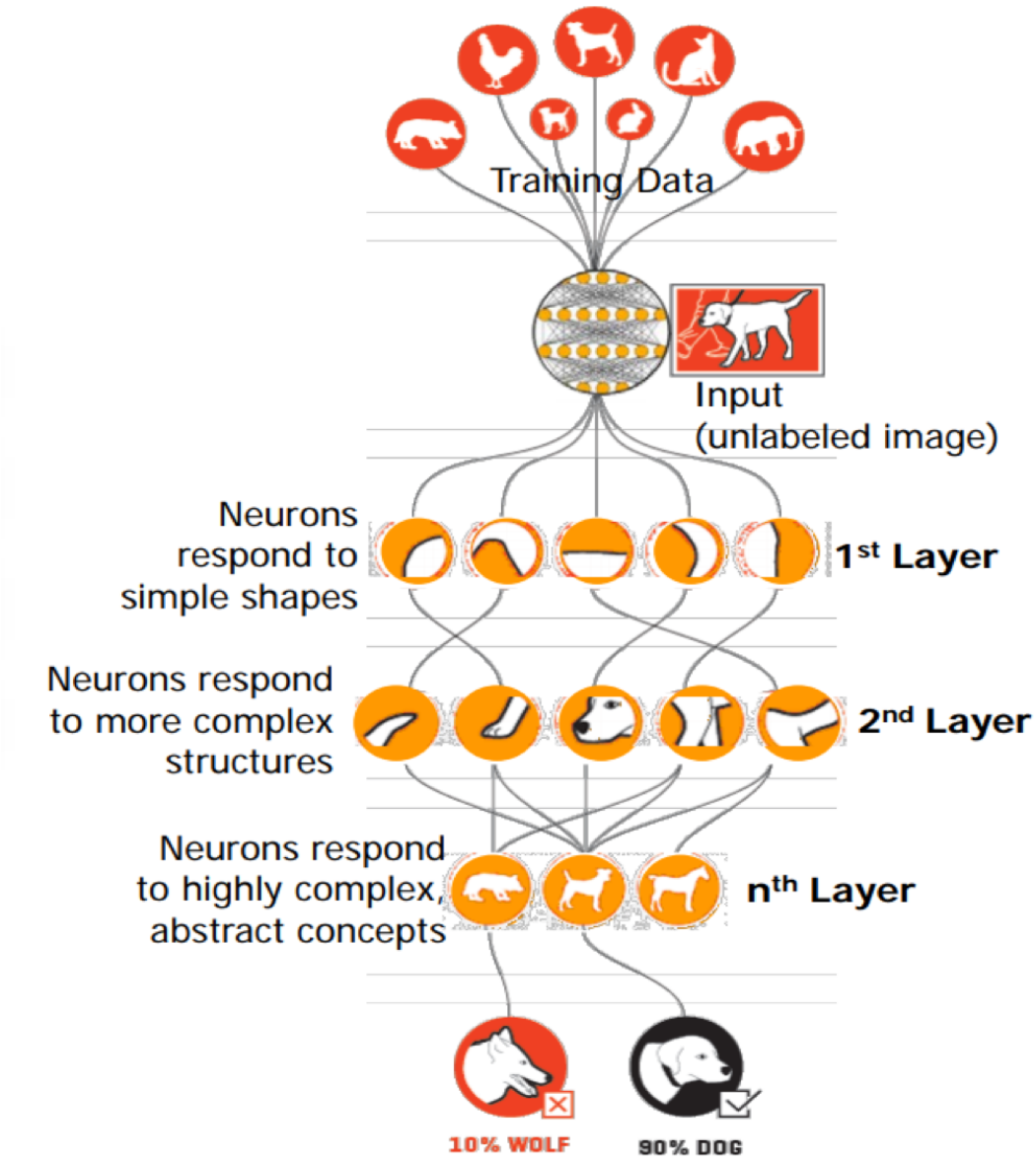


Figure 2.4: Working of a (deep) neural network for classifying animals in images. Image source: [Gun17]

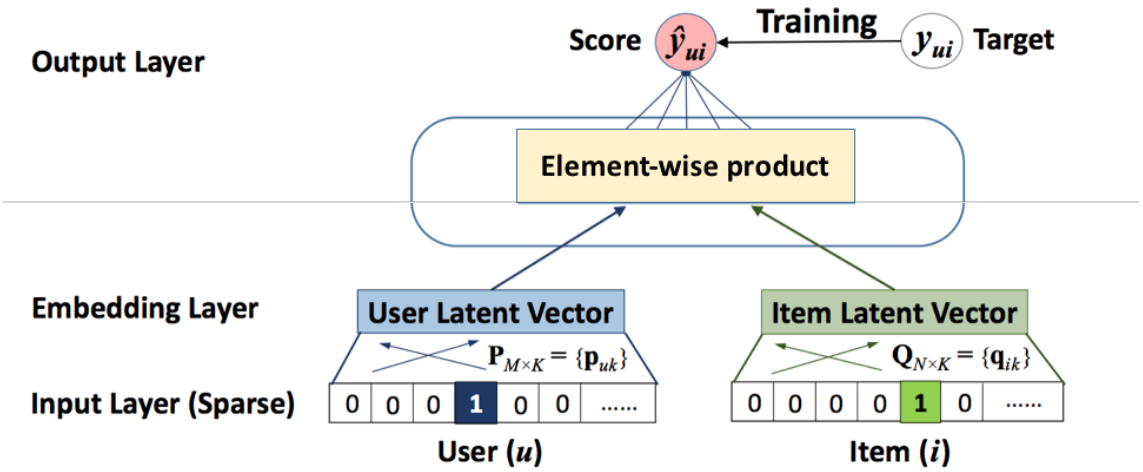


Figure 2.5: Generalized Matrix Factorization model architecture.

- Neural network models are more flexible and modular than other algorithms which are designed for particular recommendation problems. Frameworks like Tensorflow, Keras etc. make it easy to re-use and combine different neural structures and formulate powerful hybrids capable of capturing multi-modal features simultaneously.

- Particular DL techniques prove to be useful for some typical recommendation problems e.g. session-based recommendation systems can be modelled with recurrent neural networks[Hid16].

2.1.2 Models in focus

For our work in Chapter 3, we focus on collaborative filtering based neural recommendation systems. The primary idea is to learn representations for users/items from the implicit/explicit user interactions and then use those representations to predict ratings. More importantly, we will look into a subset of these models, which can explain their recommendations.

Genralized Matrix Factorization (GMF)[He17]

This is one of the simplest neural models and is able to mimic the Matrix Factorization[LS01, LS99] model. The model obtains user and item embedding vectors (via the Embedding layer in Figure 2.5) from the one-hot encodings of user and item ID. Then it takes an element wise product (also called Hadamard product) of the two latent vectors. Then it applies the weights and the activation function of the last layer to get a single scalar value representing the predicted rating of the item. One can deduce that the Matrix Factorization method is a subset of this model (hence, named Generalized Matrix Factorization).

Neural Collaborative Filtering (NCF)

This model can be seen as an extension to the GMF model discussed in the previous section. Here, instead of taking an element-wise product of the user/item latent vectors,

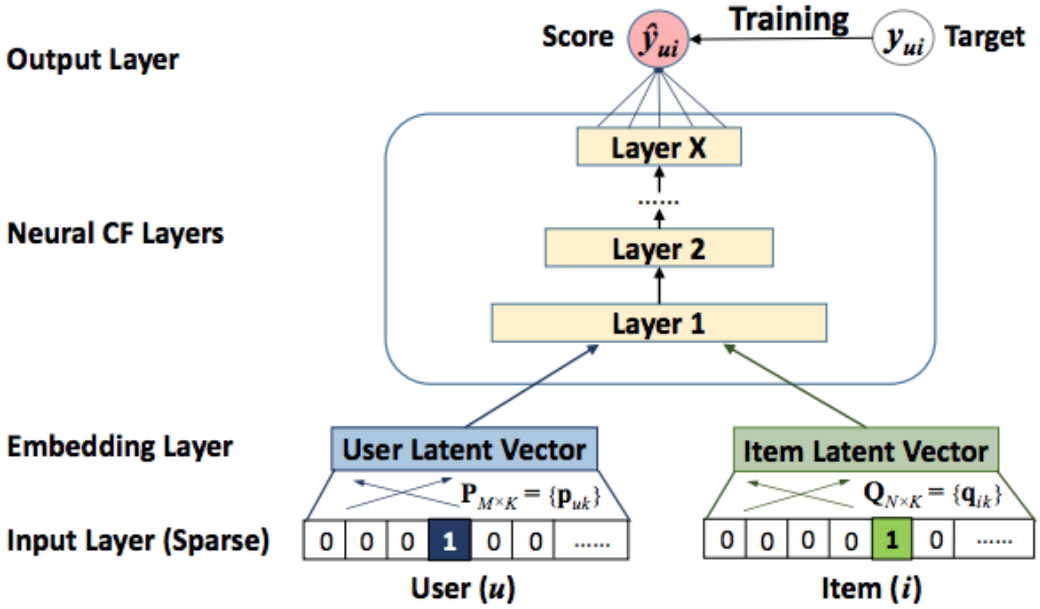


Figure 2.6: NCF model architecture. Image source: [He17]

the model concatenates them and add hidden layers on the concatenated vector, using a standard Multi Layer Perceptron (MLP) to learn the interaction between user and item latent features. This way, the model becomes more flexible and is able to learn more intricate relations.

In terms of the structure (shown in Figure 2.6), the MLP layers are stacked in a tower fashion, such that the bottom layer is the widest and each successive layer has a smaller number of neurons (see fig. 2.6). This way the model learns more abstract features with each successive layer.

Deep item-based collaborative-filtering (Deep ICF)[Xue18]

As the name suggests, deep ICF makes recommendations based on higher order item relations. Comparing to the NCF model discussed in the previous section, the deep ICF model differs at the embedding layer. Here, instead of representing users as a latent vector, it represents them as a group of item latent-vectors, one for each historical item that the user has interacted with (shown in Figure 2.7).

Next, in the pair-wise interaction layer, it takes the element-wise product of the target item's latent vector with each of the historical items' vectors. Since the number of historical items of different users may vary, the output of pairwise interaction layer will have different sizes.

Next, in the pooling layer, it produces a vector of fixed size to facilitate the deep interaction layers. This is done either via weighted average pooling or attention based pooling. The output of the pooling layer is a vector which condenses the second-order interaction between historical items and the target item. Similar to the NCF model, the higher order interactions are captured with a multi-layer perceptron.

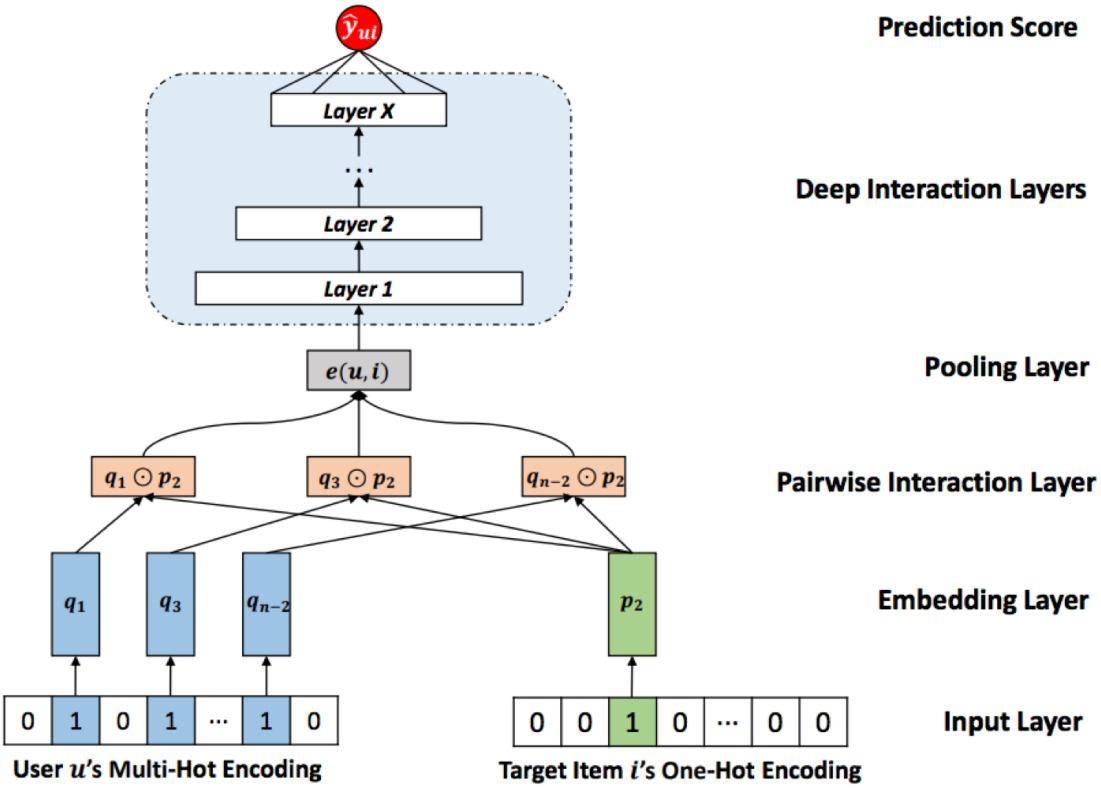


Figure 2.7: Deep ICF model architecture. Image source: [Xue18]

2.2 Explaining recommendation-systems

For recommendation systems, building models where the recommendation algorithm not only provides a recommendation list as output, but also naturally works in an explainable way and provides explanations to accompany the recommendations has gained some attention in the community[Zha18a, Zha18b]. The significance of explaining automated recommendations is widely acknowledged [HKR00, TM07]. Explanations also give users the opportunity to fix incorrect representations or recommendations. For these reasons, there has been significant research on ways to explain different types of recommendation systems. We refer the reader to [ZC18] for a detailed survey on explainable systems.

The main types of explanations that these recommendation algorithms can have are [Zha18b]:

- **User/Item based:** These recommendations are mostly for collaborative-filtering based models. This survey [Her00] discusses various ways of explaining a recommendation system to its users. Figure 2.8 shows an effective explanation which justifies a recommendation based on how the neighbors (similar users) rated the target item.

- **Feature based:** These are usually based on content based filtering algorithms where recommendations are due to some target item's feature that the user has explicitly (or implicitly) shown interest towards. For example, as we can see in Figure 2.9, the recommended item is visually explained by highlighting the decisive features in the item. Note that the highlighted parts can be different for different users.

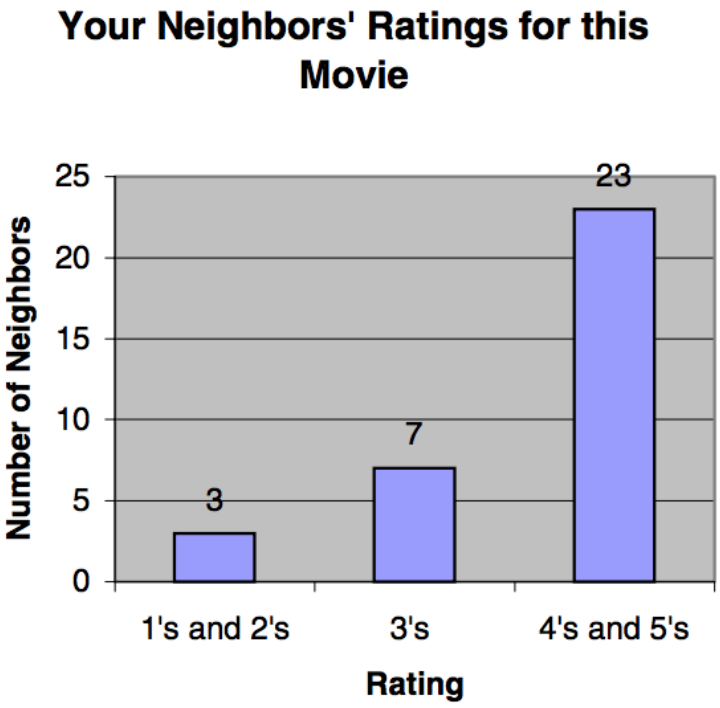


Figure 2.8: Explaining Collaborative-Filtering recommendations. Image source: [Her00]

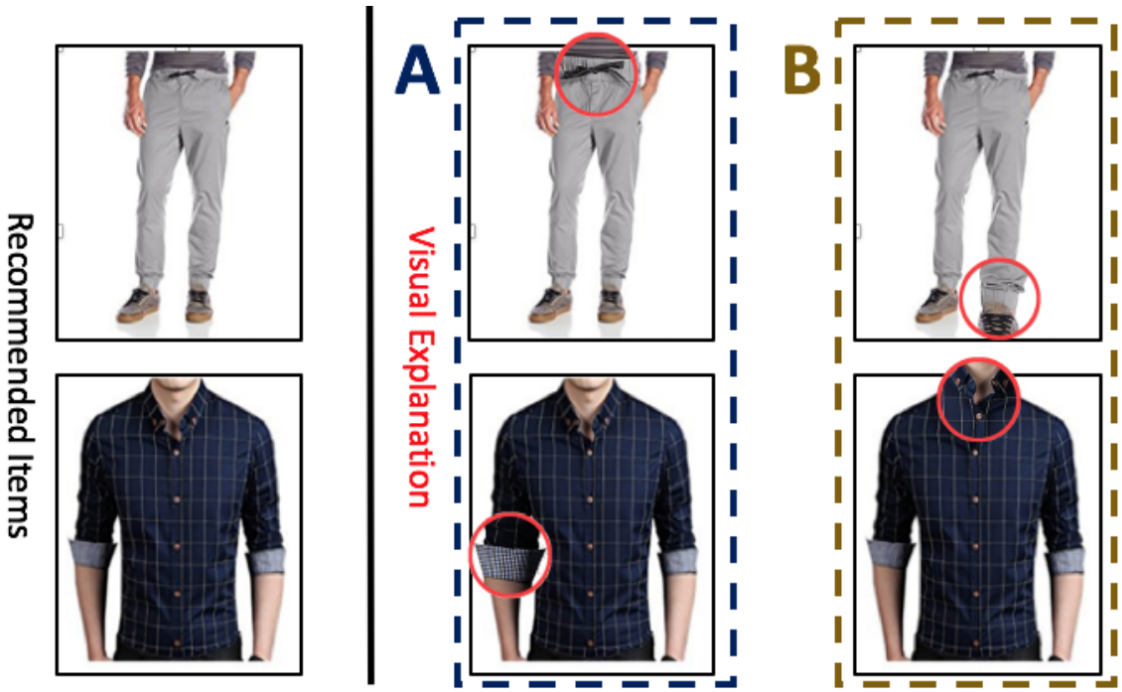


Figure 2.9: Personalised visual explanations for content based recommendations. Image source: [Che18]

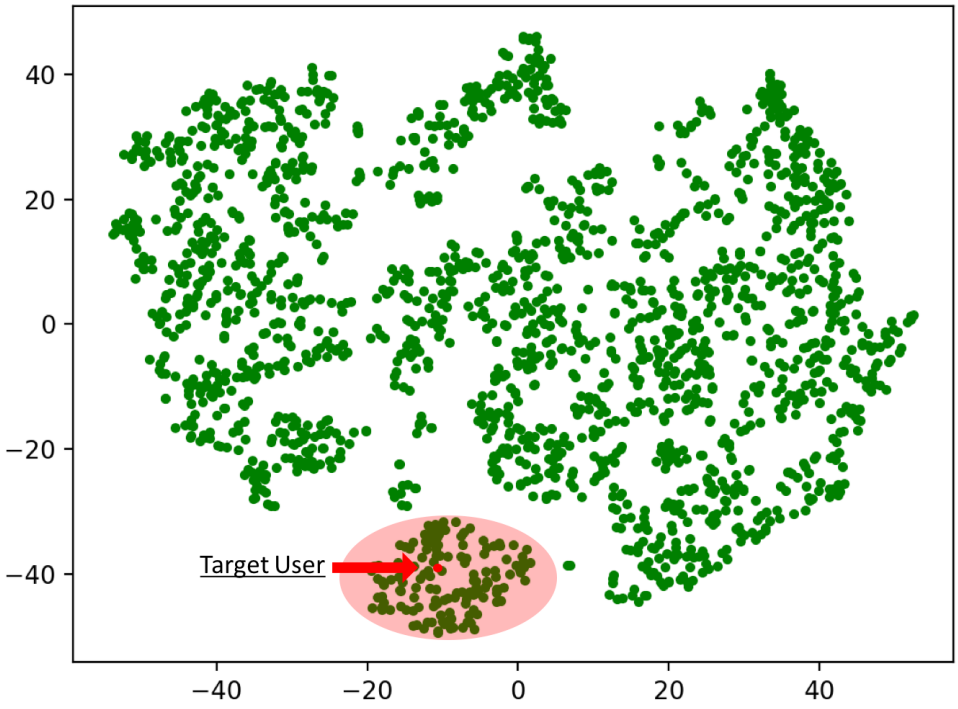


Figure 2.10: Using t-SNE for neighbor based explanations for NCF recommendations.

2.2.1 Explaining neural recommendation systems

Since explaining neural networks and explaining recommendation systems have been points of research for a few years now, their amalgamation has given us some interesting approaches to tackle our problem. In this section we will discuss a couple of techniques that have been proposed to explain their respective recommendation models.

Neighbour based explanations

As mentioned earlier, user/item based explanations work well for collaborative filtering based methods. We adapt the same technique to generate explanations for a model like Neural Collaborative Filtering. Here, we project each user latent vector (k dimensions each) onto a two dimensional plane. This can be done using techniques like PCA (Principal Component Analysis) or t-SNE (t-Distributed Stochastic Neighbor Embedding).

In Figure 2.10, clusters of users with similar interests can be seen. This way we can explain a recommendation based on the common interests of users in the same neighborhood as the target user. A similar technique has been used in this paper [Abd16] to further enhance the accuracy of the recommendation model, while making it more explainable.

Deep item-based collaborative-filtering with attention

As mentioned earlier, in deep ICF, one of the ways to perform pooling is to use an attention network on the pair-wise product vectors between the historical items and the target items' latent vectors.

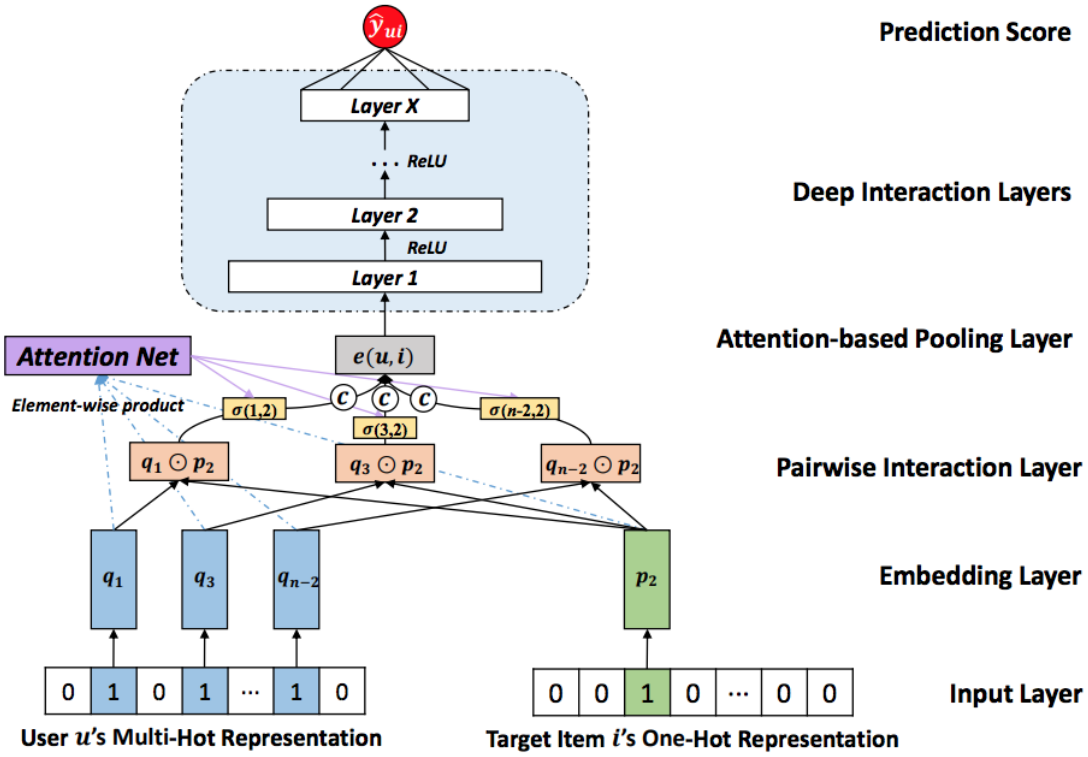


Figure 2.11: Deep ICF model architecture with attention based pooling layer. Image source: [Xue18]

With the attention pooling layer, the model is able to differentiate between the varying contributions of user's historically interacted items for the final prediction. For example, if the user gets recommended to buy a phone cover, the model's explanation should be the phones that they purchased before rather than the cameras or clothing products.

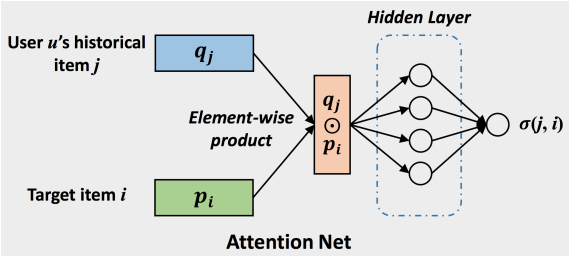


Figure 2.12: Illustration of the attention layer in the DeepICF model. The attention weights are being computed via another hidden neural layer. Image source: [Xue18]

Section 2.3 explains in detail how attention works and how it is used to generate local explanations.

2.3 Neural Attention-networks

Attention mechanisms[VSP+17], in neural networks, are known to provide the functionality for the model to focus on certain parts of the inputs or features.

Technically, the purpose of the attention later is to calculate weights corresponding to the input features. It consists of a similarity function f, which takes h as input (usually the

vector representations/embeddings of the input features). The similarity function maps h to scalar scores (one for each input feature). Then, the scalar scores are passed through a softmax activation function to get the final attention weights.

2.3.1 Additive attention

The similarity function in *Additive* attention[BCB14] is defined as:

$$f_{additive}(h) = v^T tanh(Wh)$$

2.3.2 Scaled dot-product attention

Scaled dot-product attention[VSP+17] works with the same setup as 2.3.1, except the similarity function f in this case is different.

$$f_{scaled}(h) = \frac{Wh}{\sqrt{m}}$$

where:

$h \in R^{K \times m}$

K : number of input features

m : size of input features

The softmax operation for both Additive and Scaled dot-product attention, remains the same.

$$a_k = softmax(f(h)_k) = \frac{\exp(f(h)_k)}{\sum_{j \in \{1,2...n\}} \exp(f(h)_j)} \quad \forall k \in (1, 2, ...K)$$

2.3.3 Attention in DeepICF

Figure 2.13 shows how the attention layer plays its part (in the case of DeepICF, and recommendation systems in general [HHS+18]) when the model is calculating its output. Here, a user U is represented by the list of items i_1, i_2, ..., i_n they have interacted with in the past. The purpose of the attention layer is to calculate a mask for each of these user input features such that the relevant bits of the input are highlighted for the rest of the model. This results in some input features having a bigger impact on the decision of the model than others. In case of the DeepICF model, these weights from attention layer are computed/learned by another hidden neural architecture which, in some sense, sits parallel to the model (as shown in Figure 2.12 and 2.13).

Here, the similarity function f maps each target item i_α, user input item i_x to a scalar distribution which then gets transformed to a probability distribution using a softmax operator. In the case of DeepICF the attention weights are calculated as:

$$a_k = \frac{\exp(f(i_k, i_\alpha))}{[\sum_{j \in \{1,2...n\}} \exp(f(i_j, i_\alpha))]^\beta} \quad \forall k \in (1, 2, ...n)$$

where:

$f(x, y) = h^T ReLU(W(x \odot y) + b)$

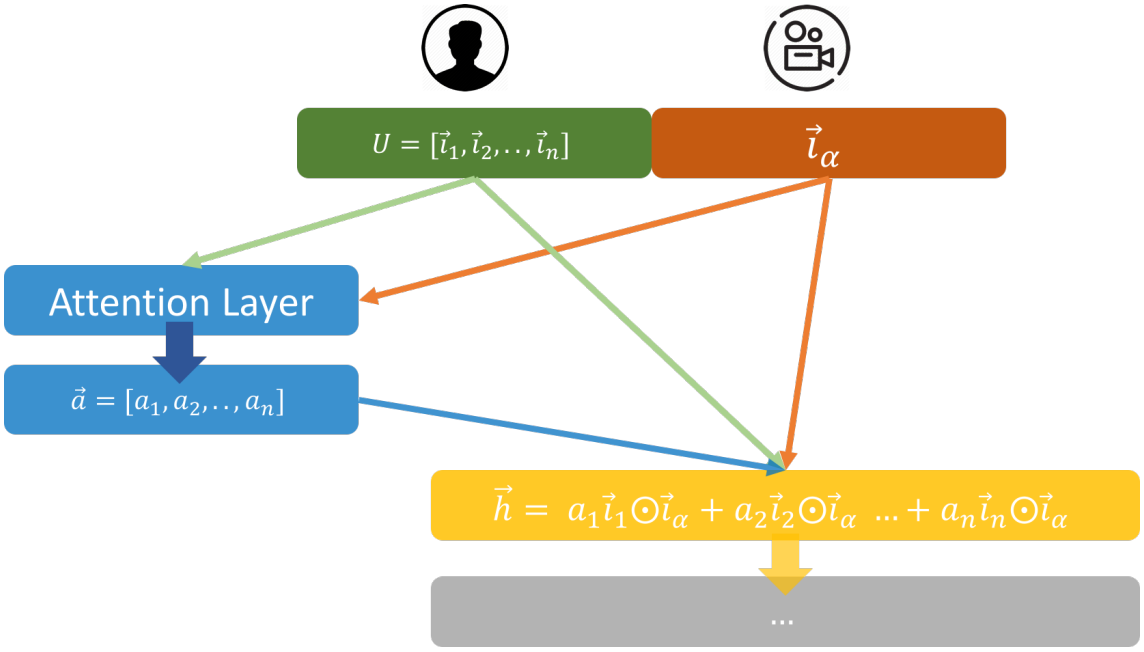


Figure 2.13: Figure showing the role of attention weights in the DeepICF model.

Note that β plays an important role here. β set to 1 would recover the original softmax function, while on the other hand, $\beta < 1$ would break the probabilistic explanation of the attention layer. We refer the reader to [HHS+18] for an explanation on the effect of different values of β.

These multiplicative masks calculated by the attention layer, intuitively, add a completely new dimension to the model. In a model without this attention layer, the input features could only interact with each other by addition. Adding this parallel neural architecture to compute weights for the input features expands the space of the functions that can be approximated by the model.

2.3.4 Explanation using Attention

In neural networks, attention is increasingly being used, not just to improve the model's performance but also as a means to explain the model's predictions. Claims about attention's interpretability are getting common in the literature, in various neural network applications like NLP[XBK+15, L+17, MA16, XMDH17], health-care[CBS+16, MWD+18], recommendation systems[Xue18, WHF+18] and others[GBY+18]. The attention maps (heat-maps) are used to indicate which input features to the model were majorly responsible for the model's predictions.

In Figure 2.14 (from a movie recommendation system from [XHW+18]), for instance, for target item #1525, the attention-network assigns the maximum weight to the input item #1254 (one of the previously interacted items of the target user). This information can be used to generate a human-readable explanation like "You are recommended to watch #1525 because you watched #1254".

More recently, there has been research on the reliability of attention-maps based explanations [JW19] and if they can be used to explain a model. In this project, we work on this line of research in the context of the model's calibration(2.4) and stability(2.5).

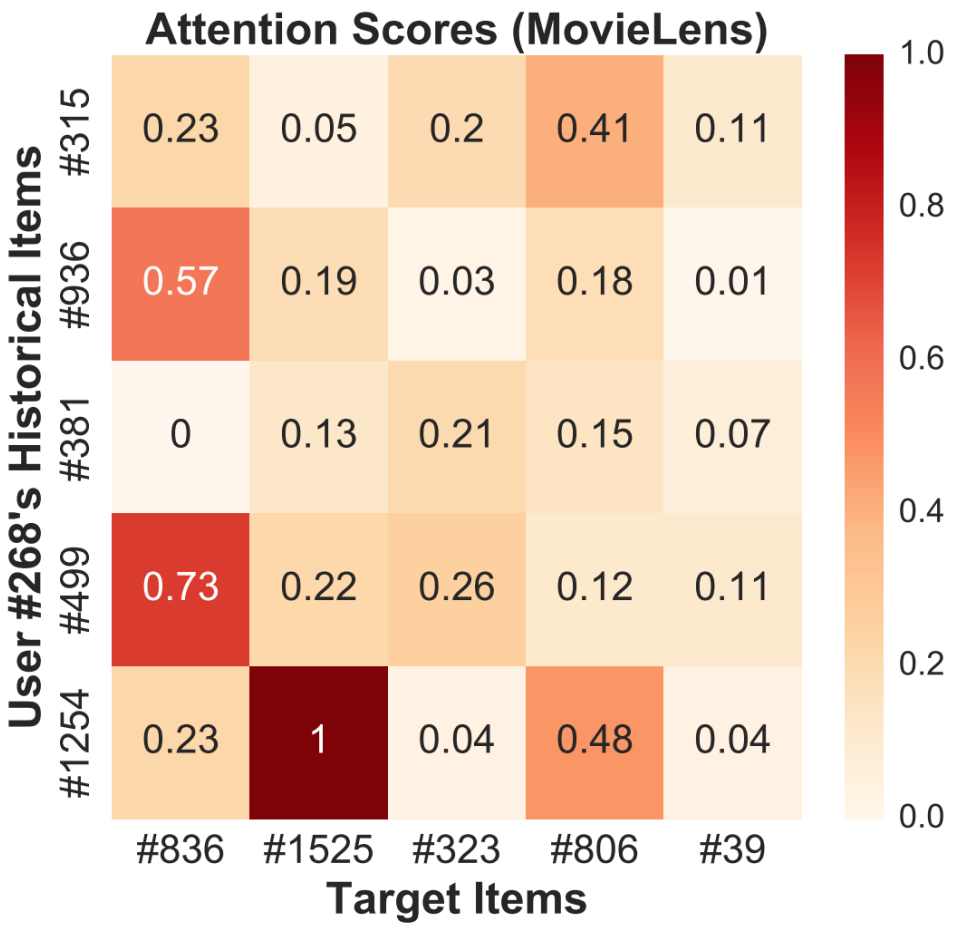


Figure 2.14: Attention map showing weights assigned to input features (historical items). Image source: [Xue18]

2.3.5 Attention is not explanation

One of the inspirations for this project is the work from [JW19]. As mentioned in Section 2.3.4, attention is increasingly being used to interpret models. In this paper, however, the authors argue about the unreliability of attention weights as means to explain a model's output. They perform various experiments on a range of datasets to show that explanations generated using attention distributions may not be stable/reliable. Figure 2.15 (taken from [JW19]), for instance, highlights the minimal effect permuting the attention weights has on the output of a binary classification model. Thus, signifying attention's un-reliable behaviour as means to explain the model's outputs. The authors perform various other experiments to showcase this issue. We would refer the reader to [JW19] to understand them in detail.

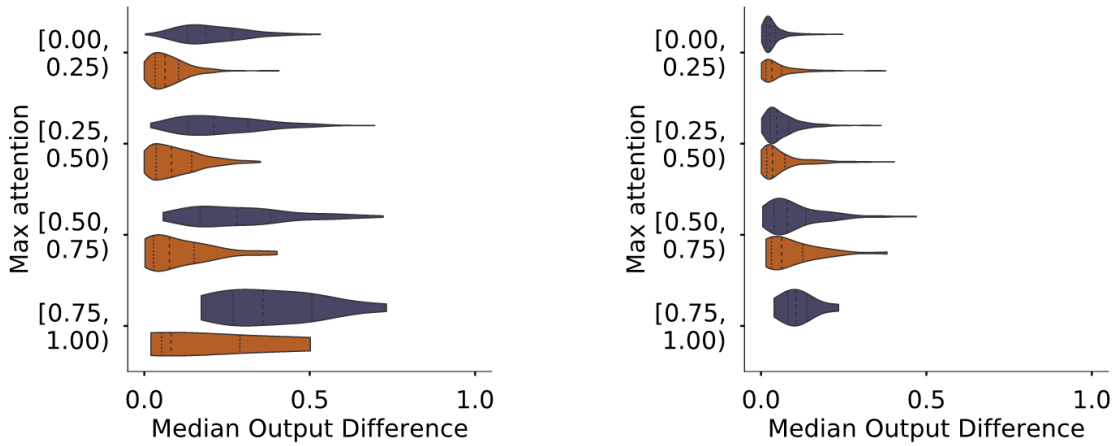


Figure 2.15: Median change in output (x-axis) densities in relation to the max attention (yaxis) obtained by randomly permuting instance attention weights (SST[SPW+13] as a binary classification dataset). Orange(■) represents instances predicted as positive, while blue(■) represents the negative ones. Image source: [JW19]

In our work (especially Chapter 4), we take a step back and look at attention instability as a part of a bigger problem, which is, model instability. We reason that an analysis on the model's stability (subject to random seeds – Section 2.5) precedes the question of attention reliability/stability (as shown in [JW19]).

2.4 Calibration of neural networks

Modern neural networks are not calibrated i.e, their probability estimates do not match with their accuracy[Guo17]. This can be an issue for real world decision making systems, like autonomous cars, where the model should not be overconfident about its decisions. This mis-calibration, as discussed in the paper [Guo17], can be caused by various factors like depth/width of the network, weight decay etc.

Classification models used as part of any decision process need to be both accurate in their predictions, and should also indicate when they are probably incorrect. Model calibration is the degree to which a model's predicted probability correlates with its true correctness likelihood. Calibration measures this property of a model. For example, if a perfectly calibrated model gives 100 different predictions, each with 80% confidence (probability), 80 of the predictions should be classified correctly.

There are a couple of techniques to test the calibration of a neural network model:

- **Reliability diagrams**: These diagrams plot the expected sample accuracy as a function of confidence. As we can see in fig. 2.13 below, the model in the left is well calibrated since the model confidence value aligns with the accuracy of the model. However, for the model on the right, the model accuracy is lower than the confidence values. This means that the model is over confident about its predictions.

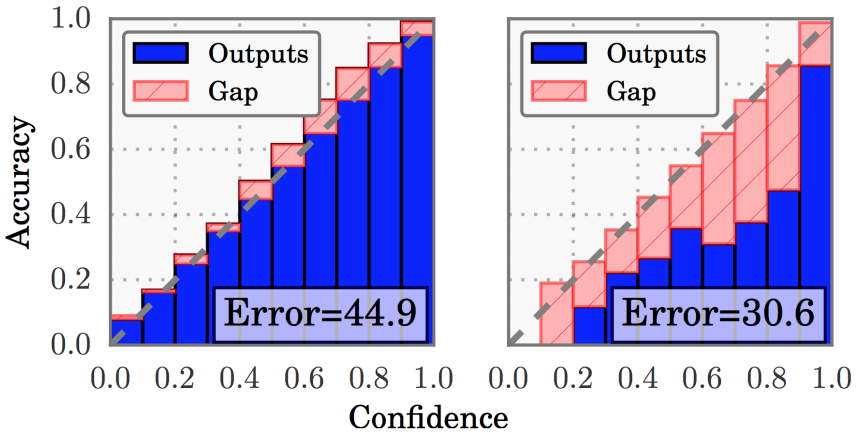


Figure 2.16: Reliability diagrams for well-calibrated (left) and poorly-calibrated (right) neural models. Image source: [Guo17]

- **Expected Calibration Error**[NCH15]: This is a scalar statistic for calibration, thus, is easy to use. ECE is just a weighted average of the gaps (absolute values) that are shown in the reliability diagrams in Figure 2.16.

Other metrics like Maximum Calibration Error and Negative Log Likelihood (NLE) [FHT01] have also been defined to do the same.

In this project, we plan on looking at the calibration aspect of neural recommendation systems in particular (Chapter 3). In the case of neural recommendation systems, due to mis-calibrations, the explainable models like deepICF might give us, not just overconfident recommendations, but also explanations that are unreliable.

2.5 Neural models as a function of random seeds

The problem of in-stability and uncertainty in models (in the context of randomly seeded trainings) has been examined in [ZKS+16] and it has been a point of discussion in the community. Most of the previous works on training the models with different seeds have been about analyzing the performance of the model.

In [EBC+10], the authors analyze the effect of random seed initializations to the training process of neural networks, especially their performance. Their primarily goal is to compare the performance of models with and without pre-training, which is not what we focus on in this project. Nevertheless, the authors highlight this problem with the variance of the test error achieved from 400 models trained with the same hyper-parameters but different random seeds. Figure 2.17 shows that there is a substantial variation in the final test error of these seeded models. This clearly highlights the problem that we are dealing with as this variance can also affect the stability of the model's interpretability.

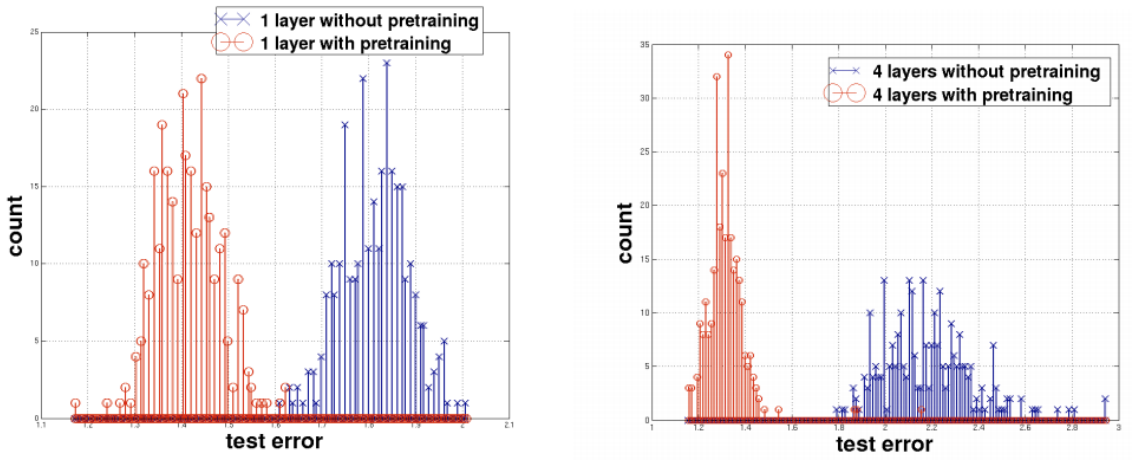
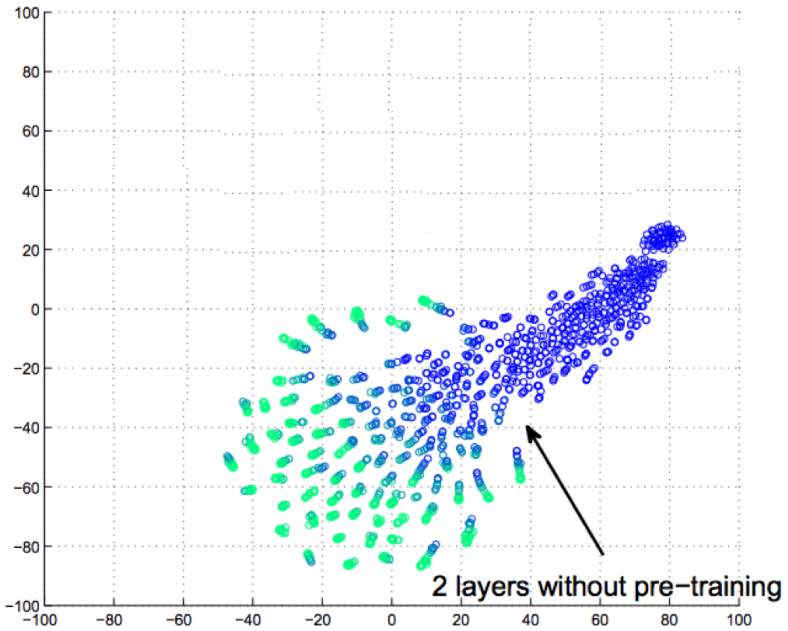


Figure 2.17: Figure from [EBC+10] showing the variance in the Test-Error for neural network model trained with the same hyper-parameters and different intialization seeds.

In Figures 2.18a and 2.18b (taken from [EBC+10]), the authors visualize the diversity in the weights learned for the same model, with different initializations. Each point in the figure represents a function learned (the state of the neural model, reduced to two dimension via Isomap[TDSL00] or t-SNE[MH08]) at the end of each iteration (after training on each batch b_i). It shows the trajectory of the learning process for the same model, with 50 different random seed initializations. The figures denote iteration progression from blue (■) to cyan (■) colored points. I would refer the reader to [EBC+10] to understand exactly how they represent the (neural-network) functions and perform dimensionality reduction techniques like t-SNE and Isomap on them.

(a) Figure showing the Isomap[TDSL00] of the functions representations, for neural network's state after each iteration.



(b) Figure showing the t-SNE[MH08] of the functions representations, for neural network's state after each iteration.

Figure 2.18: Figure highlighting the diversity in functions being learned by the same model, with different random seed initializations. The figures denote iteration progression from blue (■) to cyan (■) colored points. Image source: [EBC+10].

In [ZKS+16], the authors again discuss the robustness of neural models. They examine *explicit* ensembles, which is essentially training the same model (with same settings), under different random seed initializations. They use this approach to model the stability of the neural network. Another interesting approach, proposed by [BCKW15], called

Bayes-by-backprop can be seen as an *implicit* way to create an ensemble of models, since, here we sample models from a probability distribution on the weights of the model. I would refer the reader to [BCKW15] to understand the approach in depth.

In Chapter 4, we exclusively work on the model's stability towards random seed initializations, in the context of their interpretability.

2.6 Deep Learning based Natural Language Processing

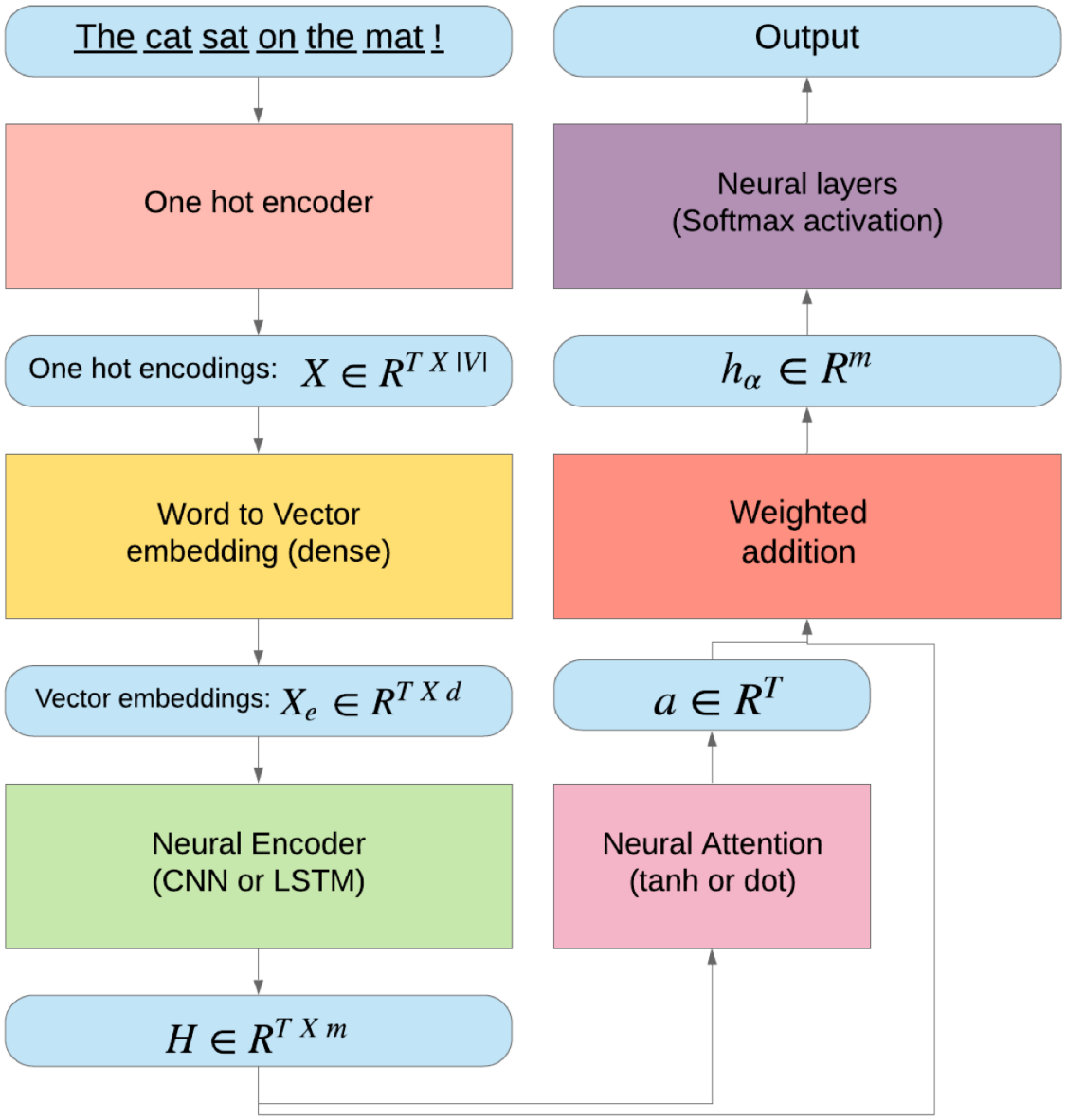
As mentioned earlier, in Chapter 4, we focus on the stability of neural networks based Natural Language Processing (NLP) models. In this Section we discuss the architecture of these models and the responsibilities of different layers in them. Our model structure, datasets and tasks are inspired from [JW19].

The basic model consists of three main parts: the encoder, the attention layer and the decoder. To start with, each word in the sentence is first converted to its one hot encoded vector. Next, these one hot encodings are converted to dense vector embeddings using a word to embedding matrix. There are multiple such word-to-vector representations publicly available for use like: GloVe[PSM14], word2vec[MCCD13] among others. The vector embeddings are then passed through the actual encoder layer which is either a CNN or an LSTM layer in our case. Both take the word embedding as input and output the hidden vector representations for each word.

Next, these hidden representations are passed through the attention layer which can be of two types in our case: `tanh` based attention or scaled-dot-product based attention. The exact implementations for both are given in Section 2.3. These attention layers assign a scalar weight to each of the words' hidden embeddings. After weighting the embedding with their attention weights, we are left with a single vector which are then passed through the final linear layer, which outputs the prediction. For binary classification tasks, a sigmoid activation function is used, while softmax is used for the multi label classifications. For binary, a binary cross entropy loss is used for training, while for multi-class/Question answering models, we use a log loss over the softmax output. Figure 2.19 shows clearly the role of each layer and the dimensionality of its inputs and outputs.

CNN based encoder Now, we look into the architecture of a CNN encoder to briefly understand what the model is learning. In the case of NLP, the *convolutional filters* slide through the words embeddings (instead of pixels, when used in the context of computer vision based applications). It is beneficial for us to understand briefly how the computations work in the case of CNNs. This can be done by understanding the dimensions of the input and the outputs in each step (from Figure 2.20). In the context of our project, it is not crucial to understand the working of CNNs in and out since we are more interested in the stability for their weights rather than how they help make accurate predictions. Nevertheless, I would refer the reader to [ZW15] and [YHPC18] to get a detailed explanation of CNNs and their application in NLP systems.

LSTM based encoder Long Short term memory (LSTM) networks are a variant of the Recurrent Neural networks (RNNs). The main strength of the RNN (and LSTM) networks is the ability to memorize the results from previous computations and use them in the current computation. This is especially beneficial when the inputs are of arbitrary length, and is also a shortcoming with CNN based models. LSTMs also fix the problem



T : *Number of words in input*

$|V|$: *Vocabulary size*

d : *Size of dense word emdebbings*

m : *Size of hidden layer*

Figure 2.19: Figure showing the basic architecture of our neural NLP model.

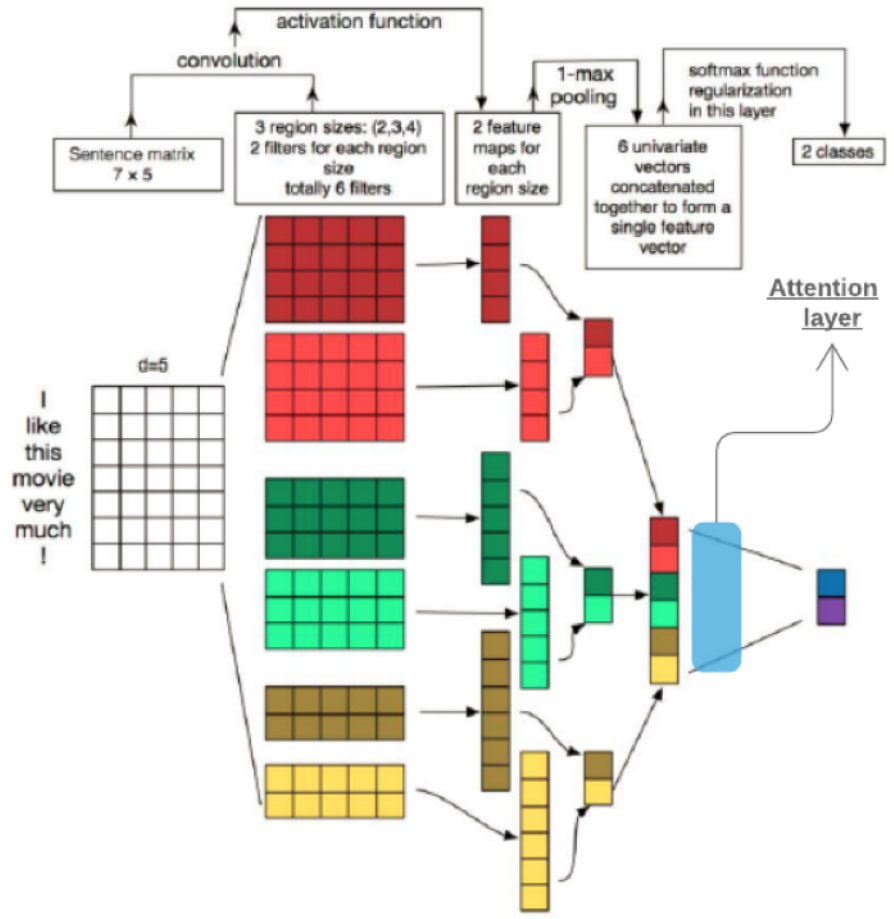


Figure 2.20: Figure showing working of a CNN encoder in an NLP model. Image source: [ZW15]

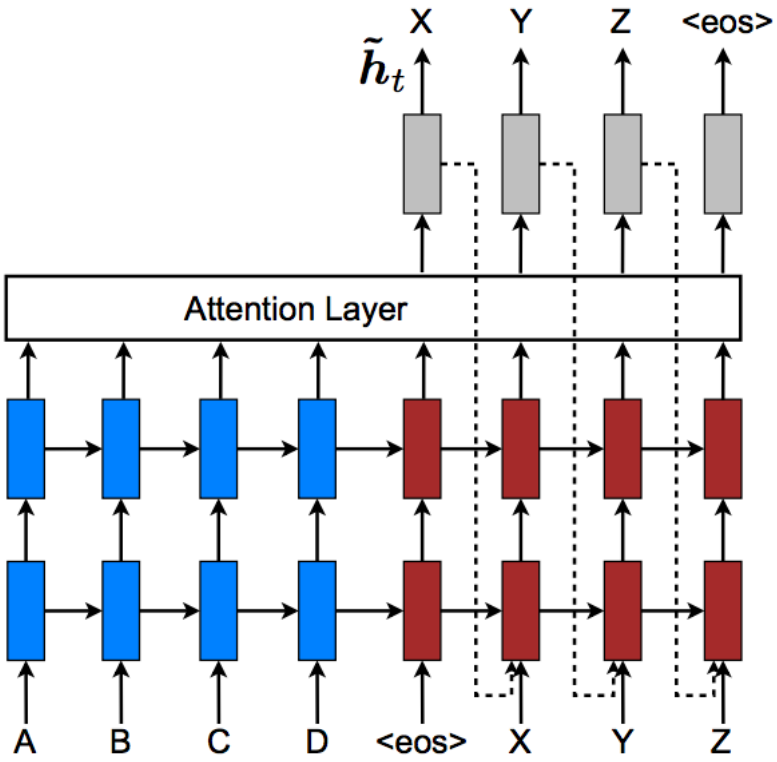


Figure 2.21: Figure showing the working of a Bi-directional LSTM encoder (with Attention Layer) in an NLP model. Image source: [LPM15]

of vanishing gradients with simple RNNs. Figure 2.21, shows the basic structure of a Bi-directional LSTM implementation. Again, it is not crucial for our project to understand the inner workings of these encoders since we are more interested in their stability aspect. I would refer the reader to [HS97] and [YHPC18] to get a detailed explanation on LSTMs and their implementation (especially in the context of NLP systems).

Attention Mechanisms The NLP models under consideration in this project use either *Additive* or *Scaled dot-product* based attention mechanisms, discussed briefly in Sections 2.3.1 and 2.3.2 respectively.

Chapter 3

Model explanations under calibration

Deep learning based recommendation systems have opened up one way of explaining neural models' outputs in the context of recommendations [ZC18] — by using attention distributions. In this context, neural attention mechanisms have gained significant focus, as they have been shown to not only help the model perform better, but also provide explanations by highlighting the input features that play a significant part in computing the model's output [XHW+18, GRMS99]. However, it is has been recently indicated that attention may not always provide a reliable form of explanation, especially in the domain of natural language processing [JMW19].

One of the emerging problems with the modern neural network models (especially deep neural networks) is their poor calibration [GPSW17]. Over-confident or under-confident predictions can make a model unreliable, especially in sensitive scenarios like health care (disease detection), autonomous driving among others [GPSW17].

In this chapter, we focus on a form of recommendation system that aims to answer *why* a certain recommendation has been made. Especially, we investigate the reliability of attention distributions in deep neural attention based recommendation systems.

3.1 Preliminary Work

In this chapter, we focus our preliminary analysis on *neural recommendation systems* explained in Section 2.1.2.

3.1.1 Models code setup

We setup the code provided by [Xue18] and [He17] on their respective implementations of DeepICF[1] and NCF[2]. Both the models are implemented in python. The NCF model implementation is using the Keras[C+15] library, while the DeepICF one is in Tensorlfow[AAB+15] itself.

Section 2.1.2 explains their architecture and gives a layer wise description for them. We decide to do our analysis on these models since they produce state-of-the-art results in recommendation systems[He17][Xue18]. The DeepICF model, is especially interesitng since it is able to explain its outputs using the attention layer (as explained in Section 2.3.4).

[1] https://github.com/linzh92/DeepICF
[2] https://github.com/hexiangnan/neural_collaborative_filtering

3.1.2 Dataset

We train, evaluate the models and perform our experiments on the MovieLens[3] dataset. This dataset has been commonly used to evaluate collaborative filtering algorithms. The dataset contains one million ratings where each user has at least 20 ratings and use the standard splits. In our study, we retain the standard procedure used in DeepICF (or NCF) where the original dataset is transformed such that each user item entry is marked as 1: when there is some interaction between the user and item and -1: when there is no interaction the between the target user and item. Table 3.1 shows the test and train split that we use for training. Another crucial feature of the dataset is its imbalance in the number of positive to negative interactions. During training, we have 4 negative samples (implying no interaction) for each positive sample while for testing we have 99 negative cases for each positive one. This imbalance is obvious since these models are *recommendation/filtering* techniques, thus, implying few positive cases amongst a lot of negative ones.

Dataset Splits	#Interactions	#Users	#Items
Full	1,000,209	6,040	3,706
Train	994,169	6,040	3,706
Test	6,040	6,040	1,921

Table 3.1: Statistics of the MovieLens dataset and standard splits.

3.1.3 Training

For training, the input to the model is a list of user item pairs, with the target label being if there was an interaction between the user and the item. As mentioned earlier, for each positive interaction, the model is trained with 4 negative interactions. Note that, we don't use the ratings given by the user to the interacted item (any rating, irrespective of its value, is taken as a positive interaction and a lack of one as a negative interaction). The output of the model is a score suggesting the likeliness of the user interacting with that item.

3.1.4 Evaluation

For evaluation purposes, the standard metrics used are HR@10 (HitRatio) and NDCG@10 (Normalized Discounted Cumulative Gain [HCKC15]). We further use the binary labelling accuracy to investigate the model performance per class, where the classes are defined as: -1 when there is no interaction between the user and items and 1 when there is an explicit interaction (user ratings for the item).

Hit-ratio calculation(HR@K): Lets denote the trained model by the function $f(u, i)$, where u is the target user and i the target item. In the test split, each test case comprises, a single user with 1 positive (interaction) test case, and 99 negative (interaction) samples for that particular user. Lets denote the positive interaction item as i_p, and the negative ones as $i_{n,1}$, $i_{n,2}$ ... $i_{n,99}$. Now, we calculate $f(u, i_{n,1})$, $f(u, i_{n,2})$ ... $f(u, i_{n,99})$ and $f(u, i_p)$. Then these function outputs are ranked in descending order. The test case is considered a *hit* if $f(u, i_p)$ is in the top K values in the sorted list. We do this for each of 6040 test

[3] https://grouplens.org/datasets/movielens/1m/

cases and calculate the average number of hits by dividing the number of hits by the total number of test cases.

NDCG calculation(NDCG@K): NDCG follows the same procedure as Hit ratio, except it accounts for the actual position of $f(u, i_p)$ in the sorted list as well. Lets say $f(u, i_p)$ is in the kth position in the sorted list. The NDCG for that prticular test case would be:

$$NDCG@K = 0 \qquad if\ k > K$$
$$NDCG@K = \frac{1}{\log_2(k+1)} \qquad if\ k \leq K$$

3.1.5 Hyperparameter Settings

For training purposes, we use the same hyper-parameters as mentioned in the paper [XHW+18]. Note that, we set β (discussed in Section 2.3) to 0.8 for training, as provided by DeepICF's implementation[Xue18][4].

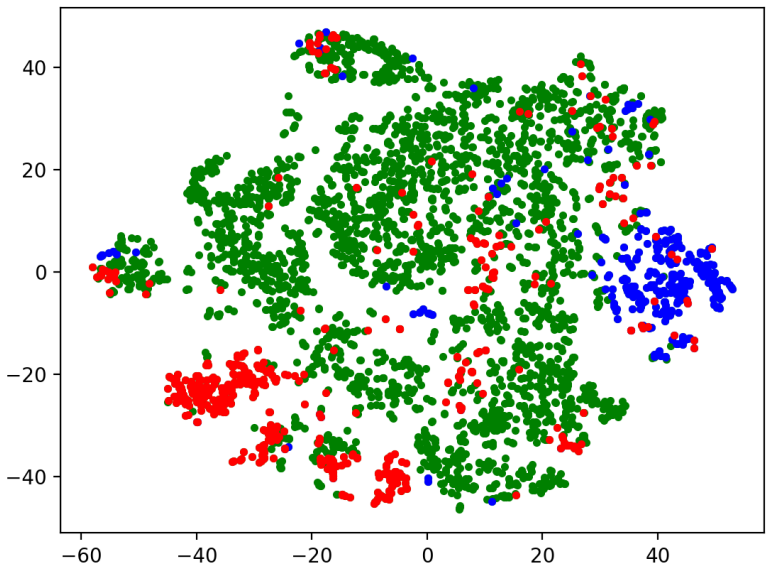
3.1.6 Embedding Vector Representations

In order to understand what the models are learning, we analyze the user and item embedding vectors learned by them. These vectors abstractly represent what the model thinks are the preferences for different users and the features of each items. In GMF[He17] (from Section 2.1.2), for instance, each user and item is represented by an 8 dimensional vector.

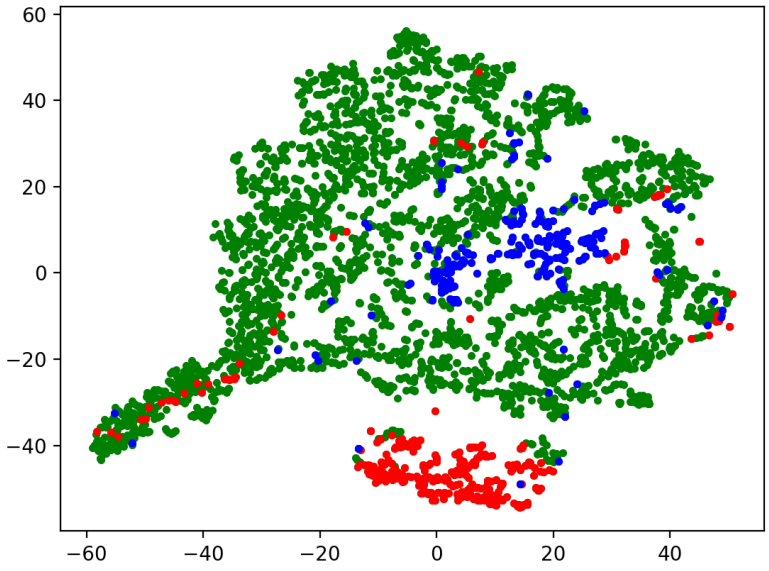
Item Embedding

First, we extract the embedding vector for all the items (movies, in our case) from the trained models. Then, in order to visualize them, we transform the 8 dimensional embeddings to 2 dimensions using different dimensionality reduction techniques like Principal Component Analysis(PCA)[Jol11] and t-SNE[MH08]. We decided to use t-SNE especially, since it is particularly well suited for visualisation of high dimensional datasets[MH08].

[4] https://github.com/linzh92/DeepICF



(a) Graph showing 2 dimensional item (movie) embeddings (from PCA) learned by the DeepICF model. The red(■) points are the *horror* movies, while the blue(■) ones are *action* movies.



(b) Graph showing 2 dimensional item (movie) embeddings (from t-SNE) learned by the GMF model. The red(■) points are the *horror* movies, while the blue(■) ones are *children* movies.

Figure 3.1: 2-D visualisation of user embeddings learned from the neural recommendation models.

In Figure 3.1b, we highlight the *horror* movies' embeddings (in red) and the *children* movies (in blue) to show how the model is able to group similar movies together just from the user item interactions. Note that, the model does not know the name, genre or any other auxiliary information about the items (movies). It is able to group similar

movies together, solely from the user item interactions during training.

User Embedding

As mentioned earlier, we transform the 8 dimensional user embeddings as well to 2 dimensions in order to visualize them on a 2-D graph. Figure 3.2 shows the user embeddings obtained from the GMF model, with the red dots highlighting the users of age 56 and above. Unlike the items, the user embeddings are not as clustered based on any of the auxiliary attributes in particular (or at least the patterns can not be highlighted by simple dimension reduction techniques).

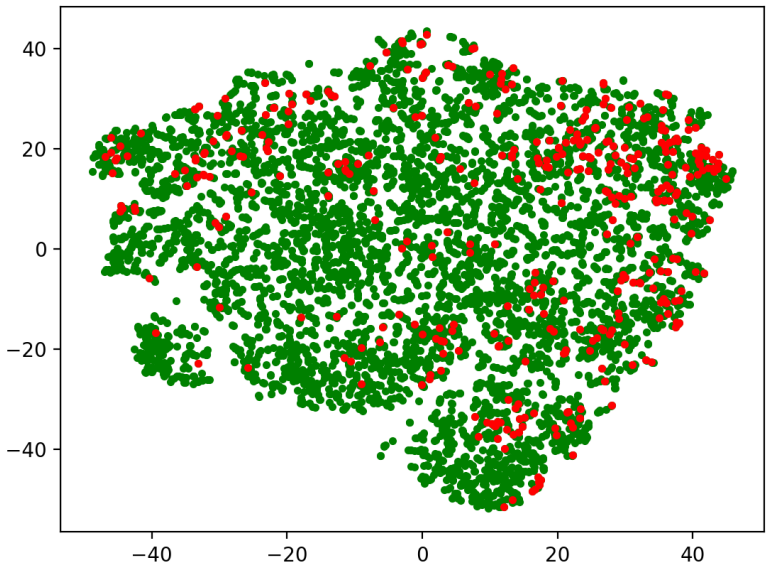


Figure 3.2: Figure shows the user embeddings obtained from GMF, mapped on to a 2-D graph using t-SNE. The red(■) points highlight users of age 56 or above. The model does not cluster the user embeddings (based on their age) as clearly as it did with the item embeddings. This signifies that the movie preferences of users (in the same age group) are diverse.

Another technique we tried in order to capture some of the intricate learnings of the model was to build a distance matrix for all the user embeddings in the dataset. We used Minkowski distance with p-norm 2 to calculate the distance between two embedding vectors, which is the same as the Euclidean[Wik19a] distance between them. We used the Scipy[JOP+] library's `spatial.distance_matrix` function to compute this. For each user, we arranged the rest of the user embeddings based on their distance from the user. This way we were able to extract the users which, according to the model, have similar behaviour/characteristics/interests. For each user we looked at their top 10 (and top 3) closest neighbors and were able to notice some patterns from it. In Figure 3.3, for instance, we observed an increase in the percentage of users belonging to the same age group, in the top 10 closest neighbors list for a particular user, compared to their general distribution in the dataset. Thus showing how users in the same age group get clustered together by the model.

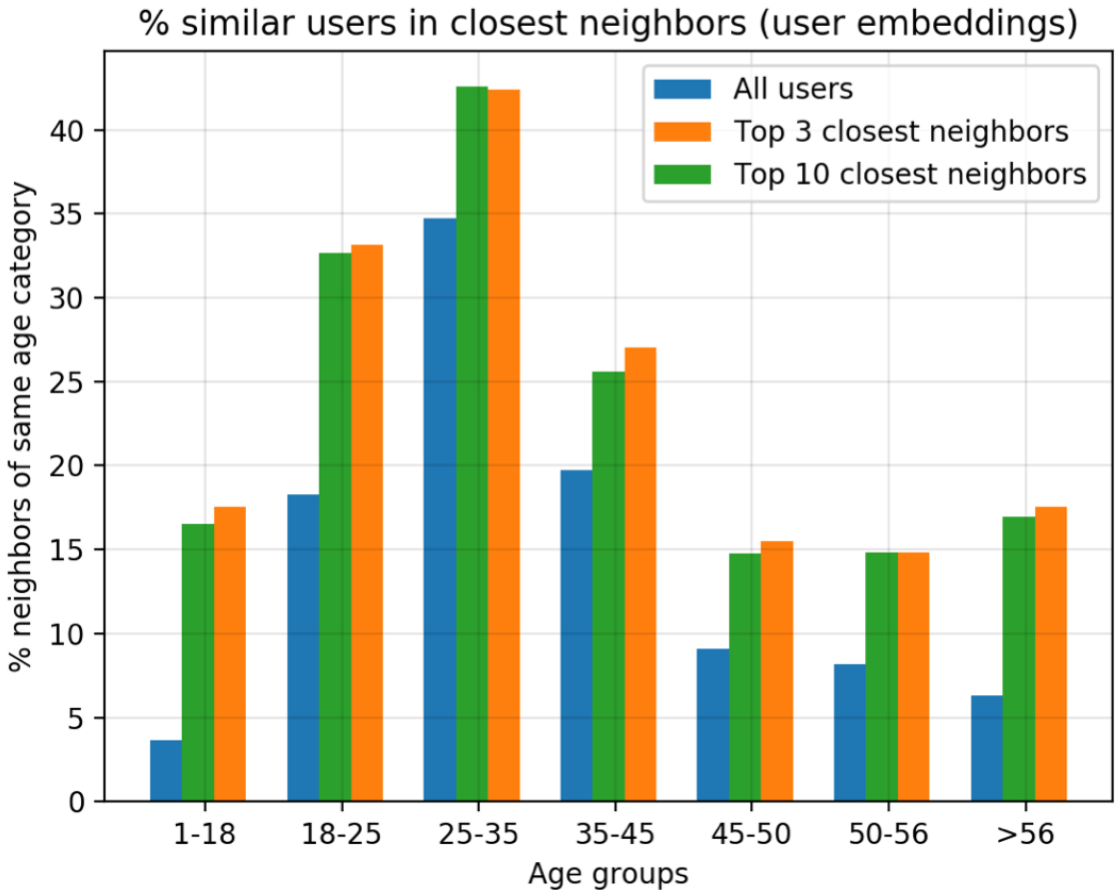


Figure 3.3: Figure highlights how the model is able to group similar (age group) users together after training. The closest neighbors to a user (embedding) is based on the **Euclidean distance**[Wik19a] to all the other user embeddings learned by the model.

All these techniques were able to give us a good idea about what the model is learning. They represent, abstractly, the model's behavior. Although these statistics provide good insight into the model, they are not enough to interpret the model or explain its outputs.

3.1.7 Sigmoid to softmax

One of the model dynamic that we look into is its *calibration* (discussed in Section 2.4). In order to plot the reliability diagrams for the NCF and DeepICF models, our first step was to obtain the prediction confidence from the model for its output. As explained earlier, both NCF and DeepICF use a sigmoid activation function to get the output score. We decided to change it to a softmax activation with two output classes to get the model's confidence values for its outputs.

Since DeepICF uses a log loss over the model outputs, changing the activation from sigmoid to softmax automatically changed the loss to a softmax cross entropy loss. Figure 3.4 shows how the loss is calculated in case of DeepICF, before and after we made the change in the activation function.

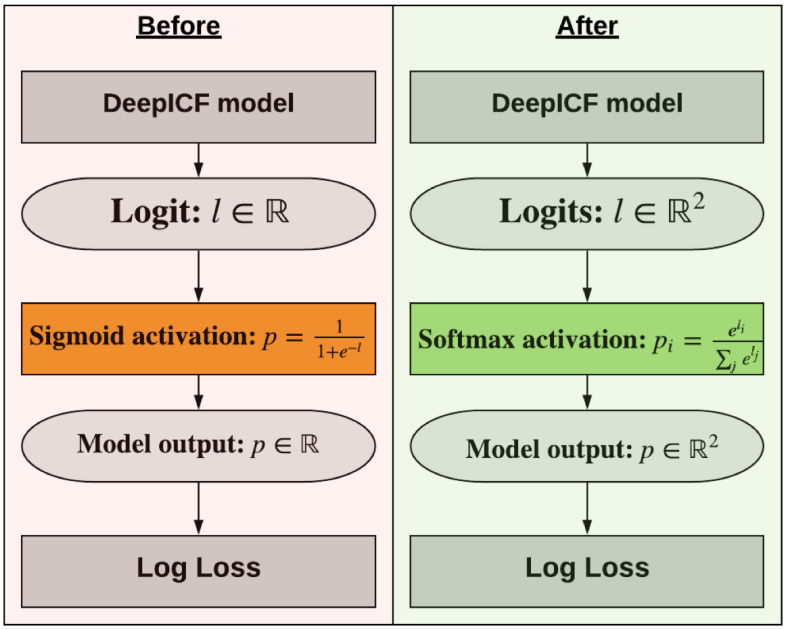


Figure 3.4: Figure showing how in DeepICF we change the activation function from Sigmoid to Softmax in order to get the confidence scores for the output labels.

3.1.8 Three output DeepICF

Another interesting experiment we tried with DeepICF was to account for the actual ratings as well when training the model. We changed the output of the model, from binary to a ternary classification problem where the three labels would be: negative interaction, no interaction, positive interaction between the user and the item. In order to implement this, we first looked into the distribution of the ratings in the training dataset. As we can see in Figure 3.5, the mean rating is around 3. We decided to go with 2 and 1 being *negative* (interaction) ratings, 3 4 and 5 being *positive* ratings (no interaction cases remained the same). Table 3.2 compares the performance of this model (DeepICF 3-output) with the original model and our other proposed variations to the model.

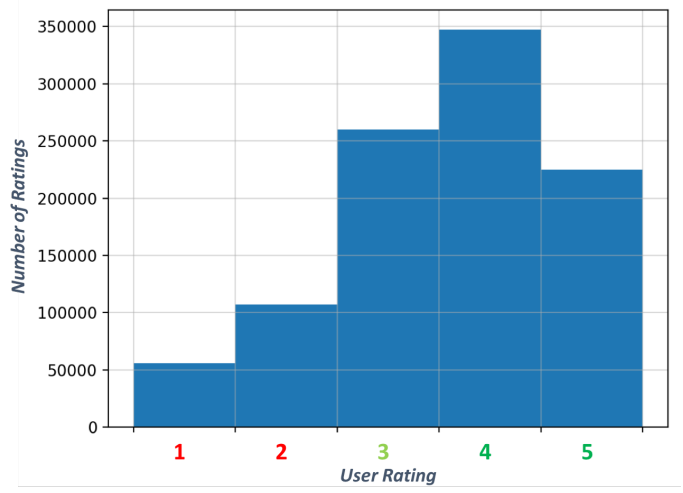


Figure 3.5: Figure showing the frequency of each rating in the **MovieLens 1M** dataset.

3.2 Problem statement and hypothesis

After having done the preliminary work, we were able to get an idea about how neural recommendation systems work in general, the nature of data they work with, and what they were learning. With our embeddings based experiments, we were also able to get a brief understanding on the model's room for explain-ability.

To start our analysis, we moved on to testing more intricate details about the models. Since our focus is on recommendation systems that aim to answer *why* a certain recommendation has been made, we perform our analysis primarily on the DeepICF model.

In the following sections, we try to discover if there exists a link between a model's calibration and its explain-ability. We discuss about the reliability of attention distributions as a means to explain deep neural based recommendation systems, especially in the context of un-calibrated models. We explain various experiments we conducted on the DeepICF model, the relevance of those experiments, and our deductions from the results.

3.3 Experiments

We investigate our hypothesis on the utility of attention with a state-of-the-art deep neural network based model with attention [XHW+18]. In this section, we describe the experiments performed for those purposes.

3.3.1 Calibration

As mentioned in section 2.4, we use the concept of calibration to plot reliability diagrams [Ham97]. A reliability diagram can be seen as the accuracy of the model as a function of its confidence.

Reliability diagrams help us visualize a model's calibration. A reliability plot which falls below the identity function suggests that the model is over-confident of its predictions (blue plot in Figure 3.7) since it means that the ground truth likelihood (accuracy) is less than the model's confidence in its predictions. On the other hand, it is considered under-confident if the reliability plot is above the identity function. For a perfectly calibrated model, the reliability plot is the identity function.

Interpretation of the DeepICF model

First, we need to have a clear understanding of the model and its output before we start looking into the plotting the reliability diagrams.

As mentioned earlier, we have an imbalance in the number of positive and negative instances, both when training and testing. Moreover, in the actual MovieLens dataset, there are only *positive* interactions. The *negative* interactions are obtained by sampling random items from the entire dataset (while making sure that they are not part of the positive interactions). This is done for both the training and the testing datasets. This sampling for negative cases is acceptable during training as these cases are only being used to train the model, which is not the point of concern for our project.

In the case of the test dataset, 99 negative test cases are sampled for each positive test case. The evaluation technique being used, commonly known as *leave-one-out*, is widely used in literature[BHKR17, HZKC16, RFGST09]. This is also acceptable since it uses specialized metrics like *HitRatio* and *NDCG* to test the performance of the model.

However, this suggests that, with the given test dataset (with randomly sampled *negative* interactions), metrics like Binary classification accuracy can not be used to evaluate the performance of the model. This also lead us to believe that the semantics of the model's output value do not match with those of a typical binary classification model.

Our Approach Given the understanding of the model (in Section 3.3.1), we decided to plot the calibration curve for the positive and the negative test cases separately. One of the main reasons we decide to do this is because we want to focus more on the positive interactions than the negative ones (especially because the negative interactions are randomly sampled out of all the remaining un-interacted items).

For instance, lets take a case with target label t and prediction for the positive label as p (note that since we are using softmax activation, the prediction of the negative label would be $1 - p$). If the target label is positive and $p > 0.5$, the test case is classified as *correct* while if it is negative, it is classified as *incorrect*. Similarly, if the target label is negative and $p < 0.5$, the test is classified as *correct*, otherwise *incorrect*. Note that when $p < 0.5$, $1 - p$ is used as the value to be bucketed. After computing that for all the test cases, we calculate the ratio of *correctly classified* test cases for each bucket separately and plot them against the bucket's score range (as shown in Figure 3.7).

Note there is a subtle but crucial difference between this approach and the general approach mentioned in [Guo17]. The x-axis (bucket scores) in the general approach represents the prediction score of the positive label (or the negative label), while, in our approach it represents the prediction confidence in the model's output (be it positive or negative). Due to this the x-axis in Figure 3.7 starts from 0.5. In the experiments, we focus on our approach for plotting the calibration for the reasons mentioned earlier.

3.3.2 Attention Permutation

Another important experiment we perform to check the reliability of attention based explanations is permuting the weights randomly and recording the effects of the permutations on the output of the model (inspired from [JW19]).

Figure 3.6, compared to Figure 2.13, shows how the attention weights are shuffled. Since the particular weights assigned to the input features are used as the basis for the explanations, permuting these weights randomly should cause the model's prediction to change by a substantial margin. In case the predictions remain unchanged it indicates that the attention necessarily doesn't contribute to the predictions. This can be concerning especially when using attention as grounds for explanations (as shown by [JW19]).

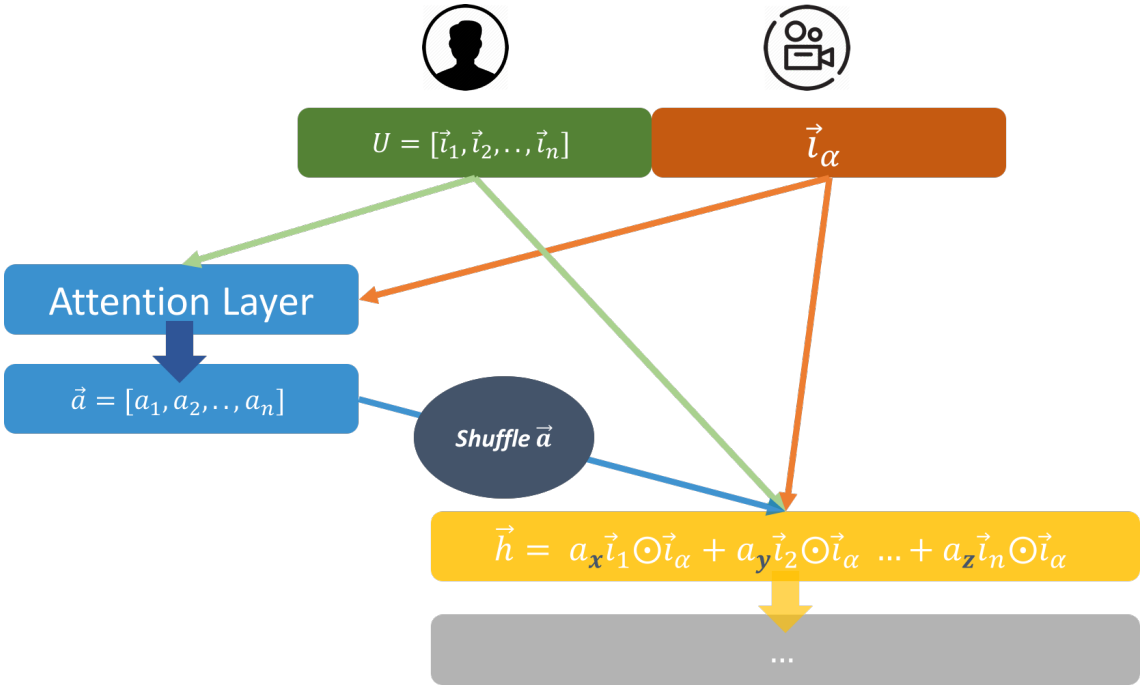


Figure 3.6: Figure shows how attention weights are shuffled in the DeepICF attention permutation experiment.

3.3.3 Class balancing loss

As the training split of the dataset is heavily imbalanced: 4 negative labels (no interactions) for every positive label, we use a simple class-weighting heuristic, to cope with this imbalance and modify the model's cross-entropy loss. The new loss is calculated by assigning weights to the losses from the test cases such that the loss contribution from both the classes (positive and negative interactions) is balanced[KZ01]. We perform our experiments on the class balanced loss model and measure any improvements in the results.

3.3.4 Model Stability

In our study, we refer to model stability as the consistency of model predictions and internal parameters with different runs of the model by only changing random seeds. [Jia03, Jia07]. The seed values are responsible for regulating the training dynamics (weight initialization, training batch generation, among others). This way, we get to measure the impact of these random processes on the output of the model (and the attention weights). We do a more detailed analysis on this model dynamic later in Chapter 4.

3.4 Results

Table 3.2 compares the performance of the softmax output model with the original DeepICF model and the state-of-the-art Neural Collaborative Filtering model[HLZ+17]. We observe that the performance of our model is highly competitive and performs as well as the DeepICF with pretraining. In the following sections, we will investigate the reliability of models and the attention distribution in the models.

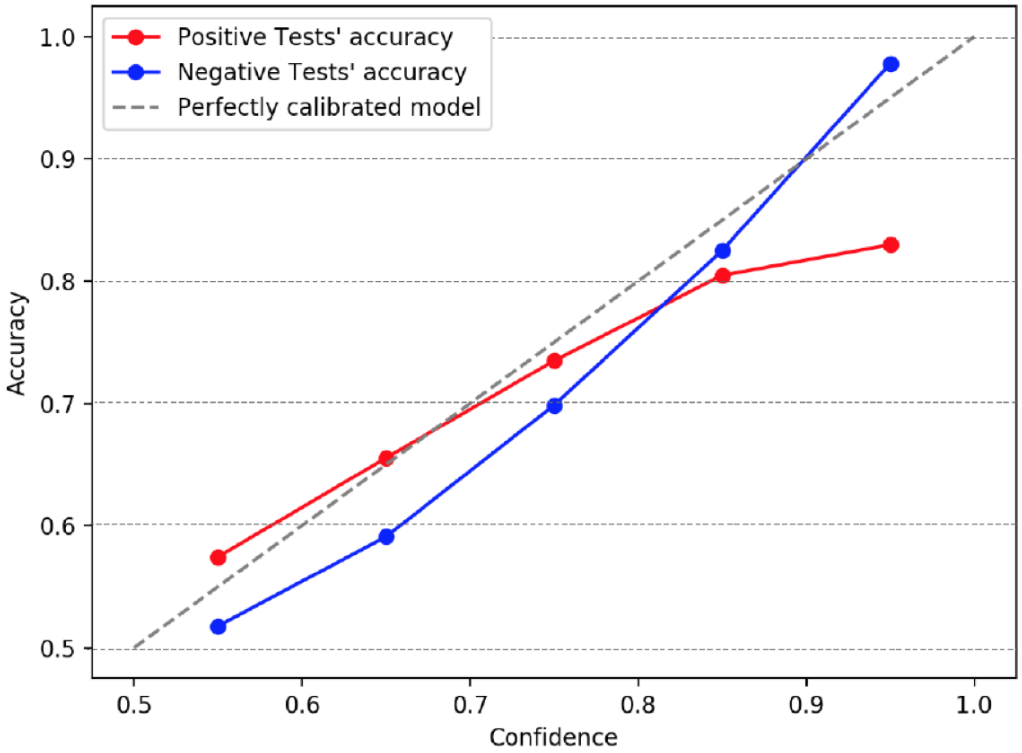


Figure 3.7: Calibration plot (**Our Approach** from Section 3.3.1) for the DeepICF model (softmax output), with positive and negative test cases plotted separately.

3.4.1 Calibration

We plot the reliability diagram for the DeepICF model by bucketing the model predictions based on their confidence and calculating the accuracy for each of the buckets (as explained in Section 3.3.1).

We see in Figure 3.7, for the positive label, the DeepICF (with attention) model tends to be over-confident as the confidence increases, where the model tends to be extremely confident about predicting the positive class without being as accurate. This can be problematic especially when dealing with real-world production systems (like medical diagnosis and autonomous cars). We also notice that the model is seemingly over-confident in predicting the negative class. This could be because of the imbalance in the training dataset where the dataset is extremely skewed towards the negative class.

3.4.2 Attention Permutation

What is the effect of over-confidence over attention? In order to test the reliability of explanations generated from attention, we permute the attention weights randomly and notice the effect of the permutation on the output of the model (as described in Section 3.3.2).

Specifically, in DeepICF, as shown in Figure 2.11 (and Figure 2.13), the attention based pooling layer assigns a weight for each of the user and item interaction, where the magnitude of the weights indicate the importance of the interaction. In this experiment, we randomly *shuffle* these weights amongst the items and record the difference in the output prediction score (originally classified interaction label). We randomly shuffle the weights 100 times (as performed in [JW19]) for each test case, and average the absolute

variations in the output predictions.

We plot the average variations in false negatives (right axis) against the confidence of the predicted output for the positive test cases in Figure 3.8. We focus on positive test cases as it is the most salient label to measure the model, and also for the reasons mentioned in Section 3.3.1. The plot also contains the reliability diagram for the model (left axis). We note that the perturbations especially have barely any effect on the miscalibrated cases. In both false positives and false negatives (these increase with miscalibration), we notice similar trends where the effect of permuting the attention weights decreases as the confidence in the predicted label increases. Thus, showing that model explanations generated from the attention distribution become less reliable with overconfident predictions.

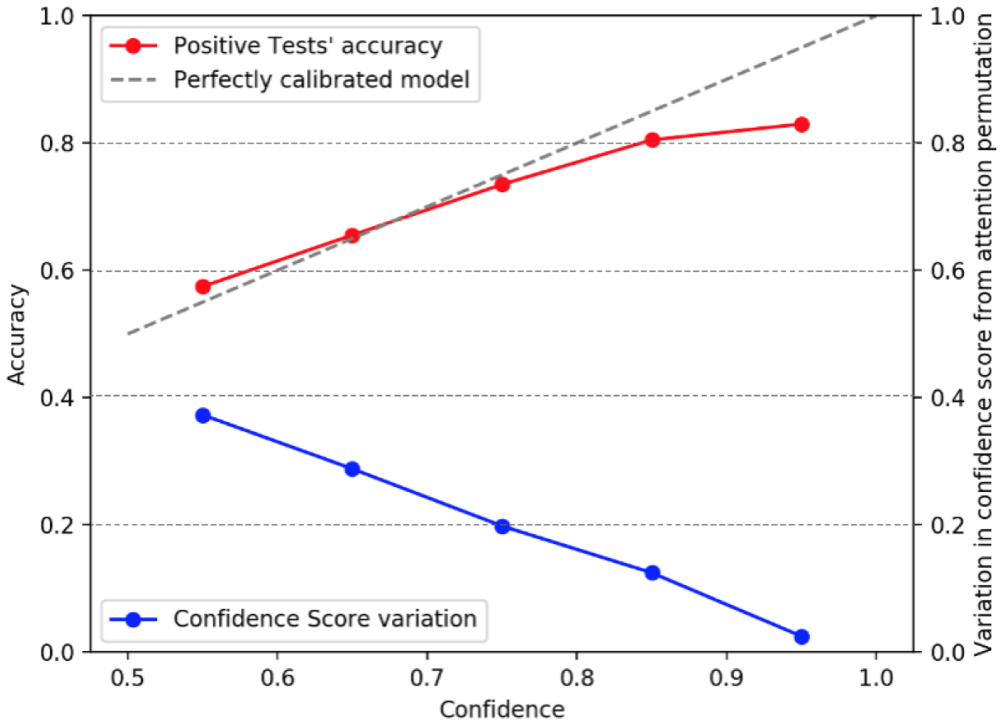


Figure 3.8: Figure showing the effect of attention permutation (right axis) on the prediction score of wrongly (negatively) classified positive test cases (false negatives).

3.4.3 Fixing the effect of Class-imbalance

As mentioned in Section 3.3.3, we try the *inverse-class weighted cross-entropy* loss in order to fix the effects of skewness in the training dataset. We retrain the model with the new loss function and were able to achieve similar HitRatio values to the original model as shown in Table 3.2. We analysed the effect of attention permutation (Section 3.4.2) on this model. Figure 3.9 compares the new model to the previous model's results. We notice that the new model is considerably more sensitive to attention permutation, compared to the original one. This suggests that attention based explanations from the class-balanced loss model are more reliable than the original model.

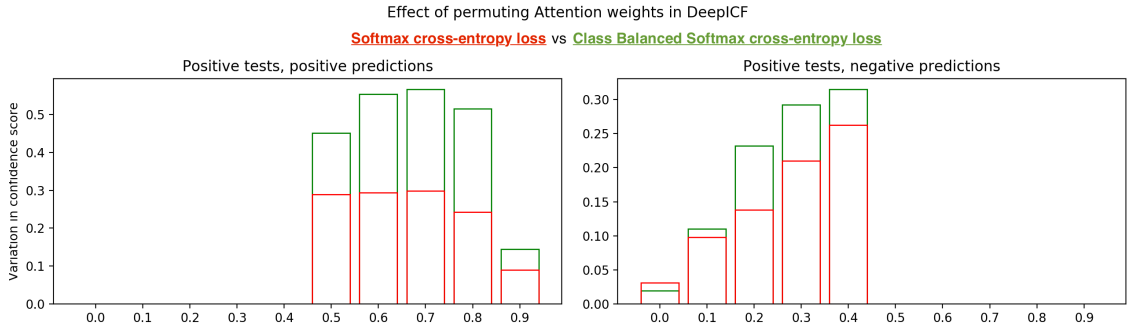


Figure 3.9: Figure showing the effect of permuting attention weights in the original model (softmax cross-entropy loss) vs the model trained with class-balanced loss. Figure highlights the increased sensitivity of attention distributions towards random permutations, hence, implying improved reliance for attention weights as means for local explanations in the class-balanced loss model.

3.4.4 Stability of DeepICF

We now consider the effect of random seeds and on initialization of model parameters and in general the model performance. We notice in Table 3.2 — the standard deviation is generally very low suggesting that the performance of the model is seemingly stable and it seems to have small deviation.

Model type	Hit Ratio@10 (%)	NDCG@10 (%)
DeepICF*	68.81	41.13
DeepICF*+Pretrain	70.84	43.80
NeuMF*+Pretrain	70.70	42.60
DeepICF (ours)	70.41($\pm$0.24)	43.00 ($\pm$0.34)
DeepICF+cls-wt	68.61	41.14
DeepICF (3-output)	68.69	40.54

Table 3.2: Performance Comparison for DeepICF and NeuMF[HLZ+17]. * indicates scores directly from the corresponding papers. The standard deviation ($\pm$) is obtained with 10 runs of the model with different random seeds.

Prediction score stability: For each positive test case, we compute the predicted class and its confidence for all the seeded models. Then we compute the average variation in the prediction score (variation from the mean) and plot it against the mean predicted score (note the prediction score lies between 0 and 1).

Figure 3.10 shows the prediction score stability in the model. We bucket the predictions according to the confidence in the positive label to see the variation in the prediction scores for different confidence levels. The prediction scores close to 0 (negative predictions) or 1 (positive predictions) are highly stable, especially when compared to the ones lying in between them. This behavior is expected as the model seems to be *unsure* about the items falling in the middle of the positive-negative spectrum. Overall, the model predictions seem stable as the average variations lie on the lower side for most of the buckets.

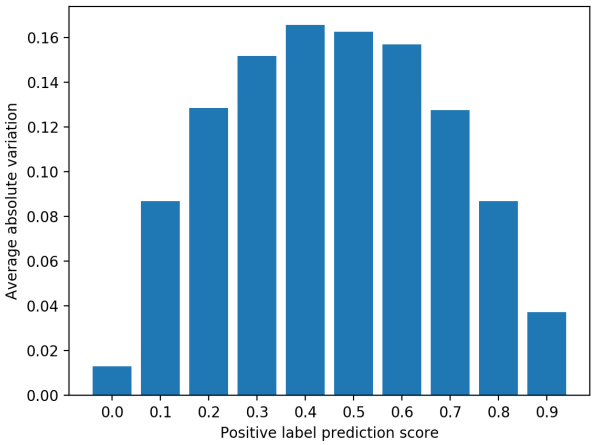


Figure 3.10: Prediction score stability from 10 differently seeded DeepICF models. Note that the range of the Y-axis in the plot is from 0 to 0.16, while the average absolute deviation for the prediction scores ranges from 0 to 1.

Attention score stability: What is the effect of random seeds on attention distribution? As we are interested in the reliability of attention *explanations*, we focus on the stability of attention scores in DeepICF. We perform the same experiment by running the same model but with 10 different random seeds and record the *top 10%* of the most attentive items (user-item interactions which get the highest attention weight assigned) for every particular test case for each model. Then we compare if these top 10 percent most attentive items for a particular test case are consistent for different runs of the models with different random seeds. We calculate the similarity between two sets of items by computing the `Jaccard Index` [Wik19b] of the sets. We calculate the Jaccard Index for every possible pair of sets of top attentive items and average over them. Figure 3.11 shows that the average `Jaccard Index` for positive predictions with high confidence is around 0.5 (where max `Jaccard Index` is 1, implying completely stable attention scores).

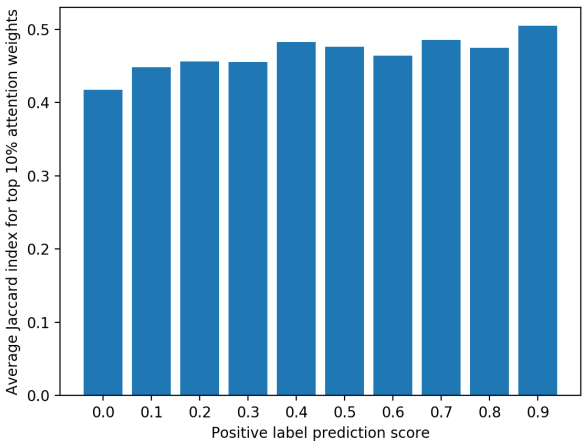


Figure 3.11: Attention weights' stability from 10 differently seeded DeepICF models. Note that the range of the Y-axis in the plot is from from 0 to 0.5, while the range for the Jaccard Index is from 0 to 1.

This highlights that the attention explanations from two identical models, trained with different seeds for the same input can vary, severely highlighting the unreliability of such explanations. Another important observation here is how the attention weights–highly unstable, are in contrast with the prediction scores–stable. We discuss this problem in more detail in Chapter 4.

3.5 Conclusion

In this chapter, we have explored the importance of model dynamics and its relation to explanation using attention. Concretely, we observe that attention may not be reliable when the selected model is especially mis-calibrated. We have explored one possible way of stabilizing the model by accounting for the class imbalance. Significantly, we noticed that using an inverse-class weighted cross-entropy formulation can help improve the stability of attention distribution. Further, we observe that over different runs of models with different random seeds, the models seem to obtain different attention distributions. We posit that our work is extremely relevant to the community and can orient towards an important discussion on the reliability of using attention as an explanation.

Chapter 4

Model stability as a function of random seeds

As mentioned before, there has been tremendous growth in deep neural based models with state-of-the-art performance. In fact, most recent end-to-end deep learning models have surpassed the performance of careful human feature-engineering based models in most NLP tasks. However, deep neural network-based models are often brittle to some sources of randomness in the training of the models. This could be attributed to several sources including, but not limited to, random parameter initializations, random sampling of examples and sampling of activations. It has been observed that these models have, more often, a set of *'random seeds'* that yield better results than others. This has also lead to research suggesting random seeds as an additional hyperparameter for tuning [Ben12][1].

One possible explanation for this could be the existence of multiple local minima in the loss surface. This is especially problematic as the loss surfaces are non-convex and may have multiple saddle points making it difficult to have a stable model, that is, to a large extent, robust to random-seed based effects.

Recently the NLP community has found new interests in interpreting and explaining deep neural models [JMW19, JW19, AMJ17]. Most of the interpretation based methods involve one of the following ways of interpreting models: a) Adversarial perturbation based interpretations: where the interpretation is based on the change in prediction score with counter-factual perturbations [JMW19, JW19]; b) Interpretations based on feature attributions using attention or input perturbation or gradient-based measures; [GFT18, FWGI+18]; c) Explanation using surrogate linear models [RSG16]. These methods can provide local interpretations based on input samples or features. However, the persisting randomness makes it difficult to accurately interpret neural models among other forms of pathologies [FWGI+18].

[1] http://www.argmin.net/2018/02/26/nominal/

i take a DRUG why do i feel the <UNK>

$$(\Pr(Y_{positive}) = 0.86)$$

i take a DRUG why do i feel the <UNK>

$$(\Pr(Y_{positive}) = 0.85)$$

Figure 4.1: Importance based on attention probabilities for two runs of the same model with **same parameters and same hyperparameters**, but with **two different random seeds** (color magnitudes: pink<magenta<red)

In this chapter we focus on the stability of deep neural models as a function of random-seeds. We are especially interested in investigating the hypothesis of model stability: do neural network based models under different random seeds allow similar explanations? In Figure 4.1, we give an illustration for this question where we have the attention distributions of two CNN based binary classification models for the *Twitter Adverse Drug Reaction* dataset[NSO+15], trained with the same settings and hyper-parameters, but with different seeds (more examples shown in Figure 4.2). We see that both models obtain the correct prediction with significantly high confidence. However, we note that both the models attend to completely different sets of words. This is problematic, especially when interpreting these models under the influence of such randomness.

We also provide a simple method that can, to a large extent, ameliorate this inherent random behaviour. In Section 4.4, we propose an aggressive stochastic weight averaging approach that helps in improving the stability of the models at almost zero performance loss while still making the model robust to random-seed based instability. We also propose an improvement to this model in Section 4.4.1 which further improves the stability of the neural models. Our proposals significantly improve the robustness of the model, on an average, by 72% relative to the original model and on Diabetes (MIMIC), a binary classification dataset, by 89% (relative improvement).

user DRUG add to regime yesterday x

$$(\Pr(Y_{negative}) = 0.99)$$

---

user DRUG add to regime yesterday x

$$(\Pr(Y_{negative}) = 0.98)$$

(a) Example taken from the *Twitter-Adverse Drug Reaction (ADR)* dataset[NSO+15]. Where both the (differently seeded) models give the same (correct) prediction, but for completely different attentive items.

new york - us stocks fell on tuesday as health insurers # qqq ; shares slid on worries that the new york attorney general # qqq probe will hit the entire industry .

$$(\Pr(Y_{business}) = 0.91)$$

---

new york - us stocks fell on tuesday as health insurers # qqq ; shares slid on worries that the new york attorney general # qqq probe will hit the entire industry .

$$(\Pr(Y_{business}) = 0.93)$$

(b) Example taken from the *AGNews* dataset [ZZL15]. Both the models correctly classify the article as *business*, but for completely different attentive items.

complete entertainment although there are many strange things in the movie that the fairy tale itself doesn't have them including the <UNK> characters mother and daughter the general concept rocks

$$(\Pr(Y_{positive}) = 0.67)$$

---

complete entertainment although there are many strange things in the movie that the fairy tale itself doesn't have them including the <UNK> characters mother and daughter the general concept rocks

$$(\Pr(Y_{positive}) = 0.77)$$

(c) Example taken from the *IMDB* dataset[MDP+11]. Both the models correctly classify the movie review as *positive*, with similar confidence score, but for completely different attentive items.

if high crimes were any more generic it would have a universal product code instead of a title

$$(\Pr(Y_{negative}) = 0.99)$$

---

if high crimes were any more generic it would have a universal product code instead of a title

$$(\Pr(Y_{negative}) = 0.98)$$

(d) Example taken from the *SST* dataset [SPW+13]. Both the models correctly classify the text sentiment as *negative*, with high confidence, but for completely different attentive items.

Figure 4.2: Examples taken from different datasets showing the importance of words based on attention probabilities for two runs of the same model with same parameters and same hyperparameters, but with two different random seeds. (Color magnitudes: pink<magenta<red).

4.1 Model Stability

Following the discussions in Sections 2.3 (especially 2.3.5), 2.5, and 2.6, we now define our interpretations of stability. We account for a range of factors (Sections 4.1.1, 4.1.2, 4.1.3) when examining the *model's* stability.

4.1.1 Prediction Stability

We define prediction stability in two parts, the first corresponds to the standard measures of the mean and the standard deviations corresponding to the accuracy of the binary classification based models on different datasets. We ensure that the models are run with exactly the same configurations and hyper-parameters but with different random seeds (setup as shown in Section 4.2). This is a standard procedure that is used in the community to report the performance of a model.

4.1.2 Attention Stability

We define attention stability using the robustness of the attention distributions. That is, we call the attention probabilities of a model stable if the model, over several runs with different random seeds (but with the same settings and hyper-parameters) has similar attention probability distributions. We consider this to be extremely important for both interpretations and explanations using attention and the utility of models in general. We also find that this is important even for interpretations using leave one out based methods or local interpretations using surrogate models. This is because, if there is variance due to random seeds (mostly due to induced randomness in the initialization of weights and biases), the general interpretations would vary for each model.

We now focus on attention distributions to quantify instability. An important part of the project was to decide on a metric which could capture the problem of unreliable attention vectors. For this, we needed two compare the attention vectors for two differently seeded models for the same input.

Before we start analyzing the usability of different metrics for our purpose, we need to understand what exactly our requirements and constraints are.

The first observation is that we are comparing two lists of the same size. This is always the case since we only need to compare attention distributions from the same test case (but for different models). Another important observation is that the lists are probability distributions (since we apply a softmax activation to get the final attention weights). This means that for each list, all the elements are between 0 and 1, and they sum up to 1.

We also need to understand what needs to be quantified when comparing two lists. The most important thing that we need to measure here is that the items (words) which get high weights assigned to them remain consistent. The metric we choose should be able to penalize the similarity score if the magnitude of the difference in the weights of any item is high. This also suggests that we do not care about items which are assigned low weights in both the distributions. For example, if *Item-A* is assigned weight 0.5 in the distribution d_1 and 0.1 in the distribution d_2, we want the similarity score to be penalized. On the other hand, if an *Item-B* is assigned weight 0.005 by distribution d_1 and 0.001 by distribution d_2, we do not want the score to be significantly impacted.

We have listed below the metrics that we considered for our analysis and chose to investigate the instability in models:

a) **Entropy quantification ($\mathcal{H}$):** Given two attention distributions for the same test case from two different models, it measures the entropy between the two probability distributions. Note that, the higher the entropy the greater the dissimilarity between the two distributions.

$$\mathcal{H} = \sum_{i \in d} \Pr_1 \cdot \log \frac{\Pr_1}{\Pr_2}$$

where, $\Pr_1$ and $\Pr_2$ are two attention distributions of the same sample from two different runs of the model and d is the number of tokens in the sample. Given n differently seeded models, for each test instance, we calculate the averaged pairwise attention distributions' entropy.

b) **Jaccard Distance ($\mathcal{J}$):** It measures the dissimilarity between two sets. Here higher values of $\mathcal{J}$ indicate larger variances. We consider top-n tokens which have the highest attention for comparison. Note that, Jaccard distance is over sets of word indices and do not take into account the attention probabilities explicitly. Jaccard distance is defined as:

$$\mathcal{J} = (1 - \frac{A \cap B}{A \cup B}) * 100\%$$

where, A and B are the sets of most relevant items. We specifically decided to use 'most' relevant (top-n items) as the tail of the distribution mostly consists of values close to 0.

```python
import numpy as np


def get_jaccard_distance(atn_dist1, atn_dist2):
    assert len(atn_dist1) == len(atn_dist2)

    n = np.math.ceil(0.1 * len(atn_dist1))

    top_n_i, top_n_j = np.argpartition(atn_dist1, -n)[-n:], \
                       np.argpartition(atn_dist2, -n)[-n:]

    set_i, set_j = set(top_n_i), set(top_n_j)

    jaccard_index = len(set_i & set_j) / len(set_i | set_j)

    jaccard_distance = (1 - jaccard_index) * 100

    return jaccard_distance


a1 = [0.2, 0.1, 0.1, 0.6]  # Attention distribution 1
a2 = [0.4, 0.1, 0.4, 0.1]  # Attention distribution 2
print(get_jaccard_distance(a1, a2))
```

Listing 4.1: `Python` code to calculate the *Jaccard distance* between the sets of top 10% attentive items from two attention distributions.

In Figure 4.3, we have shown an example scenario with attention distributions for 10 input features from 2 differently seeded models. As we can see, *Item 4* and *Item 5* have been assigned the highest weights by both the models. Hence, the *Jaccard distance* for the top 20% attentive items between the two distributions is 0. This highlights one of the major weaknesses of using Jaccard distance as a metric to compare attention distributions. It doesn't take into account the absolute weight values assigned to the items. The *Entropy*, however, is able to capture these differences as well ($\mathcal{H} = 0.66$ in this case). In this

	Attention Distribution 1	Attention Distribution 2
Item 1	0.1	0.03
Item 2	0.05	0.1
Item 3	0.001	0.1
Item 4	0.5	0.12
Item 5	0.15	0.48
Item 6	0.05	0.05
Item 7	0.05	0.05
Item 8	0.02	0.008
Item 9	0.009	0.002
Item 10	0.07	0.06

Figure 4.3: Figure showing two attention distributions for 10 input items (features). In this case, $\mathcal{H} = 0.66$; $\mathcal{J} = 0$

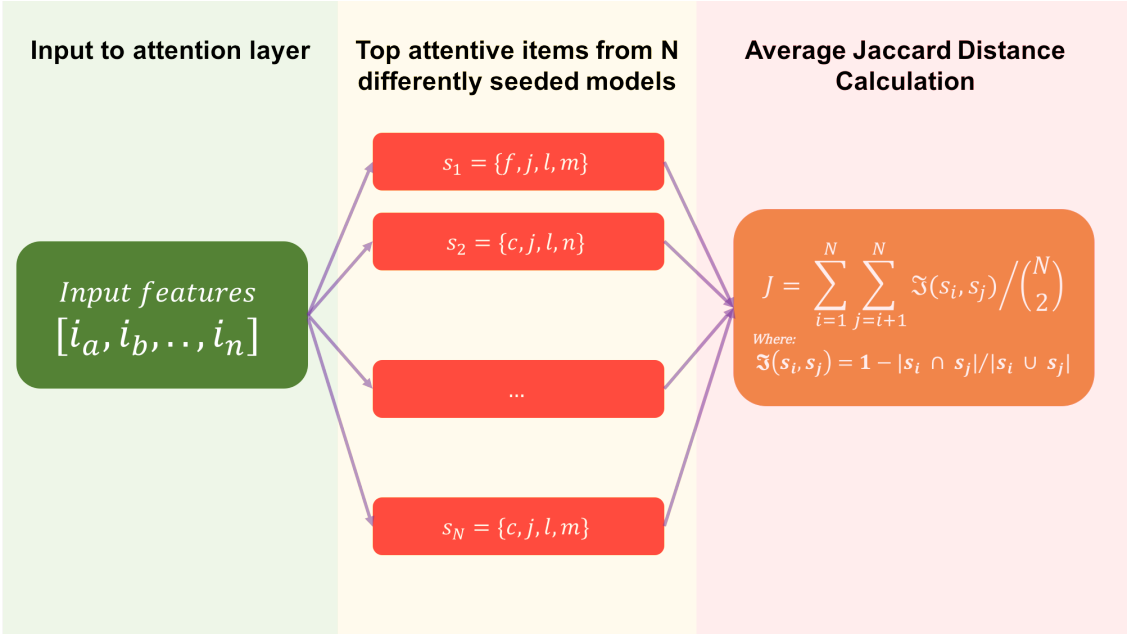


Figure 4.4: Average Jaccard distance calculation process for a particular test case, for N differently seeded models.

project, we still measure the Jaccard distance since it (leniently, in some sense) captures the reliability of attention weights as means to give explanations to the model's outputs.

Figure 4.4 shows the calculation process for the average Jaccard distance for a particular test case, with N differently seeded models.

4.1.3 Gradient-based Interpretation

Gradient-based feature importance is another way to interpret the model for local explanations. We use the input gradients of the model for each word embedding and compute the magnitude of the change as a local explanation. We refer the reader to [BSH+10] for a good introduction to gradient-based interpretations. As all of our models are differentiable, we use this as an alternative method for interpretation. We note that we do not follow [JW19] and do not disconnect the computational graph at the attention module. We follow the standard procedure as followed in [FWGI+18]. We use entropy as described in Section 4.1.2 to quantify the instability.

4.2 Reproducibility by seed initialization

In order to conduct our experiments, we follow a common practice in machine learning where we seed the random bits in the training process to make it deterministic.

The most common form of randomness introduced to the training process is by the initialization of weights of various layers in the neural model. Batch processing like splitting the test-train data, and shuffling the training batches also induces substantial randomness to the system. Other techniques like Dropouts[SHK+14] and Stochastic Optimizations[Bot10] also impart a level to randomness to the process. Moreover, when using GPUs to train the model, it is possible to have complex libraries introducing their own source of randomness (which can be particularly hard to account for).

We have to account for all these processes that can play a part in making the process hard to reproduce. The python code shows all the seed initializations we do to make sure our experiments are reproducible. Figure 4.5 shows the results from our entropy experiments (discussed in Section 4.1.2) from two different machines with the same seed. As we can see, the two plots are almost the same. The remaining differences are hard to account for, as they might be coming from sources like floating point errors, some third-party library using different sources of randomness or others. Note that, with out setup, running the seeded experiments on the same machine, gives us *exactly* the same results.

```python
import os
import numpy as np
import random
import torch
from torch.backends import cudnn


os.environ['PYTHONHASHSEED'] = str(42)
np.random.seed(42)
random.seed(42)
torch.manual_seed(42)
cudnn.deterministic = True
cudnn.benchmark = False

train_model(...)
```

Listing 4.2: `Python` code to make training *deterministic* for `pytorch`

```python
import os
import numpy as np
import random
import tensorflow as tf

tf.reset_default_graph()
os.environ['PYTHONHASHSEED'] = str(42)
np.random.seed(42)
random.seed(42)
tf.set_random_seed(42)

train_model(...)
```

Listing 4.3: `Python` code to make training *deterministic* for `tensorflow`

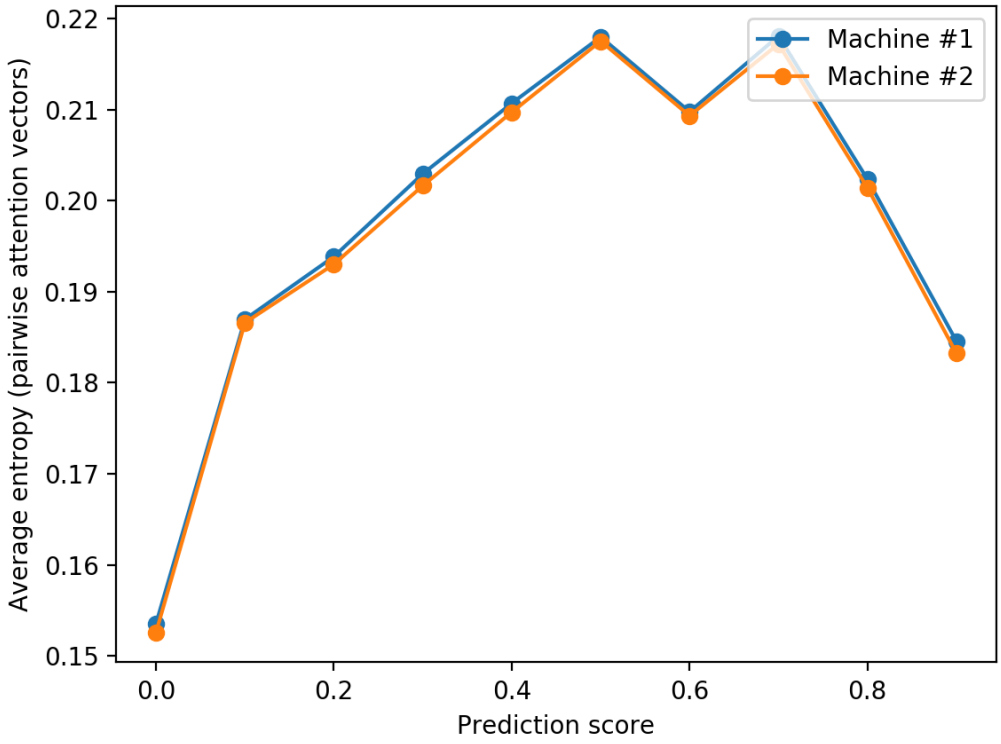


Figure 4.5: (Almost) Deterministic results (for the Entropy calculations – Section 4.1.2) for models trained on two different machines.

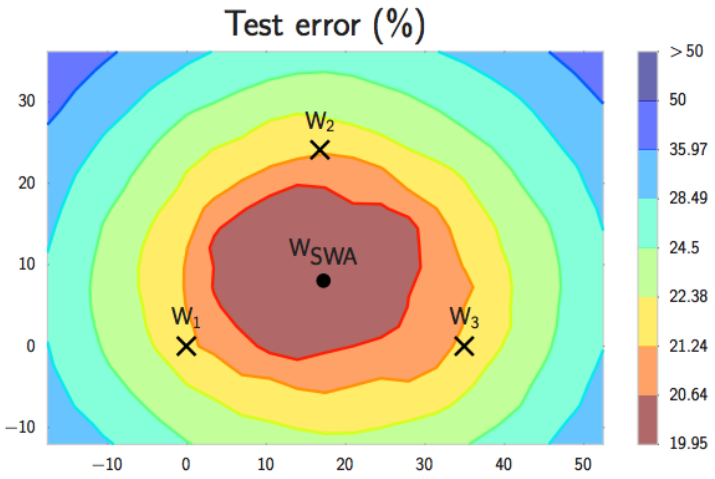


Figure 4.6: Image shows how averaging the weights (SWA) can achieve better test error. Image source: [IPG+18].

4.3 Stochastic Weight Averaging

In this chapter, we also propose a solution to the model-instability problem which is inspired from the Stochastic Weight Averaging (SWA) approach proposed in [IPG+18]. The main motivation behind SWA is to improve the training process of a neural network, and achieve better performance. They propose a modification to the normal Stochastic Gradient Descent approach. The SWA algorithm can also be seen an alternative approach to the *Fast Geometrix Ensembling* (FSE) approach proposed in [GIP+18].

The main idea behind SWA is to average the weights of multiple points in the trajectory of the gradient descent based optimizers (primarily SGD). The algorithm can be seen as a extension to SGD where, after a certain point in the training process, the learning rate for SGD is kept constant. This allows the weights to *explore* the weight space (around the local optimum). Then, the weights from different iterations are averaged to produce the final learned weights of the model. For example, if SGD (with constant learning rate) produces weight samples: w_1, w_2, ..., w_n. The final weights, according to the SWA approach is:

$$w_{swa} = \frac{\sum_{i=1..n} w_i}{n}$$

Figure 4.6, for instance, shows the SWA algorithm in action where it is able to achieve better train loss (and hence, better test loss) by averaging on the *explored* SGD attained weight samples. I would refer the reader to [IPG+18] to get a detailed understanding of the algorithm.

Note that it is also interesting to compare this approach with the FSE proposal where the final output is the *average of outputs* from an ensemble of models.

In the SWA paper[IPG+18], the authors also show that, compared to SGD (with a fixed learning rate), their proposed algorithm, has significantly higher chances of producing weights that are inside the ellipsoid near w_{opt} (the local optimum), under their given assumptions.

Overall, the principle idea in SWA is to *explore* the weight space around the local optimum and average the weights that are maximally distant from each other (while making

sure that they are around the same minima region) to converge closer to the optimum values. Although, in our case, we look at SWA from the perspective of a model's stability. In this project, we try to use this idea of the SWA approach to modify the training process such that the resulting weights (models) are more robust and stable.

4.4 Aggressive Stochastic Weight Averaging (ASWA)

Stochastic weight averaging (SWA) [IPG+18], as explained in Section 4.3, works by averaging the weights of multiple points in the trajectory of gradient descent based optimizers. The algorithm typically uses modified learning rate schedules. SWA is itself based on the idea of maintaining a running average of weights in stochastic gradient descent based optimization techniques [Rup88, PJ92]. The principle idea in SWA is averaging the weights that are maximally distant helps stabilize the gradient descent based optimizer trajectory and improves generalization. [IPG+18] use the analysis of [MHB17] to illustrate the stability arguments where they show that, under certain convexity assumptions, SGD iterations can be visualized as samples from a Gaussian distribution centred at the *minima* of the loss function. Samples from high-dimensional Gaussians are expected to be concentrated *on the surface of the ellipse* and not close to the *mean*. Averaging iterations is shown to stabilize the trajectory and further improve the width of the solutions to be closer to the *mean*.

The paper[IPG+18] suggests using SWA usually after pre-training the model (at least until 75% convergence) and after which they suggest sampling weights at different steps either using large constant or cyclical learning rates. As SWA is well defined for convex losses [PJ92], the authors connect SWA to non-convex losses by suggesting that the loss surface is *approximately* convex after convergence.

In this project, however, we focus on the stability of deep neural models as a function of random-seeds. Our proposal is based on SWA, but we extend it to the extremes and call it *Aggressive Stochastic Weight Averaging* (ASWA). We assume that, for small batch size, the loss surface is locally convex. In our algorithm we follow the same concept of SWA, where we keep a moving average of the weights of the model and update the original model weights with them at the end of each epoch, except that, we do not wait for the model to converge to start the weight averaging process. Instead we start from the first iteration of the model training itself. Also, we perform the weight averaging at every step of the batch size b (i.e, each iteration). We further relax the conditions for the optimizer and assume that the optimizer is based on some version of gradient descent — this means that our modification is valid even for other pseudo-first-order optimization algorithms including Adam [KB14] and Adagrad [DHS11].

Another subtle (but crucial) difference between the SWA and ASWA is the way we use the moving averaged weights. At the end of each epoch, instead of *swapping* the original weights with the averaged ones (as proposed by SWA), we discard the original weights and assign the averaged weights' value to them. This can be noticed in Figure 4.10 where the model, in some sense, gets a *restart* at the end of each epoch. This makes for the conservative behaviour of the algorithm which, intuitively, helps it to converge into more stable weights. Since we were using the `Pytorch` library for SWA's implementation[2], we decided to modify the original library code to implement ASWA. Figure 4.7 shows the exact change we made to the library to implement our algorithm. In the `swap` function

[2] https://github.com/pytorch/contrib

Figure 4.7: Figure showing changes proposed to the `swap` function in the original SWA implementation. In ASWA, the value of the `swa_buffer` remains unchanged, and is not replaced with the current weights.

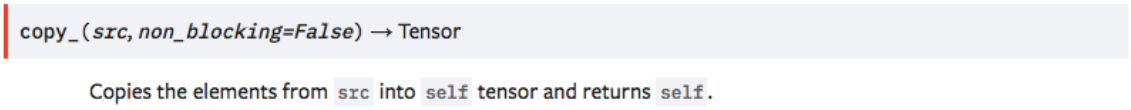


Figure 4.8: Definition of the `copy_` function from `Pytorch`.

(called at the end of each epoch) we don't replace the value stored in the moving averaged weights (`swa_buffer`). Note that the definition of the `copy_` function is shown in Figure 4.8 (taken from the official documentation of `Pytorch`[3]).

In our setup, we investigate the utility of averaging weights over every iteration (an iteration consists of one batch gradient descent).

In Figures 4.9 and 4.10, we show an SGD optimizer (with momentum) and the same optimizer *with SWA* over a 3-dimensional loss surface with a saddle point. We observe that the original SGD reaches the desired minima, however, it almost reaches the saddle point and does a course correction and reaches minima. On the other hand, we observe that SGD with ASWA is very conservative, it repeatedly restarts and reaches the minima without reaching the saddle point. We empirically observe that this is a desired property for the stability of models over runs of the same model but with different random initialization. The grey circles in Figure 4.10 highlight this conservative behaviour of SGD with ASWA optimizer, especially when compared to the standard SGD.

We note, [PJ92] show that for convex losses, averaging SGD proposals achieves the highest possible rate of convergence for a variety of first-order SGD based algorithms.

Algorithm 1 shows the implementation pseudo-code for SWA. Unlike [IPG+18], we average our weights at each batch update and assign the ASWA parameters to the model at the end of each epoch. That is, we replace the model's weights for the next epoch with the averaged weights.

[3] https://pytorch.org/docs/stable/tensors.html

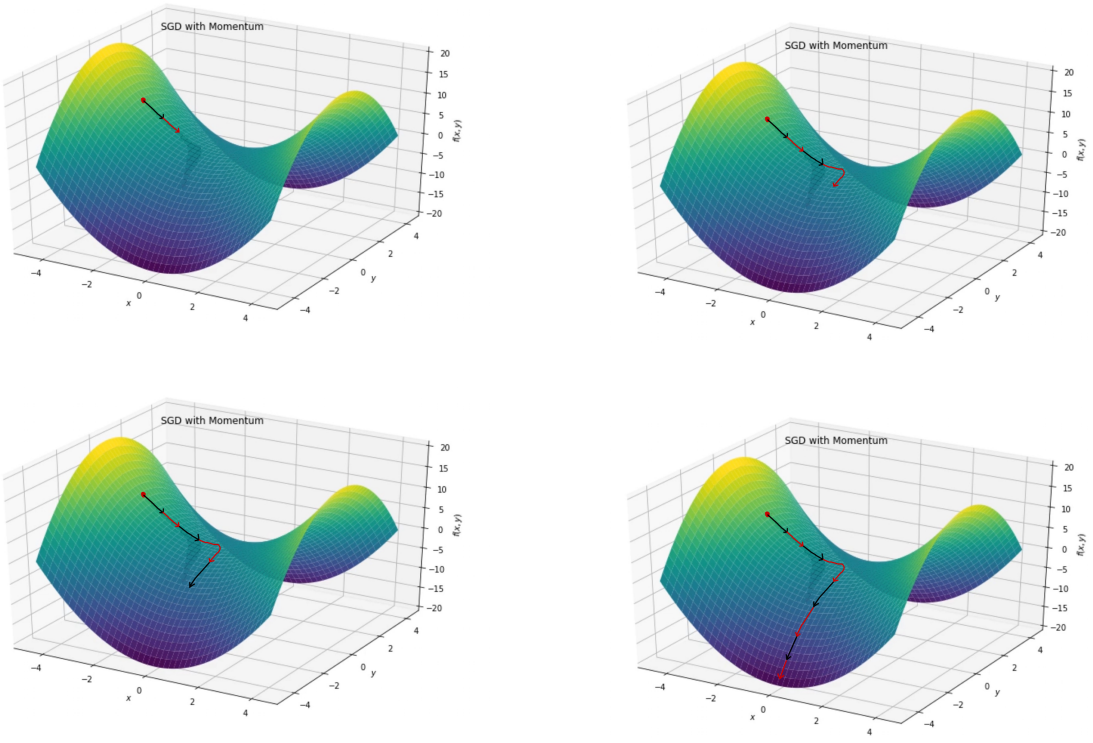


Figure 4.9: Trajectory for normal Stachastic gradient descent algorithm with red and black arrows indicating movements from consecutive epochs.

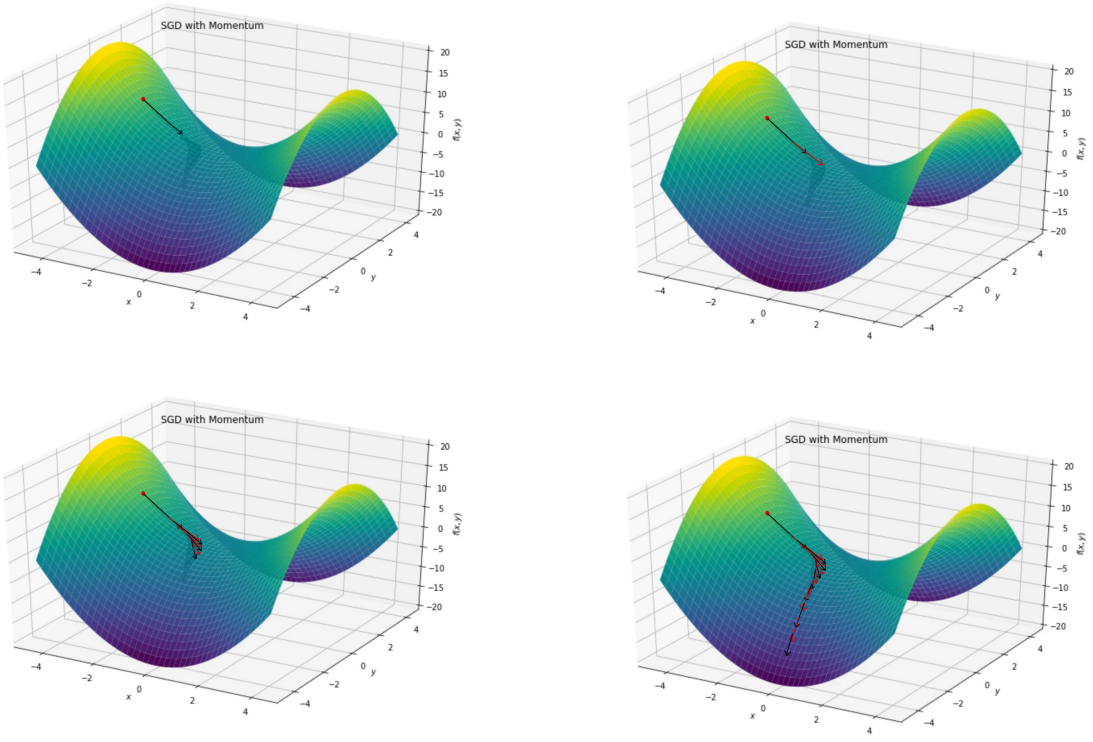


Figure 4.10: Trajectory for the ASWA algorithm with red and black arrows indicating movements from consecutive epochs *with restarts*. Conservative behaviour of ASWA algorithm helps it avoid the saddle point.

Algorithm 1: Aggressive SWA algorithm

Require:

$e =$ Epoch number
$m =$ Total epochs
$i =$ Iteration number
$n =$ Total iterations
$\alpha =$ Learning rate
$\mathcal{O} =$ Stochastic Gradient optimizer function

$e \leftarrow 0$;
while $e < m$ **do**
  $i \leftarrow 1$
  while $i \leq n$ **do**
    $W_{swa} \leftarrow W_{swa} + \frac{(W - W_{swa})}{(e*n+i+1)}$;
    $W \leftarrow W - \mathcal{O}(\alpha, W)$;
    $i \leftarrow i + 1$
  $W \leftarrow W_{swa}$;
  $e \leftarrow e + 1$

Algorithm 2: Norm-filtered Aggressive SWA algorithm

Require:

$e =$ Epoch number
$m =$ Total epochs
$i =$ Iteration number
$n =$ Total iterations
$\alpha =$ Learning rate
$\mathcal{O} =$ Stochastic Gradient optimizer function
$N_s =$ List of previous iterations' norm differences

$e \leftarrow 0$;
while $e < m$ **do**
  $i \leftarrow 1$
  while $i \leq n$ **do**
    $N_{cur} \leftarrow \|W - W_{swa}\|_1$;
    $N_{mean} \leftarrow \frac{\sum_{i=1}^{|N_s|} N_s[i]}{|N_s|}$;
    if $N_{cur} > N_{mean}$ **then**
      $W_{swa} \leftarrow W_{swa} + \frac{(W - W_{swa})}{(e*n+i+1)}$;
      $N_s \leftarrow [N_{cur}]$;
    else
      $N_s \leftarrow N_s + [N_{cur}]$;
    $W \leftarrow W - \mathcal{O}(\alpha, W)$;
    $i \leftarrow i + 1$
  $W \leftarrow W_{swa}$;
  $e \leftarrow e + 1$

4.4.1 Norm-filtered Aggressive Stochastic Weight Averaging (NASWA)

We observe that the ASWA algorithm is especially beneficial when the norm difference of the parameters of the model, at two different iterations, are high. We hypothesise that in general, the norm difference indicates the divergence between optimizers' steps and we observe that the larger the norm difference, the greater the change in the trajectory. Therefore, we propose to maintain a list that stores the norm differences of the previous iterations. If the norm difference of the current iteration is greater than the average of the list, we update the ASWA weights and reinitialize the list with the current norm difference. When the norm difference, however, is less than the average of the list, we just append the current norm difference to the list. After the completion of the epoch, we assign the ASWA parameters to the model. This is shown in Algorithm 2. We call this approach *Norm-filtered Aggressive Stochastic Weight Averaging*.

4.5 Experiments

We base our investigation on similar sets of models as [JW19] (as discussed in Section 2.6). We also use the code[4] provided by the authors for our empirical investigations for consistency and empirical validation. We describe our models and datasets used for the experiments below.

4.5.1 Models

We consider two sets of commonly used neural models for the tasks of binary classification and multi-class natural language inference. The basic structure of the model is explained in Section 2.6. We use CNN and bi-directional LSTM based models with attention. We follow [JW19] and use similar attention mechanisms using a) additive attention [BCB14]; and b) scaled dot product based attention [VSP+17]. We jointly optimize all the parameters for the model, unlike [JW19] where the encoding layer, attention layer and the output prediction layer are all optimized separately. We experiment with several optimizers including Adam [KB14], SGD and Adagrad [DHS11] but most results below are with Adam.

For our ASWA and NASWA based experiments, we use a constant learning rate for our optimizer. Other model-specific settings are kept the same as [JW19] for consistency.

Dataset	Avg. Length	Train Size	Test size
IMDB	179	12500 / 12500	2184 / 2172
Diabetes(MIMIC)	1858	6381 / 1353	1295 / 319
SST	19	3034 / 3321	652/653
Anemia(MIMIC)	2188	1847 / 3251	460 / 802
AgNews	36	30000 / 30000	1900 / 1900
ADR Tweets	20	14446 / 1939	3636 / 487
SNLI	14	182764 / 183187 / 183416	3219 / 3237 / 3368

Table 4.1: Dataset characteristics. Train size and test size show the cardinality for each class. SNLI is a three-class dataset while the rest are binary classification

[4] https://github.com/successar/AttentionExplanation

4.5.2 Datasets

The datasets used in our experiments are listed in Table 4.1 with summary statistics. We further pre-process and tokenize the datasets using the standard procedure and follow [JW19]. We note that IMDB [MDP+11], Diabetes (MIMIC) [JPS+16], Anemia (MIMIC) [JPS+16], AgNews [ZZL15], ADR Tweets [NSO+15] and SST [SPW+13] are datasets for the binary classification setup. While SNLI [BAPM15] is a dataset for the multiclass classification setup and CNN News Articles [HKG+15] for cloze style question answering. Figure 4.2 shows examples from some of these datasets. I would refer the reader to [JW19] to get a better understanding on all the different datasets.

4.5.3 Settings and Hyperparameters

We use a 300-dimenstional embedding layer which is initialized with FastText [JGB+16] based free-trained embeddings for both CNN and the bi-directional LSTM based models. We use a 128-dimensional hidden layer for the bi-directional LSTM and a 32-dimensional filter with kernels of size $\{1, 3, 5, 7\}$ for CNN. For others, we maintain the model settings to resemble the models in [JW19]. We train all of our models for 20 Epochs with a constant batch size of 32. We use early stopping based on the validation set using task-specific metrics (Binary Classification: using `roc-auc`[GS+66], Multiclass and question answering based dataset: using `accuracy`).

Dataset	CNN(%)	CNN+ASWA(%)	CNN+NASWA(%)
IMDB	89.8 ($\pm$0.79)	90.2 ($\pm$0.25)	90.1 ($\pm$0.29)
Diabetes	87.4 ($\pm$2.26)	85.9 ($\pm$0.25)	85.9 ($\pm$0.38)
SST	82.0 ($\pm$1.01)	82.5 ($\pm$0.39)	82.5 ($\pm$0.39)
Anemia	90.6 ($\pm$0.98)	91.9 ($\pm$0.20)	91.9 ($\pm$0.19)
AgNews	95.5 ($\pm$0.23)	96.0 ($\pm$0.11)	96.0 ($\pm$0.07)
Tweet	84.6 ($\pm$2.65)	84.4 ($\pm$0.54)	84.4 ($\pm$0.54)

Table 4.2: Performance statistics obtained from 10 differently seeded CNN based models. Table compares accuracy and its **standard deviation** for the normally trained CNN model against the ASWA and NASWA trained models, whose deviation drops significantly, thus, indicating increased robustness.

4.6 Results

In this section, we summarize our findings for 10 runs of the model with 10 different random seeds but with identical model settings.

4.6.1 Model Performance and Stability

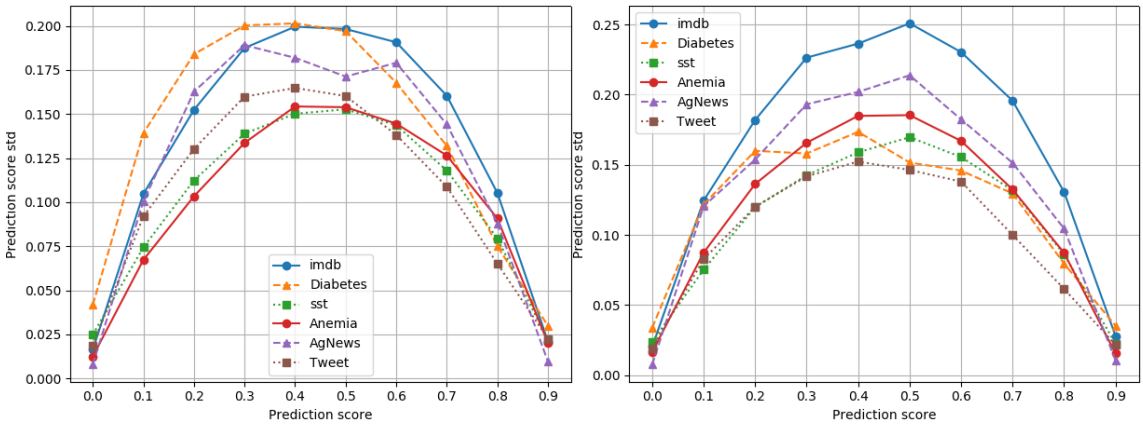
We first report model performance (classification accuracy) and prediction stability. The results are reported in Table 4.2.

We note that the original CNN based models, on an average, have a standard deviation of $\pm 1.5\%$. Which seems standard, however, we note that ADR Tweets dataset has

Dataset	LSTM(%)	LSTM+ASWA(%)	LSTM+NASWA(%)
IMDB	89.1 ($\pm$1.34)	90.2 ($\pm$0.32)	90.3 ($\pm$0.17)
Diabetes	87.7 ($\pm$1.44)	87.7 ($\pm$0.60)	87.8 ($\pm$0.55)
SST	81.9 ($\pm$1.11)	82.0 ($\pm$0.60)	82.1 ($\pm$0.57)
Anemia	91.6 ($\pm$0.49)	91.8 ($\pm$0.34)	91.9 ($\pm$0.36)
AgNews	95.5 ($\pm$0.32)	96.1($\pm$0.17)	96.1 ($\pm$0.10)
Tweet	84.7 ($\pm$1.79)	83.8 ($\pm$0.45)	83.9 ($\pm$0.45)

Table 4.3: Classification accuracy statistics obtained from 10 differently seeded LSTM based models.

a very high standard deviation of $\pm 2.65\%$. We observe that ASWA and NASWA almost always are able to get higher performance with very low standard deviation. This suggests that both ASWA and NASWA are extremely stable when compared to the standard model. They significantly improve the robustness, on an average, by 72% relative to the original model and on Diabetes (MIMIC), a binary classification dataset, by 89% (relative improvement). We observe similar results for the LSTM based models in Table 4.3.

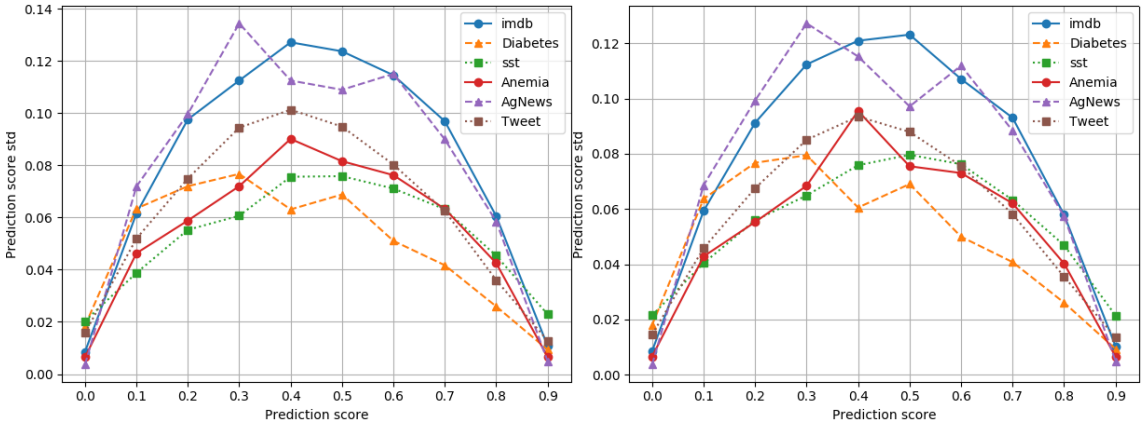


(a) CNN models

(b) LSTM models

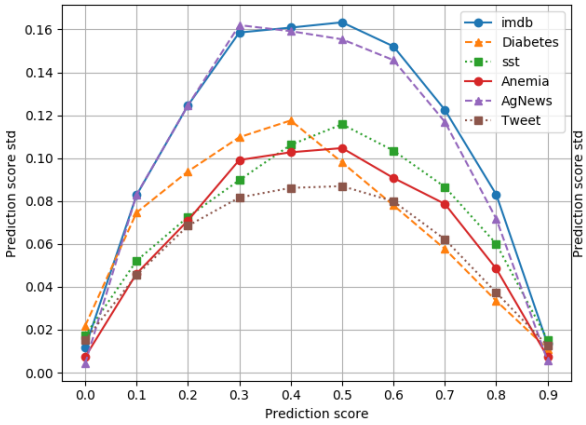
Figure 4.11: Prediction's standard deviation for CNN and LSTM based models for all binary classification datasets under consideration. Predictions are bucketed in intervals of size 0.1, starting from 0 (containing predictions from 0 to 0.1), until 0.9

We further analyze the prediction score stability by computing the mean standard deviation over the binned confidence intervals of the models in Figure 4.11a. We note that on an average, the standard deviations are on the lower side. However, we observe that the mean standard deviation of the bins close to 0.5 is on the higher side as is expected given the high uncertainty. On the other hand both, ASWA and NASWA based models are relatively more stable than the standard CNN based model (Figures 4.12a, 4.12b). We observe similar behaviours for the LSTM based models in Figure 4.11b. This suggests that our proposals, both ASWA and NASWA, are able to obtain relatively better stability without any loss in performance. We also note that both ASWA and NASWA had relatively similar performance over more than 10 random seeds.

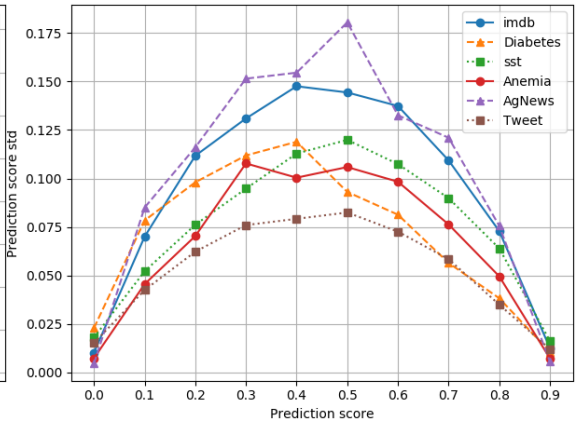


(a) CNN+ASWA

(b) CNN+NASWA



(c) LSTM+ASWA

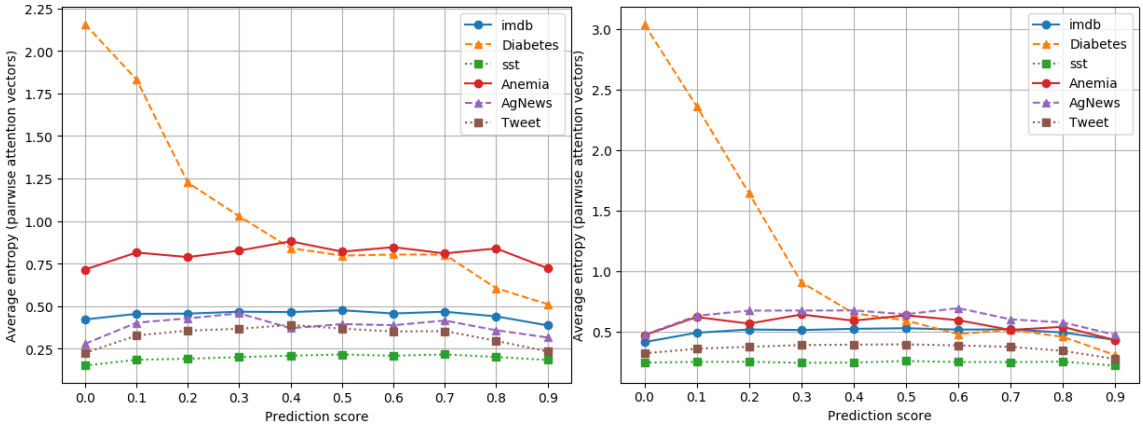


(d) LSTM+NASWA

Figure 4.12: Improved prediction stability from ASWA and NASWA for CNN and LSTM based models

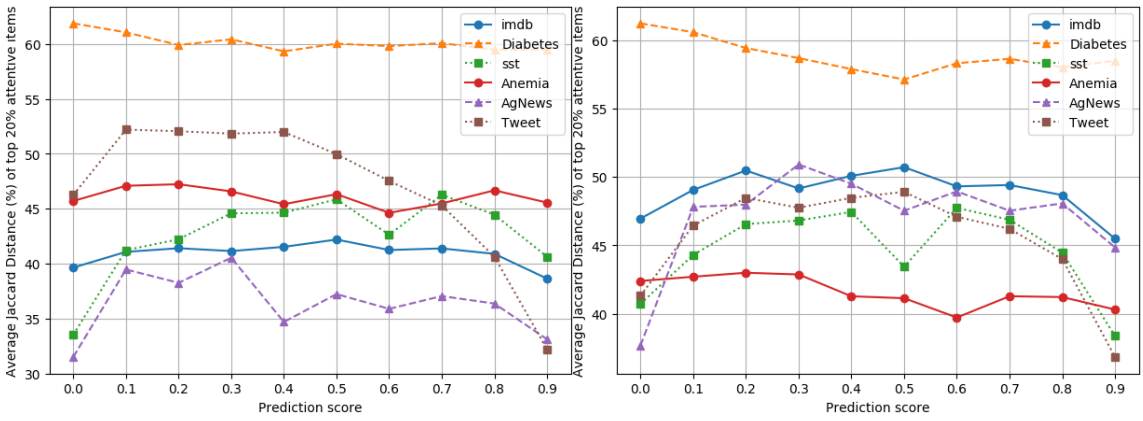
4.6.2 Attention Stability

We now consider the stability of attention distributions over as a function of random seeds. We first plot the results of the experiments for *standard* CNN based binary classification models over uniformly binned prediction scores for positive labels in Figure 4.13a. We observe that, depending on the datasets, the attention distributions can become extremely unstable (high entropy). We specifically highlight the Diabetes(MIMIC) dataset's entropy distribution. We observe similar, but relatively worse results for the LSTM based models in Figure 4.13b. In general, we would expect the entropy distribution to be close to zero however, this doesn't seem to be the case. This means that using attention distributions to interpret models may not be reliable and can lead to misinterpretations.



(a) CNN models (b) LSTM models

Figure 4.13: Average attention entropy against the bucketed predictions for CNN and LSTM based models. Figure highlights the high entropy between attention based distributions from differently seeded models (especially for the Diabetes-MIMIC datatset), indicating towards model instability.



(a) CNN models (b) LSTM models

Figure 4.14: Average Jaccard distance between attention-distributions, against the bucketed predictions for CNN and LSTM based models. Figure highlights the high Jaccard-distance between attention based distributions from differently seeded models, again, indicating towards model instability.

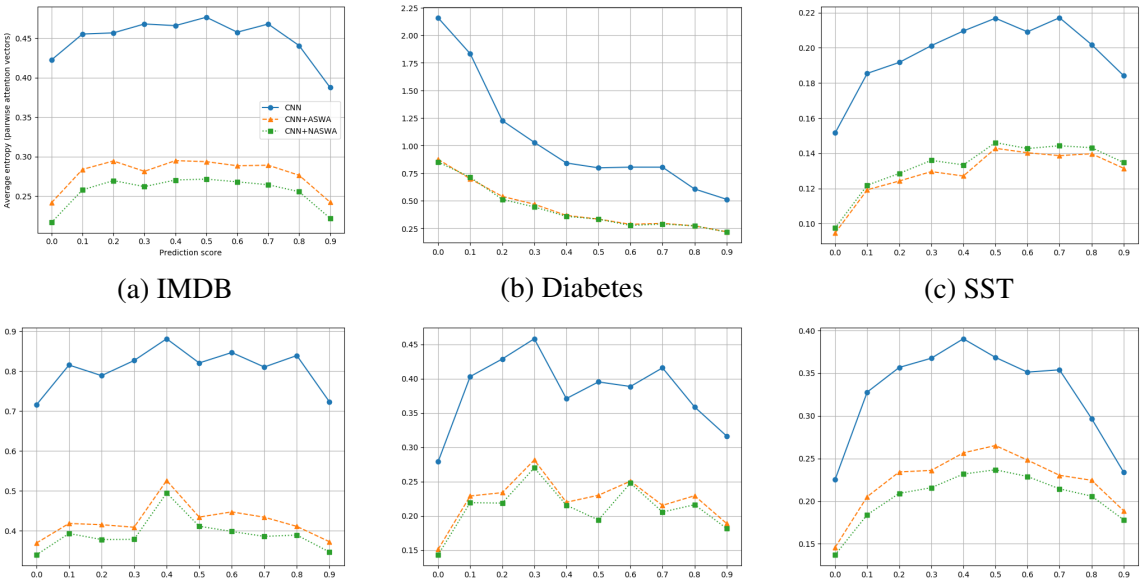
We use the top 20% of the most important items (indices) in the attention distribution

for each dataset over 10 runs and plot the Jaccard distances for CNN and LSTM based models in Figure 4.14a and Figure 4.14b. We again notice a similar trend of unstable attention distributions over both CNN and LSTM based attention distribution.

In the following sections, we focus on CNN based models with additive attention. Our results on LSTM based models are also shown in Figure 4.16. We note that the observations for LSTM models are, in most cases, similar to the behaviour of the CNN based models. Results for the scaled dot-product attention based models are also shown in 4.17 and we notice a similar trend as the additive attention.

We now focus on the effect of ASWA and NASWA on binary and multi-class CNN based neural models separately.

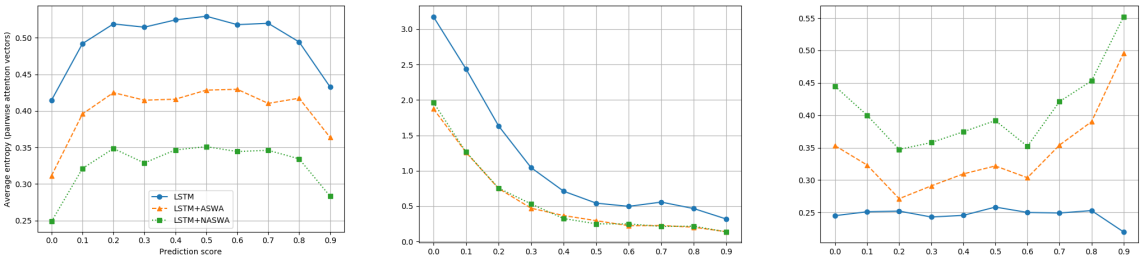
Binary Classification In Figure 4.15, we plot the results of the models with ASWA and NASWA. We observe that both these algorithms significantly improve the model stability and decrease the entropy between attention distributions. For example, in Figure 4.15b, both ASWA and NASWA decrease the average entropy by about 60%. We further notice that NASWA is slightly better performing in most of the runs. This empirically validates the hypothesis that averaging the weights from divergent weights (when the norm difference is higher than the average norm difference) helps in stabilizing the model's parameters, resulting in a more robust model.



(a) IMDB (b) Diabetes (c) SST

(d) Anemia (e) AgNews (f) ADR Tweets

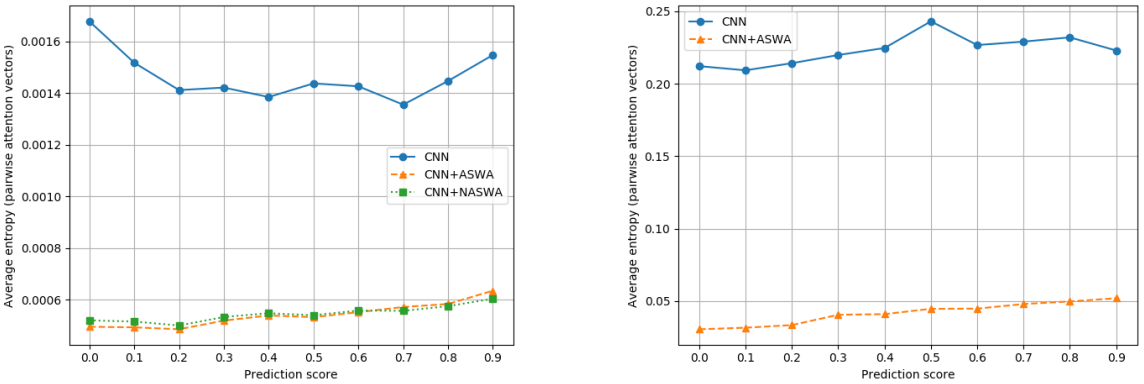
Figure 4.15: Attention stability improvement from ASWA and NASWA on CNN based models.



(a) IMDB (b) Diabetes (c) SST

(d) Anemia (e) AgNews (f) ADR Tweets

Figure 4.16: Attention stability improvement from ASWA and NASWA on LSTM based models.

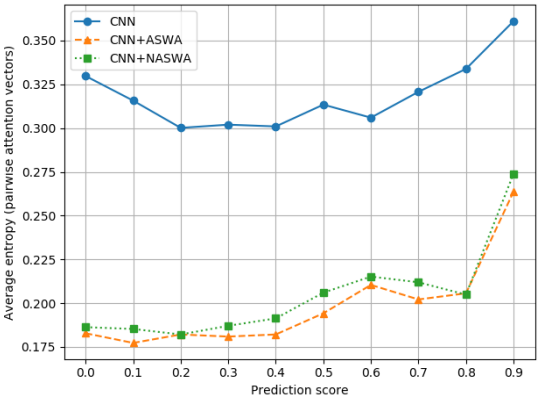


(a) Entropy improvement for dot Attention based CNN model for SST dataset.

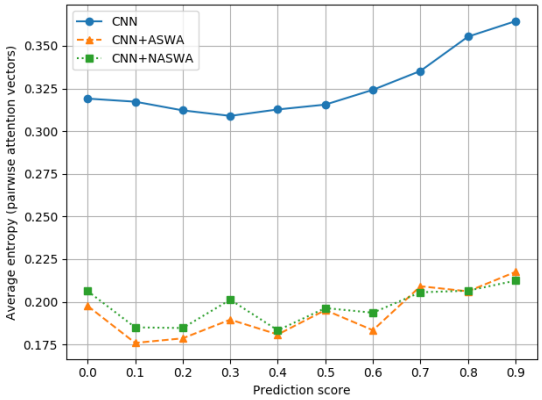
(b) Entropy improvement for dot Attention based CNN model for Diabetes dataset.

Figure 4.17

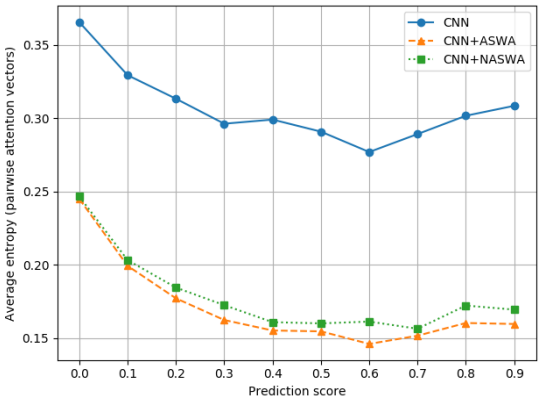
Multi-class Classification In Figure 4.18, we plot the entropy between the attentions distributions of the models for the SNLI dataset (CNN based model), separately for *each label* (*neutral*, *contradiction*, and *entailment*). We notice, similar observations as the binary classification models, the ASWA and NASWA algorithms are able to significantly improve the entropy of the attention distributions and increases the robustness of the model with random seeds.



(a) Label 0 prediction vs entropy



(b) Label 1 prediction vs entropy



(c) Label 2 prediction vs entropy

Figure 4.18: Attention stability improvement from ASWA and NASWA on CNN based model for the SNLI dataset.

Figure 4.19 highlights the achievements of our proposed techniques in terms of the attention weights' stability. It plots the standard deviation of the attention weights assigned to each word for a particular test case in the Twitter dataset (same as the one in Figure 4.1), comparing the results from the original model (trained normally), against a model trained using ASWA. As we can see, the deviation in the weight assigned to the word *"why"* is originally around 0.15. While, with the ASWA trained model, we are able to decrease the standard deviation by more than 80%. This decrease is deviation can be noticed for the rest of the words in the sentence as well. Similarly, Figure 4.20 illustrates the stability improvement on the test case shown in Figure 4.2a with our NASWA optimizer. Note that, for both the cases shown, the model prediction doesn't change significantly, although, the stability of the prediction scores improve with the ASWA and NASWA optimizers (as discussed in Section 4.6.1)

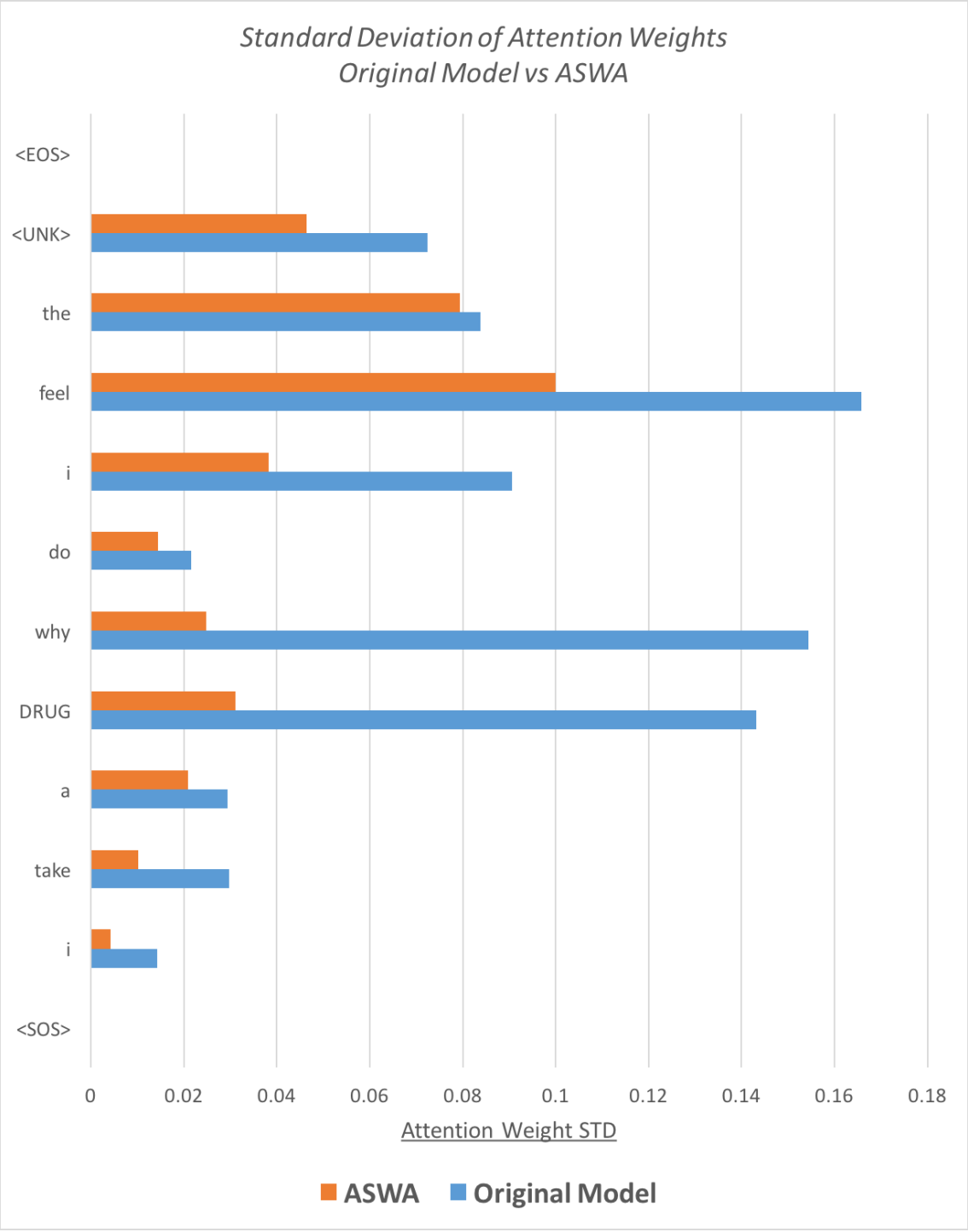


Figure 4.19: Figure highlighting the improvement in the *stability* of attention weights in a particular test case from the **Twitter** dataset.

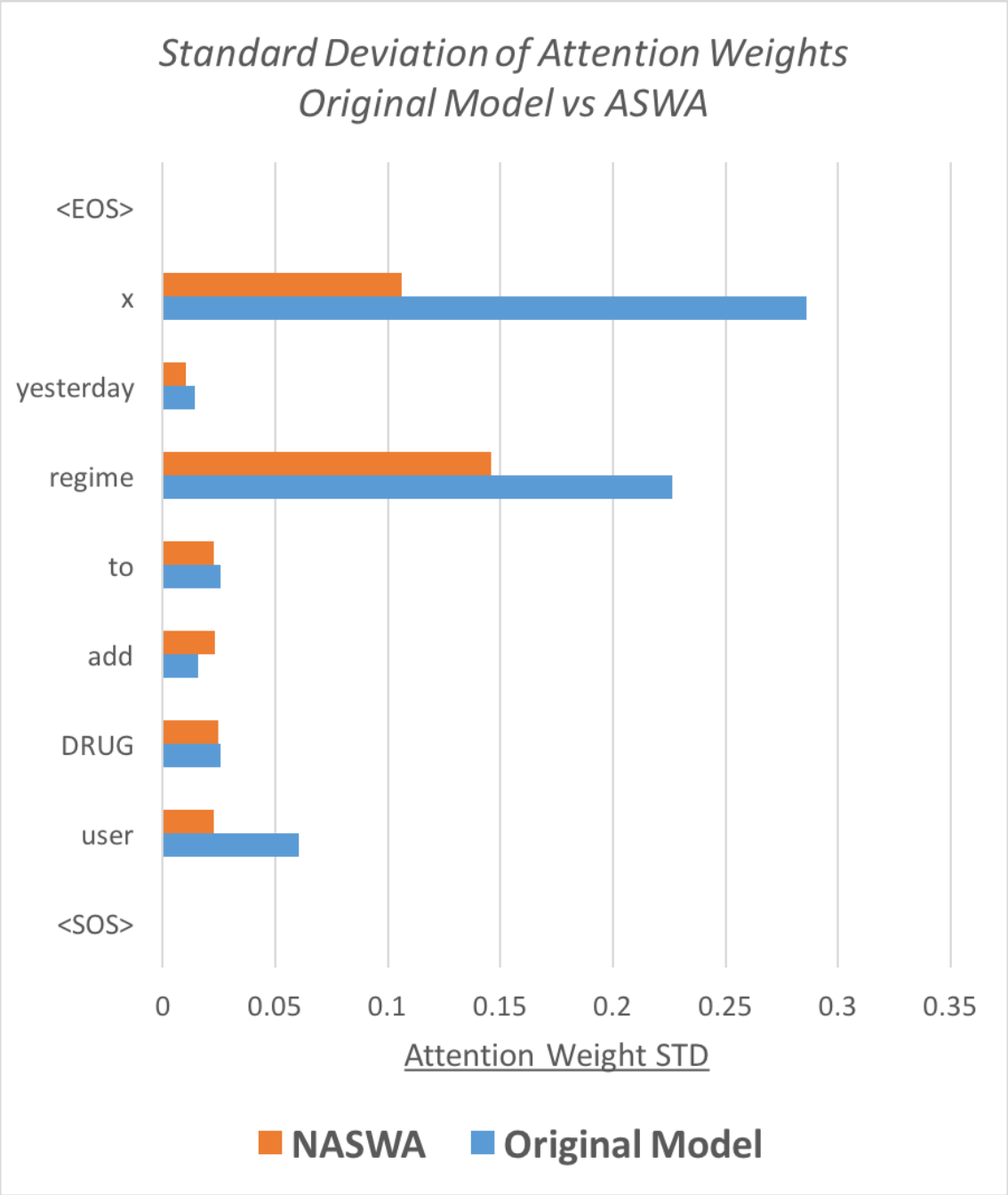
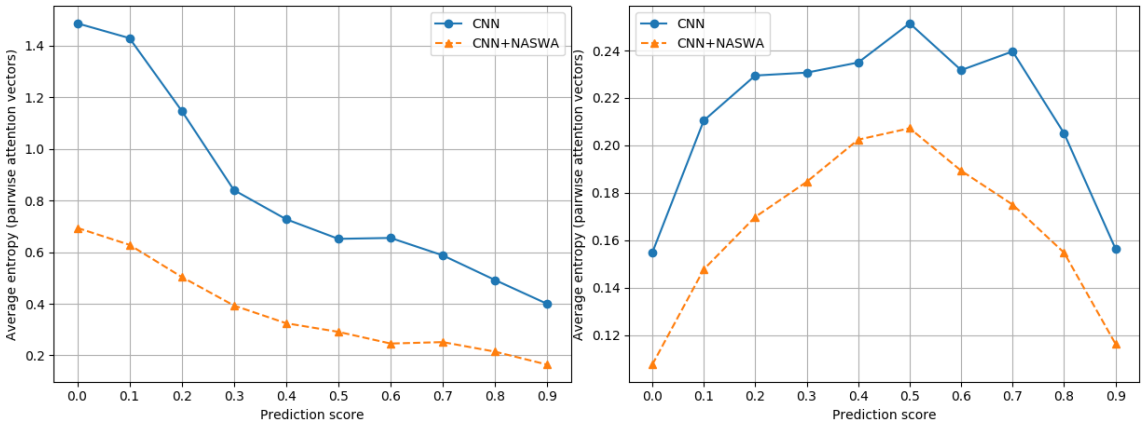


Figure 4.20: Figure highlighting the improvement in the *stability* of attention weights in a particular test case from the **Twitter** dataset.

4.6.3 Gradient-based explanations

We now look at an alternative method of interpreting deep neural models and look into the consistency of the gradient-based explanations to further analyze the model's instability. For this setup, we focus on binary classifier and plot the results on the SST and the Diabetes dataset in particular since they cover the low and the high end of the entropy spectrum (respectively). We notice similar trends of instability in the gradient-based explanations from model inputs as we did for the attention distributions. Figure 4.21 shows that the entropy between the gradient-based explanations from differently seeded models

closely follows the same trend as the attention distributions.



(a) Diabetes (b) SST

Figure 4.21: Gradient based explanation's stability improvement from NASWA on CNN based models.

This result further strengthens our claim on the importance of model stability and shows that over different runs of the same model with different seeds we may get different explanations using gradient-based feature importance. Moreover, Figure 4.21 shows the impact of ASWA and NASWA towards making the gradient-based explanations more consistent, thus, significantly increasing the stability.

4.7 Discussion

Recent advances in adversarial machine learning [NVL+15, ZKS+16] have investigated robustness to random initialization based perturbations, however, to our knowledge, no previous study investigates the effect of random-seeds and its connection on model interpretation. Our study analyzed the inherent lack of robustness in deep neural models for NLP. Recent studies cast doubt on the consistency and correlations of several types of interpretations [DVK17, JW19, FWGI+18]. We hypothesise that some of these issues are due to the inherent instability of the deep neural models to random-seed based perturbations.

Our analysis (in Section 4.5) shows that, on multiple runs of the model with equivalent settings, models may use completely different routes to reach similar decisions with high confidence. This issue of variance in all black-box interpretation methods over different seeds will continue to persist until the models are fully robust to random-seed based perturbations. Our work however, doesn't provide insights into instabilities of different layers of the models. We hypothesise that it might further uncover the reasons for the relatively lower correlation between different black-box interpretation methods as these are effectively based off on different layers and granularity.

There has been some work on using noisy gradients [NVL+15] and learning from adversarial and counter-factual examples [FWGI+18] to increase the robustness of deep learning models. [FWGI+18] show that neural models may use redundant features for prediction and also show that most of the black-box interpretation methods may not be able to capture these second-order effects. Our proposals show that aggressively averaging

weights leads to better optimization and the resultant models are more robust to random-seed based perturbation. However, our research is limited to increasing consistency in neural models. Our approach further uses first order based signals to boost stability. We posit that second-order based signals can further enhance consistency and increase the robustness.

4.8 Conclusions

In this chapter, we study the instability of deep neural models in NLP as a function of random initialization seeds. We analyze model performance and robustness of the model in the form of attention entropy and gradient-based feature importance entropy across multiple runs of the models with different random seeds. We propose a novel solution that makes use of aggressive weighted averaging and further extend it with norm-filtering and show that our proposed methods largely stabilize the model to random-seed based perturbations and, on an average, significantly reduce the standard deviations of the model performance by 72%. We further show that our methods significantly reduce the entropy in the attention distribution and the gradient-based feature importance measures across runs.

Chapter 5

Evaluation

Throughout the project we have been careful about evaluating our experiments since we wanted to make sure that our research was publishable. As mentioned earlier, we have been accepted to present our paper (based on the work in Chapter 3) in the EARS 2019 workshop[1]. We have submitted another paper (based on the work in Chapter 4) to the CoNLL[2] conference. We have ensured that the results and conclusions that we have discussed in various parts of the project are all backed with some form of empirical or theoretical results.

Performance

For our work on recommendation systems like DeepICF and NCF (Section 2.1.2), we follow the standardized metrics like HitRatio and NDCG to measure and compare the performance of our proposed models (explained in Section 3.1.4). Table 3.2 compares the performance of our modified models with the baseline *state-of-the-art* NCF and DeepICF models.

Moreover, for our work in Chapter 4, we measure the performance using the standard *classification accuracy* metric. Tables 4.2 and 4.3 compare the performance of our proposed techniques against the baseline trained models.

Calibration

In Chapter 3, we use the reliability diagrams (explained in Section 2.4) to evaluate the calibration of the models under inspection. These diagrams help us visualize and analyze the over-confidence (or under-confidence) in our models, for a range of confidence values. Figure 2.16, for instance, distinctly visualizes the degree of over-confidence in a model and how it varies with different confidence scores. We further use these reliability diagrams to form a connection between the calibration of a model and the reliability of its attention weights (as shown in Figure 3.8).

Attention stability

Measuring the stability of attention weights is an important aspect of the project. As discussed in Section 4.1.2, we did a comprehensive examination on the metrics to be used

[1] https://ears2019.github.io/
[2] https://www.conll.org/2019

to compare two attention distributions. We rely on these metrics to measure the reliability of a model's explanations (generated using their attention distributions). We mainly use *Entropy* and *Jaccard distance* for that purpose in Chapters 3 and 4.

Furthermore, in Chapter 4, we perform a thorough analysis on our proposals. We experimented with over 8 different datasets (SST, MIMIC-Diabetes, MIMIC-Anemia, Twitter, IMDB and others mentioned in Section 4.5.2) for Binary/Ternary classification, and Question-Answering tasks. We also tried different models architectures with LSTM, CNN based encoders and tanh, scaled dot-product based attention decoders.

In Chapter 3, however, we discuss our results from calibration and attention stability only from a limited number of experiments. This could have been improved, and further research/experiments should be done in order to make stronger conclusions.

For all our experiments, we also followed the standard procedure of seeding the process to make sure the experiments are easily reproducible (as mentioned in Section 4.2).

Chapter 6

Conclusion

In this project, we looked into an exceedingly important aspect of neural networks – their interpretability. Even with state-of-the-art results in a plethora of applications, their usage is restricted by their *black-box* nature. Nevertheless, there has been significant successful research in the last few years attempting to *open* this *black-box*.

This project aims to contribute to this ongoing research by opening the discussion to model dynamics like calibration and stability which, according to us, have the ability to affect a model's interpretability.

In Chapter 3, we try to draw a link between a model's calibration and the reliability of its attention weights, in the context of local explanations. We test our hypothesis on neural recommendation systems, although, we believe that the interaction between these two model dynamics should be prevalent in other neural models as well.

In Chapter 4, we look at model interpretability, from the perspective of its *stability*. We attempt to quantify the *in-stability*, imparted to the models during the training process. We further try to reduce this instability with our proposed algorithms – ASWA and NASWA (Section 4.4).

All these experiments we conduct and questions we ask correspond to the broader theme of model interpretability, especially in the context of deep learning. We believe the problems highlighted by our research are some of the basic issues that these models will have to overcome for better reception in the real world applications.

6.1 Challenges

The biggest challenge faced during the course of the project, especially in my case, was to be up-to-date with the state-of-the-art work. Given the broad range of topics that we touched upon in this project, it was very important for me to have meaningful context on them and the work relating to them. My supervisor guided me throughout the project and that significantly helped me tackle this challenge. It also helped us shape and give a direction to the project.

Another set of challenges faced during the course of this project stem from the fact that most of the research we did involves state-of-the-art work. For instance, in Chapter 3, although calibration has been a part of the literature for a long time, we could not find any work relating it to a model's interpretability. Besides, interpreting neural recommendation systems is a relatively new field of research in itself. The inspirations for our work in Chapter 4 – [JW19, IPG+18], are also very recent. This meant we had to be particularly

careful with our experiments and deductions since there is not enough research backing our work.

Working with neural networks brings its own set of challenges, given their *black-box* nature. It is especially hard to account for or reason about all the anomalies that may occur during the experiments (Figure 4.16f for instance).

Other challenges faced include debugging with libraries like `Tensorflow`, working with a variety of code bases (although, using an IDE like `PyCharm`[1] helped us navigate and understand the code) and datasets, and managing deadlines for paper submissions (for the conferences) with the college report deadlines among others.

6.2 Limitations and Future work

Although the analysis done in the project is state-of-art, it is still far from being conclusive. In the sections below, we discuss some limitations of our work and some possible extensions to the research done in the project.

6.2.1 Model explanations under Calibration (Chapter 3)

One of the shortcomings of our research in this Chapter 3 is that it is confined to a single model (DeepICF) and a single dataset (MovieLens). To the best of our knowledge, there are not many neural *recommendation* systems which use attention mechanism (due to the novelty of the field). Nevertheless, the same questions can be asked for other binary classification neural models (like sentiment analysis, image classification, and others) which use attention mechanism, especially the ones which are un-calibrated. A generalized analysis on different models and datasets would significantly strengthen the hypothesis.

Another interesting work can be looking into other ways to make the attention more reliable. For instance, in our work, we try the inverse-class weighted cross-entropy loss (Section 3.4.3), another interesting technique could be to sample lesser negative test cases for the training dataset. That way, we would not need to modify the loss, since the dataset would be balanced itself.

6.2.2 Model stability as a function of random seeds (Chapter 4)

In Chapter 4, we were able to perform a lot more comprehensive research on our hypothesis and proposal, as compared to Chapter 3. Nevertheless, there remain many interesting questions that are un-answered in this field of research.

One of the future directions for this project can be about further investigating the proposed approaches and analyzing the model's conduct when the weights get averaged at each iteration. The example in Figure 4.10 provides an intuition for how the ASWA and NASWA algorithms (Section 4.4) work, but it is also important to follow their behavior in a model as complicated as a deep neural network. In Figure 4.16f as well, we need to inspect why the LSTM based models, trained with ASWA and NASWA optmizers, give inconsistent results for the Twitter dataset, especially towards the right end of the x-axis when the confidence in the label is high.

Another extension to the project can be fine tuning the NASWA algorithm. As mentioned in Section 4.6.2, there is a possibility that averaging over weights from parameters

[1] https://www.jetbrains.com/pycharm/

whose norm difference is relatively higher might lead to better results. We can alter the NASWA algorithm to compare norm-differences from different parameters and take only specific parameters into account (instead of averaging the norm-difference from all of them) when deciding to skip a particular iteration for the running average.

One can also extend the project to datasets and models outside NLP applications and try to apply the techniques proposed and see if they work for them as well.

As mentioned earlier, we can also perform a layer-wise stability analysis for the model. That way we would be able to point out the layers/parameters that particularly contribute to the instability of the model. This could eventually help us narrow down the stability problem to granular versions and thereby enable us to find better solutions.

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[Abd16] Behnoush Abdollahi. Explainable restricted boltzmann machines for collaborative filtering. 2016.

[AMJ17] David Alvarez-Melis and Tommi S Jaakkola. A causal framework for explaining the predictions of black-box sequence-to-sequence models. *arXiv preprint arXiv:1707.01943*, 2017.

[BAPM15] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BCKW15] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[Ben12] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[BHKR17] Immanuel Bayer, Xiangnan He, Bhargav Kanagal, and Steffen Rendle. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1341–1350. International World Wide Web Conferences Steering Committee, 2017.

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[BSH+10] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010.

[C+15] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[Cas] Alex Castrounis. Artificial intelligence, deep learning, and neural networks, explained.

[CBS+16] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

[Cha17] Akshay Kumar Chaturvedi. Recommender system for news articlesusing supervised learning. 2017.

[Che18] Xu Chen. Visually explainable recommendation. 2018.

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[DVK17] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[EBC+10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[EBCV09] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[FWGI+18] Shi Feng, Eric Wallace, Alvin Grissom II, Pedro Rodriguez, Mohit Iyyer, and Jordan Boyd-Graber. Pathologies of neural models make interpretation difficult. In *Empirical Methods in Natural Language Processing*, 2018.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[GBY+18] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89. IEEE, 2018.

[GFT18] Reza Ghaeini, Xiaoli Z Fern, and Prasad Tadepalli. Interpreting recurrent and attention-based neural models: a case study on natural language inference. *arXiv preprint arXiv:1808.03894*, 2018.

[GIP+18] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

[GRMS99] Michel Goossens, S. P. Rahtz, Ross Moore, and Robert S. Sutor. *The Latex Web Companion: Integrating TEX, HTML, and XML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[GS+66] David Marvin Green, John A Swets, et al. *Signal detection theory and psychophysics*, volume 1. Wiley New York, 1966.

[Gun17] David Gunning. Explainable artificial intelligence. 2017.

[Guo17] Chuan Guo. On calibration of modern neural networks. 2017.

[Ham97] Thomas M Hamill. Reliability diagrams for multicategory probabilistic forecasts. *Weather and forecasting*, 12(4):736–741, 1997.

[HCKC15] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1661–1670. ACM, 2015.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[He17] Xiangnan He. Neural collaborative filtering. 2017.

[Her00] Jonathan L. Herlocker. Explaining collaborative filtering recommendations. 2000.

[HHS+18] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018.

[Hid16] Balazs Hidasi. Session-based recommendations with recurrent neural networks. 2016.

[HKG+15] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

[HKR00] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM, 2000.

[HLZ+17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZKC16] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558. ACM, 2016.

[IPG+18] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[JGB+16] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[Jia03] Yulei Jiang. Uncertainty in the output of artificial neural networks. *IEEE transactions on medical imaging*, 22(7):913–921, 2003.

[Jia07] Yulei Jiang. Uncertainty in the output of artificial neural networks. In *2007 International Joint Conference on Neural Networks*, pages 2551–2556. IEEE, 2007.

[JMW19] Sarthak Jain, Ramin Mohammadi, and Byron C Wallace. An analysis of attention over clinical notes for predictive tasks. *arXiv preprint arXiv:1904.03244*, 2019.

[Jol11] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[JOP+] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed <today>].

[JPS+16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[JW19] Sarthak Jain and Byron C. Wallace. Attention is not explanation. *CoRR*, abs/1902.10186, 2019.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KZ01] Gary King and Langche Zeng. Logistic regression in rare events data. *Political analysis*, 9(2):137–163, 2001.

[L+17] Tao Lei et al. *Interpretable neural models for natural language processing*. PhD thesis, Massachusetts Institute of Technology, 2017.

[Lip17] Zachary C. Lipton. European union regulations on algorithmic decision-making and a 'right to explanation'. 2017.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[LS99] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[LS01] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[LTB+13] Will Landecker, Michael D Thomure, Luís MA Bettencourt, Melanie Mitchell, Garrett T Kenyon, and Steven P Brumby. Interpreting individual classifications of hierarchical networks. In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 32–38. IEEE, 2013.

[MA16] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pages 1614–1623, 2016.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[MDP+11] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[MH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[MHB17] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017.

[MV15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[MWD+18] James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. *arXiv preprint arXiv:1802.05695*, 2018.

[NCH15] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[NSO+15] Azadeh Nikfarjam, Abeed Sarker, Karen O'connor, Rachel Ginn, and Graciela Gonzalez. Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. *Journal of the American Medical Informatics Association*, 22(3):671–681, 2015.

[NVL+15] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

[PJ92] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[RFGST09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[RJ] Pranava Madhyastha Rishabh Jain. Model explanations under calibration. In *Proceedings of the 2nd International Workshop on ExplainAble Recommendation and Search (EARS 2019)*, Paris, France.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[Rup88] David Ruppert. Stochastic approximation. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SPW+13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[TM07] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *2007 IEEE 23rd international conference on data engineering workshop*, pages 801–810. IEEE, 2007.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[WHF+18] Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. Tem: Tree-enhanced embedding model for explainable recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1543–1552. International World Wide Web Conferences Steering Committee, 2018.

[Wik19a] Wikipedia contributors. Euclidean distance — Wikipedia, the free encyclopedia, 2019. [Online; accessed 5-June-2019].

[Wik19b] Wikipedia contributors. Jaccard index — Wikipedia, the free encyclopedia, 2019. [Online; accessed 18-May-2019].

[XBK+15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[XHW+18] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. Deep item-based collaborative filtering for top-n recommendation. *arXiv preprint arXiv:1811.04392*, 2018.

[XMDH17] Qizhe Xie, Xuezhe Ma, Zihang Dai, and Eduard Hovy. An interpretable knowledge transfer model for knowledge base completion. *arXiv preprint arXiv:1704.05908*, 2017.

[Xue18] Feng Xue. Deep item-based collaborative filtering for top-n recommendation. 2018.

[YHPC18] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[ZC18] Yongfeng Zhang and Xu Chen. Explainable recommendation: A survey and new perspectives. *arXiv preprint arXiv:1804.11192*, 2018.

[Zha18a] Shuai Zhang. Deep learning based recommender system: A survey and new perspectives. 2018.

[Zha18b] Yongfeng Zhang. Explainable recommendation: A survey and new perspectives. 2018.

[ZKS+16] Tom Zahavy, Bingyi Kang, Alex Sivak, Jiashi Feng, Huan Xu, and Shie Mannor. Ensemble robustness and generalization of stochastic deep learning algorithms. *arXiv preprint arXiv:1602.02389*, 2016.

[ZW15] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

Appendix A

EARS paper

Model Explanations under Calibration

Rishabh Jain
jain.rishabh8897@gmail.com
Imperial College London

Pranava Madhyastha
pranava@imperial.ac.uk
Imperial College London

ABSTRACT

Explaining and interpreting the decisions of recommender systems are becoming extremely relevant both, for improving predictive performance, and providing valid explanations to users. While most of the recent interest has focused on providing local explanations, there has been a much lower emphasis on studying the effects of model dynamics and its impact on explanation. In this paper, we perform a focused study on the impact of model interpretability in the context of calibration. Specifically, we address the challenges of both over-confident and under-confident predictions with interpretability using attention distribution. Our results indicate that the means of using attention distributions for interpretability are highly unstable for un-calibrated models. Our empirical analysis on the stability of attention distribution raises questions on the utility of attention for explainability.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

calibration, explanations, attention, recommender systems, deep learning

ACM Reference Format:
Rishabh Jain and Pranava Madhyastha. 2019. Model Explanations under Calibration. In *Paris '19: International Workshop on Explainable Recommendation and Search at ACM-SIGIR.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recommendation systems are used for item filtering based on user preferences in a variety of areas including movies, news, books, social recommendations and products in general. Some commonly used approaches to recommendation systems include Collaborative filtering, Content-based filtering and hybrid systems. There has been an increased interest in the community in utilizing deep learning based models for recommender systems [18]. These models can alleviate several limitations of traditional models including complex non-linear transformations, interactions with different types and modalities of inputs. Deep learning based models have been particularly shown to be flexible and are known to incorporate additional data when training and can learn from large amounts of auxiliary information, which is usually available to recommendation systems. As deep models are modular than other rigid algorithms they are easily adaptable and extendable.

The significance of explaining automated recommendations is widely acknowledged [7, 13]. Explanations build user trust, improve their experience, and also give them the opportunity to fix incorrect representations or recommendations. For these reasons, there has been extensive research on ways to explain different types of recommendation systems. We refer the reader to Zhang and Chen [19] for a detailed survey on explainable systems.

Deep learning based recommendation systems have opened up one way of explaining neural models' outputs in the context of recommendations [19] — by using attention distributions. In this context, neural attention mechanisms have gained significant focus, as they have been shown to not only help the model perform better, but also provide explanations by highlighting the input features that play a significant part in computing the model's output [2, 17]. However, it is has been recently indicated that attention may not always provide a reliable form of explanation, especially in the domain of natural language processing [8].

One of the emerging problems with the modern neural network models (especially deep neural networks) is their poor calibration [3]. Over-confident or under-confident predictions can make a model unreliable, especially in sensitive scenarios like health care (disease detection), autonomous driving among others [3].

In this paper, we focus on a form of recommendation system that aims to answer *why* a certain recommendation has been made. Especially, we investigate the reliability of attention distributions in deep neural attention based recommendation systems.

2 BACKGROUND AND TOOLS

In this paper, we investigate the utility of attention with a state-of-the-art deep neural network based model with attention [17]. In this section, we succinctly describe the necessary background and the tools under consideration.

2.1 Attention Distribution

Attention mechanisms, in neural networks, are known to provide the functionality for the model to focus on certain parts of the inputs or features. An attention mechanism in recommendation systems is usually over u, a user representation, with the set of item specific representations $\{v_i\} \in \mathcal{V}$ where $\mathcal{V}$ is the domain of all item representations. A compatibility function maps u and $\{v_i\}$ to a scalar distribution, which is then typically converted into a probability distribution using a softmax operator. This usually results in a distribution where some items get more probability mass than others, indicating their influence in the decision made by the system. In this paper, we focus on such attention distributions and are interested in their reliability. We are especially interested

Paris '19, July 25, 2019, Paris, France

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

in understanding the behaviour of the models when the models are mis-calibrated.

Explanation using Attention. In neural recommendation systems (and neural networks in general), attention is increasingly being used, not just to improve the model's performance but also as a means to explain the model's predictions [1, 14, 17]. The attention maps (heat-maps) are used to indicate which input features to the model were majorly responsible for the model's predictions. In Figure 1 (from a movie recommendation system from Xue et al. [17]), for instance, for target item #1525, the attention-network assigns the maximum weight to the item #1254 (one of the previously interacted items of the target user). This information can be used to generate a human-readable explanation like "You are recommended to watch #1525 because you watched #1254".

More recently, there has been research on the reliability of attention-maps based explanations [9] and if they can be used to explain a model. In this paper, we work on this line of research in the context of recommendation systems and their calibration(2.2).

2.2 Model Calibration

Classification models used as part of any decision process need to be both accurate in their predictions, and should also indicate when they are probably incorrect. Model calibration is the degree to which a model's predicted probability correlates with its true correctness likelihood. Calibration measures this property of a model. For example, if a perfectly calibrated model gives 100 different predictions, each with 80% confidence (probability), 80 of the predictions should be classified correctly.

We use the concept of calibration to plot reliability diagrams [4]. A reliability diagram can be defined as the accuracy of the model as a function of its confidence. First, we bucket all the predictions based on the predicted probabilities. Reliability diagrams help us visualize a model's calibration. A reliability plot which falls below the identity function suggests that the model is over-confident of its predictions (blue plot in Figure 3) since it means that the ground truth likelihood (accuracy) is less than the model's confidence in its predictions. On the other hand, it is considered under-confident if the reliability plot is above the identity function. For a perfectly calibrated model, the reliability plot is the identity function.

2.3 Attention Permutation

One of the experiments we perform to check the reliability of attention based explanations is permuting the weights randomly and recording the effects of the permutations on the output of the model (inspired from Jain and Wallace [9]).

Since the particular weights assigned to the input features are used as the basis for the explanations, permuting these weights randomly should cause the model's prediction to change by a substantial margin. In case the predictions remain unchanged it indicates that the attention necessarily doesn't contribute to the predictions. This can be concerning especially when using attention as grounds for explanations.

2.4 Model Stability

In our study, we refer to model stability as the consistency of model predictions and internal parameters with different runs of the model by only changing random seeds. [10, 11]. The seed values are responsible for regulating the training dynamics (weight initialization, training batch generation, among others). This way, we get to measure the impact of these random processes on the output of the model (and the attention weights).

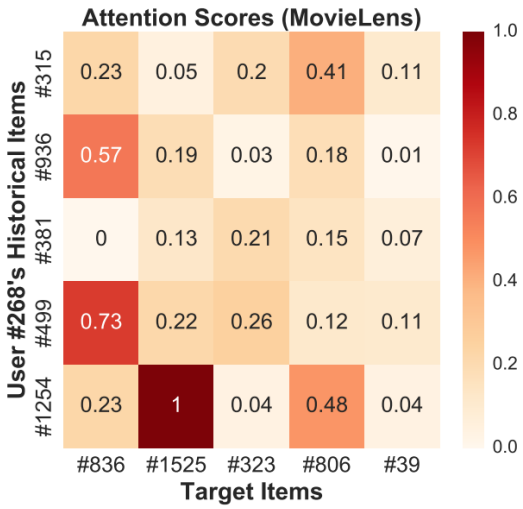


Figure 1: Attention map showing weights assigned to input features

3 EXPERIMENTAL SETUP

In the following sections we describe our experiments and observations.

3.1 DeepICF with attention

The DeepICF model uses a deep neural network to learn latent low-dimensional embeddings of users and items that capture implicit and explicit user interactions. It uses a pair-wise interaction layer, which consists of an element-wise product (also called the Hadamard product[15]) of the target item's latent vector with each of the historical items' vectors. The model then follows this with the pooling layer whose output is a vector of fixed size, to facilitate the deep interaction of layers. This is done via *attention based pooling*. The output of the pooling layer is a vector which condenses the second-order interaction between historical items and the target item (we refer the reader to Xue et al. [17] for a detailed explanation of the model). Finally, the higher order interactions are captured with a multi-layer perceptron. The output of the model is a sigmoid on the final layer's weighted sum.

Modifications: We replaced sigmoid function with a softmax with two outputs over the two classes and trained the model with cross-entropy loss.

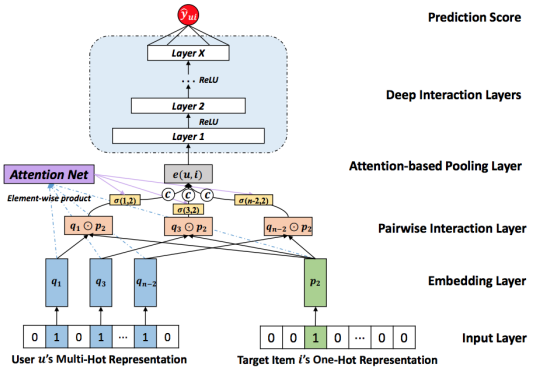


Figure 2: Deep ICF with Attention (from Xue et al. [17])

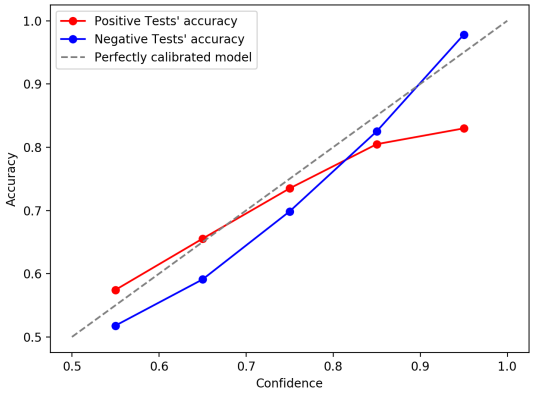


Figure 3: Calibration plot (Reliability diagram) of the DeepICF model.

3.2 Dataset, Evaluation and Hyperparameters

We train, evaluate the model and perform our experiments on the MovieLens[1] dataset. This dataset has been commonly used to evaluate collaborative filtering algorithms. The dataset contains one million ratings where each user has at least 20 ratings and use the standard splits. In our study, we retain the standard procedure used in DeepICF where the original dataset is transformed such that each user item entry is marked as 1: when there is some interaction between the user and item and -1: when there is no interaction the between the target user and item.

Evaluation: For evaluation purposes, the standard metrics used are HitRatio (HR@10) and the NDCG@10(Normalized Discounted Cumulative Gain [5]) as the main metrics. We further use the binary labelling accuracy to investigate the model performance per class, where the classes are defined as: -1 when there is no interaction between the user and items and 1 when there is an explicit interaction (user ratings for the item).

Hyperparameter Settings: For training purposes, we use the same hyper-parameters as mentioned in the paper [17]. We use the original DeepICF implementation[2].

4 RESULTS

Table 1 compares the performance of the softmax output model with the original DeepICF model and the state-of-the-art Neural Matrix Factorization model[6]. We observe that the performance of our model is highly competitive and performs as well as the DeepICF with pretraining. In the following sections, we will investigate the reliability of models and the attention distribution in the models.

4.1 Calibration

We plot the reliability diagram for the DeepICF model by bucketing the model predictions based on their confidence and calculating the accuracy for each of the buckets.

We see in Figure 3, for positive test cases, the DeepICF (with attention) model seemingly tends to be over-confident as the confidence increases, where the model tends to be extremely confident about predicting the positive class without being as accurate. This can be problematic especially when dealing with real-world production systems. We also notice that the model is seemingly over-confident in predicting the negative class. This could be because of the imbalance in the dataset where the dataset is extremely skewed towards the negative class. We also note that the test-set has a very high degree of imbalance in the number of positive and negative test cases in our test set (1 positive sample for every 99 negative tests). This is impacted in Figure 3 as it shows the curves for positive and negative test samples separately.

4.2 Attention Permutation

What is the effect of over-confidence over attention? In order to test the reliability of explanations generated from attention, we permute the attention weights randomly and notice the effect of the permutation on the output of the model (as described in Section 2.3).

Specifically, in DeepICF, as shown in Figure 2, the attention based pooling layer assigns a weight for each of the user and item interaction, where the magnitude of the weights indicate the importance of the interaction. In this experiment, we randomly *shuffle* these weights amongst the items and record the difference in the output prediction score (originally classified interaction label). We randomly shuffle the weights 100 times (as performed in Jain and Wallace [9] for each test case, and average the absolute variations in the output predictions.

We plot the average variations in false negatives (right axis) against the confidence of the predicted output for the positive test cases in Figure 4. We focus on positive test cases as it is the most salient label to measure the model. The plot also contains the reliability diagram for the model (left axis). We note that the perturbations especially have barely any effect on the mis-calibrated cases. In both false positives and false negatives (these increase with mis-calibration), we notice similar trends where the effect of permuting the attention weights decreases as the confidence in the predicted label increases. **Thus, showing that model explanations generated from the attention distribution become less reliable with over-confident predictions.**

[1] https://grouplens.org/datasets/movielens/1m/
[2] https://github.com/linzh92/DeepICF

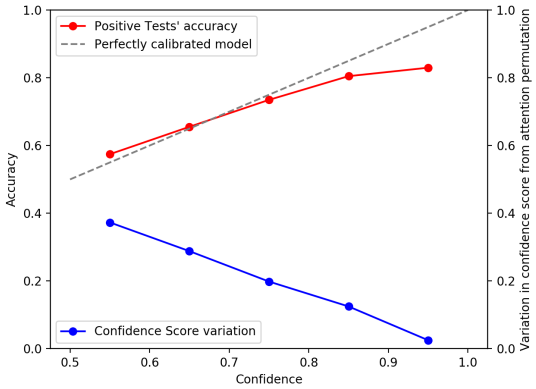


Figure 4: Figure showing the effect of attention permutation (right axis) on the prediction score of wrongly (negatively) classified positive test cases (false negatives).

4.3 Fixing the effect of Class-imbalance

As the training split of the dataset is heavily imbalanced: 4 negative labels (no interactions) for every positive label, we use a simple class-weighting heuristic, to cope with this imbalance in the training set and modify our cross-entropy loss. The new loss is calculated by assigning weights to the losses from the test cases such that the loss contribution from both the classes (positive and negative interactions) is balanced[12]. We retrain the model with the new loss function and were able to achieve similar HitRatio values to the original model as shown in Table 1. We analysed the effect of attention permutation (Section 4.2) on this model. Figure 5 compares the new model to the previous model's results. We notice that the new model is considerably more sensitive to attention permutation, compared to the original one. **This suggests that attention based explanations from the class-balanced loss model are more reliable than the original model.**

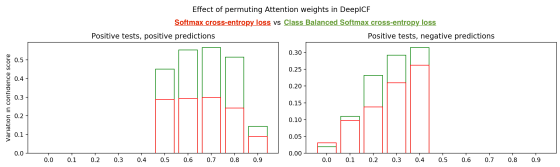


Figure 5: Effect of permuting attention weights.

4.4 Stability of DeepICF

We now consider the effect of random seeds and on initialization of model parameters and in general the model performance. We notice in Table 1 — the standard deviation is generally very low suggesting that the performance of the model is seemingly stable and it seems to have small deviation.

Attention score stability: What is the effect of random seeds on attention distribution? As we are interested in the reliability of attention *explanations*, we focus on the stability of attention scores

Model type	Hit Ratio@10 (%)	NDCG@10 (%)
DeepICF*	68.81	41.13
DeepICF*+Pretrain	70.84	43.80
NeuMF*+Pretrain	70.70	42.60
DeepICF (ours)	70.41(±0.24)	43.00 (±0.34)
DeepICF+cls-wt	68.61	41.14

Table 1: Performance Comparison for DeepICF and NeuMF[6]. * indicates scores directly from the corresponding papers. The standard deviation (±) is obtained with 10 runs of the model with different random seeds.

in DeepICF. We perform the same experiment by running the same model but with 10 different random seeds and record the *top 10%* of the most attentive items (user-item interactions which get the highest attention weight assigned) for every particular test case for each model. Then we compare if these top 10 percent most attentive items for a particular test case are consistent for different runs of the models with different random seedst. We calculate the similarity between two sets of items by computing the `Jaccard Index` [16] of the sets. We calculate the Jaccard Index for every possible pair of sets of top attentive items and average over them. Figure 6 shows that the average `Jaccard Index` for positive predictions with high confidence is around 0.5 (where max `Jaccard Index` is 1, implying completely stable attention scores). **This highlights that the**

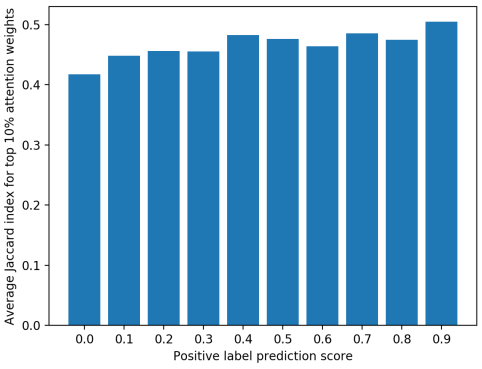


Figure 6: Attention score stability.

attention explanations from two identical models, trained with different seeds for the same input can vary, severely highlighting the unreliability of such explanations.

5 CONCLUSION

In this paper, we have explored the importance of model dynamics and its relation to explanation using attention. Concretely, we observe that attention may not be reliable when the selected model is especially mis-calibrated. We have explored one possible way of stabilizing the model by accounting for the class imbalance.

Significantly, we noticed that using an inverse-class weighted cross-entropy formulation can help improve the stability of attention distribution. Further, we observe that over different runs of models with different random seeds, the models seem to obtain different attention distributions. We posit that our work is extremely relevant to the community and can orient towards an important discussion on the reliability of using attention as an explanation.

REFERENCES

[1] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. 2018. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 80–89.

[2] Michel Goossens, S. P. Rahtz, Ross Moore, and Robert S. Sutor. 1999. *The Latex Web Companion: Integrating TEX, HTML, and XML* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1321–1330.

[4] Thomas M Hamill. 1997. Reliability diagrams for multicategory probabilistic forecasts. *Weather and forecasting* 12, 4 (1997), 736–741.

[5] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. 2015. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1661–1670.

[6] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[7] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. 2000. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 241–250.

[8] Sarthak Jain, Ramin Mohammadi, and Byron C Wallace. 2019. An Analysis of Attention over Clinical Notes for Predictive Tasks. *arXiv preprint arXiv:1904.03244* (2019).

[9] Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. *CoRR* abs/1902.10186 (2019). arXiv:1902.10186 http://arxiv.org/abs/1902.10186

[10] Yulei Jiang. 2003. Uncertainty in the output of artificial neural networks. *IEEE transactions on medical imaging* 22, 7 (2003), 913–921.

[11] Yulei Jiang. 2007. Uncertainty in the output of artificial neural networks. In *2007 International Joint Conference on Neural Networks*. IEEE, 2551–2556.

[12] Gary King and Langche Zeng. 2001. Logistic regression in rare events data. *Political analysis* 9, 2 (2001), 137–163.

[13] Nava Tintarev and Judith Masthoff. 2007. A survey of explanations in recommender systems. In *2007 IEEE 23rd international conference on data engineering workshop*. IEEE, 801–810.

[14] Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. 2018. Tem: Tree-enhanced embedding model for explainable recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1543–1552.

[15] Wikipedia contributors. 2019. Hadamard product (matrices) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Hadamard_product_(matrices)&oldid=895450816 [Online; accessed 19-May-2019].

[16] Wikipedia contributors. 2019. Jaccard index — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=890384326 [Online; accessed 18-May-2019].

[17] Feng Xue, Xiangnan He, Xiang Wang, Jiandong Xu, Kai Liu, and Richang Hong. 2018. Deep Item-based Collaborative Filtering for Top-N Recommendation. *arXiv preprint arXiv:1811.04392* (2018).

[18] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 5.

[19] Yongfeng Zhang and Xu Chen. 2018. Explainable recommendation: A survey and new perspectives. *arXiv preprint arXiv:1804.11192* (2018).

Appendix B

CoNLL paper

On Model Stability as a Function of Random Seed

Rishabh Jain
Department of Computing
Imperial College London
rishabh.jain15@imperial.ac.uk

Pranava Madhyastha
Department of Computing
Imperial College London
pranava@imperial.ac.uk

Abstract

In this paper, we focus on quantifying model stability as a function of random seeds by investigating the effects of the induced randomness on model performance and the robustness of the model in general. We specifically perform a controlled study on the effect of random seeds on the behaviour of attention and gradient-based interpretations. Our analysis suggests that random-seeds can adversely affect the consistency in models resulting in counter-factual explanations and interpretations. We propose a novel technique called *Aggressive Stochastic Weight Averaging (ASWA)* and an extension called *Norm-filtered Aggressive Stochastic Weight Averaging (NASWA)* which improves the stability of model over random-seeds. With our ASWA and NASWA implementations, we are able to improve the robustness of the original model, on an average, reducing the standard deviation of the model's performance by 72%.

1 Introduction

There has been tremendous growth in deep neural based models with state-of-the-art performance. In fact, most recent end-to-end deep learning models have surpassed the performance of careful human feature-engineering based models in most NLP tasks. However, deep neural network-based models are often brittle to some sources of randomness in the training of the models. This could be attributed to several sources including, but not limited to, random parameter initializations, random sampling of examples and sampling of activations. It has been observed that these models have, more often, a set of *'random seeds'* that yield better results than others. This has also lead to research suggesting random seeds as an additional hyperparameter for tuning (Bengio, 2012)[1].

[1] http://www.argmin.net/2018/02/26/nominal/

One possible explanation for this could be the existence of multiple local minima in the loss surface. This is especially problematic as the loss surfaces are non-convex and may have multiple saddle points making it difficult to have a stable model, that is, to a large extent, robust to random-seed based effects.

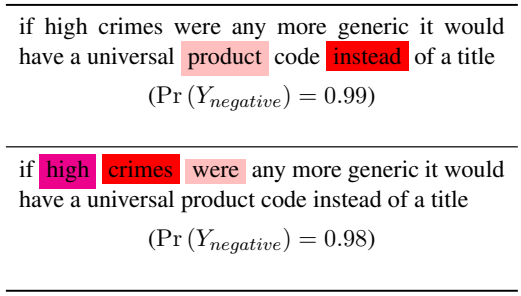


Figure 1: Importance based on attention probabilities for two runs of the same model with **same parameters and same hyperparameters**, but with **two different random seeds** (color magnitudes: pink<magenta<red)

Recently the NLP community has found new interests in interpreting and explaining deep neural models (Jain et al., 2019; Jain and Wallace, 2019; Alvarez-Melis and Jaakkola, 2017). Most of the interpretation based methods involve one of the following ways of interpreting models: a) Adversarial perturbation based interpretations: where the interpretation is based on change in prediction score with counter-factual perturbations (Jain et al., 2019; Jain and Wallace, 2019); b) Interpretations based on feature attributions using attention or input perturbation or gradient-based measures; (Ghaeini et al., 2018; Feng et al., 2018); c) Explanation using surrogate linear models (Ribeiro et al., 2016). These methods can provide local interpretations based on input samples or features. However, the persisting randomness makes it difficult to accurately interpret neural

models among other forms of pathologies (Feng et al., 2018).

In this paper, we focus on the stability of deep neural models as a function of random-seeds. We are especially interested in investigating the hypothesis of model stability: do neural network based models under different random seeds allow similar explanations? In Figure 1, we show an illustration of this question where we have the attention distributions of two CNN based binary classification models for sentiment analysis, trained with the same settings and hyper-parameters, but with different seeds. We see that both models obtain the correct prediction with significantly high confidence. However, we note that both the models attend to completely different sets of words. This is problematic, especially when interpreting these models under the influence of such randomness.

We also provide a simple method that can, to a large extent, ameliorate this inherent random behaviour. In Section 3, we propose an aggressive stochastic weight averaging approach that helps in improving the stability of the models at almost zero performance loss while still making the model robust to random-seed based instability. We also propose an improvement to this model in Section 3.1 which further improves the stability of the neural models. Our proposals significantly improve the robustness of the model, on an average, by 72% relative to the original model and on Diabetes (MIMIC), a binary classification dataset, by 89% (relative improvement).

2 Model Stability

In this section, we define prediction, attention-based explanation, and gradient-based explanation stability.

2.1 Prediction Stability

We define prediction stability in two parts, the first corresponds to the standard measures of the mean and the standard deviations corresponding to the accuracy of the binary classification based models on different datasets. We ensure that the models are run with exactly the same configurations and hyper-parameters but with different random seeds. This is a standard procedure that is used in the community to report the performance of the model.

2.2 Attention Stability

We define attention stability using the robustness of the attention distributions. That is, we call the attention probabilities of a model stable if the model, over several runs with different random seeds (but with the same settings and hyper-parameters) has similar attention probability distributions. We consider this to be extremely important for both interpretations and explanations using attention and the utility of models in general. We also find that this is important even for interpretations using leave one out based methods or local interpretations using surrogate models. This is because, if there is variance due to random seeds (mostly due to induced randomness in the initialization of weights and biases), the general interpretations would vary for each model.

We now focus on attention distributions to quantify instability. We use two metrics to investigate the instability in models: a) **Entropy quantification ($\mathcal{H}$):** Given two attention distributions for the same test case from two different models, it measures the entropy between the two probability distributions. Note that, the higher the entropy the greater the dissimilarity between the two distributions.

$$\mathcal{H} = \sum_{i \in d} \mathrm{Pr}_1 \cdot \log \frac{\mathrm{Pr}_1}{\mathrm{Pr}_2}$$

where, Pr_1 and Pr_2 are two attention distributions of the same sample from two different runs of the model and d is the number of tokens in the sample. Given n differently seeded models, for each test instance, we calculate the averaged pairwise attention distributions' entropy.

b) **Jaccard Distance ($\mathcal{J}$):** It measures the dissimilarity between two sets. Here higher values of $\mathcal{J}$ indicate larger variances. We consider top-n tokens which have the highest attention for comparison. Note that, Jaccard distance is over sets of word indices and do not take into account the attention probabilities explicitly. Jaccard distance is defined as:

$$\mathcal{J} = (1 - \frac{A \cap B}{A \cup B}) * 100\%$$

where, A and B are the sets of most relevant items. We specifically decided to use 'most' relevant (top-n items) as the tail of the distribution mostly consists of values close to 0.

2.3 Gradient-based Interpretation

Gradient-based feature importance is another way to interpret the model for local explanations. We use the input gradients of the model for each word embedding and compute the magnitude of the change as a local explanation. We refer the reader to Baehrens et al. (2010) for a good introduction to gradient-based interpretations. As all of our models are differentiable, we use this as an alternative method for interpretation. We note that we do not follow Jain and Wallace (2019) and do not disconnect the computational graph at the attention module. We follow the standard procedure as followed in Feng et al. (2018). We use entropy as described in Section 2.2 to quantify the instability.

3 Aggressive Stochastic Weight Averaging (ASWA)

Stochastic weight averaging (SWA) (Izmailov et al., 2018) works by averaging the weights of multiple points in the trajectory of gradient descent based optimizers. The algorithm typically uses modified learning rate schedules. SWA is itself based on the idea of maintaining a running average of weights in stochastic gradient descent based optimization techniques (Ruppert, 1988; Polyak and Juditsky, 1992). The principle idea in SWA is averaging the weights that are maximally distant helps stabilize the gradient descent based optimizer trajectory and improves generalization. Izmailov et al. (2018) use the analysis of Mandt et al. (2017) to illustrate the stability arguments where they show that, under certain convexity assumptions, SGD iterations can be visualized as samples from a Gaussian distribution centred at the *minima* of the loss function. Samples from high-dimensional Gaussians are expected to be concentrated *on the surface of the ellipse* and not close to the *mean*. Averaging iterations is shown to stabilize the trajectory and further improve the width of the solutions to be closer to the *mean*.

Izmailov et al. (2018) suggest using SWA usually after pre-training the model (at least until 75% convergence) and after which they suggest sampling weights at different steps either using large constant or cyclical learning rates. As SWA is well defined for convex losses (Polyak and Juditsky, 1992), the authors connect SWA to non-convex losses by suggesting that the loss surface is *approximately* convex after convergence.

In this paper, we focus on the stability of deep neural models as a function of random-seeds. Our proposal is based on SWA, but we extend it to the extremes and call it *Aggressive Stochastic Weight Averaging*. We assume that, for small batch size, the loss surface is locally convex. Hence, we propose to perform the weight averaging at every step of the batch size b. We further relax the conditions for the optimizer and assume that the optimizer is based on some version of gradient descent — this means that our modification is valid even for other pseudo-first-order optimization algorithms including Adam (Kingma and Ba, 2014) and Adagrad (Duchi et al., 2011).

In our setup, we investigate the utility of averaging weights over every iteration (an iteration consists of one batch gradient descent).

Algorithm 1: Aggressive SWA algorithm

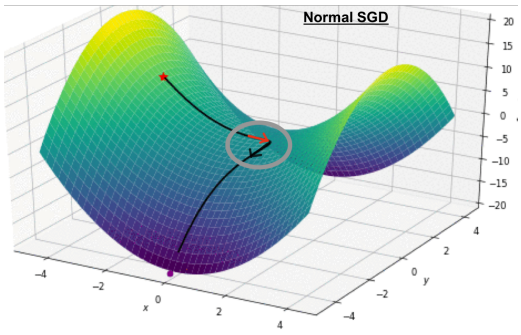
Require:

e = Epoch number
m = Total epochs
i = Iteration number
n = Total iterations
α = Learning rate
$\mathcal{O}$ = Stochastic Gradient optimizer function

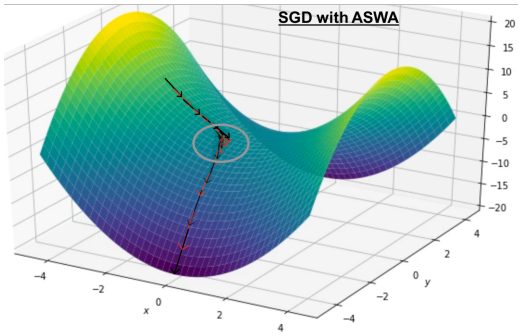
$e \leftarrow 0$;
while $e < m$ **do**
  $i \leftarrow 1$
  while $i \leq n$ **do**
    $W_{swa} \leftarrow W_{swa} + \frac{(W - W_{swa})}{(e*n+i+1)}$;
    $W \leftarrow W - \mathcal{O}(\alpha, W)$;
    $i \leftarrow i + 1$
  $W \leftarrow W_{swa}$;
  $e \leftarrow e + 1$

In Figure 2, we show an SGD optimizer (with momentum) and the same optimizer *with SWA* over a 3-dimensional loss surface with a saddle point. We observe that the original SGD reaches the desired minima, however, it almost reaches the saddle point and does a course correction and reaches minima. On the other hand, we observe that SGD with ASWA is very conservative, it repeatedly restarts and reaches the minima without reaching the saddle point. We empirically observe that this is a desired property for the stability of models over runs of the same model but with different random initialization. The grey circles in Figure 2 highlight this conservative behaviour

of SGD with ASWA optimizer, especially when compared to the standard SGD.



(a) Trajectory for Stochastic Gradient Descent



(b) Trajectory for Stochastic Gradient Descent with ASWA

Figure 2: Trajectory for gradient descent algorithms with red and black arrows on (b) indicating movements from consecutive epochs with restarts. Conservative behaviour of ASWA algorithm helps it avoid the saddle point.

We note, Polyak and Juditsky (1992) show that for convex losses, averaging SGD proposals achieves the highest possible rate of convergence for a variety of first-order SGD based algorithms.

Algorithm 1 shows the implementation pseudo-code for SWA. We note that, unlike Izmailov et al. (2018), we average our weights at each batch update and assign the ASWA parameters to the model at the end of each epoch. That is, we replace the model's weights for the next epoch with the averaged weights.

3.1 Norm-filtered Aggressive Stochastic Weight Averaging (NASWA)

We observe that the ASWA algorithm is especially beneficial when the norm difference of the parameters of the model, at two different iterations, are high. We hypothesise that in general, the norm difference indicates the divergence between optimizers' steps and we observe that the larger the norm difference, the greater the change in the trajectory. Therefore, we propose to maintain a list that stores the norm differences of the previous iterations. If the norm difference of the current iteration is greater than the average of the list, we update the ASWA weights and reinitialize the list with the current norm difference. When the norm difference, however, is less than the average of the list, we just append the current norm difference to the list. After the completion of the epoch, we assign the ASWA parameters to the model. This is shown in Algorithm 2. We call this approach *Norm-filtered Aggressive Stochastic Weight Averaging*.

Algorithm 2: Norm-filtered Aggressive SWA algorithm

Require:

e = Epoch number
m = Total epochs
i = Iteration number
n = Total iterations
α = Learning rate
$\mathcal{O}$ = Stochastic Gradient optimizer function
N_s = List of previous iterations' norm differences

$e \leftarrow 0$;
while $e < m$ **do**
  $i \leftarrow 1$
  while $i \leq n$ **do**
    $N_{cur} \leftarrow \|W - W_{swa}\|_1$;
    $N_{mean} \leftarrow \frac{\sum_{i=1}^{|N_s|} N_s[i]}{|N_s|}$;
    if $N_{cur} > N_{mean}$ **then**
      $W_{swa} \leftarrow W_{swa} + \frac{(W - W_{swa})}{(e*n+i+1)}$;
      $N_s \leftarrow [N_{cur}]$;
    else
      $N_s \leftarrow N_s + [N_{cur}]$;
    $W \leftarrow W - \mathcal{O}(\alpha, W)$;
    $i \leftarrow i + 1$
  $W \leftarrow W_{swa}$;
  $e \leftarrow e + 1$

4 Experiments

We base our investigation on similar sets of models as Jain and Wallace (2019). We also use the code provided by the authors for our empirical investigations for consistency and empirical validation. We describe our models and datasets used for the experiments below.

4.1 Models

We consider two sets of commonly used neural models for the tasks of binary classification and multi-class natural language inference. We use CNN and bi-directional LSTM based models with attention. We follow (Jain and Wallace, 2019) and use similar attention mechanisms using a) additive attention (Bahdanau et al., 2014); and b) scaled dot product based attention (Vaswani et al., 2017). We jointly optimize all the parameters for the model, unlike Jain and Wallace (2019) where the encoding layer, attention layer and the output prediction layer are all optimized separately. We experiment with several optimizers including Adam (Kingma and Ba, 2014), SGD and Adagrad (Duchi et al., 2011) but most results below are with Adam.

For our ASWA and NASWA based experiments, we use a constant learning rate for our optimizer. Other model-specific settings are kept the same as Jain and Wallace (2019) for consistency.

Dataset	Avg. Length	Train Size	Test size
IMDB	179	12500 / 12500	2184 / 2172
Diabetes(MIMIC)	1858	6381 / 1353	1295 / 319
SST	19	3034 / 3321	652/653
Anemia(MIMIC)	2188	1847 / 3251	460 / 802
AgNews	36	30000 / 30000	1900 / 1900
ADR Tweets	20	14446 / 1939	3636 / 487
SNLI	14	182764 / 183187 / 183416	3219 / 3237 / 3368

Table 1: Dataset characteristics. Train size and test size show the cardinality for each class. SNLI is a three-class dataset while the rest are binary classification

4.2 Datasets

The datasets used in our experiments are listed in Table 1 with summary statistics. We further pre-process and tokenize the datasets using the standard procedure and follow Jain and Wallace (2019). We note that IMDB (Maas et al., 2011), Diabetes(MIMIC) (Johnson et al., 2016), Anemia(MIMIC) (Johnson et al., 2016), AgNews (Zhang et al., 2015), ADR Tweets (Nikfarjam et al., 2015) and SST (Socher et al., 2013) are datasets for the binary classification setup. While SNLI (Bowman et al., 2015) is a dataset for the multiclass classification setup and CNN News Articles (Hermann et al., 2015) for cloze style question answering.

4.3 Settings and Hyperparameters

We use a 300-dimenstional embedding layer which is initialized with FastText (Joulin et al., 2016) based free-trained embeddings for both CNN and the bi-directional LSTM based models. We use a 128-dimensional hidden layer for the bi-directional LSTM and a 32-dimensional filter with kernels of size $\{1, 3, 5, 7\}$ for CNN. For others, we maintain the model settings to resemble the models in Jain and Wallace (2019). We train all of our models for 20 Epochs with a constant batch size of 32. We use early stopping based on the validation set using task-specific metrics (Binary Classification: using `roc-auc`, Multiclass and question answering based dataset: using `accuracy`).

Dataset	CNN(%)	CNN+ASWA(%)	CNN+NASWA(%)
IMDB	89.8 (±0.79)	90.2 (±0.25)	90.1 (±0.29)
Diabetes	87.4 (±2.26)	85.9 (±0.25)	85.9 (±0.38)
SST	82.0 (±1.01)	82.5 (±0.39)	82.5 (±0.39)
Anemia	90.6 (±0.98)	91.9 (±0.20)	91.9 (±0.19)
AgNews	95.5 (±0.23)	96.0 (±0.11)	96.0 (±0.07)
Tweet	84.6 (±2.65)	84.4 (±0.54)	84.4 (±0.54)

Table 2: Performance statistics obtained from 10 differently seeded CNN based models. Table compares accuracy and its **standard deviation** for the normally trained CNN model against the ASWA and NASWA trained models, whose deviation drops significantly, thus, indicating increased robustness.

5 Results

In this section, we summarize our findings for 10 runs of the model with 10 different random seeds but with identical model settings.

5.1 Model Performance and Stability

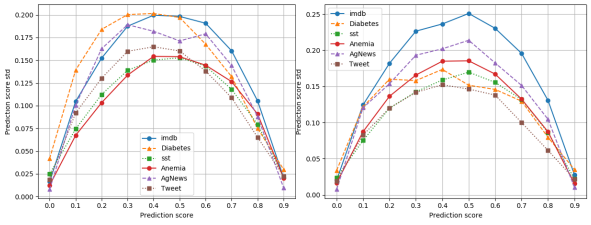
We first report model performance and prediction stability. The results are reported in Table 2.

Dataset	LSTM(%)	LSTM+ASWA(%)	LSTM+NASWA(%)
IMDB	89.1 (±1.34)	90.2 (±0.32)	90.3 (±0.17)
Diabetes	87.7 (±1.44)	87.7 (±0.60)	87.8 (±0.55)
SST	81.9 (±1.11)	82.0 (±0.60)	82.1 (±0.57)
Anemia	91.6 (±0.49)	91.8 (±0.34)	91.9 (±0.36)
AgNews	95.5 (±0.32)	96.1(±0.17)	96.1 (±0.10)
Tweet	84.7 (±1.79)	83.8 (±0.45)	83.9 (±0.45)

Table 3: Performance statistics obtained from 10 differently seeded LSTM based models.

We note that the original CNN based models, on an average, have a standard deviation of $\pm 1.5\%$. Which seems standard, however, we note that ADR Tweets dataset has a very high standard deviation of $\pm 2.65\%$. We observe that ASWA and NASWA almost always are able to get higher performance with very low standard deviation. This suggests that both ASWA and NASWA are extremely stable when compared to the standard model. They significantly improve the robustness, on an average, by 72% relative to the

original model and on Diabetes (MIMIC), a binary classification dataset, by 89% (relative improvement). We observe similar results for the LSTM based models in Table 3.

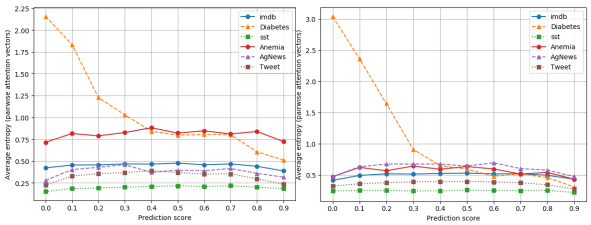


(a) CNN models (b) LSTM models

Figure 3: Prediction's standard deviation for CNN and LSTM based models for all binary classification datasets under consideration. Predictions are bucketed in intervals of size 0.1, starting from 0 (containing predictions from 0 to 0.1), until 0.9

We further analyze the prediction score stability by computing the mean standard deviation over the binned confidence intervals of the models in Figure 3a. We note that on an average, the standard deviations are on the lower side. However, we observe that the mean standard deviation of the bins close to 0.5 is on the higher side as is expected given the high uncertainty. On the other hand both, ASWA and NASWA based models are relatively more stable than the standard CNN based model. We observe similar behaviours for the LSTM based models in Figure 3b. This suggests that our proposals, both ASWA and NASWA, are able to obtain relatively better stability without any loss in performance. We also note that both ASWA and NASWA had relatively similar performance over more than 10 random seeds.

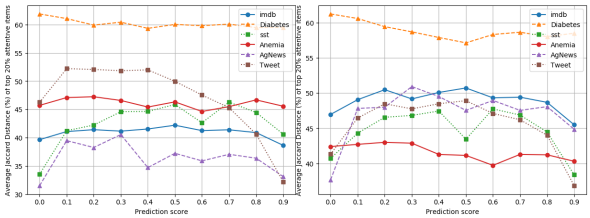
5.2 Attention Stability



(a) CNN models (b) LSTM models

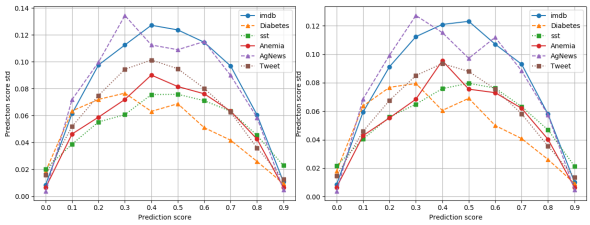
Figure 4: Average attention entropy against the bucketed predictions for CNN and LSTM based models. Figure highlights the high entropy between attention based distributions from differently seeded models (especially for the Diabetes-MIMIC datatset), indicating towards model instability.

We now consider the stability of attention distributions over as a function of random seeds. We first plot the results of the experiments for *standard* CNN based binary classification models over uniformly binned prediction scores for positive labels in Figure 4a. We observe that, depending on the datasets, the attention distributions can become extremely unstable (high entropy). We specifically highlight the Diabetes(MIMIC) dataset's entropy distribution. We observe similar, but relatively worse results for the LSTM based models in Figure 4b. In general, we would expect the entropy distribution to be close to zero however, this doesn't seem to be the case. This means that using attention distributions to interpret models may not be reliable and can lead to misinterpretations.

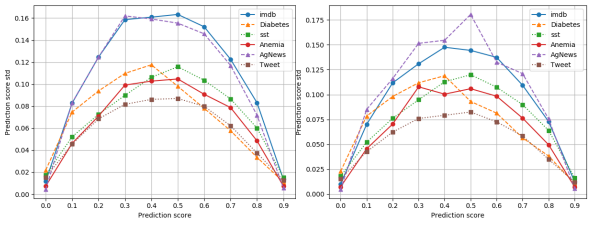


(a) CNN models (b) LSTM models

Figure 5: Jaccard distance highlighting instability in attention distributions of CNN and LSTM based models.



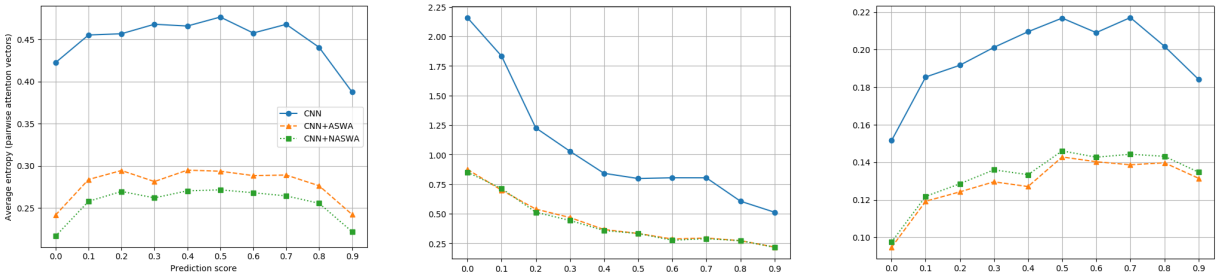
(a) CNN+ASWA (b) CNN+NASWA



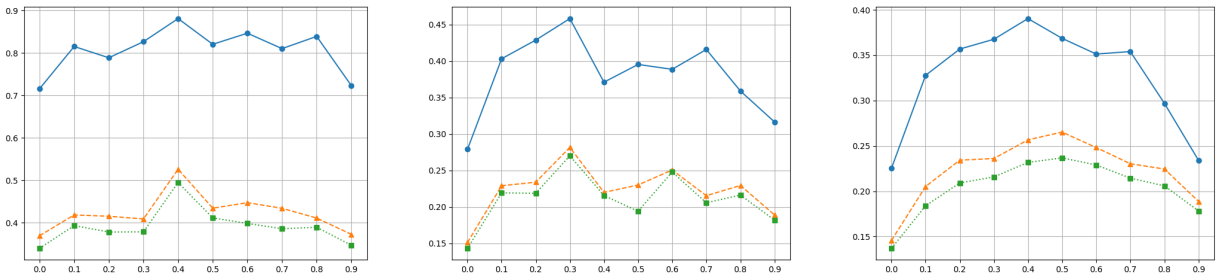
(c) LSTM+ASWA (d) LSTM+NASWA

Figure 6: Improved prediction stability from ASWA and NASWA for CNN and LSTM based models

We use the top 20% of the most important items (indices) in the attention distribution for each dataset over 10 runs and plot the Jaccard distances for CNN and LSTM based models in Figure 5a and Figure 5b. We again notice a similar

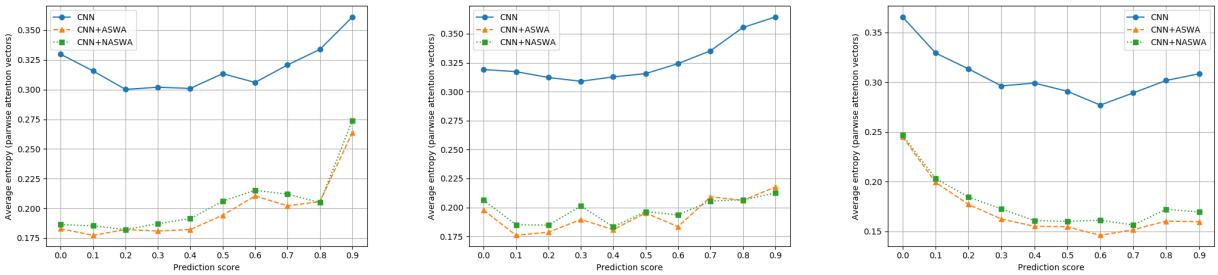


(a) IMDB (b) Diabetes (c) SST



(d) Anemia (e) AgNews (f) ADR Tweets

Figure 7: Attention stability improvement from ASWA and NASWA on CNN based models.



(a) Label 0 prediction vs entropy (b) Label 1 prediction vs entropy (c) Label 2 prediction vs entropy

Figure 8: Attention stability improvement from ASWA and NASWA on CNN based model for the SNLI dataset.

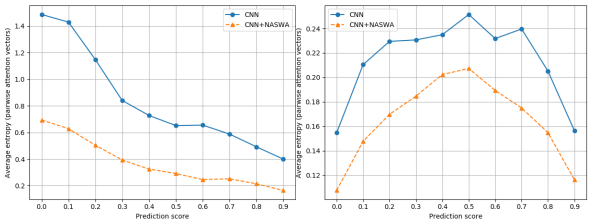
trend of unstable attention distributions over both CNN and LSTM based attention distribution.

In the following sections for space constraints, we focus on CNN based models with additive attention. Our results on LSTM based models are provided in the attached supplementary material. We note that the observations for LSTM models are, in most cases, similar to the behaviour of the CNN based models. Scaled dot-product based models are also provided in the supplementary material and we notice a similar trend as the additive attention.

We now focus on the effect of ASWA and NASWA on binary and multi-class CNN based neural models separately.

Binary Classification In Figure 7, we plot the results of the models with ASWA and NASWA. We observe that both these algorithms significantly improve the model stability and decrease the entropy between attention distributions. For example, in Figure 7b, both ASWA and NASWA decrease the average entropy by about 60%. We further notice that NASWA is slightly better performing in most of the runs. This empirically validates the hypothesis that averaging the weights from divergent weights (when the norm difference is higher than the average norm difference) helps in stabilizing the model's parameters, resulting in a more robust model.

Multi-class Classification In Figure 8, we plot the entropy between the attentions distributions of the models for the SNLI dataset (CNN based model), separately for *each label* (*neutral*, *contradiction*, and *entailment*). We notice, similar observations as the binary classification models, the ASWA and NASWA algorithms are able to significantly improve the entropy of the attention distributions and increases the robustness of the model



(a) Diabetes (b) SST

Figure 9: Gradient based explanation's stability improvement from NASWA on CNN based models.

with random seeds.

5.3 Gradient-based explanations

We now look at an alternative method of interpreting deep neural models and look into the consistency of the gradient-based explanations to further analyze the model's instability. For this setup, we focus on binary classifier and plot the results on the SST and the Diabetes dataset in particular since they cover the low and the high end of the entropy spectrum (respectively). We notice similar trends of instability in the gradient-based explanations from model inputs as we did for the attention distributions. Figure 9 shows that the entropy between the gradient-based explanations from differently seeded models closely follows the same trend as the attention distributions. This result further strengthens our claim on the importance of model stability and shows that over different runs of the same model with different seeds we may get different explanations using gradient-based feature importance. Moreover, Figure 9 shows the impact of ASWA and NASWA towards making the gradient-based explanations more consistent, thus, significantly increasing the stability.

6 Discussion

Recent advances in adversarial machine learning (Neelakantan et al., 2015; Zahavy et al., 2016) have investigated robustness to random initialization based perturbations, however, to our knowledge, no previous study investigates the effect of random-seeds and its connection on model interpretation. Our study analyzed the inherent lack of robustness in deep neural models for NLP. Recent studies cast doubt on the consistency and correlations of several types of interpretations (Doshi-Velez and Kim, 2017; Jain and Wallace, 2019; Feng et al., 2018). We hypothesise that some of these issues are due to the inherent instability of the deep neural models to random-seed base perturbation. Our analysis shows that, on multiple runs of the model with equivalent settings, models may use completely different routes to reach similar decisions with high confidence in Sections 4. This issue of variance in all black-box interpretation methods over different seeds will continue to persist until the models are fully robust to random-seed based perturbations. Our work however, doesn't provide insights into instabilities of different layers of the models. We hypothesise that it might further uncover the reasons for the relatively lower correlation between different black-box interpretation methods as these are effectively based off on different layers and granularity.

There has been some work on using noisy gradients (Neelakantan et al., 2015) and learning from adversarial and counter-factual examples (Feng et al., 2018) to increase the robustness of deep learning models. Feng et al. (2018) show that neural models may use redundant features for prediction and also show that most of the black-box interpretation methods may not be able to capture these second-order effects. Our proposals show that aggressively averaging weights leads to better optimization and the resultant models are more robust to random-seed based perturbation. However, our research is limited to increasing consistency in neural models. Our approach further uses first order based signals to boost stability. We posit that second-order based signals can further enhance consistency and increase the robustness.

7 Conclusions

In this paper, we study the inherent instability of deep neural models in NLP as a function of random seeds. We analyze model performance and robustness of the model in the form of attention entropy and gradient-based feature importance entropy across multiple runs of the models with different random seeds. We propose a novel solution that makes use of aggressive weighted averaging and further extend it with norm-filtering and show that our proposed methods largely stabilize the model to random-seed based perturbations and, on an average, significantly reduce the standard deviations of the model performance by 72%. We further show that our methods significantly reduce the entropy in the attention distribution and the gradient-based feature importance measures across runs.

References

David Alvarez-Melis and Tommi S Jaakkola. 2017. A causal framework for explaining the predictions of black-box sequence-to-sequence models. *arXiv preprint arXiv:1707.01943*.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. 2010. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Shi Feng, Eric Wallace, Alvin Grissom II, Pedro Rodriguez, Mohit Iyyer, and Jordan Boyd-Graber. 2018. Pathologies of neural models make interpretation difficult. In *Empirical Methods in Natural Language Processing*.

Reza Ghaeini, Xiaoli Z Fern, and Prasad Tadepalli. 2018. Interpreting recurrent and attention-based neural models: a case study on natural language inference. *arXiv preprint arXiv:1808.03894*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Sarthak Jain, Ramin Mohammadi, and Byron C Wallace. 2019. An analysis of attention over clinical notes for predictive tasks. *arXiv preprint arXiv:1904.03244*.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not explanation. *CoRR*, abs/1902.10186.

Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics.

Stephan Mandt, Matthew D Hoffman, and David M Blei. 2017. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. 2015. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.

Azadeh Nikfarjam, Abeed Sarker, Karen O'connor, Rachel Ginn, and Graciela Gonzalez. 2015. Pharmacovigilance from social media: mining adverse drug reaction mentions using sequence labeling with word embedding cluster features. *Journal of the American Medical Informatics Association*, 22(3):671–681.

Boris T Polyak and Anatoli B Juditsky. 1992. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*.

David Ruppert. 1988. Stochastic approximation. Technical report, Cornell University Operations Research and Industrial Engineering.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Tom Zahavy, Bingyi Kang, Alex Sivak, Jiashi Feng, Huan Xu, and Shie Mannor. 2016. Ensemble robustness and generalization of stochastic deep learning algorithms. *arXiv preprint arXiv:1602.02389*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.