NLP Assignment 2: Sentence Representation

Utkarsh Garg

1

1 Deep Averaging Network(DAN)

This is a Deep Neural Network architecture that performs a classification task, given a representation of the sentences. In other words, map an input sequence of tokens X(sentences) to one of k labels(classes). It simply averages word embeddings of the input text, then feeds them through several layers with non-linearities, followed by a softmax classification step for the output.

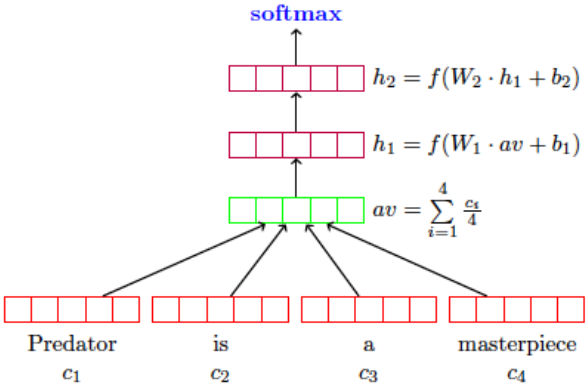

Fig:1.1 : A DAN Model with 2 hidden non-linear layers followed by Softmax.

1.1 Role of Sequence Mask

Sequence Mask is an important element in this classification task. It's a Boolean tensor of shape (batch_size, max_tokens_num), that shows the true representation of a word token in the sequence. Entries with 1 indicate that token is a real token, and 0 indicate that it's a padding token.

1.2 Dropout

It's a Regularisation technique used to improve the effectiveness of a neural network, by randomly dropping some of the neuron embeddings to 0 with some probability 'p'

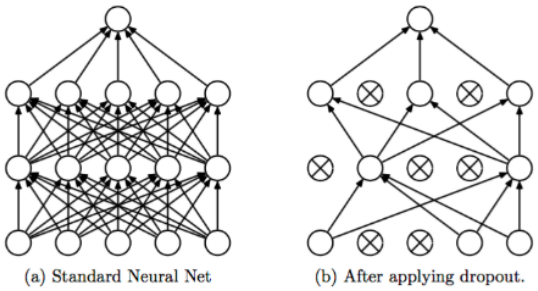

Fig:1.2 : A standard Neural Net with 2 hidden layers, Right: A dropped out version of the Neural Network. Crossed Units have been dropped.

In the case of DAN model, we select Bernoulli distribution to give the uniform random mask of 0s and 1s, each value indicating whether to drop the word embedding from the sequence or not. We use this Bernoulli mask over the input word sequences to get a regularised word vector, ready for training,post averaging, through Dense layers.

1.3 Implementation:

Here is a step by step implementation procedure for the DAN:

- We are given a vector_sequence and a sequence_mask.

- The input vector_sequence is a padded sentence embedding of size $(batch\_size, max\_num\_tokens, embedding\_dim)$.
We can relate batch_size here to the number of sentences, max_num_tokens as the max length of a sentence,i.e no. of words and embedding dims to be the number of features in each word.

- In order to obtain a original word embeddings without padding ,we create an unmasked_vector_seq by mutilplying the given vector_sequence with the given sequence_mask.

- We now apply **dropout** as a regularisation technique to remove some word tokens from this unmasked sequence, with an aim to prevent overfitting of the model.

- To achive this, we create a random uniform mask ,of the size of unmasked_vector_seq, with values normally distributed between 0s and 1s.

- From this mask, we specify a condition to track elements greater than the dropout-probability 'p' which we ultimately want to retain.

- We use this mask and apply on top of unmasked_vector_seq to obtain the final dropped out version of the word sequence, stored in our tensor drop_vector.

- The result of the drop_vector is then averaged across each sentence length, to obtain a avg_vector_seq. This is what becomes an input to our intended Deep Averaging Network.

$$z = g(w \epsilon X) = \frac{1}{|X|} \sum_{w \epsilon X} v_w$$

- We now pass this averaged vector to the Dense model, instantiated with num_layers and an activation function of 'tanh'.

- The output from each hidden dense layer is stored in the layer_representations list , while the output of the last layer from the entire model gets stored as a combined_vector.

- The combined_vector is further subjected to final non-linear softmax function(defined already in train.py) to obtain the final classification output. Whereas the layer_representations is further stacked into a tensor of shape (batch_size, num_layers, embedding_dim),for it to be

utilised for probing model(discussed later), to obtain classification score at distinct layers of the proposed model.

2 Gated Recurrent Unit(GRU)

This is an improvement over a Recurrent Neural Network model, that aims to solve the **vanishing gradient** problem associated with a long range dependency model. It does this with the aid of logic gates(**Update** and **Reset**), which control the amount of information to be passed on to the next layer.

$$\mathrm{h}_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j$$

$$\mathrm{z}_t^j = \sigma(W_z x_t + U_z h_{t-1}])^j$$

$$\tilde{h}_t^j = tanh(W x_t + U(r_t \odot h_{t-1}))^j$$

$$\mathrm{r}_t^j = \sigma(W_r x_t + U_r h_{t-1})^j (1)$$

2.1 Implementation

For this, we declare out model in _init_ step of the function, where we give it the last dimension of the input input vector sequence that this SentenceToVector encoder will encounter, along with an activation function(here, tanh). We also pass an additional parameter "return_sequences=True" to return the output at last time step from each layer, in the output sequence.

In the call() method, we initialise a loop for num_layers, to create GRU layers. For the first layer we pass sequence mask as an input along with the vector_sequence, to the created GRU layer. For each subsequent layers, we simply pass the last hidden_layer output as input to the next layer. Note that for each time step, the model remembers the previous state and the current input token. So we take the last time step output from each layer, and store it as a combined_vector for that layer. This is further appended to layer_representations list , and the last output from this list of tensor is outputted as the combined_vector.

3 Probing Model

We define a Probing model to test the classification accuracy at different layers of the Deep Neural Network(for both DAN and GRU).

This analysis is helpful in realising how many layers are sufficient to perform the classification task , without accounting for additional computations(for subsequent layers) and training overheads.

3.1 Implementation

The function call to DAN and GRU, gives us the layer_representations for each layer in the form of a list of tensors. We use the i^{th} index of the layer_representations as an input to the Probe Model, and perform a softmax operation on the output from this layer. While probing the i^{th} layer in the Deep Neural Network, we make sure the weights and biases till that layer are frozen, by making Training=False enabled during call to the underlying probed Model(DAN/GRU).

4 Analysis

4.1 Learning Curves:

The provided code has scripts to generate plots for the following. Run them and explain in brief, what changes as we:

Increasing the training data

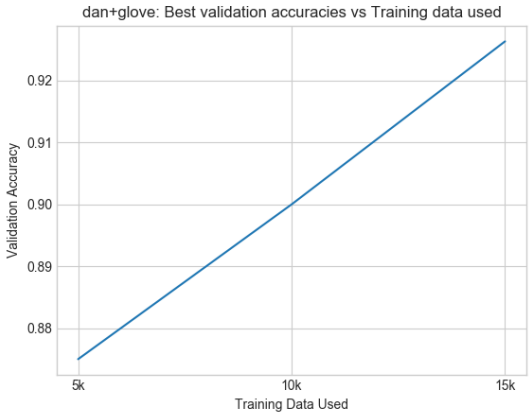

Fig:4.1.1 : DAN Performance on increasing training data

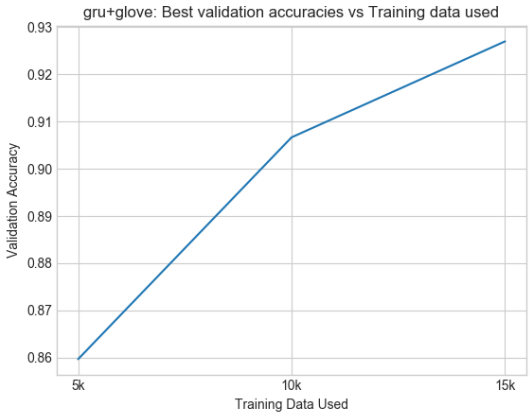

Fig:4.1.2 : GRU Performance on increasing training data

The Validation accuracy is directly proportional to the size of training data for both the models. The main difference is the slope of learning curve . In case of DAN, the model learns linearly with the vocab its trained on, whereas for GRU, we see a steep learning curve as the model is trained on higher dataset, and sets to linear with increased vocabulary.

Increase training time (number of epochs)

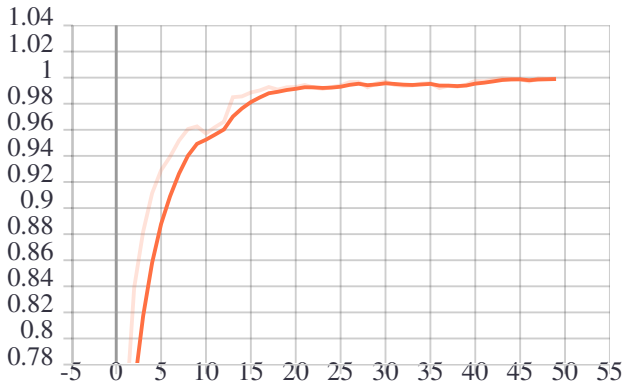

Fig: 4.1.3 : Accuracy during Training

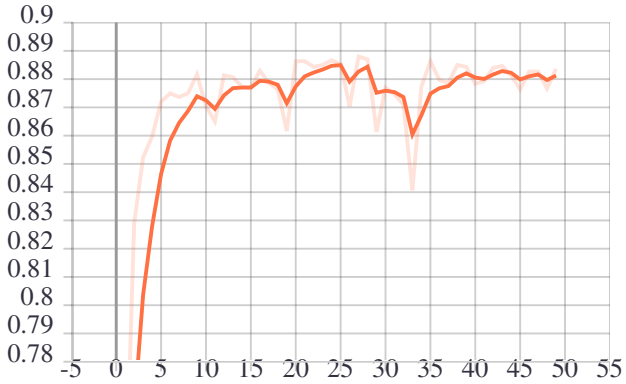

Fig:4.1.4 : Accuracy during Validation

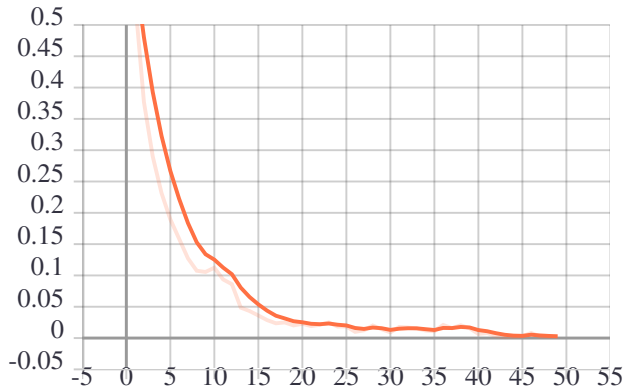

Fig:4.1.5 : Loss during Training

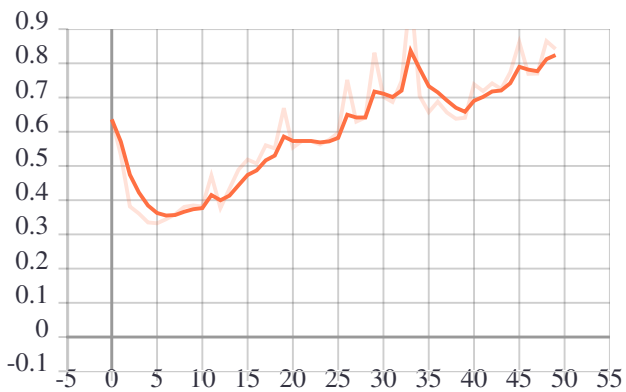

Fig:4.1.6 : Loss during Validation

Increasing the training time/epochs doesn't effect DAN much during training in terms of accuracy. We can see that training the Model for many epochs beyond say 10), hinders the performance on the validation set(as evident from the dip in the 2nd plot). Consequently, the loss has an opposite effect as we see it getting increased along higher epochs.

4.2 Error Analysis:

Explain one advantage of DAN over GRU and one advantage of GRU over DAN.

Advantages of DAN over GRU: DAN is a simplistic Neural Network that works best with 2-3 hidden layers, and eliminates the need to perform complex computations to remember syntactic word representations; they prefer averaging the word representations. DANs perform comparably well on "single negations" or "contrastive conjunctions" when it comes to accuracy v/s computation trade-off . On the other hand, GRU is a complicated network with deeper layers and additional gates logic to remember the word sequences and their relative dependencies in order.

Advantages of GRU over DAN: GRU is better in handling **double negations**, while DAN fails at capturing such contrasting words in the same sentence.(eg: movie was not bad). DAN model will ideally treat it as 2 negatives and give an overall negative response to this sentence.

5 Probing Tasks

5.1 Probing Sentence Representation for Sentiment Task

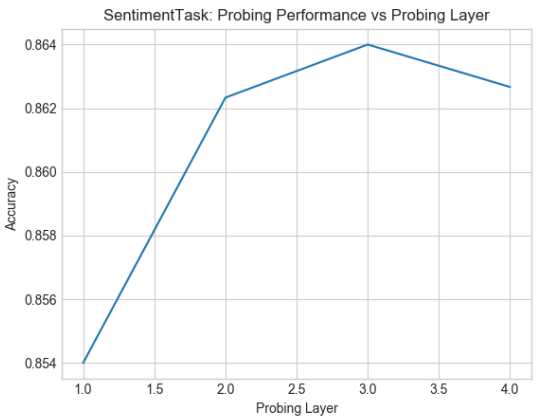

Fig:5.1.1 : Probing Performance on Sentiment Analysis Task with DAN

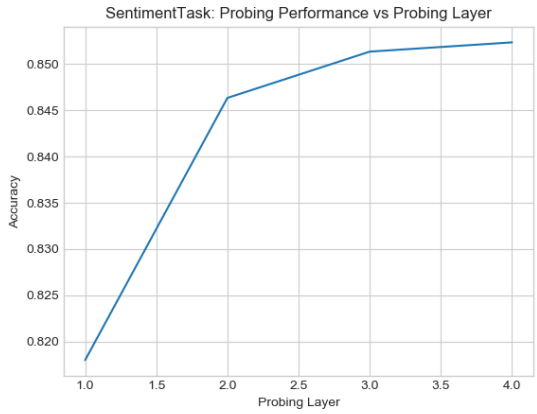

Fig:5.1.2 : Probing Performance on Sentiment Analysis Task with GRU.

In Sentiment Analysis Task, GRU performs a bit better than DAN. This is inherently due to DAN's averaging ability over the words,thereby ignoring the word-order sequence in the sentence. We can see that for DAN, the classification performance after layer 3 has declined .

For the GRU, the performance on Sentiment Task increases proportionally as we increase the layers , at the cost of high computations.

5.2 Probing Sentence Representations for Bigram Order

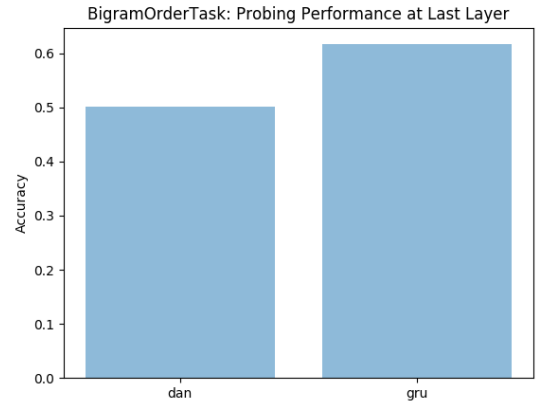

Fig:5.2 : Probing Performance on Bigram Task.

This explains the word-order learning capacity of the GRU model, which is a dominating factor over the DAN(which ignores the time sequence of word representations in a sentence). So if the example sentence has word order reversed (New York becomes York New), GRU model will account for this change in word order , while DAN remains unchanged.

Also,from the plot ,we can infer that the GRU performs significantly better on bigram dataset with an accuracy of 0.62 as compared to 0.50 (as good as a random guess).

5.3 Analysing Perturbation Response of Representations

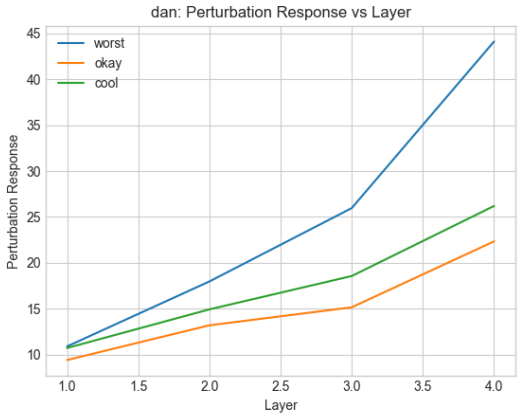

Fig:5.3.1 : Perturbation Response with DAN.

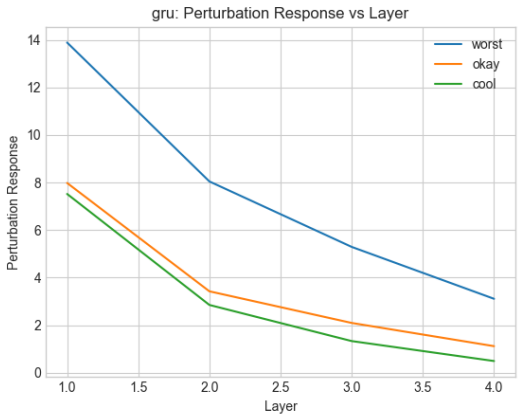

Fig:5.3.2 : Perturbation Response with GRU.

For a deep feed forwarding DAN network, the negative words add drastic effect to the overall classification , as the network is trained through more n more layers, and thus change the meaning of the sentence completely(as evident by the steep perturbation response curve in the plot).

In the case of GRU, the network learns the complete representation of a sentence until that particular time-step or token. So changing 1 word in the sentence doesn't impact the classification severely, as depicted by the gradual slope decrease in the plot.