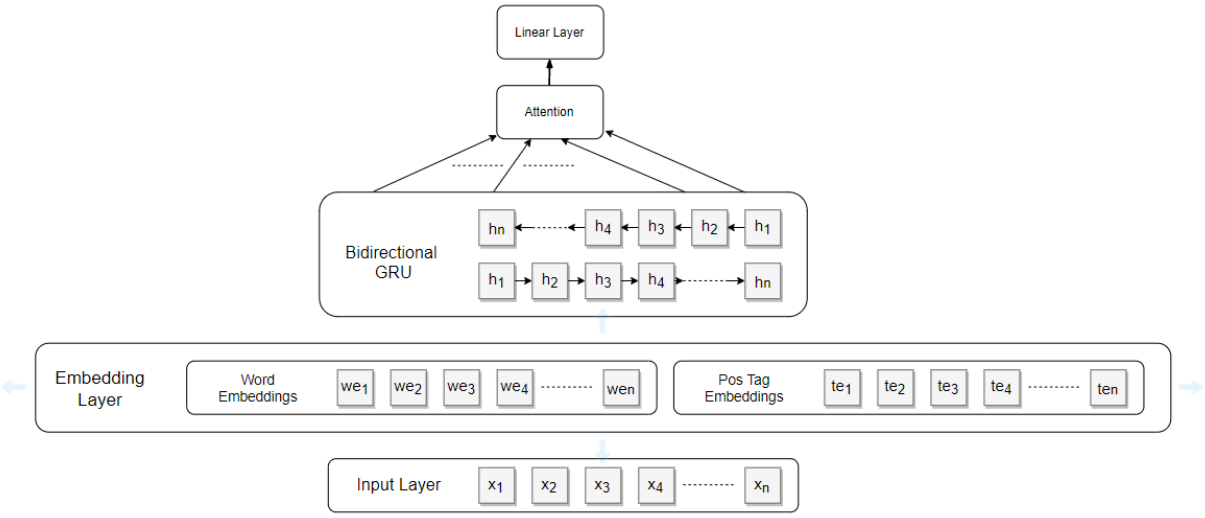
RELATION EXTRACTION

112685069, Mitkar Abdullah

Explanation of GRU + attention implementation

Architecture



Need for Bidirectional GRU

Properties of GRU

- Fast because they have computations only for 2 gates as compared to LSTM having 3 gates.
- Almost similar results as compared to LSTM.

GRUs are sophisticated neural networks that models the long-term dependencies in a sentence. However, they are feed forward networks and are only able to model the memory that it has seen [Previous words]. However, in the case of relation extraction, the relation for the words that are ahead is also needed to be modelled. To model the relation of both directions, we need Bidirectional GRU.

Need for Attention

Attention helps in choosing which part of the input to the attention layer to focus on. It helps in concentrating on the relation part of the features extracted by the bidirectional GRUs. Focusing only on the features extracted helps the model to learn in a better manner.

Steps

Model (Inputs, Pos Inputs)

1. Word embedding = EmbeddingLookup (Inputs)
2. Pos Embedding = EmbeddingLookup (Pos Inputs)
3. Embedding = Concat(Word Embedding, Pos Embedding)
4. RNN Output = Bidirectional(Embedding)
5. AttentionOp= Atten(RNN Output)

6. Logits = Decoder(AttentionOp)

Attention(RNNOutputs)

1. M = tanh(RnnOutputs)
2. Alpha = Softmax(MatMul(M, Omegas))
3. R = Mul(RnnOutputs,Alpha)
4. H* = tanh(R)
5. Output = ReduceSum(H*, axis =1)

F1 Score: 0.5659; LR: 0.5

Observation of GRU experiment 1 [Only Word Embeddings]

Using only the word embedding gives us an F1 score of 0.449. This is so because using word embeddings only model can perform fairly as it can model the relationships based only word embedding. The absence of the dependency structure increases the training time considerably. It performs better in the absence pos tag embeddings as it gets to focus and learn weights only from the word embeddings.

Epoch0
Train loss for epoch: 4.774015426635742
Val loss for epoch: 3.1827
Val F1 score: 0.2788

Epoch1
Train loss for epoch: 2.7988812923431396
Val loss for epoch: 3.8482
Val F1 score: 0.2908

Epoch2
Train loss for epoch: 1.9270520210266113
Val loss for epoch: 2.8423
Val F1 score: 0.3657

Epoch3
Train loss for epoch: 1.3411195278167725
Val loss for epoch: 3.1858
Val F1 score: 0.3936

Epoch4
Train loss for epoch: 1.0254889726638794
Val loss for epoch: 3.4206
Val F1 score: 0.449

The losses show that model learns faster in the absence of dependency structure.

Observation of GRU experiment 2 [Word + Pos Tag Embeddings]

Using only the word embedding gives us an F1 score of 0.40. Adding Pos Tag Embedding to the restricts the model to learn from the important features. Using the entire pos tag and word embedding digresses the model from understanding the relationship completely and restricts it from learning the weights. The absence of the dependency structure increases the training time considerably.

Epoch0
Train loss for epoch: 5.7586350440979
Val loss for epoch: 4.0917
Val F1 score: 0.2185

Epoch1
Train loss for epoch: 3.2509500980377197
Val loss for epoch: 2.9232
Val F1 score: 0.2999

Epoch2
Train loss for epoch: 2.4286561012268066
Val loss for epoch: 3.8447
Val F1 score: 0.3632

Epoch3
Train loss for epoch: 1.8484472036361694
Val loss for epoch: 4.2135
Val F1 score: 0.3138

Epoch4
Train loss for epoch: 1.3462451696395874
Val loss for epoch: 3.6901
Val F1 score: 0.4044

The losses show that model learns slowly in the absence of dependency structure.

Observation of GRU experiment 3 [Word + Dep Structure]

Using only the word embedding gives us an F1 score of 0.5858. Using Only the word embeddings and the dependency structure the model gets the only the important features it need to focus only on the important part and learns about the relationship in a better manner. It outperforms the usage of Word + Pos Tag + Dep Structure because it does not have the pos tag embedding to distract it from learning the weights in a better manner.

Motivation of advanced model choice

Paper: The advanced model that was chosen is loosely based on the paper "*Relation Classification via Convolutional Deep Neural Network*, *D Zeng et al*". The neural network used for relation extraction is a convolutional neural network.

Why CNN?

Convolutional neural network uses the convolution which captures the details of the input based on the kernel size (one kernel size at a time). Using CNN, on the word + pos tag embedding we can capture the relations between the words of the sentence and hence the features of the sentence. CNNs are heavily used in Computer Vision for image extraction because over the layers they capture features nicely. This feature of CNN can be used for capturing features in text too. For the instead of using a 2D kernel, we can use a 1D for convolving over the sentences and extracting features.

Based on the kernel size, different kind of features and dependency are extracted from the input text. Hence, we use different types of filter on the text and then concatenate all the features.

Attention

Attention proves to be vital in capturing what part of the text the model should focus on as seen above. We leverage the same in this model too where we apply attention on the embedding to capture and focus on the vital part of the text.

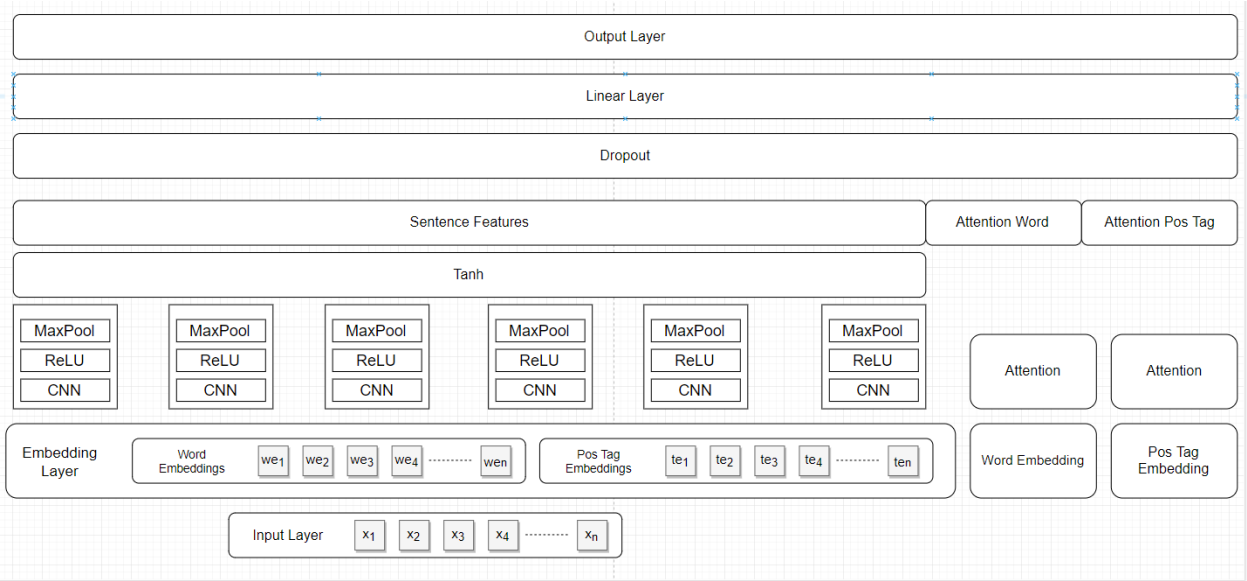
Concatenation of features extraction

The features from the sentence and the attention are concatenated to form a final layer of features based on which the linear layers will learn to classify.

Classification

Paper mentioned suggests using 1 hidden layer followed by tanh layer. It can be seen

Architecture



Accuracy: 0.6261

Training setting: 10 epochs, LR: 0.5