Statistical Natural Language Processing
Assignment 2

November 7, 2024

Victor Pekkari, epekkari@ucsd.edu

Introduction

This assignment consists of two main parts. Part 1 involves programming a neural network with the transformer architecture to classify text into three different categories, depending on whether it was spoken by Barack Obama, George W. Bush, or George H.W. Bush. The second part of the assignment involves creating a decoder that outputs the cross-entropy between the estimated probability distributions for the next word (as predicted by the decoder) and the actual next word. Additionally, this part includes calculating the decoder's perplexity score.

Run Code

The classifier can be run be typing: "python3 main.py ENCODER"
The classifier can be run be typing: "python3 main.py DECODER"

1 Text-Classifier

This text-classifier is used to determine if a text was said by barrack obama, h bush, or w bush. it does this by taking in a string and outputing probabilities for each of the three options.

The classifier consisists of an encoder and neural network with one hidden layer of 100 perceptrons. the input gets passed into the encoder, the output (vector with length of 64) from the encoder is then passed as input into the neural network.

1.1 Encoder

an embeddings layer that embeds the words into vectors, An encoder that takes in the vector embeddings, and lastly a linear layer that takes in an a vector with the size of embeddings dimension.

1.1.1 Number of parameters in classifier

feedforward: $100 \cdot 64 + 100 \cdot 3 = 6700$
embeddings: $2 \cdot 5755 \cdot 64 = 736640$
Block
  feedforward: $2(64 \cdot 64 \cdot 4) = 32768$
  multihead: $32 \cdot 2 \cdot 64 + 2(3 \cdot 64 \cdot 32) = 16384$
total: 792 492

1.1.2 Embedding Layer

The embedding layer is responsible for converting the tokens as numeric vectors containing information about their position in the token-sequence as well as the their meaning. these vectors get sent to the multiple heads that perform attention operations. the embeddings are learned during training.

1.1.3 Heads

The encoder contains two attention-heads. these two heads operate on vectors of size 32 (original embedding_dim=64 / 2 = 32). they generate attention matrices seperatly by taking the query matrix (matrix with all query vectos for each token) matrix-mulitplied with the key matrix (matrix contaning all key vectors for each token). Every row_r in the attention matrix contains how relevant all the other tokens are for $token_r$. All rows have to add up to 1. The attention matrix is finally multuplied with the value vector. the final ouput from the heads is a vector with lengh 32. the two vectors from head1 and head2 are then concatenated to form a vector with lenght 64 (same dimension as was fed into the Multiple heads). all vectors in the heads are learned during training.

the value-, query-, and key-vectors are all learnable parameters that are learned during training. The values in the vectorshave a dropout percentage of 20% during training. The benefit of having several heads is that the heads often pick up on different relations between the tokens during training.

An important difference from the heads in the decoder is that the heads in the encoder allow tokens to interact with future tokens in the sequence, $token_i$ can interact with $token_{i+1}$ that is. This is an important difference from the attention in the decoder where current tokens can only interact with previous tokens.

1.2 Feed forward - Neural network

the neural network takes 64 values provided from the encoder. The network has a hidden layer of 100 neurons, it applies a ReLu function in the hidden layer, as well as a dropout percentage of 10% for each node in the hidden layer. the output layer consists of three neurons, one for each class. it then applies a softmax function to the three output values to make them sum up to 1 (like probabilities have to do).

2 Decoder

The decoder is used to compute cross entropy between the predicted probability distribtion (predicted in the decoder but not outputed) for the next token and the target token.

the decoder consist of token-embedding vectors that are learned during training exaclty like the encoder. A feedforward neural network. and two attention heads.

The data is passed very similary between the compoments like it is in the encoder. There are some big differences however.

2.0.1 Number of parameters

feedforward: $100 \cdot 64 + 100 \cdot 5755 = 581900$
embeddings: $2 \cdot 5755 \cdot 64 = 736640$
Block
  feedforward: $2(64 \cdot 64 \cdot 4) = 32768$
  multihead: $32 \cdot 2 \cdot 64 + 2(3 \cdot 64 \cdot 32) = 16384$
total: 1 367 692

2.1 Masking

The decoder has to calculte the probability distributions for the next token to calculate the crossentropy. we therefore have to mask future tokens during training. This means

that the attention matrix:s upper right triangle will be filled with minues infinity values.

2.2 feedforward - neural network

Almost the same as the one in the encoder, but uses a dropout percentage of 35% for the hidden layer.

3 Results

3.1 Attention-matrices

These are my attention matrices for the encoder and the decoder after training 500 epochs for the string:
"Because I know our work has not only helped so many Americans, it has inspired so many Americans, especially so many young people out there, to believe that you can make a difference, to hitch your wagon to something bigger than yourselves."

3.1.1 Decoder

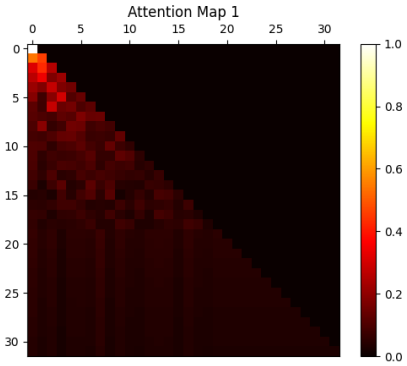

Figure 1: Head 1

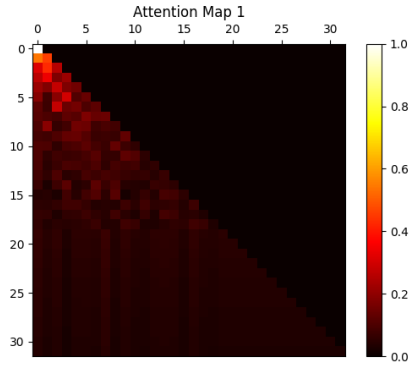

Figure 2: Head 2

the things we want to see in the images are that the rows add up to one. this means that the very top row should have one white pixel, and that the colors should get darker as the row number increases.

we also want to see that the current token on row i is only looking at tokens with column nunber greater or equal to i. this is true since the upper corner is black.

3.1.2 Encoder

Here we can clearly see how the token at row i is looking interacting with tokens on columns with columns number both greater and less than i.

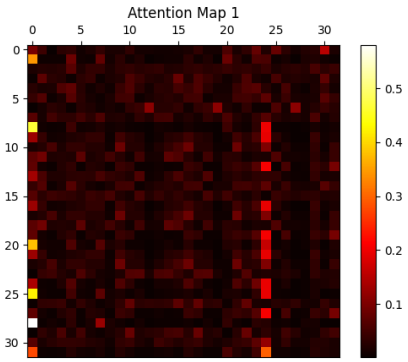

Figure 3: Head 1

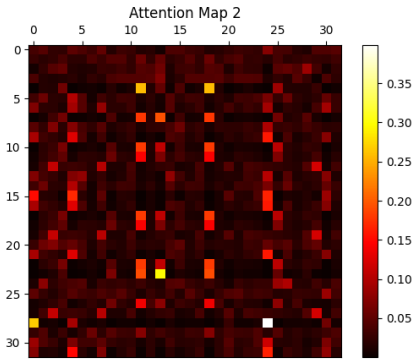

Figure 4: Head 2

3.2 Accuracy

3.2.1 Decoder

The perplexity-score can be seen as a measure on how suprised the model is, it may vary because some of the texts may have a richer vocabularies, the sentence structure can vary. Basically the more alike the testing text file is to the txt file that the model is trained on, the lower the perplexity score will be. From this we know that the actual training text is most like it self, and wbush deviates the most from the training text.

perplexities after 0 iterations
training data test: 6120.3916015625
obama data test: 6106.3876953125
hbush data test: 6183.30126953125
wbush data test: 6183.0361328125

---

perplexities after 100 iterations
training data test: 475.818603515625
obama data test: 608.6070556640625
hbush data test: 623.0281982421875
wbush data test: 698.2257690429688

---

perplexities after 200 iterations

training data test: 333.212890625
obama data test: 480.7460632324219
hbush data test: 508.9027404785156
wbush data test: 578.4375610351562
————————-
perplexities after 300 iterations
training data test: 256.042236328125
obama data test: 430.0272216796875
hbush data test: 459.07904052734375
wbush data test: 522.388916015625
————————-
perplexities after 400 iterations
training data test: 202.99819946289062
obama data test: 406.1135559082031
hbush data test: 436.88787841796875
wbush data test: 504.4370422363281
————————-
perplexities after 500 iterations
training data test: 167.9344482421875
obama data test: 386.6236877441406
hbush data test: 415.71649169921875
wbush data test: 493.95318603515625
————————-

3.2.2 Encoder

The following table summarizes the training and testing accuracy across the epochs:

Epoch	**Loss**	**Train Accuracy (%)**	**Test Accuracy (%)**
1	1.082	44.646	33.333
2	1.041	59.273	52.800
3	1.071	66.157	57.600
4	0.827	73.901	65.467
5	0.993	75.574	66.933
6	0.861	77.629	71.333
7	0.893	84.082	73.200
8	0.722	85.182	73.867
9	0.721	86.998	74.933
10	0.824	85.325	73.867
11	0.616	87.524	75.067
12	0.639	88.432	78.667
13	0.720	89.293	77.867
14	0.673	91.109	80.267
15	0.749	91.013	80.667
Final Test Accuracy			**80.67**

Table 1: Training and Test Accuracy for the Encoder across Epochs