DS-GA 1011: Natural Language Processing, Fall 2020
Homework 2

Due: Thursday, November 12, 2020 at 17:00EST

Instructions

1. Complete the provided Jupyter notebook (`.ipynb` file) for each part, three in total.
2. Each part accounts for 30 points, with 10 points allocated for the discussion session.
3. You can use material from the labs in the homework.
4. Cite any other open-source code which helps you to complete this assignment.
5. Submission should be done through NYU Classes by one person per team with the 3 Jupyter notebooks and any utility code not already provided.
6. Suffix your team number to the name of each submitted file.
7. Grading discussion sessions would be scheduled on 19^{th} (Thu) & 20^{th} (Fri) November.

DS-GA 1011: Natural Language Processing, Fall 2020 Homework 2

Due: Thursday, November 12, 2020 at 17:00EST

Part 1: BoW based Natural Language Inference (30 pts)

For question 1, you will train a Bag-of-Words encoder to tackle the Stanford Natural Language Inference (SNLI) task. Because the SNLI data set is relatively large, we recommend starting the assignment early (training may take hours). Likewise, because you are training on only a subset of the full data and using a relatively simple BoW encoder, you should not expect to get accuracies comparable to the published leaderboard.

1 Dataset (6 pts)

The SNLI task poses the following problem: Given two pieces of text, a premise and a hypothesis, you (or a model) have to choose one of the following:

1. The premise **entails** the hypothesis.
2. The premise **contradicts** the hypothesis.
3. The premise neither entails nor contradicts the hypothesis, and is thus **neutral** to it.

For example:

1. *Premise*: A man inspects the uniform of a figure in some East Asian country.
   Hypothesis: The man is sleeping.
   This is a **contradiction**, since the man cannot both be sleeping and inspecting the uniform.
2. *Premise*: A soccer game with multiple males playing.
   Hypothesis: Some men are playing a sport.
   This is an **entailment**, because if the former is true, the latter must be true.
3. *Premise*: An older and younger man smiling.
   Hypothesis: Two men are smiling and laughing at the cats playing on the floor.
   This is **neutral**, because the premise does not contain any information about cats that the hypothesis posits.

Being based on natural text, there will be some ambiguity regarding the answers. Nevertheless, the NLI-style tasks have shown to be effective for training models to capture aspects of semantic information from text. Refer to `https://nlp.stanford.edu/projects/snli/`, or the original paper (`https://nlp.stanford.edu/pubs/snli_paper.pdf`) for more details. You will be provided with tokenized training and validation pickle files. Pre-tokenized .tsv files are also provided, you can tokenize the data by yourself or use some pre-trained embeddings if you want. The data set has already been reduced to include 100,000 examples in the training set and 1,000 examples in the validation set.

DS-GA 1011: Natural Language Processing, Fall 2020
Homework 2

Due: Thursday, November 12, 2020 at 17:00EST

2 Model (8 pts)

At its core, the SNLI is a 3-class classification problem, where the input is two separate strings of text. Choosing what approach to use to integrate information from both strings of text is where the problem becomes interesting, and various approaches have been proposed. In our case, we will take the following simple approach:

1. We will use a BoW encoder (refer to code provided in Lab 8 - `nnet_models_new.py`) to map each string of text (hypothesis and premise) to a fixed-dimension vector representation. At this point, we have one vector representation corresponding to hypothesis and one for premise.

2. We will interact the two representations and perform classification on this. For combining the two vector representations, you should try two methods (sum, element-wise product, concatenate) to combine the representations and make a vector.

3. Once we've the combined representation, now we will to do a 3-class classification problem on this input vector. For this we will use Logistic Regression Model as shown in Lab 6 - `06b-neural_networks.ipynb`.

3 Training and Validation (16 pts)

Your task is to implement, perform hyperparameter tuning, and analyze the results of the model. Perform tuning over at least two of the following hyperparameters:

1. size of vocabulary

2. embedding dimension

3. optimizer itself (SGD vs Adam)

4. learning rate

For each mode of hyperparameter tuning, report the training and validation losses and accuracies (in plotted curves), as well as the number of trained parameters in each model. Discuss the implications of each hyperparameter tuned.

Finally, take your best model based on validation performance and highlight 3 correct and 3 incorrect predictions in the validation set. Describe why the model might have gotten the 3 incorrect predictions wrong.

What you present: Training & validation loss curves for each hyperparameter. Selected 3 correct and 3 incorrect predictions from your best model. Be ready to explain your results and answers for the written questions.

DS-GA 1011: Natural Language Processing, Fall 2020
Homework 2

Due: Thursday, November 12, 2020 at 17:00EST

Part 2: Neural Language Modeling (30 pts)

1 LSTM and Hyper-parameters (10 pts)

In this part you will train a neural recurrent language model on **Wikitext-2**. This task is similar to what was covered in Lab 7 - RNN Language Model. However, instead of a vanilla RNN, you will use an LSTM (see `nn.LSTM`). There are several hyper-parameters that can affect performance, such as the input size, hidden size, number of layers, etc. You need to:

1. In the training code, implement validation-based early stopping, such that if the validation performance does not improve in consecutively m epochs, terminate the training process early. m is called "max patience". Make sure that after early stopping, **you have saved the model with the best validation performance, not the model at the last epoch.** (5 pts)

2. Find and choose two hyper-parameters, and for each one produce 2 plots showing respectively the training and validation losses over the course of training, for varying hyper-parameter settings. Thus, each plot should have at least 4 curves. *Training may be time-consuming; we suggest you to train one model, use that model to finish the rest of Q2, and then come back to hyper-parameter tuning.* (5 pts)

What you present: Training & validation loss curves for each hyper-parameter. Be ready to explain which hyper-parameters you chose and why.

2 Learned Embeddings (6 pts)

In this part you will analyze the word embeddings that your best model (best validation perplexity) from the previous part learns.

1. Choose a set of 5 words, e.g. {**the, run, dog, where, quick**}, and for each word in the set, find the 10 'closest' words and 10 'furthest' words according to cosine similarity of vectors from the trained LSTM model's projection layer (the `nn.Linear` weight matrix of size $(|V|, d)$). You will need to implement your own cosine similarity function. (3 pts)

2. Use **UMAP** to visualize the model's `nn.Linear` layer. In particular, run **UMAP** on the entire `nn.Linear` weight matrix, and plot the points corresponding to the 100 words $(5 \times (10 + 10))$ selected from the previous part. (3 pts)

What you present: Selected 5 words with 10 closest and 10 furthest words, and visualization. Be ready to explain how to interpret the results. Are the 'closest' and 'furthest' words interpretable? Where are these words in the plots you produced? Are there interpretable patterns in the plots? Negative results are also good to discuss.

DS-GA 1011: Natural Language Processing, Fall 2020
Homework 2

Due: Thursday, November 12, 2020 at 17:00EST

3 Sampling (14 pts)

In Lab 3, we saw how to sample sequences from n-gram language models. In this section, you will implement sampling for your recurrent language model. Here is pseudo-code of this process:

```
$h_0 \leftarrow \vec{0}$;
$x_0 \leftarrow$ <bos>;
while $x_t \neq$ <eos> do
    $h_t = \textbf{LSTM}(h_{t-1}, x_t)$;
    $p = \textbf{softmax}(\textbf{projection}(h_t))$;
    $x_{t+1} \sim \textbf{multinomial}(p)$;
end
Result: sampled sequence (<bos>, $x_1, x_2, x_3, ..., x_T$, <eos>)
```

p is a $|V|-$dimensional vector giving conditional probabilities of each possible next-token. The typical way of obtaining these probabilities is by projecting the **LSTM** output and then taking the **softmax**. For `multinomial(p)` see `torch.multinomial`. Your task is to:

1. Implement sampling. (4 pts)
2. Sample $1,000$ sequences from the best model you trained above and calculate their average log probability. Recall that

$$\log p(x_1, ..., x_T) = \log \prod_{t=1}^{T} p(x_t | x_{<t}) = \sum_{t=1}^{T} \log p(x_t | x_{<t}).$$

   (4 pts)
3. Compare the sampled sequences against $1,000$ sequences from the validation set in terms of number of unique tokens and sequence length. (3 pts)
4. Choose 3 sampled sequences and discuss their properties, for instance: Can you tell that these 3 sequences are machine-generated rather than human-generated? Do these samples stay on topic? Are they grammatically correct? (3 pts)

What you present: Generated samples and comparison with the sequences in the validation set, Selected 3 sequences, Be ready to explain how to interpret the model's performance with the generated samples.

DS-GA 1011: Natural Language Processing, Fall 2020
Homework 2

Due: Thursday, November 12, 2020 at 17:00EST

Part 3: Neural Machine Translation (30 pts)

1 Transformer Encoder (18 pts)

The goal of this part is to plug-in a transformer encoder instead of RNN and make it work. You are not required to outperform the RNN-based baseline model demonstrated in Lab 8. The idea is to understand the differences in those encoders.

1. Build the encoder per below (refer to Lab 9 - Masked Language Modeling). (9 pts)

    - Positional embedding: Please use sinusoidal encoding as mentioned in the Transformer paper.
    - Transformer encoder (stack of layers): To simplify your work, please use PyTorch official transformer modules.
    - Hyper-parameters: Use same configuration as the NMT lab, i.e. with 512 embedding and hidden size, 1 layer in encoder and others. 2 heads for the transformer. Can explore the same if you have time.

    In the end the output from Transformer encoder should be similar to RNN encoder.

2. Pair encoder with following from the lab and train for ∼20 epochs. (9 pts)

    - Basic RNN decoder (no attention)
    - RNN Decoder with Encoder attention
    - RNN Decoder with Encoder & Self attention

    Each architecture should take approximately 2 hours to train on the Google Colab with single GPU.

What you present: Training & validation curves for both loss and BLEU. Be ready to explain how positional embedding works and its other types.

2 Visualizing the attention (12 pts)

In this part you will analyze the attention mechanism in your models. Your task is to:

- Check the model code and find how attention weights are logged. Make sure to understand how each type of attention works.
- Implement a function which takes in the attention weights and plot the corresponding heat map.

- Make sure that on the axes of the attention map you show the actual tokens from sequences.

What you present: Select 3 examples (from validation set) and print: input, target, attention map for each encoder-decoder combination (all heads, source & self). Be ready to explain how attention is computed on every time step.