DS-GA 1011 Fall 2018
RNN/CNN-based Natural Language Inference

Anant Gupta and Jason Phang

October 2018

In this assignment, you will train RNN- and CNN-based models to tackle the Stanford Natural Language Inference (SNLI) task.

1 Data set

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:

- 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.
- 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.
- 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 (`http://nlp.stanford.edu/pubs/snli_paper.pdf`) for more details.

You will be provided with pre-tokenized training and validation sets of the SNLI in a `.tsv` format. The data set has already been reduced to include 100,000 examples in the training set and 1,000 examples in the validation set.

2 Model

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 an encoder (either a CNN or an RNN) to map each string of text (hypothesis and premise) to a fixed-dimension vector representation.
2. We will interact the two hidden representations and output a 3-class softmax. (To keep things simple, we will simply concatenate the two representations, and feed them through a network of 2 fully-connected layers.)

Of course, we will also use frozen, pretrained word embeddings (e.g. fastText) to embed our word tokens. We recommend using one of the fastText word vector sets.[1]

For the encoder, we want the following:

- For the CNN, a 2-layer 1-D convolutional network with ReLU activations will suffice. We can perform a max-pool at the end to compress the hidden representation into a single vector.
- For the RNN, a single-layer, bi-directional GRU will suffice. We can take the last hidden state as the encoder output. (In the case of bi-directional, the last of each direction, although PyTorch takes care of this.)

3 Task

3.1 Training on SNLI

Your task is to implement, perform hyperparameter tuning, and analyze the results of the model.

Because the SNLI data set is relatively large, we recommend starting the assignment early. Likewise, because you are training on only a subset of the

[1] https://fasttext.cc/docs/en/english-vectors.html

full data, you should not expect to get accuracies comparable to the published leaderboard.

Perform tuning over **at least two** of the following hyperparameters:

- Varying the size of the hidden dimension of the CNN and RNN
- Varying the kernel size of the CNN
- Experiment with different ways of interacting the two encoded sentences (concatenation, element-wise multiplication, etc)
- Regularization (e.g. weight decay, dropout)

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.

We recommend that you refer to the PyTorch code from the lab session on CNN and RNN models to tackle the assignment. Copying code from existing public implementations without acknowledgement or demonstrating understanding will be treated as plagiarism. Please let us know if you have questions on Piazza or visit us during office hours.

3.2 Evaluating on MultiNLI

The Multi-Genre Natural Language Inference (MultiNLI) task is a variant of SNLI which covers spoken and written text from different sources, or "genres".

You task is to take your best trained model, one each for RNNs and CNN, and evaluate them on the provided MultiNLI data, for each genre. This means you should have a table of validation accuracies for an RNN and a CNN model, for each genre. Briefly discuss the results, comparing them both to SNLI accuracies as well as comparing the results within MultiNLI across genres.

You are provided with subsets of the MultiNLI training and validation data set in the same `.tsv` format as the above, except with an addition "genre" column. Perform your evaluation on the validation data set for MultiNLI separately for each genre. (The training subset for MultiNLI is only used for the optional section below.)

3.3 Fine-tuning on MultiNLI (Bonus)

(This section is optional, but highly recommended. If you're already completed the above, this should not take much more code or thought.)

In this section, we will *fine-tune* our best trained SNLI model (pick the best over RNNs or CNNs in this case). This means that we will take our pre-trained model, and training it further on a new task / data set, to fine-tune it for that

task. Often, fine-tuning is done on tasks where we may have less data but wish to use information from both a pre-training task where we have a lot of data, and data from the new task.

Specifically, we will fine-tine our SNLI model on training data for each MultiNLI genre (training set), and evaluate it on that genre (validation set). Since our SNLI model is already trained, we do not need (nor want) to over-train our model–hence, you can continue training just on a small number of epochs, or use a lower learning rate. Take note that we will fine-tune separately for each genre, so be sure to save/load the `state_dict` of your respective models when you fine-tune on a new genre.

In short:

1. For each genre, take your best SNLI model and train it a number of epochs on the training set of each genre.

2. Evaluate your fine-tuned models on that genre.

Report the validation accuracies for each genre with and without fine-tuning. If you have the time, further evaluate each fine-tuned model on every other genre, to see if that fine-tuning carries over to the other genres.

4 Submission

Your submission will be a LaTeX-formatted PDF with analysis and discussion of results, as well as a link to a public GitHub repository with your code.

To obtain full marks, we expect that you achieve at least 65% accuracy on the validation set. You will get 0 marks if you do not submit the written report and the link to a public GitHub repository. We will take away 15 marks if you submit the link to GitHub repository late or make significant commits to the repository after the assignment deadline.

5 Hints

- SNLI is a relatively large data set, so you could try to check that your implementation is successfully training on an even smaller subset before tuning on full subset we provide.

- **Start Early**.

- When fine-tuning, make sure you keep track of states are being saved and copied, and which optimizers and models you are using.

- Think about how you are handling the variable text lengths in each model, both from a programming and a modeling perspective. There are at least 2 different ways of handling the variable text lengths for RNNs programmatically.