DS-GA 1008: Deep Learning, Spring 2019
Homework Assignment 3

Due: 6pm on Friday, Mar 29, 2019

`If you get, give. If you learn, teach. Maya Angelou (1928 - 2014)`

1. Fundamentals

1.1. Dropout

Dropout is a very popular regularization technique.

(a) [2 pts] List the `torch.nn` module corresponding to 2D dropout.
Answer: Dropout2d

(b) [8 pts] Read on what dropout is and give a short explanation on what it does and why it is useful. You might find these references helpful:

1. Section 7.12 of the Deep Learning Book by Ian Goodfellow et al: https://www.deeplearningbook.org/contents/regularization.html.
2. Original paper: link

Short Answer: Dropout is a regularization technique that alleviates overfitting and helps with generalization in an efficient way. During training, dropout removes a unit (with all its incoming and outgoing weights) from the network with probability p. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time.
Long Answer: The following are helpful excerpts from the original paper.
"Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights."
Description of the meaning of "co-adaptation": *"In a standard neural network, the derivative received by each parameter tells it how it should change so the final loss function is reduced, given what all other units are doing. Therefore, units may change in a way that they fix up the mistakes of the other units. This may lead to complex co-adaptations. This in turn leads to overfitting because these co-adaptations do not generalize to unseen data. We hypothesize that for each hidden unit, dropout prevents co-adaptation by making the presence of other hidden units unreliable. Therefore, a hidden unit cannot rely on other specific units to correct its mistakes. It must perform*

well in a wide variety of different contexts provided by the other hidden units. To observe this effect directly, we look at the first level features learned by neural networks trained on visual tasks with and without dropout."
Test time behavior: *"At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. The weights of this network are scaled-down versions of the trained weights. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in Figure 2. This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time. By doing this scaling, 2n networks with shared weights can be combined into a single neural network to be used at test time."*

1.2. Batch Norm

(a) [2 pts] What does mini-batch refer to in the context of deep learning?
Answer: A mini-batch is a subset of the training data. It is used to compute a single update of the weights of a model with some optimization algorithm. It makes learning more efficient as it provides a less computationally expensive alternative to the (sometimes even impossible) scenario of using the entirety of the training dataset in a single optimization step.

(b) [8 pts] Read on what batch norm is and give a short explanation on what it does and why it is useful. You might find these references helpful:

1. This blog post: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
2. Original paper: link

Answer: The abstract of the original paper sums it up nicely.
"Training Deep Neural Networks is complicated by the fact that the distribution of each layers inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout."

DS-GA 1008: Deep Learning, Spring 2019
Homework Assignment 3

Due: 6pm on Friday, Mar 29, 2019

2. Language Modeling

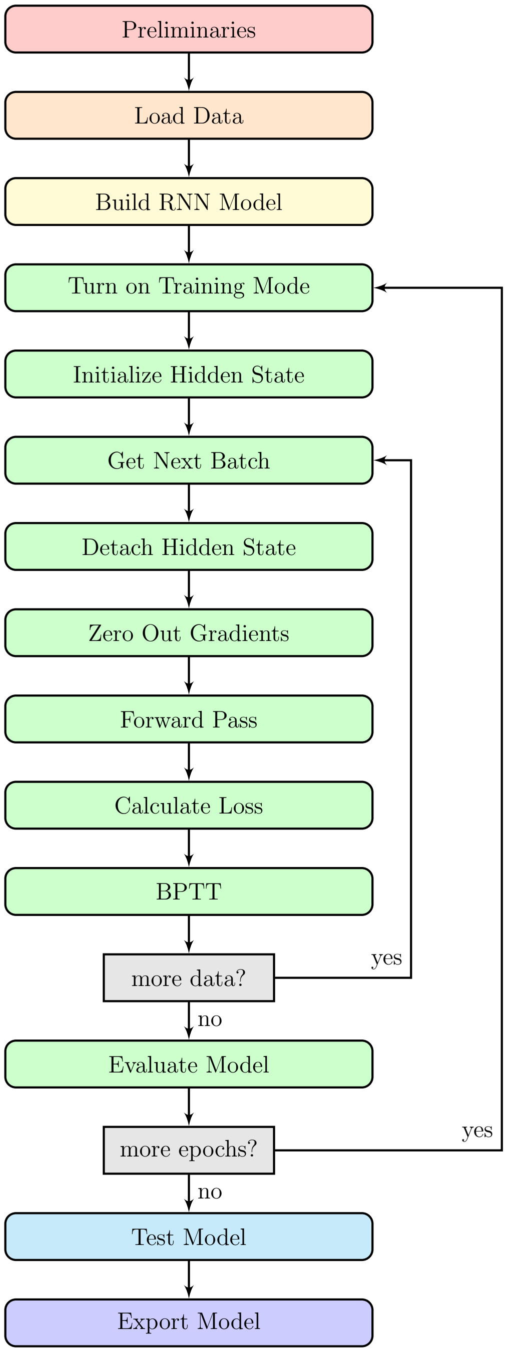
This exercise explores the code from the `word_language_model` example in *PyTorch*.

(a) Go through the code and draw a block diagram / flow chart (see this tutorial) which highlights the main interacting components, illustrates their functionality, and provides an estimate of their computational time percentage (rough profiling). [10 pts]

Answer: There are many possibilities here. The flowchart below is based on Erica Dominic's submission.

DS-GA 1008: Deep Learning, Spring 2019
Homework Assignment 3

Due: 6pm on Friday, Mar 29, 2019



Preliminaries: $<1\%$ computational time

- Parse arguments
- Set random seed for reproducibility
- Set device (CPU or CUDA)

Load Data: 1% computational time

- Load WikiText-2 dataset
- Batchify the training set, validation set, and test set separately

BUILDING: $<1\%$ computational time

- Build RNN model and set criterion (i.e. cross entropy loss)

TRAINING: 96% computational time

- Turn on training mode
- Initialize hidden state
- Get next batch
- Detach hidden state
- Zero out gradients so they don't accumulate
- Forward pass through model
- Calculate loss
- Backpropagate through time
- Evaluate on validation data; save model or anneal learning rate when appropriate

TESTING: 3% computational time

- Run model on as yet unseen test data

EXITING: $<1\%$ computational time

- Export model in ONNX format

Note: percent computation time is based on running one epoch without a GPU. Percentages are similar for $25-40$ epochs on a GPU.

DS-GA 1008: Deep Learning, Spring 2019
Homework Assignment 3

Due: 6pm on Friday, Mar 29, 2019

(b) Find and explain where and how the back-propagation through time (BPTT) takes place (you may need to delve into *PyTorch* source code). [5 pts]
Answer: BPTT takes place in line 159 where `loss.backward()` is called. `Autograd` (PyTorchs automatic differentiation package) uses the computational graph of operations performed on tensors during the forward pass and computes the gradients for all of the tensors for the given loss. The gradients are propagated back through time only within a given data batch (see Part (c)) which is a *truncated* version of BPTT.
Code reference: *to be added*

(c) Describe why we need the `repackage_hidden(h)` function, and how it works. [5 pts]
Answer: `repackage_hidden(h)` takes `torch` tensors as input and returns `torch` tensors with the same `.data` but removed from the computational graph of the original tensors. [1] As hinted in the comments in lines 153-154 of the code, `repackage_hidden(h)` is needed in the `for` training loop over data batches for the `hidden` tensors because otherwise gradients will be backpropagated to the beginning of the dataset which can lead to vanishing/exploding gradients.

(d) Why is there a `--tied (tie the word embedding and softmax weights)` option? [5 pts]
Answer: The `--tied` option is there to tie the weights of the encoder and the decoder together. As suggested in "Using the Output Embedding to Improve Language Models" (arXiv), tying the input and output embeddings it can reduce the size (i.e., the number of parameters) in neural translation models to less than half of their original size without harming their performance. Additionally, it leads to an improvement in the perplexity of various language models, irrespective of whether were using dropout or not.

(e) Compare LSTM and GRU performance (validation perplexity and training time) for different values of the following parameters: number of epochs, number of layers, hidden units size, input embedding dimensionality, BPTT temporal interval, and non-linearities (pick just 3 of these parameters and experiment with 2-3 different values for each). [10 pts]
Answer: Based on the parameters you chose, there are many possible answers. The following tables are taken from Joanna Bitton's submission for experiments ran on NYU HPC with using a GPU. (Please note, beside the parameter values explicitly shown, all other settings have been set to the default ones.)

[1] Note that whenever an operation is performed on a Tensor, this operation is stored in a computational graph that `Autograd` uses to compute gradients when `.backward()` is called). Also, as mentioned in `.grad`, every call of `.backward()` will accumulate gradients in a tensor's `grad` attribute.

DS-GA 1008: Deep Learning, Spring 2019
Homework Assignment 3

Due: 6pm on Friday, Mar 29, 2019

LSTM:

# Epochs	# Hidden Units	# Layers	Total Time	Validation Loss	Validation Perplexity
25	200	2	18:54	4.78	118.54
25	200	3	16:46	4.79	119.93
25	400	2	30:22	4.73	113.69
25	400	3	36:16	4.77	117.60
40	200	2	33:11	4.74	114.80
40	200	3	39:38	4.78	118.63
40	**400**	**2**	**46:47**	**4.73**	**113.17**
40	400	3	37:52	4.74	114.38

GRU:

# Epochs	# Hidden Units	# Layers	Total Time	Validation Loss	Validation Perplexity
25	200	2	21:08	5.14	171.25
25	200	3	24:34	5.01	150.31
25	400	2	28:07	4.79	120.64
25	400	3	34:05	5.57	263.11
40	200	2	33:27	5.09	161.89
40	200	3	38:13	5.01	149.31
40	**400**	**2**	**44:47**	**4.79**	**120.49**
40	400	3	51:38	4.83	125.42

(f) Why do we compute performance on a test set as well? What is this number good for? [5 pts]
Answer: Evaluating performance on the test set provides an estimate of how a model would perform on unseen data, i.e. it is an indication of how well the model generalizes.

3. Programming

Complete exercises in `DS-GA-1008-HW_assignment_3.ipynb` [80 pts].

4. Submission

You are required to write up your solutions to Part 1 and Part 2 using LaTeX.
Submit the following files to **NYU Classes** by the deadline (6pm on Friday, March 29, 2019):

- `First-name_Last-name_netID_A3.pdf` file for Part 1 and Part 2 (containing a read-only link to your Overleaf project for Part 1 and Part 2)

- `First-name_Last-name_netID_A3.tex` file for Part 1 and Part 2
- `First-name_Last-name_netID_A3.ipynb` file for Part 3

5. Disclaimers

You are allowed to discuss problems with other students in the class but have to write up your solutions on your own.

As feedback might be provided during the first days, the current homework assignment might be undergoing some minor changes. We'll notify you if this happens.