CMU 10-715: Homework 6

Implementation of Dropout Classifiers

DUE: Oct. 29, 2020, 11:59 PM.

Instructions:

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books, again after you have thought about the problems on your own. Please don't search for answers on the web, previous years' homeworks, etc. (please ask the TAs if you are not sure if you can use a particular reference). There are two requirements: first, cite your collaborators fully and completely (e.g., "Alice explained to me what is asked in Question 4.3"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only.

- **Submitting your work:** Assignments should be submitted as PDFs using Gradescope unless explicitly stated otherwise. Each derivation/proof should be completed on a separate page. Submissions can be handwritten, but should be labeled and clearly legible. Else, submission can be written in LaTeX.

- **Late days:** For each homework you get three late days to be used only when anything urgent comes up. No points will be deducted for using these late days. We will consider an honor system where we will rely on you to use the late days appropriately.

- **Training NNs**: It is feasible to train the network in this homework on a CPU. However, to speed up the training process, you may use Google colab[1], a free GPU service provided by Google. To enable GPU, under the Runtime tab, select Change runtime type and choose GPU. To mount a colab notebook to your local file systems or google drive, refer to the link here[2].

[1] `https://colab.research.google.com/notebooks/intro.ipynb`

[2] `https://colab.research.google.com/notebooks/io.ipynb`

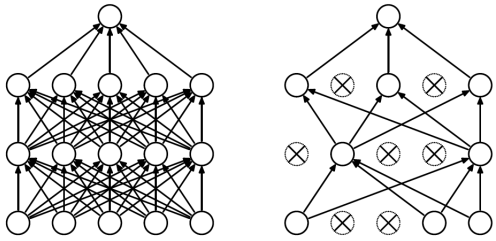
1 Implementation of Dropout Classifiers [100 points]

In this homework you are going to replicate the results of the original dropout paper[3] and you will also explore and learn how to use **PyTorch** which is one of the most popular, open source deep learning frameworks.

1.1 Dropout Regularization

As you already learnt in class, deep neural networks are extremely expressive models which can learn complicated relations. They are also therefore prone to overfit, that is, fit to the noise rather than the true separator. Dropout regularization is one of the most popular techniques to prevent deep neural networks to overfit, improving their performance on unseen test data.

The main idea of the dropout is to randomly drop neurons and 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 approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned (without dropping units) network that has smaller weights (by multiplying weights by the probability of dropout).



(a) Standard Neural Net (b) After applying dropout.

1.2 PyTorch

Before you start, please checkout this useful link[4] that explains the basics of **PyTorch** library, so that you feel comfortable with its classes and methods.

We provided you with a *setup.sh* file that has conda instructions to create an environment with the packages you will need for this assignment. We also provided you with the *data.py* file that creates the dataloaders for train and validation with 10,000 images from the MNIST dataset that you explored before.

[3] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958. Avaliable at `https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf`

[4] `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

1.3 Instructions

- Read **PyTorch**'s documentation and load the MNIST data.
- You will need to complete the missing code (marked with #TODO) in the script provided, ***dropout_classifier.py***.
- Complete the function ***_initialize_network*** method on *dropout_classifier.py* that defines the architecture of the neural network. The neural network must have 3 hidden layers (the number of neurons of each layer it is specified in the hyper-parameters) with ReLU activations. You need to add dropout regularization after the input layer and on each hidden layer. (Hint: Use the nn.Linear(), nn.ReLU(), nn.Dropout() modules.)
- Complete the ***fit*** method. You need to add the forward pass (compute predictions), compute the cross entropy loss [5] and define the backward step. (Hint: We recommend you to use a linear layer for the output_layer and cross entropy loss meant for these logits (note that in deep learning people refer to the logits as these final activations)).
- Complete the ***evaluate_cross_entropy*** method which computes the cross entropy loss for the full sets. *Hint: this function is analogous to the* ***evaluate_accuracy*** *method.*
- Train your classifiers, we provided you with beginning code in the ***main*** function. (Hint: trajectories will be available in the ***trajectories*** attribute of the classifier, save the weights of your classifiers so that your progress is not lost.)
- You should be able to reach around 96.5% test accuracy with 0.5 dropout probability. The original paper runs the optimization for a million steps, try to run your classifiers for at least **a hundred thousand steps** (refer to the initial instructions on how to train NNs using Google colab), keep in mind that you should be able to almost perfectly fit the training set (close to 100% accuracy).

1.4 Expected Results

Finally, for the following values of dropout probability (parameter *hidden_dropout_prob* in params list) $p \in \{0.0, 0.2, 0.5, 0.8\}$ report the following :

a (10 points) Train your classifiers (using GPU) for **a hundred thousand steps**. You may run your algorithm for **ten thousand steps** (using CPU) and you will be awarded full points for the next results, but won't receive this 10 points.

[5] see `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

b (25 points) In the same plot, display the train and test loss vs the gradient steps (only for the given display steps specified in the hyper-parameters, each display is expensive to compute).

c (25 points) Plot gradient steps vs train and test accuracy (only for the given display steps specified in the hyper-parameters).

d (30 points) Plot the input_layer's learnt filters (weights) of 25 units (of your selection), plot on a 5x5 grid figure. For this filters plot, the weights must be normalized to be between 0 and 1, using the min and max of all the weights. For this use the ***get_filters*** method.

e (10 points) Report the final loss and accuracy for train and test sets. Provide your comments on the relationship between generalization and regularization, what are your thoughts on the filters?.