CS 7180 Special Topics in AI: Deep Learning
Northeastern University, Spring 2019

Assignment 1 (5 Points)

Due 11:59 p.m. Sunday, Jan 27, 2019

Submission Instructions

Please submit your assignment as a .zip archive with filename `LastnameFirstname.zip` (replacing `Lastname` with your last name and `Firstname` with your first name), containing a PDF of your assignment writeup **in the main directory** with filename `LastnameFirstname_1.pdf` using the provided LaTeX template. (`http://roseyu.com/CS7180/latex_template.tex`). Submit your code as Jupyter notebook .ipynb files or .py files, and **include any images generated by your code along with your answers in the solution .pdf file.** We ask that you use Python 3.6 for your code, and that you comment your code such that the TAs can follow along and run it without any issues. Failure to do so will result in a **1 point deduction**.

1 Convolutional Neural Network

Read the convolutional neural network chapter (Chapter 9) of the Deep Learning book and build a deep neural network to classify the handwritten digits.

1.1 Dataset

MNIST is a classic dataset in computer vision. It consists of images of handwritten digits (0 - 9) and the correct digit classification. In this problem you will implement a deep network using Pytorch to classify MNIST digits. Specifically, you will explore what it really means for a network to be "deep", and how depth vs. width impacts the classification accuracy of a model. You will be allowed at most N hidden units, and will be expected to design and implement a deep network that meets some performance baseline on the MNIST dataset.

1.2 Installation

Before any modeling can begin, Pytorch must be installed. Pytorch is an open source deep learning platform that provides a seamless path from research prototyping to production deployment.

To install Pytorch, follow the steps on `https://pytorch.org/get-started/locally/`. We recommend using the Anaconda install instructions if you installed python via the Anaconda distribution, or using the Pip install instructions if you didn't.

Once you have finished installing, write down the version numbers for Pytorch that you have installed.

1.3 Classify MNIST with CNN

Using Pytorch to build a deep *convolutional* network to classify the handwritten digits in MNIST. You are **only** allowed to use the following layers:

- **Dense:** A fully-connected layer
  - In convolutional networks, Dense layers are typically used to knit together higher-level feature representations.
  - Particularly useful to map the 2D features resulting from the last convolutional layer to categories for classification (like the 1000 categories of ImageNet or the 10 categories of MNIST).
  - Inefficient use of parameters and often overkill: for A input activations and B output activations, number of parameters needed scales as $O(AB)$.
- **Conv2D:** A 2-dimensional convolutional layer
  - The bread and butter of convolutional networks, conv layers impose a translational-invariance prior on a fully-connected network. By sliding filters across the image to form another image, conv layers perform "coarse-graining" of the image.
  - Networking several convolutional layers in succession helps the convolutional network knit together more abstract representations of the input. As you go higher in a convolutional network, activations represent pixels, then edges, colors, and finally objects.
  - More efficient use of parameters. For N filters of $K \times K$ size on an input of size $L \times L$, the number of parameters needed scales as $O(NK^2)$. When N, K are small, this can often beat the $O(L^4)$ scaling of a Dense layer applied to the L^2 pixels in the image.
- **MaxPooling2D:** A 2-dimensional max-pooling layer
  - Another way of performing "coarse-graining" of images, max-pool layers are another way of ignoring finer-grained details by only considering maximum activations over small patches of the input.
  - Drastically reduces the input size. Useful for reducing the number of parameters in your model.
  - Typically used immediately following a series of convolutional-activation layers.
- **BatchNormalization:** Performs batch normalization (Ioffe and Szegedy, 2014). Normalizes the activations of previous layer to standard normal (mean 0, standard deviation 1).
  - Accelerates convergence and improves performance of model, especially when saturating nonlinearities (sigmoid) are used.
  - Makes model less sensitive to higher learning rates and initialization, and also acts as a form of regularization.
  - Typically used immediately before nonlinearity (Activation) layers.
- **Dropout:** Takes some probability and at every iteration sets weights to zero at random with that probability

  - An effective form of regularization. During training, randomly selecting activations to shut off forces network to build in redundancies in the feature representation, so it does not rely on any single activation to perform classification.

- **Activation (ReLU):** Sets negative weights to 0
- **Activation (Softmax):** Sets highest weight to 1, rest to 0
- **Flatten:** Flattens any tensor into a single vector (required in order to pass a 2D tensor output from a convolutional layer as input into Dense layers)

Your tasks. Build a network with only the allowed layers that achieves **test accuracy of at least** 0.985. You are required to use categorical cross entropy as your loss function and to train for 10 epochs with a batch size of 32. Note: your model must have fewer than 1 million parameters, as measured by *'model.count_params()'*. Everything else can change: optimizer (RMSProp, Adam, ???), initial learning rates, dropout probabilities, layerwise regularizer strengths, etc. You are not required to use all of the layers, but *you must have at least one dropout layer and one batch normalization layer in your final model*. Try to figure out the best possible architecture and hyperparameters given these building blocks!

In order to design your model, you should train your model for 1 epoch (batch size 32) and look at the final **test accuracy** after training. This should take no more than 10 minutes, and should give you an immediate sense for how fast your network converges and how good it is.

Set the probabilities of your dropout layers to 10 equally-spaced values $p \in [0, 1]$, train for 1 epoch, and report the final model accuracies for each.

You can perform all of your hyperparameter validation in this way: vary your parameters and train for an epoch. After you're satisfied with the model design, you should train your model for the full 10 epochs.

In your submission. Turn in the code of your model, the test accuracy for the 10 dropout probabilities $p \in [0, 1]$, and the final test accuracy when your model is trained for 10 epochs. We should have everything needed to reproduce your results.

Discuss what you found to be the most effective strategies in designing a convolutional network. Which regularization method was most effective (dropout, layerwise regularization, batch norm)?

Do you foresee any problem with this way of validating our hyperparameters? If so, why? *Hints:*

- You are provided with a sample network that achieves a high accuracy. Starting with this network, modify some of the regularization parameters (layerwise regularization strength, dropout probabilities) to see if you can maximize the test accuracy. You can also add layers or modify layers (e.g. changing the convolutional kernel sizes, number of filters, stride, dilation, etc.) so long as the total number of parameters remains under the cap of 1 million.

- You may want to read up on successful convolutional architectures, and emulate some of their design principles. Please cite any idea you use that is not your own.
- **Make sure to normalize the input vectors (dividing all values by 255).**
- Dense layers take in single vector inputs (ex: *(784,)*) but Conv2D layers take in tensor inputs (ex: *(28, 28, 1)*): width, height, and channels. You will need to reshape the training/test X to a 4-dimensional tensor (ex: *(num_examples, width, height, channels)*) using *'np.reshape'*. For the MNIST dataset, *channels=1*. Typical color images have 3 color channels, 1 for each color in RGB.
- If your model is running slowly on your CPU, try making each layer smaller and stacking more layers so you can leverage deeper representations.
- Other useful CNN design principles:
  - CNNs perform well with many stacked convolutional layers, which develop increasingly large-scale representations of the input image.
  - Dropout ensures that the learned representations are robust to some amount of noise.
  - Batch norm is done after a convolutional or dense layer and immediately prior to an activation/nonlinearity layer.
  - Max-pooling is typically done after a series of convolutions, in order to gradually reduce the size of the representation.
  - Finally, the learned representation is passed into a dense layer (or two), and then filtered down to the final softmax layer.