Introduction to Machine Learning, Spring 2016

Problem Set 7: Deep learning with neural networks

Due: Monday, May 9, 2016 at 10pm (upload to NYU Classes.)

Important: *See problem set policy on the course web site.*

Instructions. You must show **all** of your work and be rigorous in your writeups to obtain full credit. Your answers to the below, plots, and all code that you write for this assignment should be uploaded to NYU Classes.

1. **Back-Propagation** (5 pts)

   Logistic regression is a popular technique in machine learning to classify data into two categories (y=0 or 1). This technique builds over linear regression by using the same linear model but it is followed by the sigmoid function which converts the output of the linear model to a value between 0 and 1. This value can then be interpreted as a probability:

   $$p(y = 1|\mathbf{x}) = \sigma(z) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \tag{1}$$

   where $\mathbf{x}$ is the input feature vector, $z = \mathbf{w}^T\mathbf{x} + b$, and σ represents the sigmoid function. Further denote $\hat{y} = p(y = 1|\mathbf{x})$. Answer the follow questions:

   (a) Suppose that we have a loss function $l(\hat{y}, y)$ which measures the dissimilarity between the prediction $\hat{y}$ and the true label y. Assume that we already calculated $\frac{\partial l}{\partial \hat{y}}$, write down the expression for $\frac{\partial l}{\partial \mathbf{w}}$ given $\frac{\partial l}{\partial \hat{y}}$.

   (b) We decide to use the cross entropy loss (which corresponds to maximum likelihood estimation): $l(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$. Suppose that the current weights are given by $\mathbf{w}_t$ and bias b_t, and consider the input $\mathbf{x}$ and true label y given by:

   $$\mathbf{w}_t = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad b_t = 0, \quad \mathbf{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad y = 1$$

   Using these, calculate $\hat{y}$, $l(\hat{y}, y)$ as well as $\frac{\partial l}{\partial \mathbf{w}}$ for this data point, evaluated at $\mathbf{w}_t, b_t$.

2. **Convolutional Neural Nets for Digit Recognition** (12 pts)

   In PS3, we experimented with SVM-based approaches on MNIST dataset. In this assignment, we will attack the same digit recognition problem with convolutional neural networks using `TensorFlow`. The MNIST dataset was processed and pickled to make it easier to use in python.[1]

   Sample code is provided to you. `data.py` defines a data class that loops through the dataset and provides mini-batches during training; `model.py` defines a simple 2-layer convolutional neural network model; `train.py` defines the training process. Execute `python train.py` to start training.

   (a) Read through the sample code, making sure you understand each part. Refer to the TensorFlow Python API for specific usage of functions. What are the kernel sizes and strides of the first and second convolutional layers (counting from the input layer)?

   (b) Pooling and convolution normally change the size of the input. Suppose you have an input image of 96 by 96 pixels, what is the size of the output after a convolutional layer (without padding) with kernel size 5 by 5 and stride 1 followed by a pooling layer with kernel size 3 by 3 and stride 3? Explain how you get the result.

[1] `http://deeplearning.net/data/mnist/mnist.pkl.gz`

(c) In `train.py`, accuracy is reported every 100 iterations. Which part of the data are these accuracies evaluated on? Implement the function `get_accuracy` and report accuracy evaluated on the whole validation set every 100 iterations.

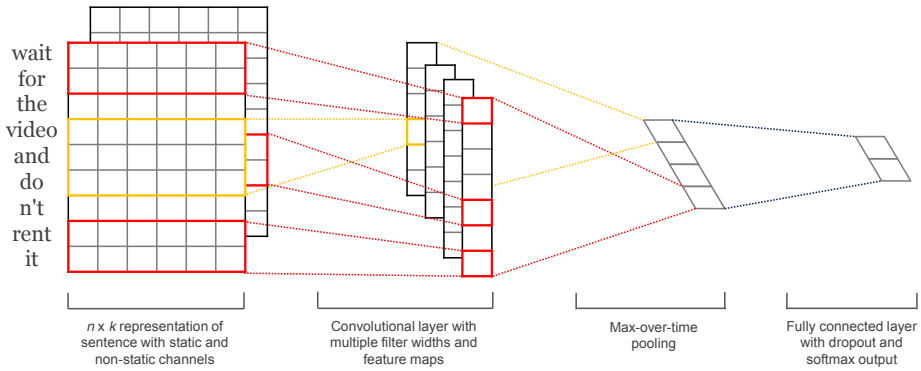
(d) A technique that is often used to prevent over-fitting in neural networks is early-stopping. A simple version is to stop training after the best validation error does not change for a certain number of iterations. Implement early-stopping in `train.py` to stop training after the validation error does not decrease for 300 iterations (validation error evaluated every 100 iterations). How many iterations does it take before training stops?

(e) Choose at least one of the hyper-parameters (kernel size, number of filters, number of convolutional layers, dropout probability, etc.) and experiment with the effect of the hyper-parameter on model performance. Report the validation accuracy for each configuration you tried as a table. After you finish this, evaluate the best model on the test set and report your results. Make sure to use early-stopping as implemented in the previous question.

(f) (bonus 2 pts) Visualize the 32 kernels in the first convolutional layer. Hint: you can save and restore a session containing all the trained weights using `tf.train.Saver`. The value of a variable `var` can be retrieved using `session.run(var)`. Refer to `maplotlib.pyplot.imshow` for visualizing a 2D array.

3. **Convolutional Neural Nets for Text Classification** (6 pts)

In this section, we are going to study word embeddings and convolutional neural networks (CNNs) for text classification. We have provided you with sample code which trains the model shown in the below figure (Kim, EMNLP 2014). Word embeddings are a technique often used in Natural Language Processing to get a vector representation of a word. There are multiple techniques to obtain such embeddings, one of the most popular ones being Word2Vec [2]. The embeddings learned by Word2Vec exhibit nice properties : similar words tend to cluster together ("lunch", "dinner", "breakfast" for example). In this assignment, we provide you with embeddings learned using Word2Vec. You are going to evaluate the quality of these embeddings, and then use them to do sentiment analysis on Rotten Tomatoes movie reviews.



(a) Read the code in the file `nearestneighbors.py`, and implement the function `getNN`. This function should return the list of words whose embeddings are the closest to the embedding of a given word in terms of cosine similarity. The cosine similarity betweens two vectors $\vec{u}$ and $\vec{v}$ is $\frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\|\vec{v}\|}$. In your report, show the 10 nearest neighbors of the following words: "movie", "man", "computer".

(b) Read the code for text classification using CNN (files `train.py`, `data_helpers.py` and `text_cnn.py`. The first layer of this model is the embedding layer. Change this layer to use the pre-trained embeddings of the file `embeddingMatrix.npy`, and do not include this layer in the trainable parameters (thus keeping the embeddings fixed[3]). Report the final cross-validation accuracy you obtain.

[2] `https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf`

[3] See `https://www.tensorflow.org/versions/r0.8/api_docs/python/state_ops.html#trainable_variables` for reference