6.867: Homework 3

This homework will be structured in the same way as homeworks 1 and 2. Details of the submission and grading process are repeated at the end.

- We *strongly* encourage you to do this assignment in groups of two students, though you may do it individually if you wish.

- You submit your paper via the CMT site by 11:59 PM on Thursday, November 10.

- Each student who was an author or co-author on a submission will be assigned 3 papers to review.

- Reviews must be entered on the CMT site by 11:59PM on Tuesday, November 15.

- All reviews will be made visible shortly thereafter, and students will have a two day period in which to enter rebuttals of their reviews into CMT if they wish.

Introduction

1. Section 1 asks you to implement a "traditional" fully-connected neural network and test it on several data sets. Although it's tempting to use an existing implementation, you won't really understand the operation of these networks unless you do an implementation from scratch.

2. Section 2 asks you to experiment with an existing implementation of convolutional neural nets so that you can get some idea of the space of "architectural" decisions one faces when applying this approach.

1   Neural Networks

In this question, we will implement backpropagation in a vector/matrix style, as presented in class. You should review the neural network lecture notes.

Use Matlab or Python to implement a program to train fully-connected neural nets using stochastic gradient descent. Do your implementation in sufficient generality so that you can vary the number of nodes in each hidden layer and the number of hidden layers. The gradient computation for gradient descent is given by the pseudo-code for Backprop, in the course notes. Make sure that you understand the dimensions of each of vectors and matrix in the pseudo-code. When properly implemented, you should be able to do essentially all the computations layer-wise in matrix form.

First test your implementation on the simple two-dimensional, 3-class data set we provided you in (`data/data_3class.csv`). Try networks with one and two hidden layers. This is primarily for debugging, you do not need to report on this in detail.

1. ReLU + Softmax:

   Assume that the **hidden** units are all ReLU; whose activation functions are $f(z) = \max(0, z)$. Also assume that the output layer is a Softmax layer for k classes. A Softmax layer maps a vector of k inputs (the outputs of the previous layer) to a vector of k "probabilities" that add up to one. The activation function for each output is:

   $$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad \text{for } i = 1 \dots k$$

   We assume that the target outputs are represented as a "one-hot" (1-of-k) vector. Let the loss function be cross entropy, that is, for a particular training point (x, y), the loss is $- \sum_{i=1}^{k} y_i \log(f(z)_i)$ where $f(z)$ is the Softmax output; see also equations 5.24 and 5.25 in Bishop.

   Describe how the choice of a Softmax output layer and cross-entropy loss is incorporated into your implementation.

2. Initialization:

   A common way of initializing the weights for a neural net is to use random weights distributed as a Gaussian with zero mean and standard deviation $1/\sqrt{m}$ for a weight matrix with dimensions $[m, n]$. Imagine that all the units are sigmoids, why would this be a reasonable choice?

3. Regularization:

   If we wanted to add weight regularization during training:

   $$J(w) = \ell(w) + \lambda(\|w^{(1)}\|_F^2 + \|w^{(2)}\|_F^2)$$

   where $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$ is the matrix Frobenius norm. How would this affect the pseudo-code? Be specific. You dont need to implement and test this (unless you want to).

4. Binary classification:

   Test your implementation on the four 2D data sets from HW 2. During the training process, use the training set to fit the weights and use the validation set to decide when to stop training. Report the final accuracy on the training data and on the test data.

   Pay careful attention to the form of the target values you need for a Softmax output unit. You will need to convert the data in the files to that form.

   Experiment with some architectures (single small hidden layer, single large hidden layer, two small hidden layers, two large hidden layer) and report your results. You will need to experiment with learning rates to find ones that work for the individual architectures. Discuss any trends that you see and compare your best results to those you obtained on these data sets in HW 2. You should be able to adapt the basic data reading and display code from HW 2.

5. Multi-class classification:

   Repeat this process for the 10-class digit classification problem, using the MNIST data from HW 2. Note that in HW 2 we addressed several binary classification problems, such as odd vs even, with the MNIST data, now we are asking you to predict the digit directly. Remember to normalize the data as you did in HW 2 (what happens if you don't?). Try it with at least 200 samples per class but try more if you can.

2   Convolutional Neural Networks

In this question, you will be asked to experiment with a convolutional neural network implementation to perform image classification. The dataset we will use for this assignment was created by Zoya Bylinskii, and contains 451 works of art from 11 different artists all downsampled and padded to the same size. The task is to identify which artist produced each image. The original images can be found in the *art_data/artists* directory included with the assignment. The composition of the dataset and a sample painting from each artist are shown in Table 1.

Table 1: Artists and number of paintings in the dataset
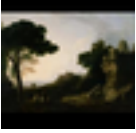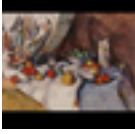
Artist	#	Example	Artist	#	Example
Canaletto	20		Paul Gauguin	37	
Claude Monet	54		Paul Sandby	36	
George Romney	63		Peter Paul Rubens	28	
J. M. W. Turner	70		Rembrandt	40	
John Robert Cozens	40		Richard Wilson	23	
Paul Cezanne	40				

Figure 1 shows an example of the type of convolutional architecture typically employed for similar image recognition problems. Convolutional layers apply filters to the image, and produce layers of feature maps. Often, the convolutional layers are interspersed with pooling layers. The final layers of the network are fully connected, and lead to an output layer with one node for each

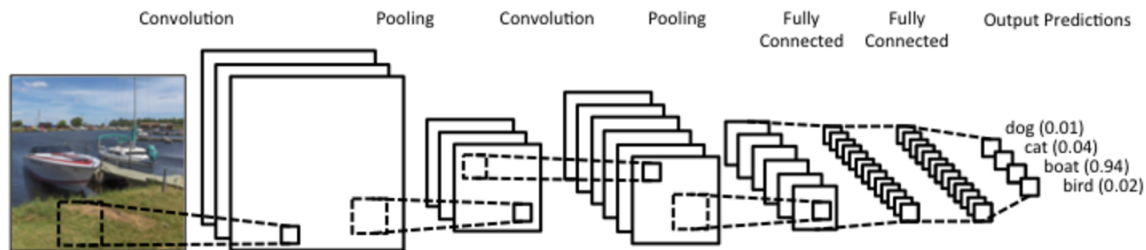of the K classes the network is trying to detect. We will use a similar architecture for our network.

Figure 1: A typical convolutional architecture. Image taken from http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

For this question, we will help you by providing the code you will need to perform the data processing and network training steps. The conv net code will be implemented in Tensorflow, a popular library for training deep neural networks. You will need to install Tensorflow and its dependencies on your machine by following the instructions here: https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html.

In the code, the file *make_datasets.py* creates a dataset from the artists' images by downsampling them to 50x50 pixels, and transforming the RGB values to lie within the range [-0.5, 0.5]. The resulting dataset is saved in the file *art_data.pickle*. The code to train the network can be found in *conv.py*[1]. You do not need to understand all of this code to complete the assignment, but you will need to make changes to the hyperparameters and network structure found in the function *define_tensorflow_graph*. Note that Tensorflow uses deferred computation; it sets up a computation *graph*, containing *tensors* - variables that do not necessarily store actual values until they are evaluated. The graph determines the flow of information through the network. Only once we are ready to train the model do we initialize a *session* and start feeding data into the graph. You can read more about how Tensorflow works here.

1. **Convolutional filter receptive field**

   First, it is important to develop an intuition for how a convolutional layer affects the feature representations that the network learns. Assume that you have a network in which the first convolutional layer applies a 5x5 patch to the image, producing a feature map Z_1. The next layer of the network is also convolutional; in this case, a 3x3 patch is applied to the feature map Z_1 to produce a new feature map, Z_2. Assume the stride used in both cases is 1. Let the *receptive field* of a node in this network be the portion of the original image that contributes

---

[1]Note that this code was optimized for simplicity, not best programming practices

information to the node (that it can, through the filters of the network, "see"). What are the dimensions of the receptive field for a node in Z_2? Note that you can ignore padding, and just consider patches in the middle of the image and Z_1. Thinking about your answer, why is it effective to build convolutional networks deeper (i.e. with more layers)?

2. **Run the Tensorflow conv net**

   Study the provided code in the *define_tensorflow_graph* function in *conv.py*. Answer the following questions about the initial implementation:

   - How many layers are there? Are they all convolutional? If not, what structure do they have?

   - Which activation function is used on the hidden nodes?

   - What loss function is being used to train the network?

   - How is the loss being minimized?

   Now that you are familiar with the code, try training your network! This can be accomplished by simply running the command *python conv.py*. It should take between 30-60 seconds to train your network for 1500 steps. What is the training accuracy for your network after training? What is the validation accuracy? What do these two numbers tell you about what your network is doing?

3. **Add pooling layers**

   We will now max pooling layers after each of our convolutional layers. This code has already been provided for you; all you need to do is switch the *pooling* flag in the hyperparameters to **True**, and choose different values for the pooling filter size and stride. After you applied max pooling, what happened to your results? How did the training accuracy vs. validation accuracy change? What does that tell you about the effect of max pooling on your network?

4. **Regularize your network!**

   Because this is such a small dataset, your network is likely to overfit the data. Implement the following ways of regularizing your network. Test each one individually, and discuss how it affects your results.

   - **Dropout**: In Tensorflow, this is implemented using the tf.nn.dropout function, which takes a value called the *keep_prob*, representing the probability that an activation will **not** be dropped out. This value should be between 0.5 and 1.0 during training, and 1.0 for evaluation and testing. To change the value, you can use a placeholder variable, and feed in different values for *keep_prob* at training and testing. An example of how this works is available here. Dropout has already been integrated into the *conv.py* code. To activate

it, simply set the *dropout_prob* hyperparameter to some value < 1.0. You should explore different values to find one that works well.

- **Weight regularization**: As in the last question, we will add an additional term to the loss function to penalize the norm of your network's weights. Again, this has already been implemented, and you simply need to find a good value for the *weight_penalty* hyperparameter.

- **Data augmentation**: Another effective regularization techniques is to augment the data with transformed versions of the original samples. The file *augmented_art_data.pickle* contains randomly flipped and distorted versions of the original training images (note that the validation images remain unchanged). Change the *DATA_FILE* constant at the top of your *conv.py* code to point to the augmented dataset, and try running your code on this new data. The dataset is 4 times as large, so you should increase the number of training steps accordingly.

- **Early stopping**: Stop training your model after your validation accuracy starts to plateau or decrease (so that you do not overtrain your model). The number of steps can be controlled through the *num_training_steps* hyperparameter.

Discuss your results. Which regularization technique was most effective?

5. **Experiment with your architecture**

All those parameters at the top of *define_tensorflow_graph* still need to be set. You cannot possibly explore all combinations; so try to change some of them individually to get some feeling for their effect (if any). Optionally, you can explore adding more layers. Report which changes led to the biggest increases and decreases in performance. In particular, what is the effect of making the convolutional layers have a) a larger filter size, b) a larger stride and b) greater depth? How does a pyramidal-shaped network in which the feature maps gradually decrease in height and width but increase in depth (as in Figure 2) compare to a flat architecture, or one with the opposite shape?
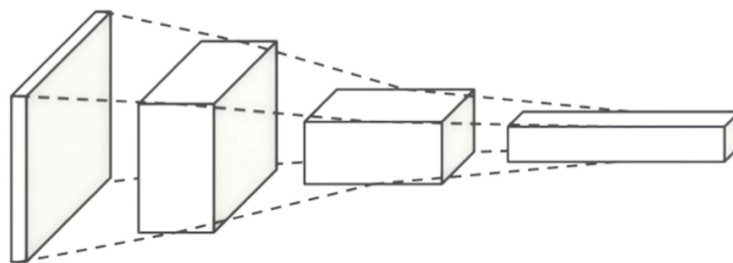

Figure 2: Pyramidal architecture with 3 convolutional layers. Figure was adapted from https://www.udacity.com/course/deep-learning--ud730
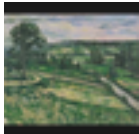
6. **Optimize your architecture**

Based on your experience with these tests, try to achieve the best performance that you can on the validation set by varying the hyperparameters, architecture, and regularization methods. You can even (optionally) try to think of additional ways to augment the data, or experiment with new techniques like local response normalization layers using tf.nn.lrn. Report the best performance you are able to achieve, and the settings you used to obtain it.

7. **Test your final architecture on variations of the data**

Now that you have optimized your architecture, you are ready to test it on new data! The file *invariance_art_data.pickle* contains versions of the validation images that have been distorted using the transformations shown in Table 2. To test your code on this data, run the command *python conv.py invariance*. Report your performance on each of the transformed datasets. Are you surprised by any of the results? Which transformations is your network most invariant to, and which lead it to be unable to recognize the images? What does that tell you about what features your network has learned to use to recognize artists' images?

Table 2: Translations made to images

Transformation	Example	Transformation	Example
Normal		High contrast	
Translated		Low contrast	
Brightened		Flipped upside-down	
Darkened		Colors inverted	

Procedure, grading, etc.

You will find a zip file with some useful code and data in the Stellar Materials page. You can do these assignments in any computational system that you are used to. We recommend Matlab or the Pylab/Numpy/Scipy/Matplotlib cluster of packages and we'll try to provide help for those two systems. If you use anything else, you're on your own...

You will be turning in a single, readable "paper" (a single PDF file) with your solutions. In the Stellar Materials page we have provided some sample solutions from a previous year. The content in these examples is not directly relevant, but note that there are coherent explanations and nice illustrations of results in figures and tables. You should write your paper as if it is going to be read by someone who has taken a class in machine learning but has not read this assignment handout.

We will be emulating the process of submitting papers for publication to a conference. We will be using an actual conference review system (CMT) to have these papers peer reviewed (by the other students). This means that your answers have to be readable and understandable to your peers and, where possible, interesting. Note that when explanations are called for, you will need to convince the reviewers that you understand what you're talking about. The course staff will serve as the Program Committee for the conference and make all final decisions.

Grading rubric

Your paper must be anonymous (no identifying information should appear in the PDF file). If it is not, it will automatically receive a 20% deduction, and will be graded by a grumpy staff member.

The paper must be no more than 6 pages long in a font no smaller than 10 point. It should include whatever tables, graphs, plots, etc., are necessary to demonstrate your work and conclusions. *It should not include code.*

Each of the **four** parts of the assignment will be graded on a scale from 0 to 5 (where 0 is failing and 5 is an A) on two aspects:

- **Content:** Did the solution answer the questions posed? Were the answers correct? Were the experiments well-designed or examples well chosen?

- **Clarity:** Were the results written up clearly? Were the plots labeled appropriately and described well? Did the plots support the points in the paper? Did the discussion in the paper illuminate the plots?

As a reviewer, you will be asked to provide a score for each section, and at at least two paragraphs of feedback, per review, explaining things that were done well and things that could have been improved upon.

Your overall score for this assignment will be:

- **80%**: The average of all 8 scores on your assignment given by all three reviewers.

- **20%**: A score for the quality of your reviews. This will be full credit, by default. But we will skim reviews and examine some carefully and may reduce this grade for review commentary that is sloppy or wrong.

The course staff will spot-check submissions and reviews, paying careful attention to cases where there were rebuttals. The staff will act as the program committee and determine a final score. Our overall goals in this process are:

- To motivate you to work seriously on the problems and learn something about the machine learning material in the process

- To engage you in thinking critically and learning from other students' solutions to the problems

We will arrange to give full credit to anyone who submits a serious and careful solution to the problems and who gives evidence of having read carefully the solutions they were assigned and who writes thoughtful reviews of them.

The questions are the points that your paper should cover in order to receive full credit. Your presentation should roughly follow the order of these questions so that your reviewers can see what you're doing.