10-601: Homework 3

Due: 9 October 2014 11:59pm (Autolab)

TAs: Henry Gifford, Jin Sun

Name: ______________________________

Andrew ID: ____________________________

Please answer to the point, and do not spend time/space giving irrelevant details. You should not require more space than is provided for each question. If you do, please think whether you can make your argument more pithy, an exercise that can often lead to more insight into the problem. Please state any additional assumptions you make while answering the questions. You need to submit a single PDF file on autolab. Please make sure you write legibly for grading.

You can work in groups. However, no written notes can be shared, or taken during group discussions. You may ask clarifying questions on Piazza. However, under no circumstances should you reveal any part of the answer publicly on Piazza or any other public website. The intention of this policy is to facilitate learning, not circumvent it. Any incidents of plagiarism will be handled in accordance with CMU's Policy on Academic Integrity.

⋆: Code of Conduct Declaration

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: ______________________ (e.g. *Jane explained to me what is asked in Question 3.4*)
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: ______________________ (e.g. *I pointed Joe to section 2.3 to help him with Question 2*).

⋆: Notification

This is the handout for programming questions in homework 3, you need to download the handout for theoretical part as well. You should not submit programming assignment handout. If you have any questions, please post it on Piazza or email:

Henry Gifford: hgifford@andrew.cmu.edu

Jin Sun: jins@andrew.cmu.edu

1 Digit Classification Competition (TA:- Jin Sun)

This section will only count up to 5 extra credits for this assignment. Since finding the best neural net model is very time consuming, if you are not planning to work on this problem, **it is strongly recommended to at least read through all the text**. In this section, you are asked to construct a neural network using a dataset in real world. The training samples and training labels are provided in the handout folder. Each sample is a 28*28 gray scale image. Each pixel (feature) is a real value between 0 and 1 denoting the pixel intensity. Each label is a integer from 0 to 9 which corresponds to the digit in the image.

1.1 Rules and Grading Policy

In this competition, you can only use **neural networks**, and **the data we provided**. Using any models other than neural networks or extra data will be treated as cheating. Any machine learning toolboxes are also prohibited.

Submission site will be closed at October 9th, 11:59pm. After that, all submissions will be re-evaluated and final scores will be posted on the scoreboard. The grading policies are as follows:

- Students ranking from 1 to 5 will get 5 extra points on this assignment.
- Students ranking from 6 to 10 will get 3 extra points on this assignment.
- Students participating this competition with **a valid neural network model** and **a better-than-random-guess classification accuracy (10%)** will get 1 extra point on this assignment.

1.2 Getting Started

1.2.1 Getting Familiar with Data

As mentioned above, each sample is an image with 784 pixels. Load the data using the following command:

```
load('digits.mat')
```

Visualize an image using the following command:

```
imshow(vec2mat(XTrain(i,:),28)')
```

where $X \in \mathbb{R}^{n\times 784}$ is the training samples; i is the row index of a training sample.

1.2.2 Neural Network Structure

In this competition, you are free to use any neural network structure. A simple feed forward neural network with one hidden layer is shown in Figure 1. The input layer has a bias neuron and 784 neurons with each corresponding to one pixel in the image. The output layer has 10 neurons with each representing the probability of each digit given the image. You need to decide the size of the hidden layer.

1.2.3 Code Structure

You should implement your training algorithm (typically the forward propagation and back propagation) in 'train_ann.m' and testing algorithm (using trained weights to predict labels) in 'test_ann.m'. In your training algorithm, you need to store your initial and final weights into a mat file. In the simple example below, two weight matrices "W_{ih}" and "W_{ho}" are stored into "weights.mat":

```
save('weights.mat','Wih','Who');
```

Be sure your "test_ann.m" runs fast enough. It is always good to vectorize your code in Matlab. Please refer to "hints for fast code" post on Piazza.

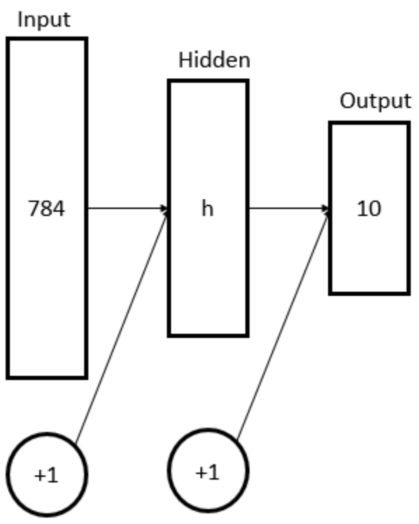

Figure 1: Feed forward neural net

1.2.4 Separating Data

'Digits.mat' contains 3000 instances which you used in previous section. The number of instances are pretty balanced for each digit so you do not need to worry about skewness of the data. However, you need to handle the overfitting problem. Neural networks are very powerful models which are capable to express extremely complicated functions but very prone to overfit.

The standard approach for building a model on a dataset can be described as follows:

- Divide your data into three sets: a training set, a validation set, a test set. You can use any sizes for three sets as long as they are reasonable (e.g. 60%, 20%, 20%). You can also combine the training set and the validation set and do k-fold cross-validation. Make sure to have balanced numbers of instances for each class in every set.
- Train your model on the training set and tune your parameters on the validation set. By tuning the parameters (e.g. number of neurons, number of layers, regularization, etc...) to achieve maximum performance on the validation set, the overfitting problems can be somehow alleviated. We will have further discussions in the following material and later in the course. The following webpage provides some reasonable ranges for parameter selection: `http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics`
- If the training accuracy is much higher than validation accuracy, the model is overfitting; if the training accuracy and validation accuracy are both very low, the model is underfitting; if both accuracies are high but test accuracy is low, the model should be discarded.

1.3 Bag of Tricks for Training a Neural Network

Okay the fun begins ...

1.3.1 Overfitting vs Underfitting

This is related to the model selection problem that we are going to discuss later in this course. It is extremely important to determine whether the model is overfitting or underfitting. The table below shows several general approaches to discover and alleviate these problems:

	Overfit	Underfit
Performance	Training accuracy much higher than validation accuracy	Both accuracies are low
Data	Need more data	If two accuracies are close, no need for extra data
Model	Use a simpler model	Use a more complicated model
Features	Reduce number of features	Increase number of features
Regularization	Increase regularization	Reduce regularization

There are other ways to reduce overfitting and underfitting problems particular for neural networks, and we will discuss them in other tricks.

1.3.2 Early Stopping

A common reason for overfitting is that the neural net converges to a bad minimum. In Figure 2, the solid line corresponds to the error surface of a trained neural net while the dash line corresponds to the true model. Point A is very likely to be a bad minimum since the "narrow valley" is very likely to be caused by overfitting to training data. Point B is a better minimum since it is much smoother and more likely be the true minimum.

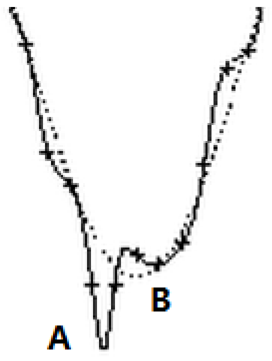

Figure 2: Minima of Neural Net

To alleviate overfitting, we can stop the training process before the network converges. In Figure 3, if the training procedure stops when network achieves best performance on validation set, the overfitting problem is somehow reduced. However, in reality, the error surface may be very irregular. A common approach is to store the weights after each epoch until the network converges. Pick the weights that performs well on the validation set.

1.3.3 Multiple Initialization

When training a neural net, people typically initialize weights to very small numbers (e.g. a Gaussian random number with 0 mean and 0.005 variance). This process is called **symmetry**

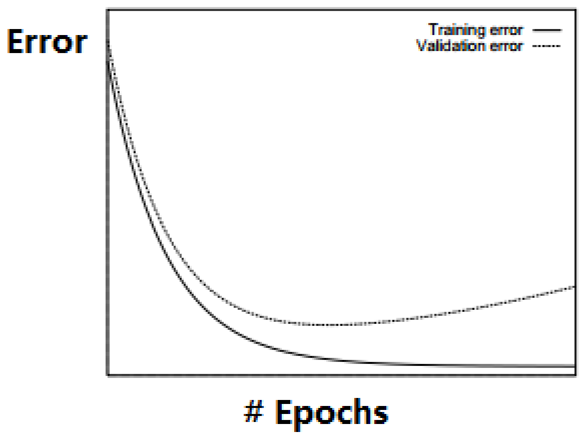

Figure 3: Training Error vs Validation Error

breaking. If all the weights are initialized to zero, all the neurons will end up learning the same feature. Since the error surface of neural networks is highly non-convex, different weight initializations will potentially converge to different minima. You should store the initialized weights into "ini_weights.mat".

1.3.4 Momentum

Another way to escape from a bad minimum is adding a momentum term into weight updates. The momentum term is $\alpha * \Delta W(n-1)$ in Equation 1, where n denotes the number of epochs. By adding this term to the update rule, the weights will have some chance to escape from minimum. You can set initial momentum to zero.

$$\Delta W(n) = \nabla_W J(W, b) + \alpha * \Delta W(n-1) \tag{1}$$

The intuition behind this approach is the same as this term in physics systems. In Figure 4, assume weights grow positively during training, without the momentum term, the neural net will converge to point A. If we add the momentum term, the weights may jump over the barrier and converge to a better minimum at point B.

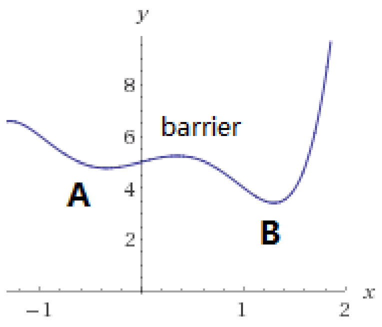

Figure 4: Local Minimum

1.3.5 Batch Gradient Descent vs Stochastic Gradient Descent

As we discussed in the lectures, given enough memory space, batch gradient descent usually converges faster than stochastic gradient descent. However, if working on a large dataset (which exceeds the capacity of memory space), stochastic gradient descent is preferred because it uses memory space more efficiently. Mini-batch is a compromise of these two approaches.

1.3.6 Change Activation Function

As we mentioned in the theoretical questions, there are many other activation functions other than logistic sigmoid activation, such as (but not limited to) rectified linear function, arctangent function, hyperbolic function, Gaussian function, polynomial function and softmax function. Each activation has different expressiveness and computation complexity. The selection of activation function is problem dependent. Make sure to calculate the gradients correctly before implementing them.

1.3.7 Pre-training

Autoencoder is a unsupervised learning algorithm to automatically learn features from unlabeled data. It has a neural network structure with its input being exactly the same as output. From input layer to hidden layer(s), the features are abstracted to a lower dimensional space. Form hidden layer(s) to output layer, the features are reconstructed. If the activation is linear, the network performs very similar to Principle Component Analysis (PCA). After training an autoencoder, you should keep the weights from input layer and hidden layers, and build a classifier on top of hidden layer(s).

For implementation details, please refer to Andrew Ng's cs294A course handout at Stanford: `http://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf`

1.3.8 More Neurons vs Less Neurons

As mentioned above, we should use more complicated models for underfitting cases, and simpler models for overfitting cases. In terms of neural networks, more neurons mean higher complexity. You should pick the size of hidden layer based on training accuracy and validation accuracy.

1.3.9 More Layers?

Adding one or two hidden layers may be useful, since the model expressiveness grows exponentially with extra hidden layers. You can apply the same back propagation technique as training a single hidden layer network.

However, if you use even more layers (e.g. 10 layers), you are definitely going to get extremely bad results. Any networks with more than one hidden layer is called a *deep* network. Large deep network encounters the "vanishing gradient" problem using the standard back propagation algorithm (except convolutional neural nets). If you are not familiar with convolutional neural nets, or training stacks of Restricted Boltzmann Machines, you should stick with a few hidden layers. Generally we accept and offer special grading for deep neural nets. However, 1) they may not work outstandingly well with small datasets, 2) they are very hard to train, and 3) they are much beyond your scope.

1.3.10 Sparsity

Sparsity on weights (LASSO penalty) forces neurons to learn localized information. Sparsity on activations (KL-divergence penalty) forces neurons to learn complicated features.

1.3.11 Other Techniques

All the tricks above can be applied to both shallow networks and deep networks. If you are interested, there are other tricks which can be applied to (usually deep) neural networks:

- Dropout
- Model Averaging

  You can find these information in coursera lectures provided by Geoffrey Hinton.

1.4 Submission

The submission should be a tar file containing "train_ann.m", "test_ann.m", "weights.mat", and "ini_weights.mat". **Please do not change file names**. Any files other than these four files will be ignored.

You should write a brief description in "train_ann.m" to illustrate all the tricks that you applied, e.g. at which epoch you stopped. Based on your description, TAs will run your training code with the initialized weights and check if the final weights match the weights you provide.

Good luck and have fun!