# Homework 3: Neural Networks and Topic Models

14 November 2017

## 1 Neural Network

Neural networks (NN) are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. They have found most use in applications difficult to express in a traditional computer algorithm using rule-based programming.

In this section, we will implement a fully-connected neural network in python from scratch using stochastic gradient. In a fully-connected neural network, every node takes input from every node in the previous layer. The weights of each node in a layer are represented as a weight matrix which is multiplied by inputs from the previous layer and fed to an activation function to produce the output of that layer. To reduce the number of gradients computed during gradient descent, we use a backpropogation paradigm in a vector/matrix style.

### 1.1 ReLU and Softmax

We first experiment with the activation function of our neural network. For our implementation, we choose the activation function for each node in the hidden layer to be ReLU i.e. f(z) = max(0,z).

The outputs from the final layer are fed into a Softmax layer for k classes( here, k=3), which maps an input of k vectors to a vector of k probabilities that add upto one. The activation function for each output is:

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

Let the loss function be cross-entropy. Therefore, for a point (x,y) the loss is,

$$l = -\sum_{i=1}^{k} y_i log(f(z)_i)$$

To incorporate softmax and cross-entropy in our implementation, our backpropogation algorithm remains the same except for the first step, when we calculate $\delta^L$ for the output layer. So, for the output layer:

$$\delta^L = D[f'(z^L)]\Delta_a loss$$

So, we incorporate the differentiation of the cross-entropy function in the $\Delta_a loss$ part and assume its input is the output of Softmax. And we include the Softmax function in the output from the neural network and consider it to be the new activation function.

### 1.2 Initialization

A good initialisation is paramount for the successs of a neural network. A good initialisation leads to a faster convergence of the network and also prevents it from getting stuck on a local minima cause if stuck on a local minima it might take the network a long to unlearn this bias and it might prevail till the end.

In our analysis, we used the Xavier initialisation technique for weight initialisation with biases initialised to 0. Note that, we cannot initialize the weights to 0 as well because then the output of each layer will be 0 as well, which means that the input of each hidden layer will be 0 and thus all nodes will receive the same signal.

To break this symmetry, in Xavier initialisation, we randomly assign the initial weights so that the initial variance of any unit's value is independent of the size of the previous layer according to a zero mean gaussian where variance is the key parameter that we need to choose wisely.

If we choose a low variance, it again leads to same problem as initializing all weights to the same value. If we choose a high variance, it might take longer to train the data and the large weights might add biases to network. Therefore according to Xavier initialization a good choice of variance $Var(W)$ is,

$$Var(W) = \frac{2}{n}, where\ n\ is\ the\ number\ of\ nodes\ in\ the\ hidden\ layers.$$

### 1.3 Regularisation

L2 regularization is perhaps the most common form of regularization. It can be implemented by penalizing the squared magnitude of all parameters directly in the objective. So, our loss function becomes,

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

where $||A||_F$ is the matrix Frobenius form.

The L2 regularization has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors. Due to multiplicative interactions between weights and inputs this has the appealing property of encouraging the network to use all of its inputs a little rather that some of its inputs a lot.

To accommodate for the L2 regularization term in our implementation, we would add a regularizing term in the final gradient calculation for each layer. So,

$$\frac{\delta loss}{\delta W^l} = a^{l-1}\delta^l + 2 * \lambda * W^l$$

### 1.4 Binary Classification

In this section we will be using our neural network to do a task of binary classificaiton. The dataset that we use is the 2D data set from Homework 2. We report our results in Table 1.

Table 1: Accuracy of Neural Network on 2D datasets

| Dataset | 1 Layer, 3 units each | | 1 Layer, 150 units each | | 2 Layer, 3 units each | | 2 Layer, 150 units each | |
|---|---|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test | Train | Test |
| 1 | 100.0 | 100 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 | 100.0 |
| 2 | 92.0 | 90.5 | 92.0 | 91.5 | 93.6 | 91.5 | 91.8 | 91.5 |
| 3 | 99.5 | 97.0 | 98.5 | 97.5 | 98.5 | 97.0 | 99.0 | 97.0 |
| 4 | 96.5 | 94.5 | 99.0 | 95.50 | 95.25 | 94.50 | 96.5 | 95.25 |

We observe that, for a given number of hidden layers, the neural network with more units per layer outperforms the one with less number of units. For, the same number of units per layer, we observe that on increasing the number of layers the testing accuracy either remains same or increases slightly.

Comparing these to the results we obtained in Homework 2, we observe that the neural net performs as good as logistic regression for Data set 1 and 3. But for Data set 2, we observe that the NN outperforms the LR by 10% and in data set 4, it outperforms LR by 40%. We observe the similar pattern between the neural network and SVM.

### 1.5 Multi-class Classification

Now, we use our neural network for digit classification using the MNIST data. We trained our network for 200 normalised samples of each digit. The results we get are mentioned in Table 2.

Table 2: Accuracy of Neural Network on MNIST datasets

| | $\lambda = 0.001$ | | $\lambda = 0.01$ | | $\lambda = 0.1$ | | $\lambda = 1$ | |
|---|---|---|---|---|---|---|---|---|
| | Train | Test | Train | Test | Train | Test | Train | Test |
| 1 Layer, 3 units each | 94.06 | 93.3 | 99.11 | 98.01 | 100.0 | 98.7 | 83.45 | 83.33 |
| 1 Layer, 150 units each | 97.75 | 97.4 | 98.3 | 96.2 | 100.0 | 98.54 | 82.50 | 83.60 |
| 2 Layer, 3 units each | 92.4 | 90.8 | 99.4 | 97.85 | 100.0 | 96.6 | 81.90 | 83.10 |
| 2 Layer, 150 units each | 98.0 | 97.6 | 99.7 | 97.3 | 100.0 | 98.40 | 100.0 | 83.01 |

# 2 Convolutional Neural Network

In this section, we will experiment with a convolutional neural network implemen- tation to perform image classification. The dataset we will use for this experiment 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.

Convolutional architecture typically employed for image recognition problems contain convolutional layers applying filters to the image, and producing 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 create 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 then saved in a file.

## 2.1 Convolutional filter receptive field

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, we connect each neuron to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). So,The "receptive field" of a "neuron" in a layer would be the cross section of the previous layer from which neurons provide inputs.

For example, assume that we 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. The receptive field for a node in $Z_1$ is 5X5. The receptive field for a node in $Z_2$ is 7X7.

The basic idea of a CNN is to extract local features and then combine them to make more complex patterns. That translates to local transformations and therefore the idea of receptive fields. The deeper we build our network the more non-linearity we get. Also for a 7X7 one layer deep CNN we would require $\propto 49$ parameters whereas for a two layer deep CNN with the same receptive field we would require $\propto 25{+}9 = 34$ parameters. Therefore, we require less parameters for the same receptive field and we get more non-linearity in our model. Therefore, it is better to build a deeper CNN.

## 2.2 The convolutional net

We use the staff provided implementation of the CNN. This implementation has the following architecture:

**(a)** The CNN has 4 layers. The first 2 layers are convolutional whereas the last 2 layers are a fully connected one. Both the convilutional layers use a filter size of 5 pixels with a stride of 2 pixels. The first fully-connected layer contains 64 units. The second fully-connected layer consists of units equal to number of output labels i.e. 11.

**(b)** All nodes in the convolutional as well as the first fully connected layers use ReLu as the activation function. The second fully connected layer doesnot use any activation function.

**(c)** The loss function used to train the network is the cross entropy loss summed over all the points after calculating the softmax probabilities for them.

Let, $a_i^n = \frac{e^{z_i^n}}{\sum e^{z_j^n}}$

Then $L = -\frac{1}{N}\sum_{n,\forall i} y_i^n log(a_i^n)$

**(d)** We use gradient descent with learning rate 0.01 to optimize our model by minimizing the loss.

The training accuracy that we achieve using these parameters and architecture is 94.2% and the validation accuracy is 62.1%. It took 30.8 seconds to train the CNN. The training accuracy tells us how many points in our training data set our fully-trained model can correctly predict. The validation accuracy helps us determine if we overfit our model to the training data. Since, the training accuracy is much larger than the validation accuracy, we can say that we might have overfitted the model.

## 2.3 Network Features

In this section we will be experimenting with some common techniques used to boost CNN performance in practice. Many of these will have some regularization effect (among other effects) which is important on such a small dataset.

### 2.3.1 Early Stopping

In early stopping, we stop training our model after our validation accuracy starts to plateau or decrease (so that we do not overtrain our model).

We observe that if we keep training our model for the default number of steps given in the implementation i.e. 1500, we achieve a training accuracy of 88.1% and a validation accuracy of 50.6%. The validation accuracy peaks near step 900, where we get a training accuracy of 88.8% and a validation accuracy of 62.1%.

We retrain our models for 1500 steps and 900 steps. For 1500 steps, we get a testing accuracy of 47.1% whereas for 900 steps, we get a testing accuracy of 52.9%. Although, this not a huge difference, this result was as expected since parameters with a higher validation accuracy performed better on test set as well.

#### 2.3.2 Pooling Layers

We will now add max pooling layers after each of our convolutional layers. We will switch between two pooling layers POOL1 and POOL2. We will try different combinations of parameters like filter size and stride for both pooling layers, also turning off one of them for a few combinations.

Table 3: Accuracy of different Pooling layers

| POOL1 filter size | POOL1 stride | POOL2 filter size | POOL2 stride | Training Accuracy | Validation Accuracy | Test Accuracy | Time |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 75.5 | 50.6 | 48.3 | 21.4 |
| 2 | 1 | 2 | 1 | 89.9 | 59.8 | 55.2 | 32.8 |
| 3 | 1 | 3 | 1 | 94.6 | 64.4 | 55.2 | 33.2 |
| 3 | 2 | 3 | 2 | 75.5 | 58.6 | 49.4 | 20.9 |
| 4 | 2 | 4 | 2 | 75.1 | 67.8 | 57.5 | 21.8 |
| 5 | 2 | 5 | 2 | 66.4 | 64.4 | 51.7 | 23.6 |
| 5 | 1 | 5 | 1 | 78.7 | 63.2 | 56.3 | 35.5 |
| 5 | 3 | 5 | 3 | 86.6 | 52.9 | 47.1 | 34.6 |
| 6 | 1 | 6 | 1 | 72.2 | 54 | 59.8 | 34.9 |
| 7 | 1 | 7 | 1 | 71.1 | 62.1 | 56.3 | 34.7 |
| 3 | 1 | 5 | 1 | 83 | 60.9 | 59.8 | 33.3 |
| 0 | 0 | 5 | 1 | 83.3 | 57.5 | 52.9 | 30.3 |
| 0 | 0 | 3 | 1 | 93.9 | 62.1 | 56.3 | 30.2 |
| 3 | 2 | 0 | 0 | 77.6 | 56.3 | 54 | 23.1 |
| 3 | 1 | 0 | 0 | 96 | 60.9 | 48.3 | 33.6 |
| 4 | 1 | 0 | 0 | 98.9 | 66.7 | 56.3 | 33.2 |
| 5 | 1 | 0 | 0 | 87.7 | 49.4 | 54 | 32.8 |
| 5 | 2 | 3 | 1 | 83 | 59.8 | 52.9 | 23.6 |
| 5 | 1 | 3 | 2 | 86.3 | 62.1 | 56.3 | 34.9 |
| 3 | 1 | 5 | 2 | 79.1 | 56.3 | 56.3 | 31.6 |

As we can observe from the table, the combination that gives the best accuracy is the filter size 6 and stride 1 for both the pooling layers. Filter size 3 and stride 1 for the first pooling layer and filter size 5 and stride 1 for the second pooling layers also give similar results.

#### 2.3.3 Batch Normalisation

Batch normalization can be added after a network layer to normalize the activations before passing them to the next layer. This safe gaurds against internal covariate shift allowing for larger learning rates and much faster training times. We apply batch normalisation to the data after each convolutional layer and after the first fully-connected layer.

We observe that the training accuracy achieved by the CNN without batch normalisation is 98.6% and validation accuracy is 63.2%. With batch normalisation, the CNN achieves a training accuracy of 99.6% and a validation accuracy of 65.5%. Thus our results support our claims.

### 2.4 Testing on variations of data

To optimize out model accuracy, we use the CNN with 1500 steps and two pooling layers, each with filter size of 6 pixels and stride 1 pixel with batch normalization applied to the output after each layer. This helps us achieve a training accuracy of 99.3% and validation accuracy of 72.4%.

We obtain the results mentioned in Table 4 when we test this architecture on distorted versions of the images.

- We get similar accuracies for normal, high contrast and low contrast images. This shows us that our network is almost invariant to changes in the contrast of the image. This might be because the only thing changing is the color distribution of the image but the relative RGB value of each pixel still remains the same.
- We lose some accuracy on brightening or darkening the image. This was expected since to increase the brightness of each pixel we must add constant to its RGB values which will make the CNN less accurate.
- We also lose some accuracy on translating and flipping the image. This was expected since now most parts of the image are at a different location then they were at during training. But since our neural net also uses features like

Table 4: Accuracy on variations of data

| Transformation | Accuracy |
|---|---|
| Normal | 66.7 |
| Translated | 42.5 |
| Brightened | 49.4 |
| Darkened | 52.9 |
| High Contrast | 66.7 |
| Low Contrast | 58.6 |
| Flipped Upside-down | 47.1 |
| Colors Inverted | 8.0 |

color distribution to recognize the image and the color distribution remains the same after translation, we are still able to recognize the images.

- Inverted the colors almost leads the CNN unable to recognize the images. This informs us that our network highly depends on the color distribution in the images to recognize them.

Data Augmentation makes our model more invariant to certain transformation since we are already training on augmented versions of the images. Batch Normalisation also helps our CNN be more robust again changes in the brightness and darkness of the images. Finally, max pooling helps our CNN stay invariant to translations in the images.

# 3 Topic Models

In this section, we will be performing an unsupervised learning task using the latent Dirichlet allocation (LDA) model to perform topic modeling on a corpus of text.

On a high level, LDA views documents as being generated from a mixture of latent topic variables that have an associated distribution over word probabilities. When learning these topics LDA further assumes a sparse Dirichlet prior for these distributions which models the fact that documents generally cover a small number of topics which in turn are primarily associated with a small number of words.

## 3.1 Training with 100 topics

We learn an LDA model with 100 topics over 19997 articles for 20 categories taken from the Usenet newsgroup collection. We use the sklearn library function to implement and train the LDA.

After training our LDA with 100 topics, a few of the topics generated by it are:

- Topic 3: team hockey players player nhl play teams league canada cup bob ice best fans north sweden candida ticket expansion said
- Topic 9: printer font print fonts copies postscript laser resolution adobe truetype tt dpi outline atm darn hawk 300 diff outlines maker
- Topic 15: israel people israeli peace lebanese zone poor cultural think occupied large education try numbers just group nation northern want kids

We notice that Topic 3 is probably about hockey, Topic 9 is probably about printers and Topic 15 is about Israel. We see that most of the words in a topic are relevant to that topic. Therefore, from a qualitative point of view, we can say that our LDA does a good job of generating relevant topics.

## 3.2 Exploring the topics

Now we will choose two of the 20 article categories and report the top three most represented topics in each of them, by averaging the topic distributions of the documents in the category.

First we train our model on the category soc.religion.christian. The top three most represented topics we get are:

- Topic 1: god people jesus does believe say think christian don life
- Topic 2: don like just think time good know way make really

- Topic 3: church catholic pope ex st texas orthodox canon mass council

Next, we train our model on the category comp.os.ms-windows.misc. The top three most represented topics we get are:

- Topic 1: use windows program software using version available output set does
- Topic 2: window widget lib return x11 contrib r5 mit xt event
- Topic 3: card db memory video bus monitor ram pc board drivers

We can see that the top three topics from these category give a good overview of most of the common words used in the articles of that category. We also notice that Topic 2 in soc.religion.christian doesn't make very sense or does not provide any information specific to the category. This might be expected since it is not always possible to avoid garbage/stop words.

### 3.3 Variation in Topics with parameters

Now we retrain our LDA model while changing the parameters such as the number of topics with which the LDA trains.

Since LDA extracts the topics in a way similar to clustering and also does that in an unsupervised fashion, the parameters of the model largely affect its accuracy. One measure that we can use to determine the optimum parameters is to look at the log-likelihood of the trained model.

Table 5 summarizes the number of topics extracted versus the corresponding log-likelihood. We can observe from the data that the likelihood is maximum for 40 topics and decreases if we increase or decrease the number of topics.

Table 5: Likelihood for different number of topics

| Number of Topics | Log-Likelihood | Number of Topics | Log-Likelihood |
|---|---|---|---|
| 10 | -7510838 | 60 | -7455195 |
| 20 | -7483766 | 70 | -7448499 |
| 30 | -7483766 | 80 | -7520075 |
| 40 | -7428673 | 90 | -7476186 |
| 50 | -7431716 | 100 | -7516023 |

Some of the topics generated when we ran our model with topic size 10 are:

- Topic 2: said went didn got year home came hit left took saw started going time armenians told runs right later
- Topic 3: 00 10 game team 25 15 11 12 20 games 16 14 13 play season 18 17 30 hockey league
- Topic 4:armenian turkish war armenians jews people health medical russian 000 turkey government population turks

The results obtained when we ran our model with topic size 40 are:

- Topic 1:boxes tyre vice dialog com1 decenso ink cartridge cartridges versa com3 partition aix greek com2 latin
- Topic 2: armenian armenians turkish jews turkey russian war armenia turks killed genocide jewish muslim soviet
- Topic 3: team hockey play league nhl season players new pittsburgh teams san points gm period montreal cup goal

The results obtained when we ran our model with topic size 100 are:

- Topic 3: team hockey players player nhl play teams league canada cup bob ice best fans north sweden candida ticket
- Topic 9: printer font print fonts copies postscript laser resolution adobe truetype tt dpi outline atm darn hawk 300
- Topic 15: israel people israeli peace lebanese zone poor cultural think occupied large education try numbers just

As we can see, that the words in the topics make most sense when the topic size is 40 and start becoming more general or vague as we move further away from that topic size.