Neural Networks Tutorial
COMS3007

Benjamin Rosman

April 29, 2018

Use your notes and any resources you find online to answer the following questions. This does not need to be submitted.

1. Consider a neural network with three input nodes, one hidden layer with two nodes, and one output node. All activation functions are sigmoids $\sigma(\cdot)$. The initial weights are as follows (including bias nodes which take the form of $x_0 = 1$):

$$\Theta^{(1)} = \begin{bmatrix} 1 & -1 & 0.5 & 1 \\ 2 & -2 & 1 & -1 \end{bmatrix}, \quad \Theta^{(2)} = \begin{bmatrix} -1 & 2 & 1 \end{bmatrix}.$$

   (a) Draw the network and label all weights.

   (b) Neural networks can be written out as functions of their input variables, as combinations of weighted sums and non-linearities. Write out this network in the form $f(x_1, x_2) = ...$

   (c) What is the network output for the following input data?

      i. $(0, 0)$
      ii. $(3, 2)$
      iii. $(-1, 1)$

   (d) Update the weights of the network using the backpropagation algorithm when trained on the point $(0, 0)$ with target (class) 0. Use learning rate $\alpha = 0.1$. Repeat the process for point $(3, 2)$ with target 1, and point $(-1, 1)$ with target 0.

2. As in the logistic regression tut, we will generate our own data to train a neural network. This time, we will generate data in the region $[0, 1]^2$ with a complicated decision boundary. In your favourite language, perform the following tasks:

   (a) Define the function $f(x) = x^2 \sin(2\pi x) + 0.7$.

   (b) Generate a uniform random point $(x_1, x_2) = [0, 1]^2$. Associate with this point the class 0 if $f(x_1) > x_2$, and class 1 otherwise.

   (c) Generate 100 points in this way. Plot them with different symbols for the two classes.

3. For this question, use the dataset you generated for question 2 above. We are now going to train a neural network model to classify this data.

   (a) First we need to choose an architecture for the network. This data is 2D, in x_1 and x_2. We therefore need two input parameters for the model, and one output variable. It is a classification problem, so we can choose the final activation function to be a sigmoid. We know the decision boundary is

non-linear because we made the data – otherwise we may need to visualise some of it to figure this out, and so we need at least one hidden layer. Let's use three nodes on the hidden layer, and sigmoids for all activation functions.

(b) Implement forward propagation for this network. Do this is a vectorised way, so we can generalise to different architectures. For any input vector, we need a vector of activations at every layer.

(c) Now compute the error δ at the final layer as the difference between the final activation and the target.

(d) Moving backwards through the layers, compute the δ of each layer (in this case this would just be for the hidden layer).

(e) Given the activations and δs, compute the gradients for each parameter.

(f) Finally, perform a weight update. Use the learning rate of $\alpha = 0.1$.

(g) Repeat the update in a loop. Technically there are two nested loops: for each epoch (iteration of the outer loop) we run through all our data points and update the weights. We then usually use two terminating conditions on the outer loop: the first is to look at the normed difference between the parameter vector between two successive iterations and we terminate when this is small, i.e. $||\theta_{new} - \theta_{old}|| < \epsilon$, with $\epsilon = 0.05$. We also usually set a maximum number of epochs, e.g. 1000, just in case. Run the learning until convergence. What is the error on the training data?

(h) Generate 100 more datapoints from the procedure in question 2. This will be our validation data. Classify them using your trained model and tabulate the results in a confusion matrix. Compare the error on the training data to the error on the validation data. What do you notice?

(i) Change the hyperparameters. Before, we considered changing the learning rate α and termination threshold ϵ. Now try add another node or two to the hidden layer, and even add in a second hidden layer. For each different setting of the hyperparameters, retrain your model on the *training data* and evaluate it on the *validation data*.

(j) Keep the best values of your hyperparameters. Now generate 100 more datapoints. This will be our testing data. Classify them using your trained model and tabulate the results in a confusion matrix. This is the final performance of the classifier with optimised hyperparameters!

(k) Why is it important to have the three data sets: training, validation, and testing?

4. Consider the file **digits.csv** from the logistic regression tut, which contains 1797 8x8 images of hand written digits (modified from the sklearn digits dataset). Note the data here is low resolution. Each row of the file has 65 elements: the first 64 indicate the pixel value for the 8x8 pixel image, and the last element is the class number (0-9). To visualise the nth digit, reshape the nth row into an 8x8 matrix.

(a) Choose a neural network architecture for this problem with at least one hidden layer. Remember, we do multiclass classification by having an output node per class. We can then get a prediction by taking the max of these. Train your neural network classifier on a random 60% of the full dataset. Use 20% of the data to optimise the hyperparameters, and use the remaining 20% to generate a confusion matrix showing its performance. How do these results compare to the logistic regression results from last week? Are they reasonable? How accurate can you get the classifier?

(b) For fun, open an image editor and draw examples of 8x8 digits by hand. Preprocess these so that the background values are 0, and pixel values range up to 16. Reshape the images into 1x64 vectors. Feed these into your classifier, and see if it correctly labels the images.

5. As we saw in class, neural networks can be used to learn boolean functions. Design a neural network (choose the architecture, weights and activation functions) to take two inputs x_1 and x_2, and the output

the XOR function, i.e. $x_1 XOR x_2$. Recall that this function has output 1 when the inputs differ. It may be useful to remember that this is the function that was shown to not be able to be modelled by a single-layer perceptron. Hint: use your notes to design the two separate pieces of the network, and then combine them.

6. A hospital manager wants to predict how many beds will be needed in the geriatric ward. She asks you to design a neural network for making this prediction. She has weekly data for the last five years that cover:

   - The number of people in the geriatric ward each week.
   - The weather (average day and night temperatures).
   - The season of the year (spring, summer, autumn, winter).
   - Whether or not there was an epidemic on.

   Design a suitable neural network for this problem, considering how you would set up the input data and the output data, as well as the preprocessing of the data. How many hidden layers and nodes in each layer would you choose? Do you expect the system to work?