Neural Networks Tutorial 2
COMS3007

Benjamin Rosman, Devon Jarvis

Instructions: Use your notes and any resources you find online to answer the following questions. You need to submit a PDF of your answers on Moodle. Make sure all your names and student numbers appear on the document!

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) What is the network output for the following input data?
      i. $(0, 3 - 1)$
      ii. $(1, 2, 1)$
      iii. $(-1, 1, 2)$

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