EAI 320
Practical Session 3

Presented By: Michael Teles

Original: Hugo Coppejans and Johan Langenhoven

27 February 2020

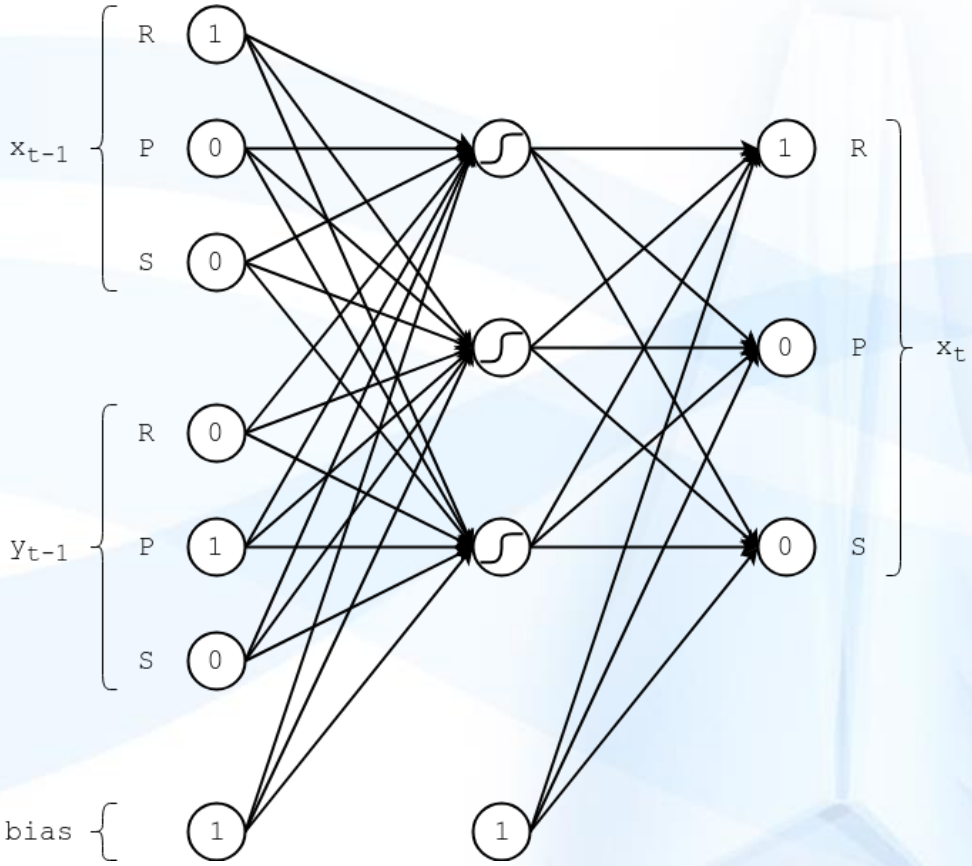
Today's focus

- Feedback
- Neural Networks
- Implementation
  - Checklist
- Gathering results
- Practical 3
  - Check online for the guide.
  - Neural Networks

Practical Feedback

- Comments.
- Overcomplication.
- Rubric.

RPS Neural Network Example



Implementation – Input layer

- For this subject, we will always use all of the available variables/features. Make sure that you always use all the data available
  - PS : In practice there are multiple method to determine if a variable is redundant or useless in a given NN. This is however beyond the scope of the work.
- For a neural network to function properly, all the input values need to be normalized to the same range. This is to insure that each variable has an equal shot at contributing to the final output value early on. This also reduces the time to train a NN
  - Thus it eliminates the effect where a very large value can dominate the NN early on, before the weights have updates sufficiently.
- For this subject we use a fully connected NN. That means that every single neuron has all the previous layers neurons/outputs as its own inputs.
  - There are multiple other types of NN that are not fully connected. Each has its own purposes and specifics. When creating a generalized NN (thus it can be used for any type of data) we use a fully connected NN because we assume no specific knowledge of the data

Implementation – Input layer

- The bias value is used to adjust the positioning of the activation function (sigmoid, tanh etc.) This is critical to the NN process, as it allows the activation function to fit the data better. Remember that the Bias value itself also has a weight, and needs to be changed using backpropagation.
  - Note that you can use a just about any bias value (except 0), but it is good practice to choose it as either your maximum or minimum value in the normalized dataset. Thus if your data is normalized between [0,1], you would use Bias = 1. If your normalized data is between [-1, 1], you can use either Bias = 1 or -1.

Implementation - Checklist

Input layer	
Did you use all the input variables	
Did you normalize the data (or make sure it is already)	
Make sure you have a fully connected NN	
Did you add a bias value	

Implementation – Hidden layer

- For this practical we will be using at least one hidden layer. The number of hidden layers that a NN should have is a hotly debated issue. There is no consensus on how many a generalized NN should have, but many studies indicate that the number has very little to no effect on the output.
  - There are some NN applications where multiple hidden layers are advantageous, but then again there are some where there are no hidden layers at all.
- The weights within a neuron is associated with each individual input into that neuron. Thus every single input (including the bias) will have a unique weight associated with it.
  - Each weight will be updated using the previous value, as well as the input value. Remember also that weights can be negative numbers.

Implementation - Checklist

Hidden layers	
Decide on the number of hidden layers you require	
Do you have a weight for every input at each neuron (including bias values)	
Initialize each weight individually using a random function	
Choose an appropriate activation function, and apply it to EVERY neuron	
Remember to save the summed value of each neuron **before** and **after** the activation function is applied	
Remember to also add a bias value that feeds to the next layer	

Implementation – Hidden layer

- When the NN is initialized, the each weight is individually (and separately) created. Every single weight should be randomly generated. There is no real rule that dictates the range that a weight should be initialized as. One that has however become popular is the following:

$$\frac{-1}{\sqrt{hidden}} < w_{ij}^{k} < \frac{1}{\sqrt{hidden}}$$

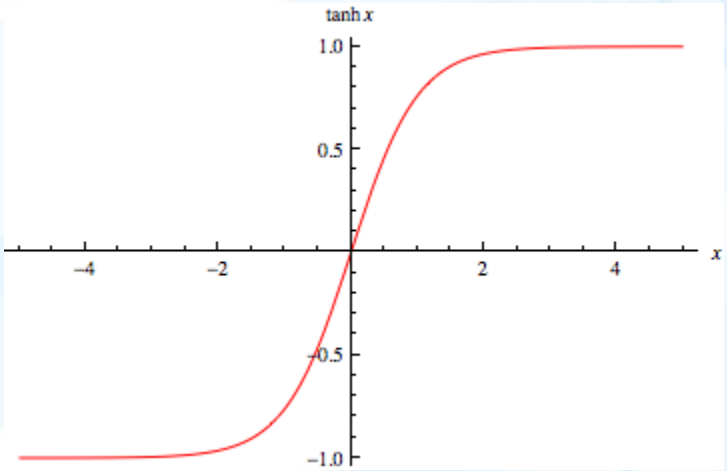
where *hidden* is the number of neurons in the hidden layer.

- This specifies a range that the weights should be random in. Make sure to use a uniform distribution for the RNG.

Implementation – Hidden layer

- Choose the activation function you want to use. This should correlate with the range that your data was normalized in. There are some activation functions that allow for an easy derivation.

  For instance the hyperbolic tangent (tan**h** (x)) is an excellent choice when your data lies between [-1 , 1] but it also works well for [0,1].



$$\frac{d}{dx}\tanh(x) = 1 - \tan^2(x)$$

Implementation – Output layer

- The type of output will differ between different implementation. Each implementation can be considered to have a unique goal, and that goal will determine what type of output is required. A NN can be used for regression and classification.
  - Note however that a NN will output an actual numerical value. Thus if we wish to classification between (example) TRUE and FALSE, we need to assign a numerical value to these classes. Thus TRUE=1 and FALSE = 0. Then we train the NN with these numerical values, and it will output a numerical value that we then have to classify (easiest way is use a step function at 0.5).

    Thus an output of 0.7 will equate a TRUE class, while 0.499 will equate a FALSE class (however there is a very high level of uncertainty then)

Implementation - Checklist

Output layers	
Determine what type of output is required (this will determine the number of output neurons)	
Do you have a weight for every input at each neuron (including bias values). Remember the inputs for the output layer comes from the hidden layer	
Initialize the weights for the output neurons (same as hidden layer)	
Remember to fit your activation function with the desired output (check prac guide)	
Determine what type of error function you will use.	
There's no bias value (because there are no more layers after this)	

Backpropagation

- For the error function, the squared error should be used. Remember however that if there are multiple outputs, you have to consider each error individually.
  - When reporting on the error (thus giving results), use the mean squared error (MSE) to portray the results.
  - On the class notes, you will notice that the squared error is used to derive all the following equations.

- Note what the derivation of the activation function is, and make sure you use the correct derivative at each layer. For instance the activation function might be different between he hidden layer and the output layer.
  - Making a mistake with this can severally effect your NN performance.

Backpropagation

- Remember when updating your weights, you only change the old weight by a little bit. The learning rate will determine how much of a change will occur on the weight.
  - Weights can potentially have negative values.
  - Remember to always use the old weights when doing backpropagation. Thus do not use the newly calculated weights at the output layer to calculate the new values for the hidden layer.

Backpropagation - Checklist

BackProp	
Calculate the error using the chosen error function (make sure you double check this, there are a few that can be chosen from).	
Make sure that you use the correct derivative of the activation function at each point. Remember that the activation function might not be the same over all the layers.	
Make sure to use the correct weights and input values at each stage	
Remember to update the weight associated with the bias value	
Remember that the new weight is only a slight alteration of the old weight.	
Do not make the learning rate too high or low.	

Simulation procedure

- Use the training data to actually train your NN. This data will be fed through the NN, and error will be calculated, and backpropagation will be used to alter the weights.
  - There are different methods of learning using the full dataset. For this practical you should update the weights for each individual data sample.
- The testing dataset will solely be used to calculate the performance of your NN. It should NOT be used to alter any weights in the NN. The reason for this is that you do not overtrain your NN
  - Remember to use the mean squared error to present your NN performance.
- You have to run multiple epochs for your NN. A single epoch is when you run the entire training dataset through your NN once. Thus if you have run through your dataset four times, you have run 4 epochs.
  - This is the standardized way of measuring how fast/slow your NN trains.

Simulation procedure - Checklist

Simulations	
Use the training dataset to change the weight values of all the neurons	
Use the testing dataset to measure your actual NN performance.	
Remember you have to run multiple epochs to train your NN	
Check that you do not overtrain/overfit the data	
Due to the random nature of the weights, you have to run each simulation multiple times. A good number would be 10 – 20 iterations	
Report on the average error, the classification error and also give the standard deviation.	

Simulation procedure

- During training, the training dataset is used to adjust the weights of the neurons to generalize over the training set. Because we keep on using the same dataset, it is possible that the NN will start to overfit the data. What this means is that the NN is starting to match to the training data too much, thus losing the ability to generalize. This is a natural occurrence with NN, and it will always happen when you keep on training on the same data.
- To prevent this, we need some type of stop criteria. There are multiple stop criteria that can be considered.
  - You can stop the NN when it reaches a certain number of epochs. This method is considered as the worst one available.
  - You can stop your NN after a certain error rate has been achieved. This method is better than the previous only if the goal error rate is known before hand.
  - You can stop your NN using the testing dataset. This entails calculating the average error over each epoch. Once the error starts going up, you know you are overfitting the data.

Practical assignment 3

- The Practical guide will be posted on ClickUP.