# Individual Assignment

## Network Analysis

## Hans Alberto Franke

## Feb/21

**Individual Assignment questions**

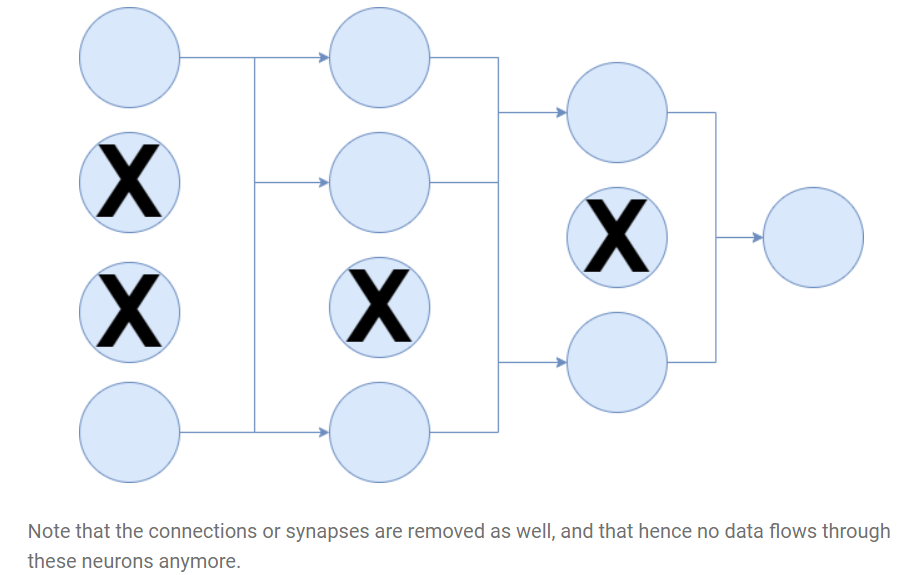
The questions below are related to the group assignment and lectures but are not suitable for group work. Please answer these questions individually in writing, and submit the result on Blackboard by the end of March 9th.

## Describe the principles of overfitting and how dropout can reduce this (Question 1, 5 points)

### **Answer**:

**Overfitting,** being too sensitive to your training data. In other words, your model predict “too good” in training set, but not so well in real data or even in test set. Both underfitting and overfitting are to be avoided, as your model will perform worse than it could perform theoretically. Fortunately, certain techniques – called regularizes – can be used to reduce the impact of overfitting. **Dropout** is one of them

With Dropout, the training process essentially drops out neurons in a neural network. They are temporarily removed from the network, which can be visualized as follows:



Source: <https://www.machinecurve.com/index.php/2019/12/16/what-is-dropout-reduce-overfitting-in-your-neural-networks/>

This removal of neurons and synapses during training is performed at random, with a parameter p that is tunable (or, given empirical tests, best set to 0.5 for hidden layers and close to 1.0 for the input layer). This effectively means that, according to the authors, the “thinned” network is sampled from the global architecture, and used for training.

At test time, “it is not feasible to explicitly average the predictions from exponentially many thinned models” (Srivastava et al., 2014). That’s true: it would become a computational burden when hundreds of thousands of epochs/minibatches have to be averaged, especially when networks become really large.

Fortunately, there is a solution – which is simple, but produces the same result. By using one neural network, where the weight outputs are scaled down according to the p with which a unit was retained during training. This means that the expected output at training time is the same as the true output at test time, resolving the computational issue and making Dropout usable in practice.

Computing the gradient is done with respect to the error, but also with respect to what all other units are doing (Srivastava et al., 2014). This means that certain neurons, through changes in their weights, may fix the mistakes of other neurons. These, Srivastava et al. (2014) argue, lead to complex co-adaptations that may not generalize to unseen data, resulting in overfitting.

Dropout, then, prevents these co-adaptations by – as we wrote before – making the presence of other hidden [neurons] unreliable. Neurons simply cannot rely on other units to correct their mistakes, which reduces the number of co-adaptations that do not generalize to unseen data, and thus presumably reduces overfitting as well.

**Dropout** prevents **overfitting** due to a layer's "over-reliance" on a few of its inputs. Because these inputs aren't always present during training (i.e. they are dropped at random), the layer learns to use all of its inputs, improving generalization.

--------------------------------------------------------------------------------------------------------------------------------------

Read the research paper “Performance-optimized hierarchical models predict neural responses in higher visual cortex”, available from:

<http://www.pnas.org/content/pnas/111/23/8619.full.pdf>

## Write a short (~500 word) summary of the experimental approach and results. (Question 2, 10 points)

**Answer:**

The article compares a neural network with biological functions of humans in task of easily recognize objects in scenes. This ability is known to be supported by a network of hierarchically interconnected brain areas. The authors looks to use computational techniques to identify a neural network that matches human performance. First, they construct a scenario considering of natural categories, the objects are putted on random natural scenes to ensure the background is uncorrelated with object. Using multiple electrode arrays, they collected responses from 168 IT neurons to each image, then using high-throughput computational methods to evaluate thousands of candidate neural network model on these images, measuring object categorization performance x IT neural predictive for each model.

CNNs approximate the general retinotopic organization of the ventral stream via spatial convolution, with computations in any one region of the visual field identical to those elsewhere. Each convolutional layer is composed of simple and neuronally plausible basic operations, including linear filtering, thresholding, pooling, and normalization. These layers are stacked hierarchically to construct deep neural networks.

Network depth ranged from one to three layers, and filter weights for each layer were chosen randomly from bounded uniform distributions whose bounds were model parameters. Models were selected for evaluation by one of three procedures: (i) random sampling of the uniform distribution over parameter space. (ii) optimization for performance on the high-variation eight-way categorization task and (iii) optimization directly for IT neural predictivity.

They evaluate the performance of the network in different difficult scenarios, from simple image position to high(p.e 180° rotations on all axes, 2.5× dilation, and full-frame translations..). The comparison was made with human performance and other networks, to proper evaluate performance and find clues where the model can be improved.

After comparison, they propose extend network performance with a mixture of Deep Neural Networks which correspond intuitively to architecturally specialized subregions like those observed in the ventral visual stream. They used hierarchical modular optimization (HMO) procedure embodies a conceptually simple hypothesis for how high-performing combinations of functionally specialized hierarchical architectures can be efficiently discovered and hierarchically combined, without needing to prespecify the subtasks ahead of time. Algorithmically, HMO is analogous to an adaptive boosting procedure interleaved with hyperparameter optimization.

To gain further insight, a exploratory analysis of the parameters of the learned HMO model, evaluating each parameter both for how sensitively it was tuned and how diverse it was between model mixture components. Two classes of model parameters were especially sensitive and diverse: (i) filter statistics, including filter mean and spread, and (ii) the exponent trading off between max-pooling and average-pooling.

The results added in neuroscience on assumption that visual neuroscience is that understanding the tuning curves of neurons in lower cortical areas will be a necessary precursor to explaining higher visual cortex, the results indicate that it is useful to complement this bottom-up approach with a top-down perspective characterizing IT as the product of an evolutionary/developmental process that selected for high performance on recognition on tasks like those used in our optimization.

---------------------------------------------------------------------------------------------------------------------------------------

It should be clear from the Group Assignment that even a relatively simple deep convolutional learning network is quite computationally intensive to run on a personal computer. Here we will move to a web-based interface for deep learning, at: http://playground.tensorflow.org/

Here, you can classify the object positions in different data sets (left panel) using deep convolutional network of differing complexity, different numbers of feature maps (number of ‘neurons’ in each hidden layer), different numbers of layers and different inputs (‘features’ column). At the top, you can also change the activation function and add normalisation (regularization). In the left column, you can change the ratio of training and test data, and add noise to the network to improve generalization to simulate imperfect inputs.

## Play around with these settings and see how they affect your ability to learn classification of different data sets. Write down what you found and how you interpret the effects of these settings. This question is intentionally open to allow you to explore the process. (Question 3, 8 points)

The website is a visual representation of a formal neural network. First as question states we can select only classification problems. Besides classification problem you can select different parameters, the same you can chose in a model on Keras for example, thus this visual is clear to see the impact of changing each one.

Definitions:

* Parameters:
  + Learning rate => The amount that the weights are updated during **training** is referred to as the step size
  + Activation function
    - is a **function** that is added into an artificial **neural network** in order to help the network **learn** complex patterns in the data. When comparing with a neuron-based model that is in our brains, the **activation function** is at the end deciding what is to be fired to the next neuron
  + Regularization:
    - As we can see from the formula of **L1 and L2 regularization**, **L1 regularization** adds the penalty term in cost function by adding the absolute value of weight(Wj) parameters, while **L2 regularization** adds the squared value of weights(Wj) in the cost function
  + Regularization Rate
  + Ratio of training and test
    - Split between train and test
  + Input Features
    - (normal or quadratic, and interaction x1 \* x2, sin() )
  + Batch size
    - How many row per batch to update weights.
* Modeling:
  + Number of hidden layers
  + Number of neurons in hidden and output layers
* Extra:
  + Noise => to simulate real input problems
* Output:
  + Loss: in train and test
  + Visual plot with predict values (and if you click the real values)

The colors of the plot represent positive (blue) or negative (orange) impact. In the hidden layers, the lines are colored by the weights of the connections between neurons. Blue shows a positive weight, which means the network is using that output of the neuron as given. An orange line shows that the network is assigning a negative weight. One can look at number of lines to understand how backpropagation are updating the weights of the network based on the loss function. If one put the mouse over a node, it shows the impact of that node on final output.

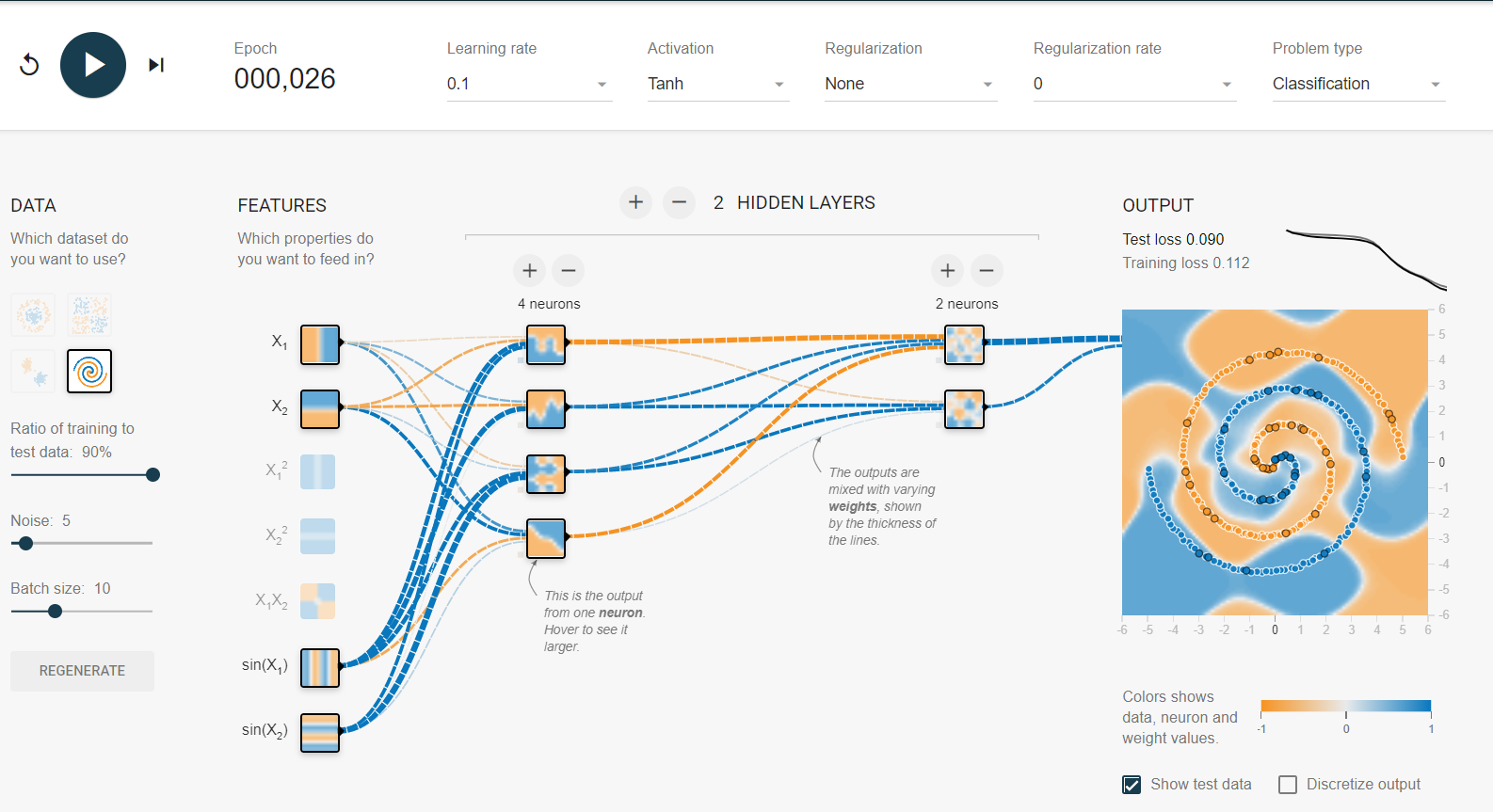
The interface of the website allow one to change parameters and simulate a network. Therefore one can compare how fast a network converge (number of epochs) to a desired loss output.

## What is the minimum you need in the network to classify the spiral shape with a test set loss of below 0.1? (Question 4, 7 points)

Activation function **Tanh** with **sin**(x1) and **sin**(x2)

**Simulate create new features**. The best features I can come up with is to transform the coordinates to [spherical coordinates](https://en.wikipedia.org/wiki/Spherical_coordinate_system) (sin). So, with ~30 epochs we achieve test loss < 0.102

* Number of neurons:
  + First layer: 4 (less than that the converge rate almost double)
  + 2nd layer: 2 (it is need, if not the converge rate takes almost more 100 epochs, 4 neurons don’t affect performance too much so avoided to turn network simpler)
* For smaller batch sizes like 1, 0.1 is too high a learning rate as the model fails to converge as it jumps around the global minima. So, if you would like to keep a high learning rate(0.1), keep the batch size high(10) as well. This usually gives a slow yet smoother convergence.
* Ratio of training to test data: 90% it converge fast and almost same overfitting than with 70% (difference from training loss to test loss)
* Noise => 5 (higher than that the model takes long to converge and have a lot of overfitting almost double loss in test)



Source:

<http://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.1&regularizationRate=0&noise=5&networkShape=4,2&seed=0.19939&showTestData=true&discretize=false&percTrainData=90&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false>

**Backpropagation of error:**

So far, we have avoided explaining backpropagation in detail. We have discussed what backpropagation does, and used Keras’s implementation to train our networks. But we have not looked at how it works because the mathematics are complex and do not fit well with the goals of this course. However, this is a major principle in machine learning with neural networks, so the final part of the assignment will attempt to explain it.

To start this part of the assignment, we suggest watching videos 3 and 4 from the playlist at the following link. You may like to start by watching videos 1 and 2, as these set things up for videos 3 and 4:

https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU

Please to answer the following question after viewing the content above:

## Explain the principle of backpropagation of error in plain English in about 500 words. This can be answered with minimal mathematical content and should be IN YOUR OWN WORDS. What is backpropagation trying to achieve, and how does it do so? (Question 5, 8 points)

One would be interested for example, what will be my grade in Network Analysis as a function of grade in individual assignment and group assignment? The algorithm needs sample dataset, for example, examples of other students. Then I can try to predict my grade (y^) with my own inputs like: my grade on individual assigment (x1) and my grade on group assigment(x2).

In any ANN we have the following items: Neurons and Weights. Neurons store the values that will be calculated to define the Weights, where these weights are the "key" to the functioning of all ANN, it is by the weight that ANN can identify that object is round and not square.

Every time you submit a input( Student 1 (individual = 5, group = 6) ) and a certain output for this (student 1 exam grade = 6), it regulates the weights (which are the lines that connect the neurons) to try to get as close as possible to the result you have. So when obe submit an input you do not know the desired result (which is the case for a new draw), it should respond as close as possible to the actual result , because he was able to learn this relation (f(y) = W1.x1 + W2.x2).

The way that ANN learn it **weights**, or solve this equation, can be expressed by gradient descend for example, or in another words, it minimizes the loss function of y for y^. What backpropagation does, it pick this loss\_function and reproduces the error backwards on network, thus it adjust the weights of layers based on their contribution on the overall error.

**Example:**

* **Inputs**:
  + Individual assigment = x1
  + Group assigment = x2
* **Hidden** Layer of 2 nodes (N1 to N4) where **W** = weights and **a** = outputs
  + W1 = link from x1 to N1 = > w1\*x1 = a1
  + W2 = link from x2 to N1 = > w2\*x2 = a2
  + W3 = link from x1 to N2 = > w3\*x1 = a3
  + W3 = link from x2 to N2 = > w4\*x2 = a4
* **Output** layer (O1) = 1 node => predicted grade or y^
  + W5 = link from N1 to O1
  + W6 = link from N2 to O1
  + SumFUNCTION (w5\*a1 + w6\*a2 ) = > **y^**
* **Error / Cost Function =** Sum squared error (y^- y) ^2 = E

Thus, E1 is used to adjust every **W** in the network based on their contribution to total error (E1). Let assume a equation like (e1\*W1, e2\*W2, ….).

This assumptions are made trying to simulate neuron of human brain, or in other words, adjusting weights so that neurons fire or not depends on the added value that they have to the network.

W1

W2

W3

W1

W5

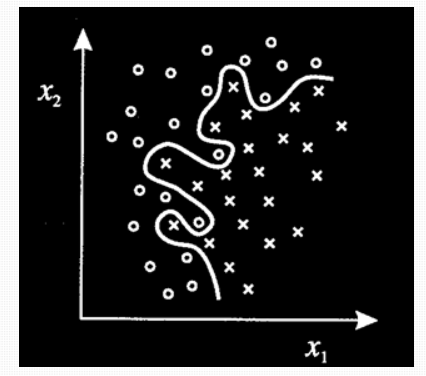
W6

Backpropagation

Feedforward Information

**Figure1**. **Backpropagation**: The sizes of yellow circles represents the contribution of each weight(W) based on the overall error

The main advantage of using Backpropagation is that it works with multilayers and solves "nonlinearly separable" problems and some algorithms do not solve. In summary, a "nonlinearly separable" problem is one where we cannot separate 2 distinct classes on the two-dimensional Cartesian axis just by tracing a line.



Another important feature is that Backpropagation is feedforward, that is, the connection between neurons is not cyclic, going from start to finish you will not find a cycle.

**BONUS QUESTION:**

If you are also attempting to answer question 6, you should also look at the page here:

https://github.com/stephencwelch/Neural-NetworksDemystified/blob/master/Part%204%20Backpropagation.ipynb

## Describe the process of backpropagation in mathematical terms. Here, explain (in English, in about 500 words) what each equation you give does, and relate this to the answers given in Question 5. You are welcome to express equations in your own R or python code rather than using equation layout, but you need to make clear you understand what each line is doing. (Question 6, 5 points).