

Training neural networks: the nitty-gritty

We know that the weights \mathbf{W}^* that minimize the average loss

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} L(\hat{\mathbf{y}}, \mathbf{y}; \mathbf{W})$$

can be found by using Back Propagation ([see the earlier module \(Training Neural Network Backprop.ipynb\)](#)) to compute the derivatives needed by the Gradient Descent algorithm.

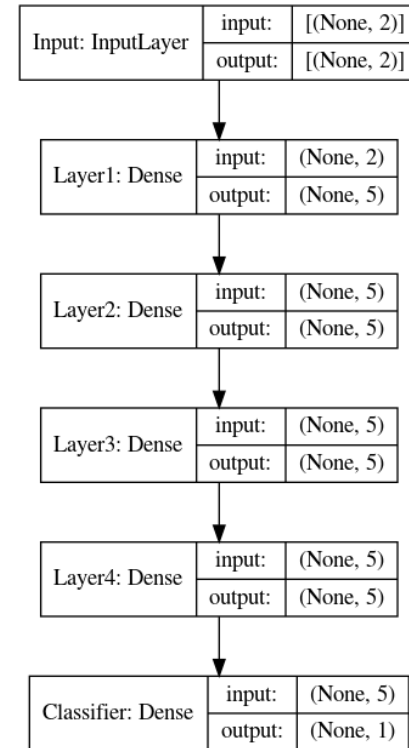
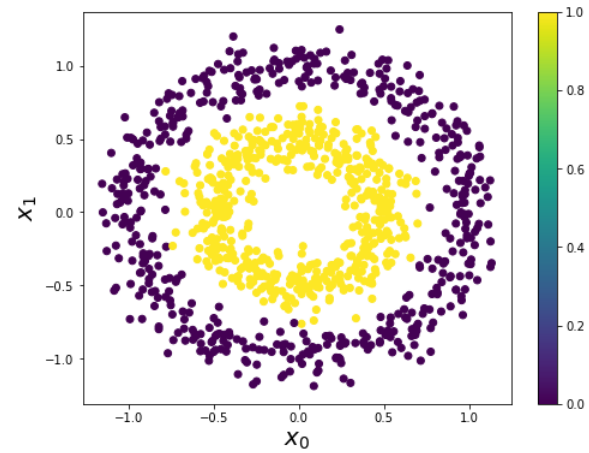
So training a Neural Network sounds simple enough.

But we also recalled several "AI Winters" in which progress in Deep Learning stalled.

History indicates that Training is more complex than it appears.

We will illustrate the complexities via a running example

- Training a binary classifier
- On the dataset shown on the left
 - Two features: x_0, x_1
 - Targets: 0, 1
- Using the Neural Network architecture shown on the right



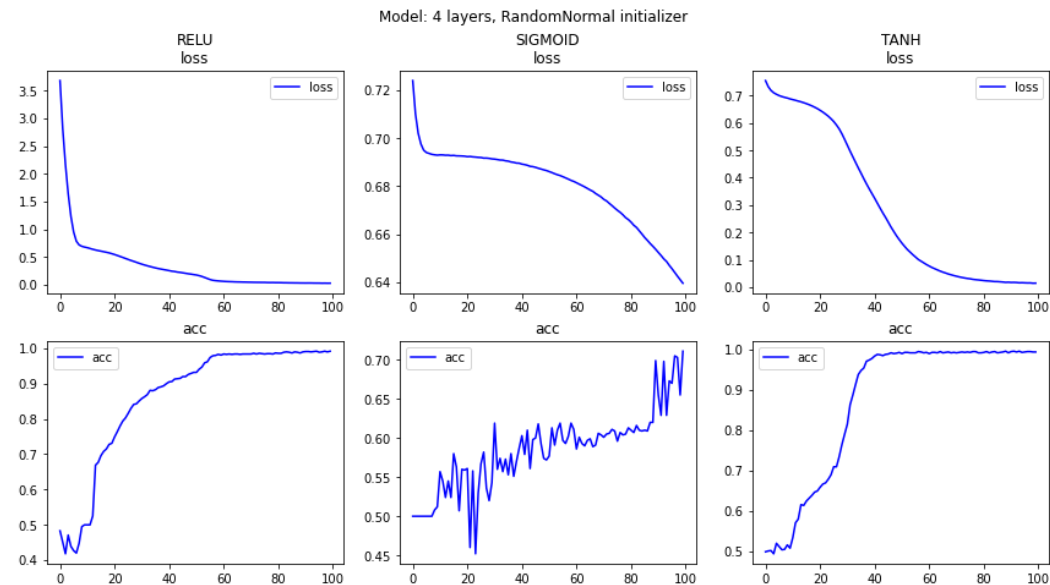
Effect of different Activation functions

Let's see the effect of choosing among ReLU, Sigmoid and Tanh activation functions

- Keeping the architecture identical
- but changing the activation functions of the Dense layers
 - the Classifier Dense layer always uses the sigmoid (as a classifier must)

We see a difference in Loss and Accuracy

- note the difference in the scale of the vertical axes between plots



Effect of weight initialization

Training is the process of discovering optimal weights/parameters for the components of the Neural Network.

- updating an initial choice via Gradient Descent

Yet, we have not specified how to initialize the weights.

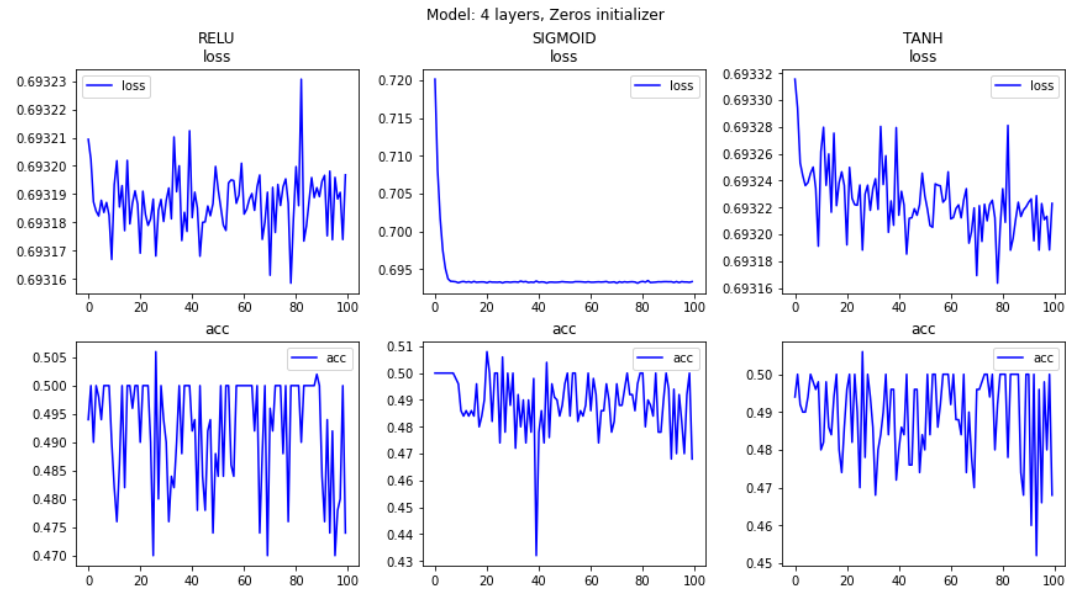
Does it matter ?

In the diagram above, we saw the Loss and Accuracy when initializing weights

- according to a Random Normal distribution with mean 0 and unit variance

Here are the identical plots when initializing weights

- to all zero



The loss and accuracy of **all** the networks is notably worse than the Random Normal initialization

- The Loss hardly changes with training: the network is not "learning"
- The Accuracy is not better than a coin flip

Understanding what happens during training

As we saw from our experiments

- choices that seem minor
 - activation function
 - weight initialization
- can have a major impact on the success of training a Neural Network.

We now spend some time investigating the causes, and solutions, to the difficulty of training networks.

Broadly speaking the issues are

- Gradients becoming zero or infinity, inhibiting learning (weight updates in Gradient Descent)
- Proper scaling of the inputs
- Initialization of learnable weights
- Making sure that the proper scaling of inputs continues to each layer, not just the input

Vanishing and Exploding Gradients

Although Backpropagation is mechanically simple, there are some mathematical subtleties.

Let's explore the problem of [Vanishing and Exploding Gradients](#)
([Vanishing and Exploding Gradients.ipynb](#)).

Initializing and maintaining layer inputs

Neural Networks are sensitive to the scale of the layer inputs.

Creating the correct situation to learn is the subject of [Scaling and Initialization](#)
([Training Neural Networks Scaling and Initialization.ipynb](#)).

Improving trainability

Apart from the mathematical issues of preventing activations and gradients from exploding/vanishing, there are many ways to make training successful.

Let's explore techniques for [Improving trainability](#)
([Training Neural Networks Tweaks.ipynb](#)).

How big should my NN be ?

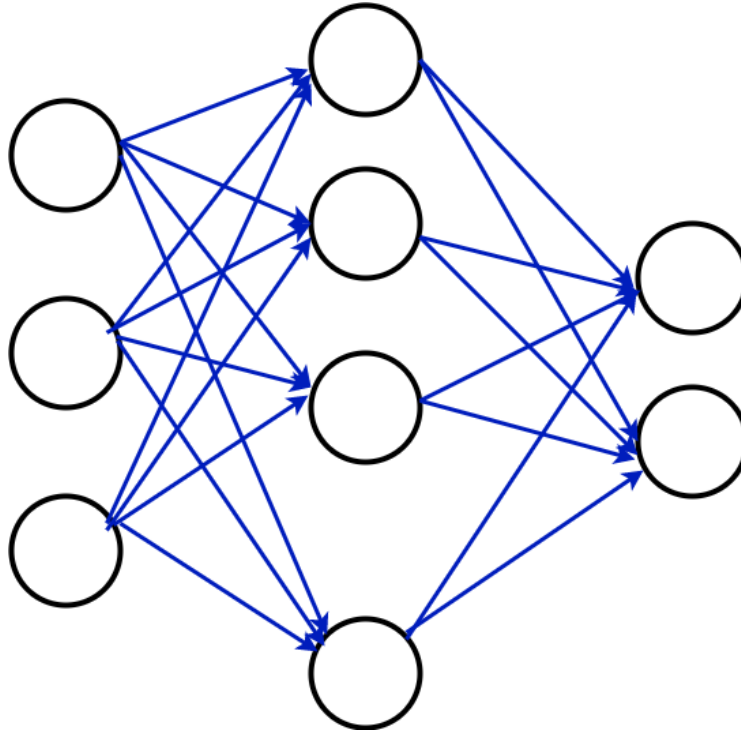
There is a paradox in building Neural Networks:

- Start off training an overly large NN (many units)
- Many units turn out to be "dead": near zero weights
- Reduce the number of units
- Can't train !

Given a fixed number of layers: it is easier to train a big NN than a small one.

"Somewhere in this big mess must be something valuable"

"Big" NN -- some nodes are dead



[The Lottery Ticket Hypothesis \(https://arxiv.org/abs/1803.03635\)](https://arxiv.org/abs/1803.03635) is an inter that addresses this issue.

For now:

- Use bigger than necessary NN's
- With regularization to "prune"

Conclusion

We sometimes take training Neural Networks for granted.

After all, Gradient Descent seems like a simple procedure.

It turns out that there are *many* subtleties.

Uncovering and solving the subtle problems were the key contributions in rapid advance of Deep Learning.

Without them, we'd still be living in "AI Winter".

