Back propagation

The key to training a Neural Network is find the weights $\mathbf{W}^*$ that minimize the average loss

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

and Gradient Descent is the tool we use.

Let's quickly review minimizing a function using Gradient Descent (Gradient_Descent.ipynb#Gradient-Descent:-Overview)

Although this minimization sounds simple, there are substantial details and pitfalls to be aware of.

Let's explore Back propagation (Training_Neural_Network_Backprop.ipynb)

Analytical derivatives made easy

In order for the magic of Gradient Descent to work, we need to compute derivatives of a function.

As explained in the first lecture on Neural Networks

- We prefer *analytical* derivatives
- To *numerical* derivatives

Numerical differentiation applies the mathematical definition of the gradient

$$\frac{\partial f(x)}{\partial x} = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

- It evaluates the function twice: at $f(x)$ and $f(x+\epsilon)$
- Is expensive and is only an approximation (exact only in the limit)

Analytical derivatives are how you learned differentiation in school

- As a collection of rules, e.g.,
$$\frac{\partial(a+b)}{\partial x} = \frac{\partial a}{\partial x} + \frac{\partial b}{\partial x}$$

This is very efficient.

Tensorflow and other toolkits implement analytical derivatives.

Let's explore the simple trick that makes this possible (Training_Neural_Network_Operation_Forward_and_Backward_Pass.ipynb)

What took so long ?

We had briefly discussed this topic in a prior lecture.

We will now dive more deeply into the multitude of reasons for why it took so long for Deep Learning to achieve success.

An historical perspective:

- Perceptron invented 1957
- Mid-1970's: First "AI Winter"
- Late 1980's: secibd "AI Winter"
- 2010: Re-emergence of AI

The promise of AI led to great expectations, that were ultimately unfulfilled. The difficulty was the inability to train networks.

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 ?

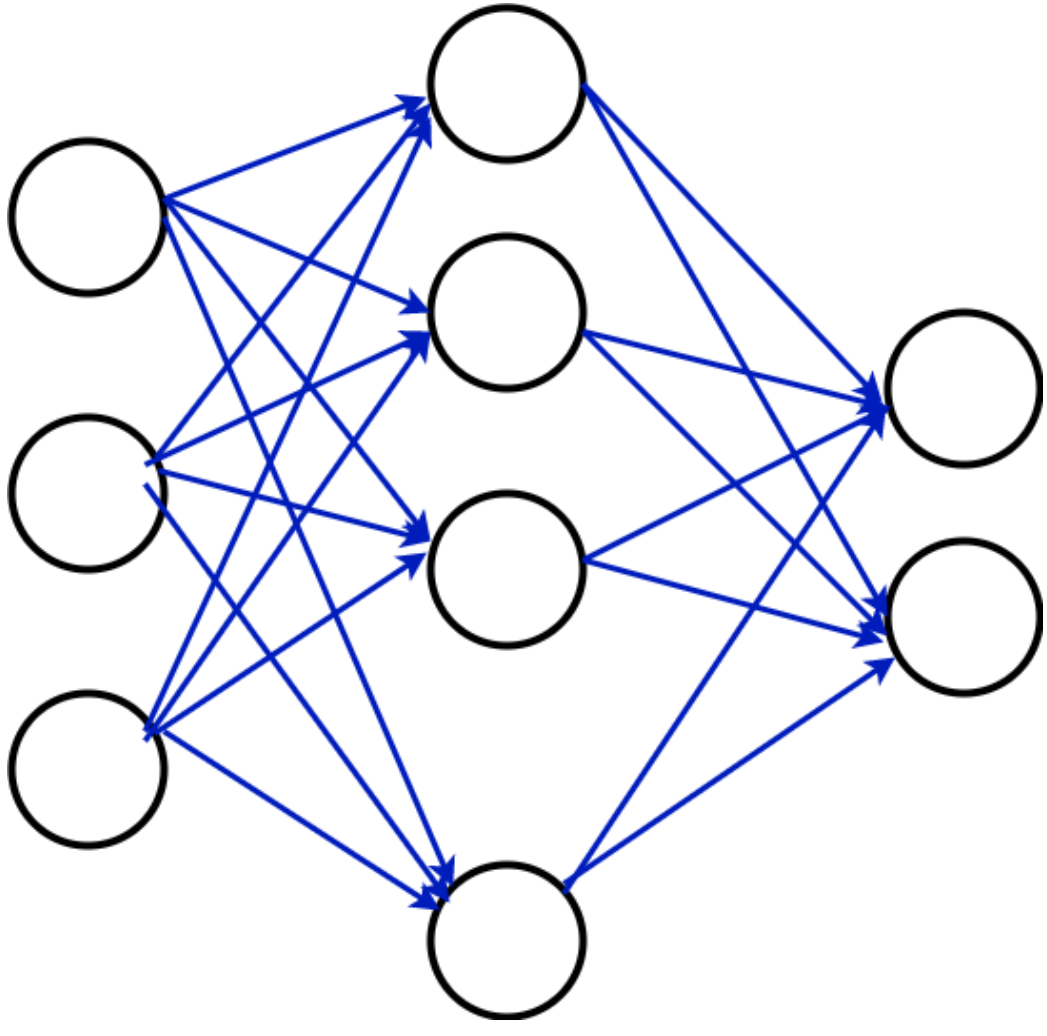
There is a paradox in building Neural Networks:

- 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 with dead nodes



The Lottery Ticket Hypothesis (https://arxiv.org/abs/1803.03635) is an inte
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 ir
rapid advance of Deep Learning.

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

In [1]:

```
Done
```

"Big" NN after dead nodes have been pruned