Back propagation

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

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

and Gradient Descent is the tool we use.

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

In [3]:

```
print("Done")
```

Done