NEURAL NETWORKS FOR REGRESSION

LECTURE 6
SECTION 1
JUNE 11TH

INSTITUTE FOR APPLIED
COMPUTATIONAL SCIENCE
AT HARVARD UNIVERSITY

UNIVERSITY of
RWANDA

MOTIVATION

COMPARING CLASSIFIERS:

We now have a large number of classifiers to choose from. Which model should we use for which application?

Besides accuracy and AUC, we also need to think about which model is more interpretable, easier to store, faster to train, faster at making predictions.

	Interpretability	Space Complexity	Train Time	Predict Time
Logistic Reg. Linear Boundary	Easy to interpret	Easy to store: w	Gradient descent is fast	Fast: $w^T x$
Logistic Reg Poly. Boundary	Harder to interpret	Easy to store: w	Gradient descent is fast	Fast: $f_w(x)$
K-Nearest Neighbours	Easy to interpret	non-parametric: needs to store all training data	No training time	slow: need to compare to every training pt.
Decision Tree	Easy to interpret if tree is not deep	need to store 2^{depth} nodes	Training is slower	slower: 2^{depth} comparisons
Random Forest	Harder to interpret	need to store large # of trees	Training can be much slower	slower: $2^{depth} \times$ # of trees comparisons

MOTIVATION FOR NEURAL NETWORKS:

If we want a model that is fast to train and takes little space to store, and we want to capture non-linear trends, then we may want to use a polynomial model.

If we choose a degree that is too small, we might underfit.

If we choose a degree that is too large, computation becomes unstable, e.g. $x^{100,000}$.

We want a model that is:

1. fast to train (uses an algorithm like gradient descent)
2. doesn't take up too much space
3. fast to compute (make predictions)
4. can approximate complex non-linear trends in a numerically stable way.

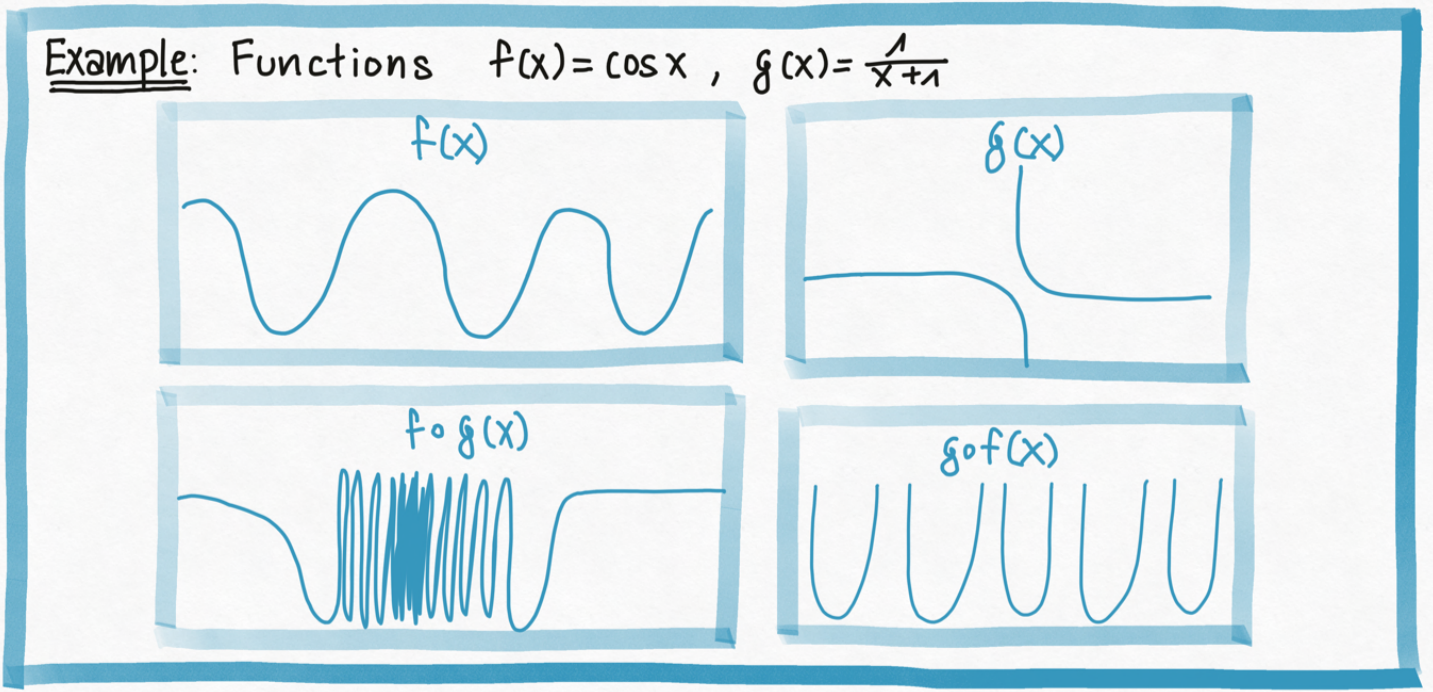
NEURAL NETWORKS

THE CENTRAL IDEA BEHIND NEURAL NETWORKS:

Building a model that is both flexible (i.e. complex) and simple to manipulate is easy if we exploit a simple algebraic fact:

The composition of two functions is often much more complex than their sum.

Example: Functions $f(x) = \cos x$, $g(x) = \frac{1}{x+1}$



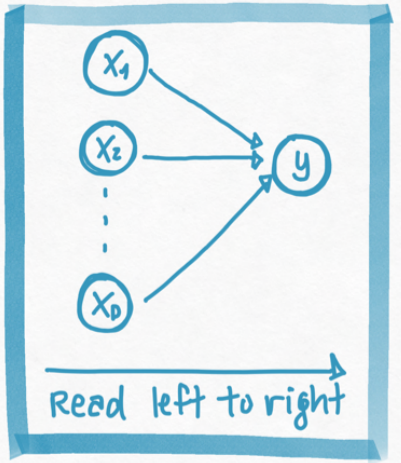
GRAPHICAL REPRESENTATION OF SIMPLE FUNCTIONS:

We will build complex functions by composing simple functions.
We will consider simple functions of the form:

$$f_w(x) = h(w^T x + b)$$

Where h is a fixed choice of a non-linear function, e.g. $h(z) = e^x$.
We call h the activation function.

We will represent our simple function as a graph:

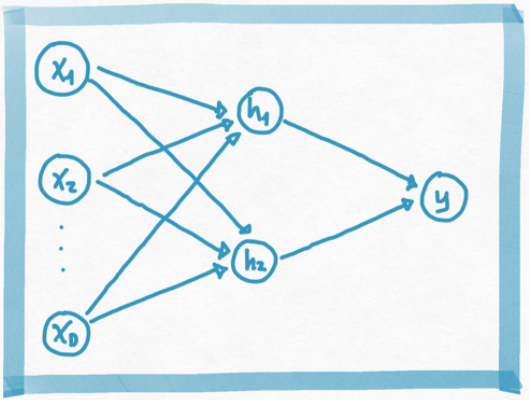


Each edge in this graph represents multiplication by a different constant w_d.

We call each w_d a **weight**.

ARCHITECTURE OF NEURAL NETWORKS:

Now we can express the composition of simple functions as graphs:



y is the function: $y = h(w_1 a_1 + w_2 a_1 + b_{11})$

h_1 is the function: $a_1 = h(\sum_{d=1}^{D} w_{d1} x_d + b_{01})$

h_2 is the function: $a_2 = h(\sum_{d=1}^{D} w_{d2} x_d + b_{02})$

The graph represents the function:

$$y = h(w_1 h(\sum_{d=1}^{D} w_{d1} x_d + b_{01}) + w_2 h(\sum_{d=1}^{D} w_{d2} x_d + b_{02}) + b_{11})$$

We see that the graph is an extremely compact way of representing a complex function.

This function we built is a neural network. The vertices of the graph are called nodes. The nodes for x_d are called input nodes, y is called the output node. The column of nodes between the input and output is called a hidden layer and each h_i is a hidden node.

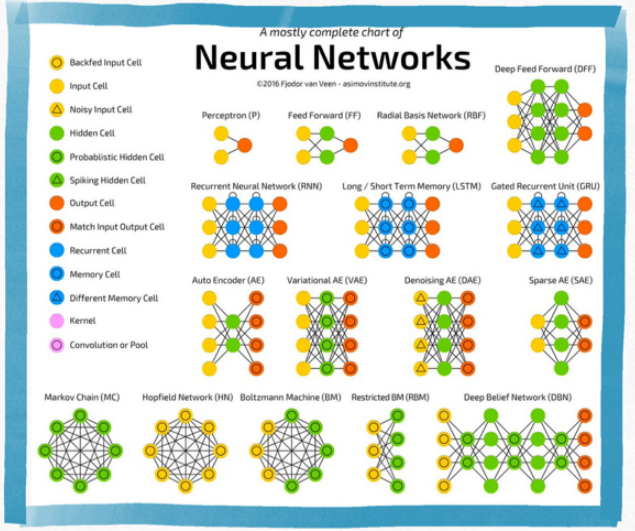
The graph and the choice of activation function h defines the architecture of the neural network.

THE EXPRESSIVENESS OF NEURAL NETWORKS

A ZOO OF NEURAL NETWORK ARCHITECTURES:

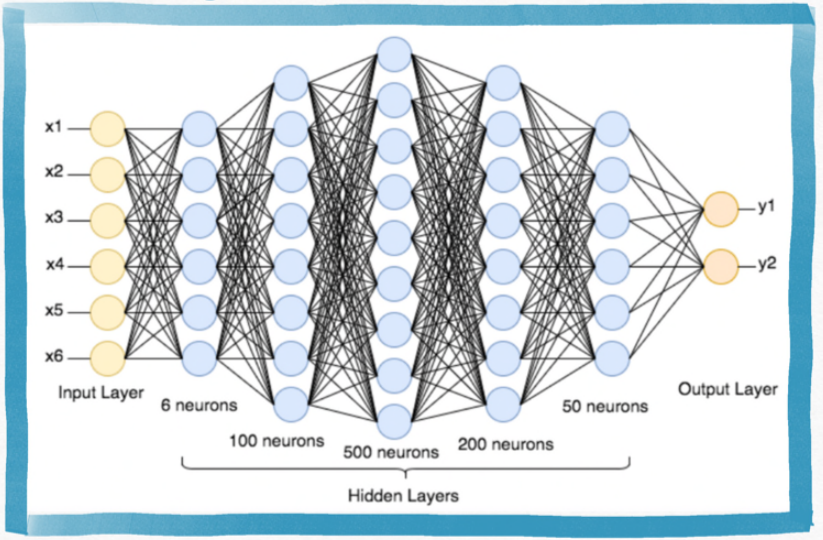
Using the graphical representation of neural network architectures, we can easily express a wide range of complex functions.

Architecture Matters!



Different architectures give rise to functions with very different properties, that are appropriate for different applications.

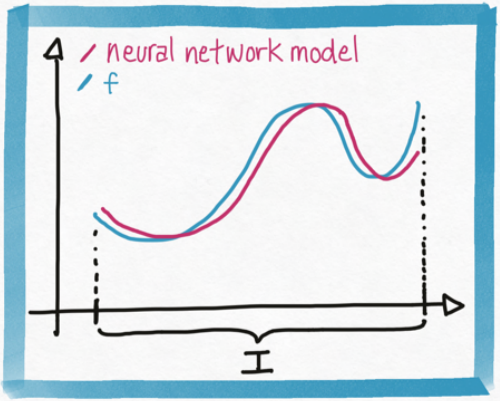
Size Matters!



We can easily define and implement large architectures that define functions so complicated it would be nearly impossible to write down the functional form explicitl

NEURAL NETWORKS AS UNIVERSAL APPROXIMATORS:

We've seen that neural networks can represent complex functions, but are there limitations on what a neural network can express?



Theorem: (Universal Approximation Theorem)

"For any continuous function f defined on a bounded domain I, we can find a neural network that approximates f with an arbitrary degree of accuracy."

This means that neural networks can be used in any application where we require a non-linear function.

NEURAL NETWORKS FOR REGRESSION

NEURAL NETWORKS FOR REGRESSION:

Model

Given a dataset $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$, we assume a non-linear probabilistic model for the data:

$$y = f_w(x) + \varepsilon \;, \quad \varepsilon \sim N(0, \sigma^2)$$

where f_w is a neural network with weights collectively denoted by w.

This is a probabilistic model.

Inference

We find weights w that maximizes the log-likelihood of D:

$$w_{MLE} = \arg\max_w \prod_{n=1}^{N} \log N(y^{(n)}; f_w(x^{(n)}), \varepsilon)$$

This is equivalent to minimizing the MSE of f_w on D:

$$w_{MLE} = \arg\min_w \frac{1}{N} \sum_{n=1}^{N} (y^{(n)} - f_w(x^{(n)}))^2$$

We find the maximum likelihood estimator of w

Optimization

To optimize the MSE, we use gradient descent, with learning rate η, we repeatedly update:

$$w^{(i+1)} = w^{(i)} - \eta \nabla_w \text{MSE}(w)$$

until $\nabla_w \text{MSE}(w) = 0$.

Gradient descent will find a stationary point, but will this point be a global minimum?