

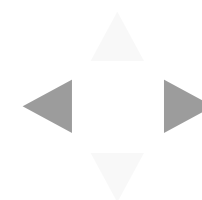
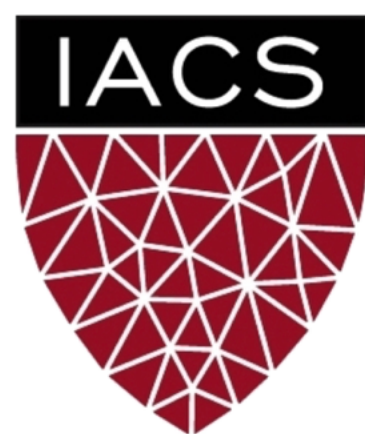
Lecture #16: Neural Network Models for Regression

AM 207: Advanced Scientific Computing

Stochastic Methods for Data Analysis, Inference and Optimization

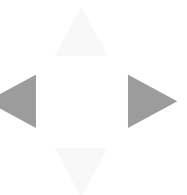
Fall, 2020



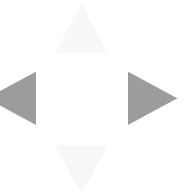


Outline

1. Regression as Generalized Linear Models
2. Neural Networks
3. Automatic Differentiation and BackPropagation



Regression as Generalized Linear Models



Linear Regression Models

Here is a generalized linear model (GLM) you've known since day one!

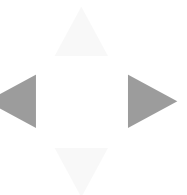
$$\mu = \mathbf{w}^\top \mathbf{X}^{(n)}$$
$$Y^{(n)} \sim \mathcal{N}(\mu, \sigma^2)$$

Alternatively, we can write this model as

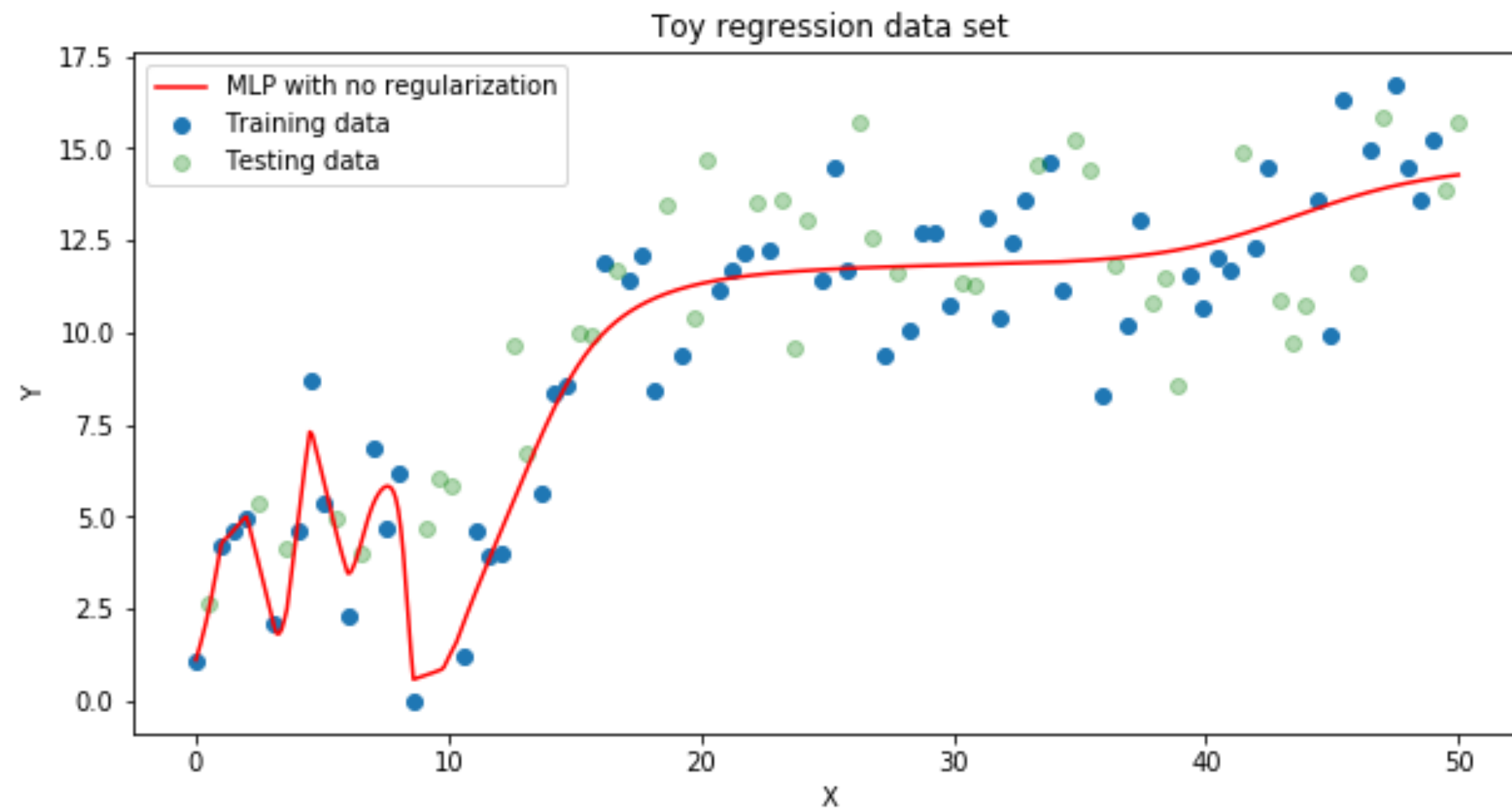
$$Y^{(n)} = \mathbf{w}^\top \mathbf{X}^{(n)} + \epsilon; \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$$

That is, injecting covariates into a normal likelihood is precisely linear regression! Just like in the case of logistic regression, we can form scientific hypotheses by examining the parameters of a linear regression model:

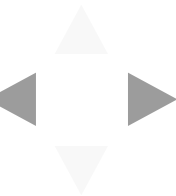
$$\widehat{\text{income}} = 2 * \text{education (yr)} + 3.1 * \text{married} - 1.5 * \text{gaps in work history}$$



How Would You Parameterize a Non-linear Trend?

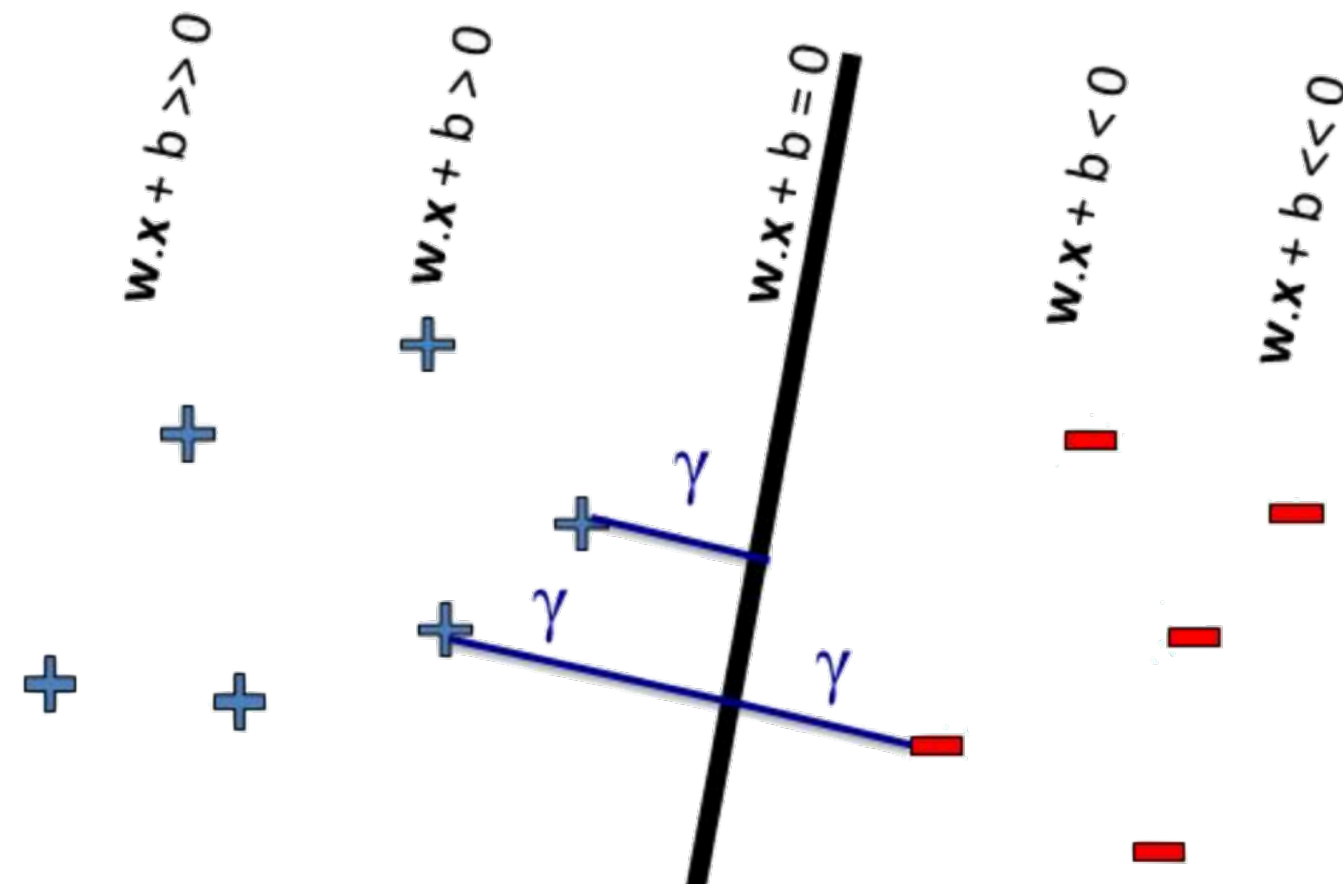


It's not easy to think of a function $g(x)$ can capture the trend in the data, e.g. what degree of polynomial should we use?

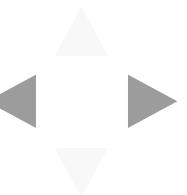


Review of the Geometry of Logistic Regression

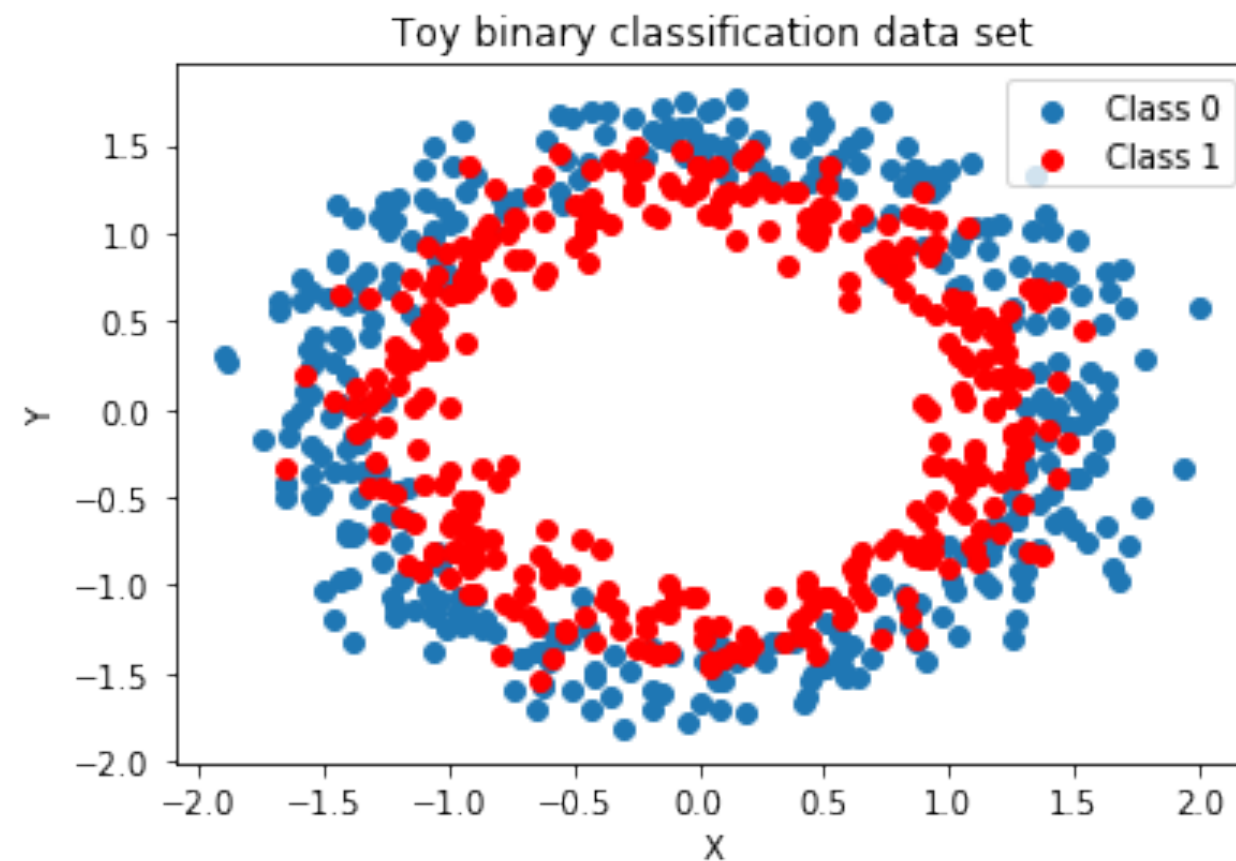
In **logistic regression**, we model the probability of an input \mathbf{x} being labeled '1' as a function of its distance from the hyperplane parametrized by \mathbf{w}



That is, we model $p(y = 1 | \mathbf{w}, \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$. Where $\mathbf{w}^\top \mathbf{x} = 0$ is the equation of the decision boundary.



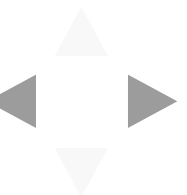
How would you parametrize a elliptical decision boundary?



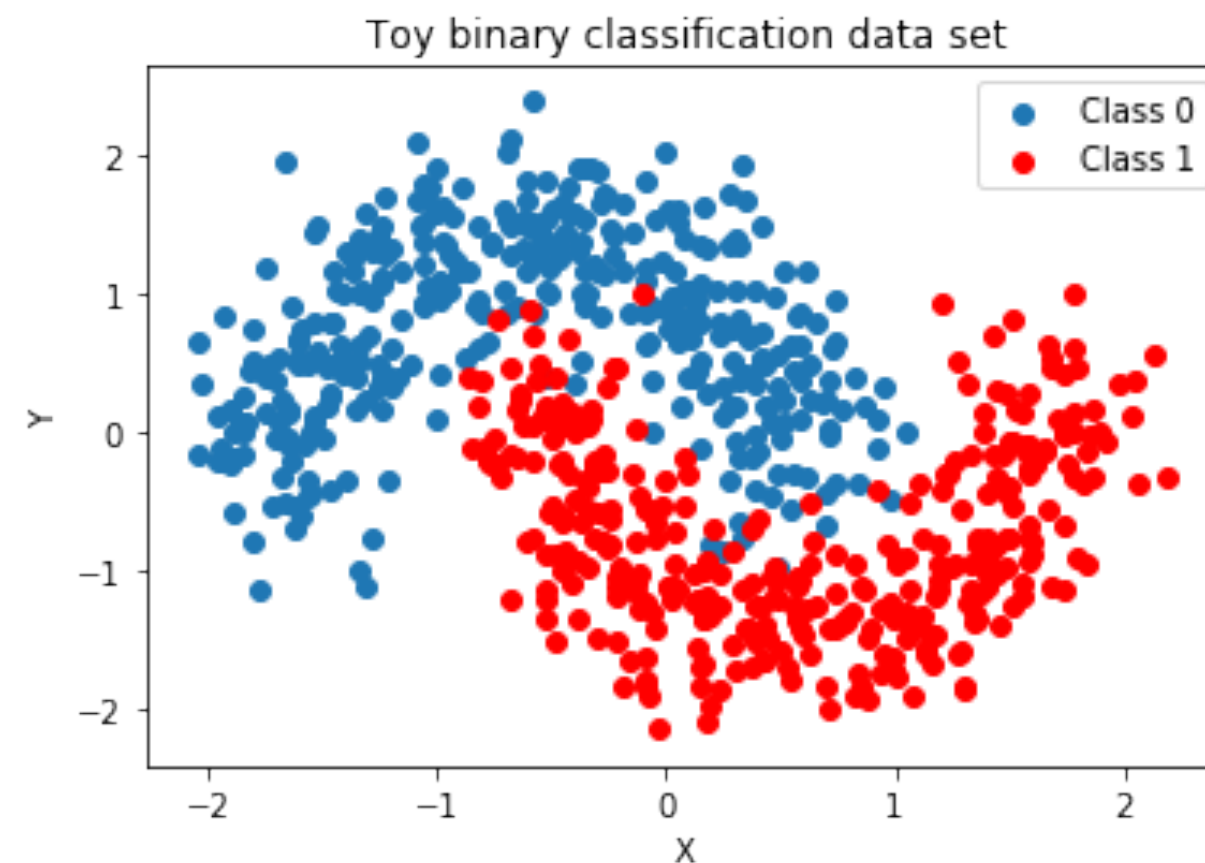
We can say that the decision boundary is given by a **quadratic function** of the input:

$$w_1 x_1^2 + w_2 x_2^2 + w_3 = 0$$

We say that we can fit such a decision boundary using logistic regression with degree 2 polynomial features

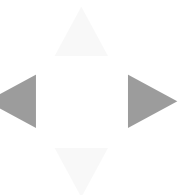


How would you parametrize an arbitrary complex decision boundary?

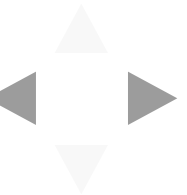


It's not easy to think of a function $g(x)$ can capture this decision boundary.

GOAL: Find models that can capture *arbitrarily complex* functions.



Neural Networks

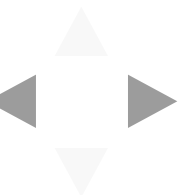


Approximating Arbitrarily Complex Decision Boundaries

Given an exact parametrization, we could learn the functional form, g , of the decision boundary directly.

However, assuming an exact form for g is restrictive.

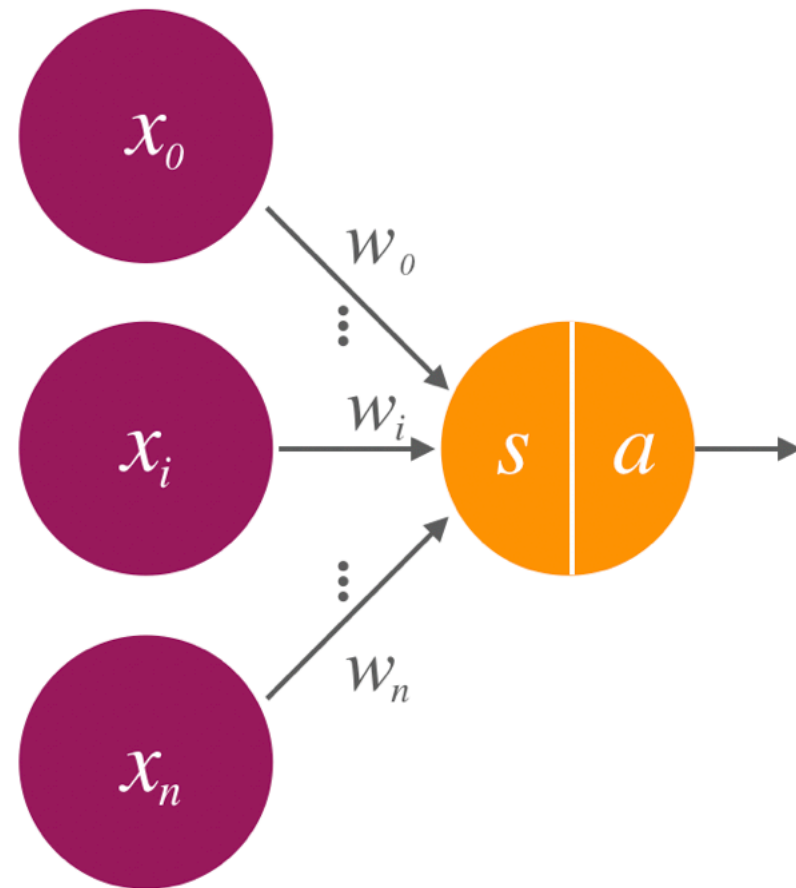
Rather, we can build increasingly good approximations, \hat{g} , of g by composing simple functions.



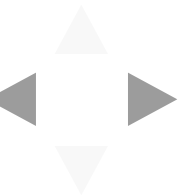
What is a Neural Network?

Goal: build a good approximation \hat{g} of a complex function g by composing simple functions.

For example, let the following picture represents $a = f\left(\sum_i w_i x_i\right)$, where f is a non-linear transform, and we denote the intermediate value $\sum_i w_i x_i$ by s .

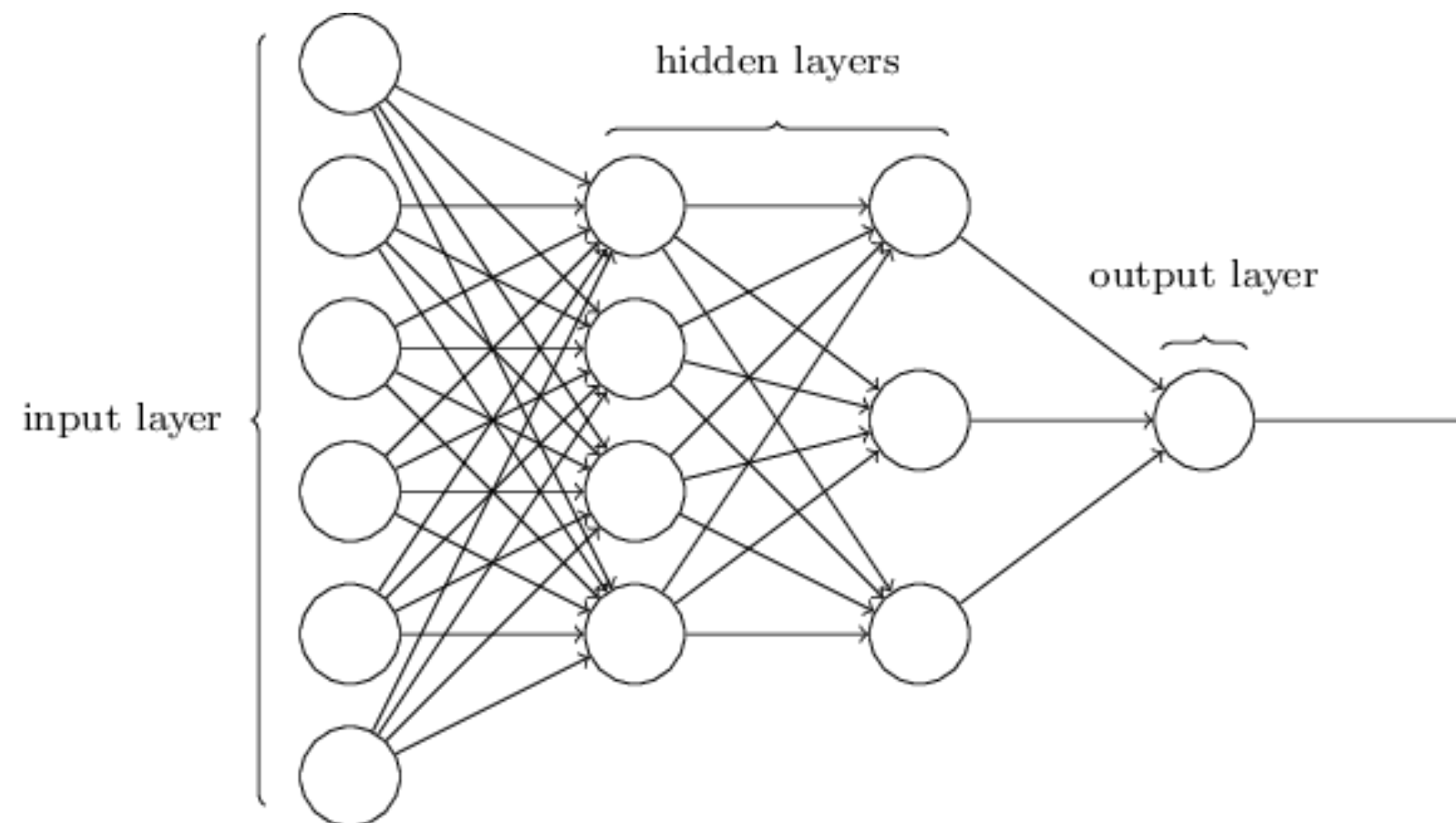


Note: we always assume that $x_0 = 1$ and hence w_0 is the intercept or ***bias*** of the linear expression $\sum_i w_i x_i$.

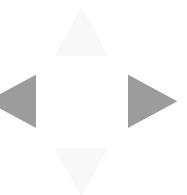


Neural Networks as Function Approximators

Then we can define the approximation \hat{g} with a graphical schema representing a complex series of compositions and sums of the form, $f\left(\sum_i w_i x_i\right)$



This is a **neural network**. We denote the weights of the neural network collectively by \mathbf{W} . The non-linear function f is called the **activation function**.



A Flexible Framework for Function Approximation

A mostly complete chart of

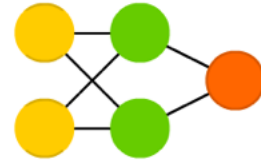
Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

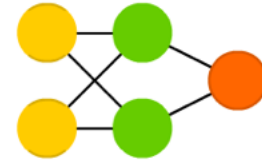
Perceptron (P)



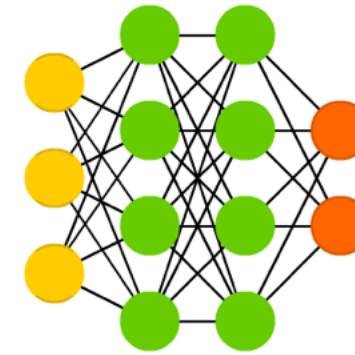
Feed Forward (FF)



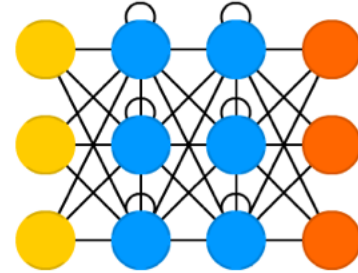
Radial Basis Network (RBF)



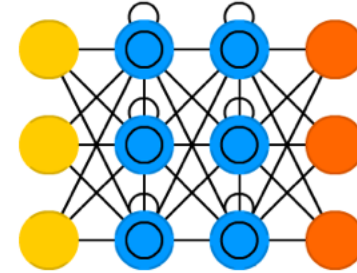
Deep Feed Forward (DFF)



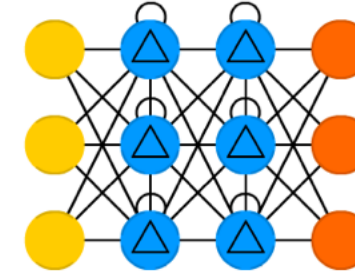
Recurrent Neural Network (RNN)



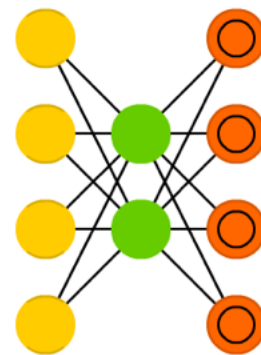
Long / Short Term Memory (LSTM)



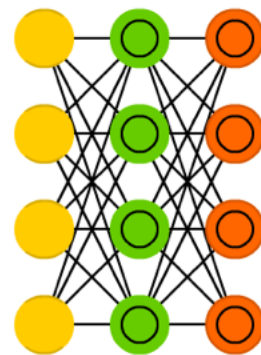
Gated Recurrent Unit (GRU)



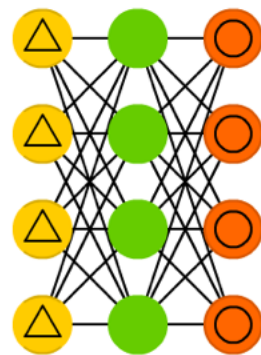
Auto Encoder (AE)



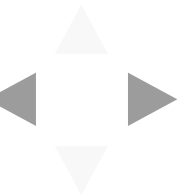
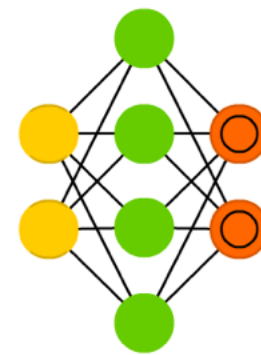
Variational AE (VAE)



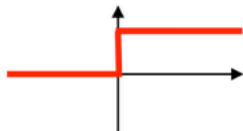
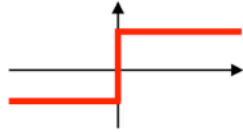
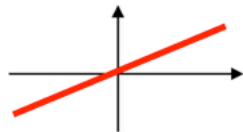
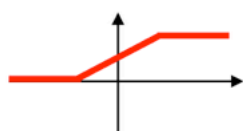
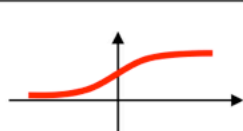
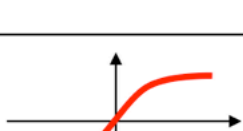
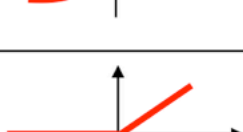
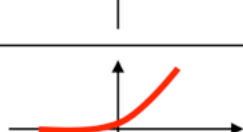
Denoising AE (DAE)



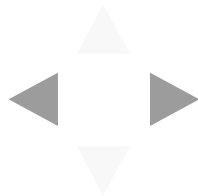
Sparse AE (SAE)



Common Choices for the Activation Function

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016
(<http://sebastianraschka.com>)

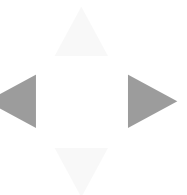


Neural Networks are Universal Function Approximators

So what kind of functions can be approximated by neural networks?

Theorem: (Hornik, Stinchcombe, White, 1989) Fix a "nice" activation function f . For any continuous function g on a compact set K , there exists a feedforward neural network with activation f , having only a single hidden layer, which approximates g to within an arbitrary degree of precision on K .

For this reason, we call neural networks *universal function approximators*.



Neural Networks Regression

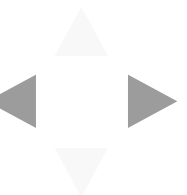
Model for Regression: $Y^{(n)} \sim \mathcal{N}(\mu, \sigma^2)$, $\mu = g_{\mathbf{W}}(\mathbf{X}^{(n)})$, where $g_{\mathbf{W}}$ is a neural network with parameters \mathbf{W} .

Training Objective: find \mathbf{W} to maximize the likelihood of our data. This is equivalent to minimizing the Mean Square Error,

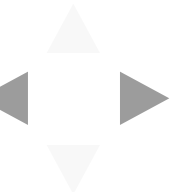
$$\max_{\mathbf{W}} \text{MSE}(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^N (y_n - g_{\mathbf{W}}(x_n))^2$$

Optimizing the Training Objective: For linear regression (when $g_{\mathbf{W}}$ is a linear function), we computed the gradient of the MSE with respect to the model parameters \mathbf{W} , set it equal to zero and solved for the optimal \mathbf{W} analytically (see Homework #0). For logistic regression, we computed the gradient and used (stochastic) gradient descent to "solve for where the gradient is zero".

Can we do the same when $g_{\mathbf{W}}$ is a neural network?



Automatic Differentiation and Backpropagation

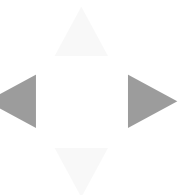
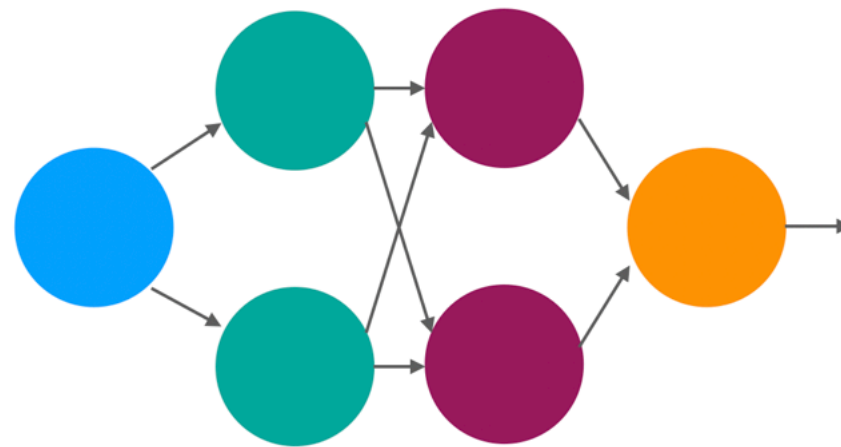


Gradient Computation for Neural Networks

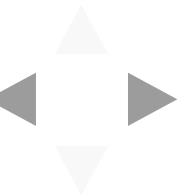
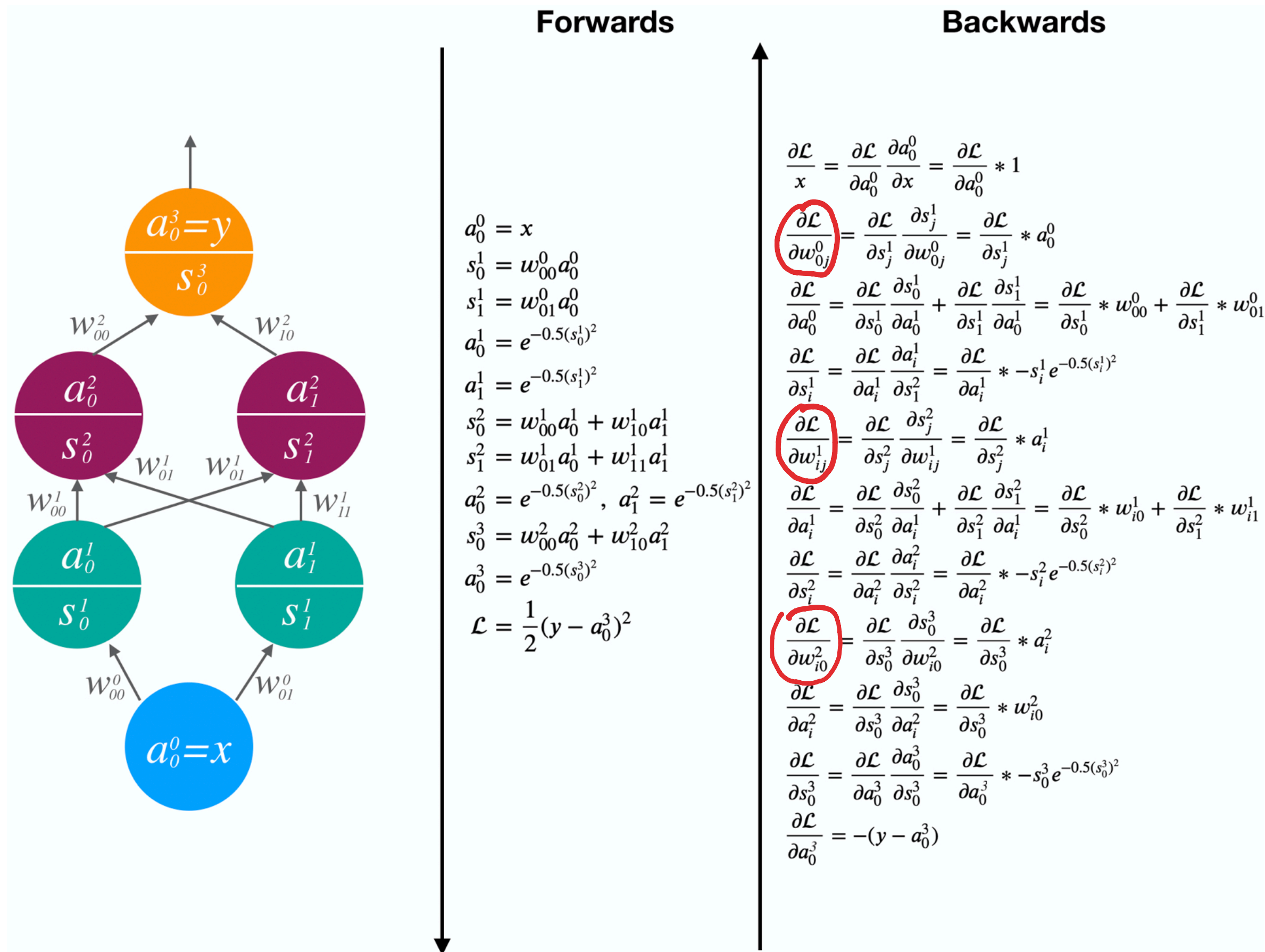
Computing the gradient for any parameter w_{ij}^l in the following network requires us to use the *chain rule*:

$$\frac{\partial}{\partial t} g(h(t)) = g'(h(t))h'(t), \quad \text{or} \quad \frac{\partial g}{\partial t} = \frac{\partial g}{\partial h} \frac{\partial h}{\partial t}$$

This is because a neural network is just a big composition of functions.



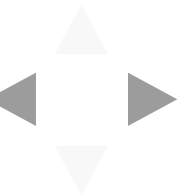
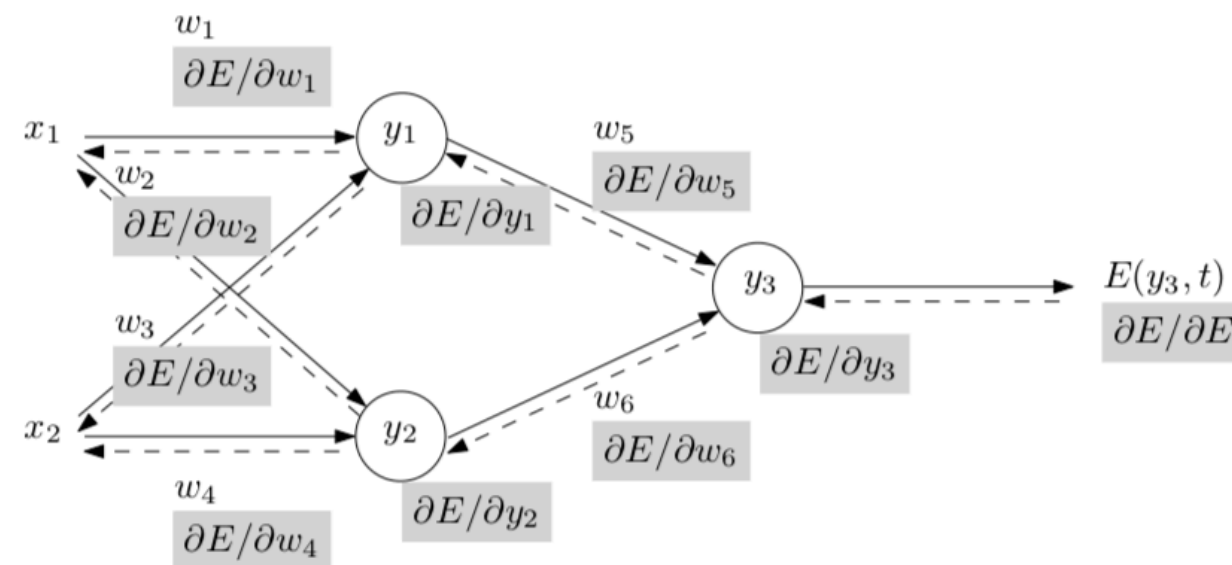
Example: Computing Neural Network Gradients



Backpropagation: Gradient Descent for Neural Networks

The *backpropagation* algorithm consists of three phases:

1. **(Initialize)** initialize the network parameters \mathbf{W}
2. Repeat:
 - A. **(Forward Pass)** compute all intermediate values s_{ij}^l and a_{ij}^l for the given covariates \mathbf{X}
 - B. **(Backward Pass)** compute all the gradients $\frac{\partial \mathcal{L}}{\partial w_{ij}^l}$
 - C. **(Update Parameters)** update each parameter by $-\eta \frac{\partial \mathcal{L}}{\partial w_{ij}^l}$

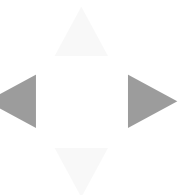
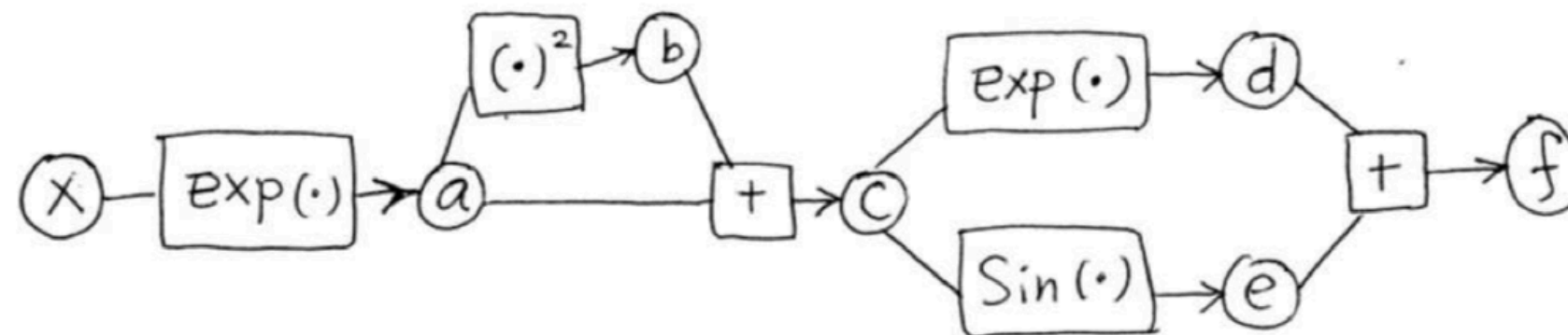


Gradient Computation with Automatic Differentiation

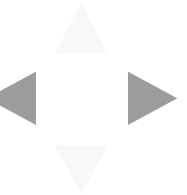
The forwards-backwards way of computing the gradient lends itself to an algorithm that automates gradient computation for any neural network.

This is a special instance of **reverse mode automatic differentiation** -- a method of algorithmically computing exact gradients for functions defined by combinations of simple functions, by drawing graphical models of the composition of functions and then taking gradients by going forwards-backwards.

$$f = \exp(\exp(x) + \exp(x)^2) + \sin(\exp(x) + \exp(x)^2)$$

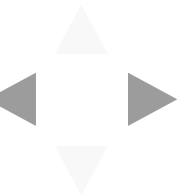
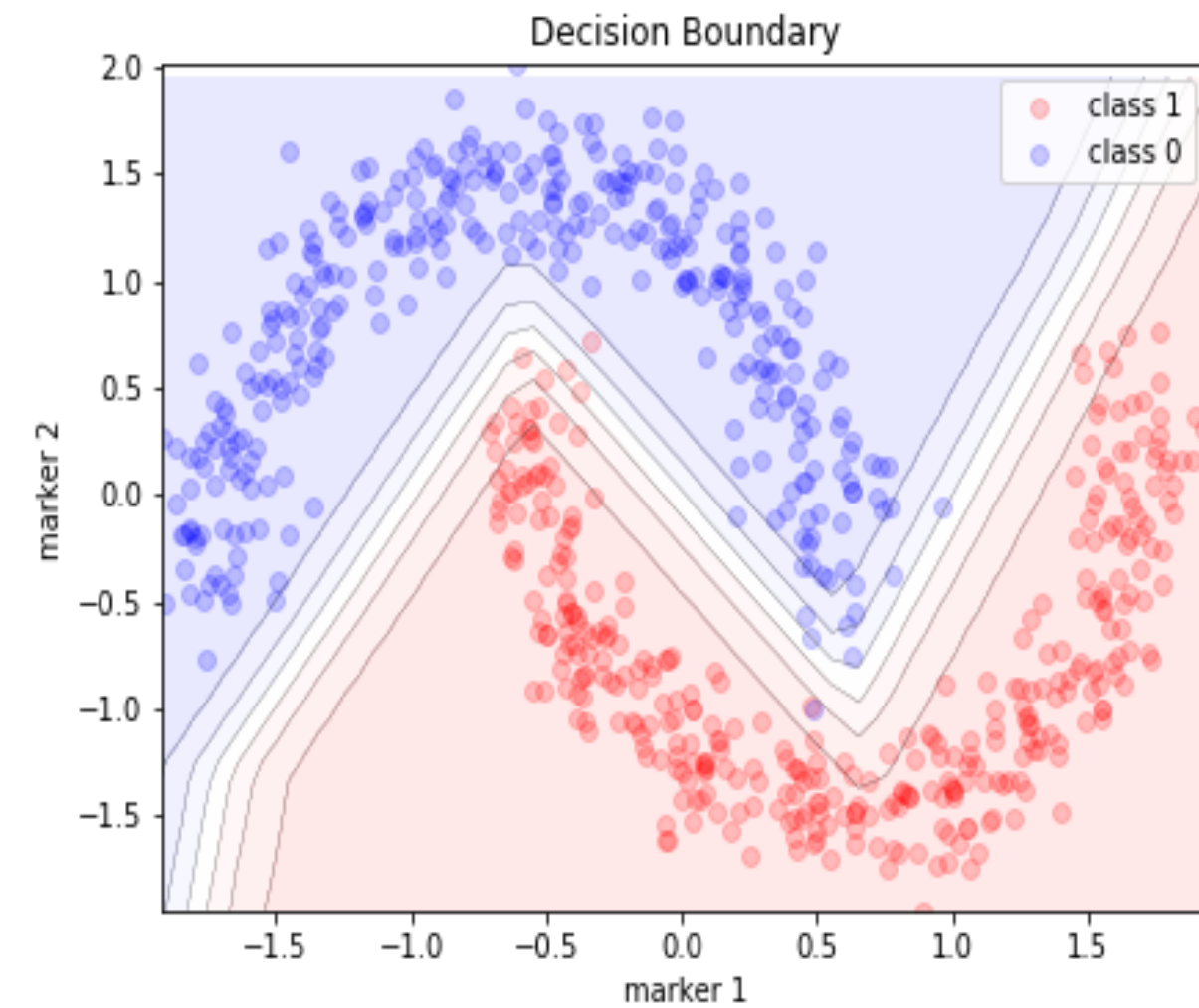
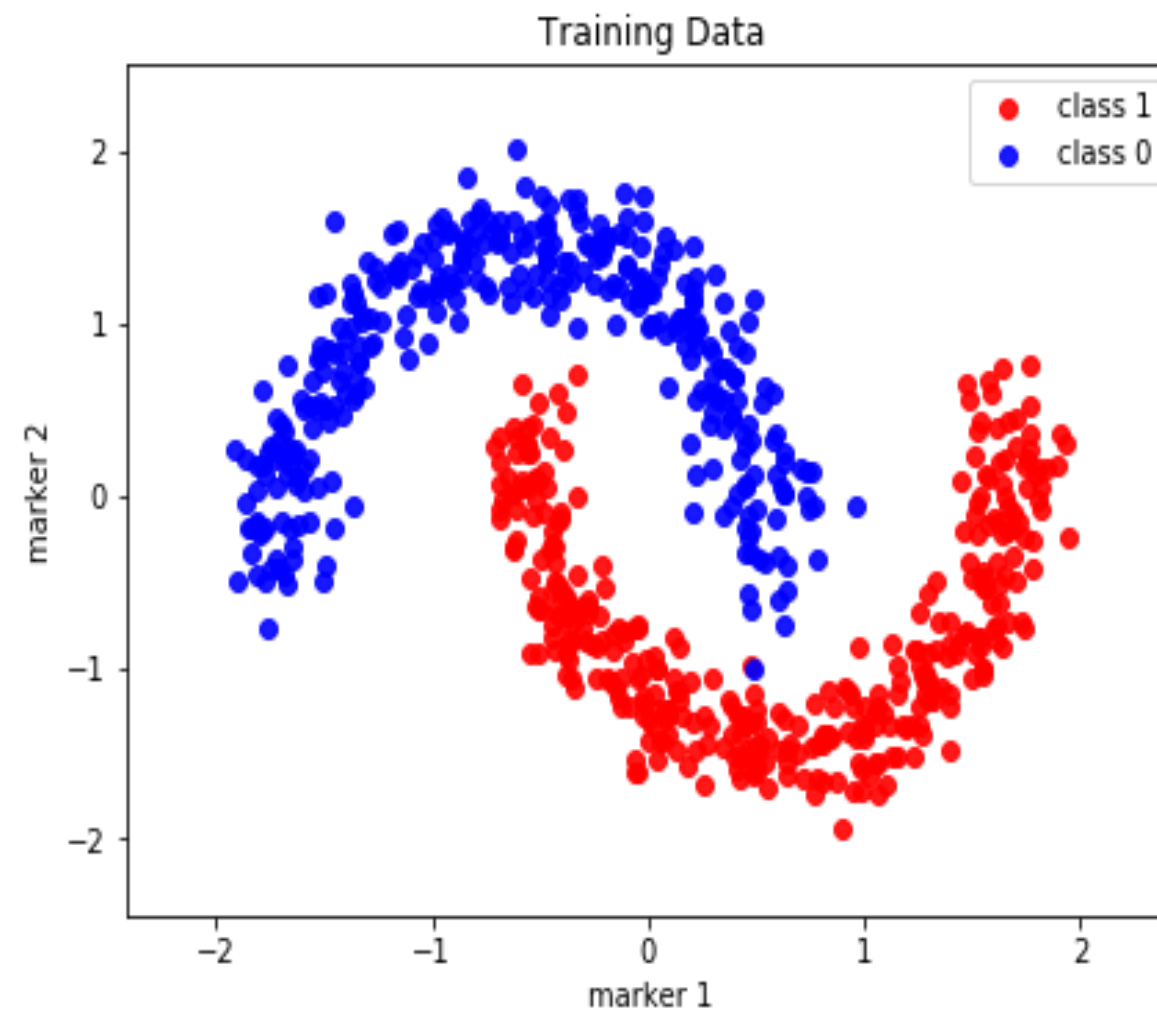


What Does a Neural Network Learn?



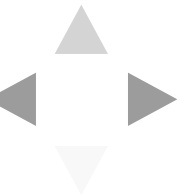
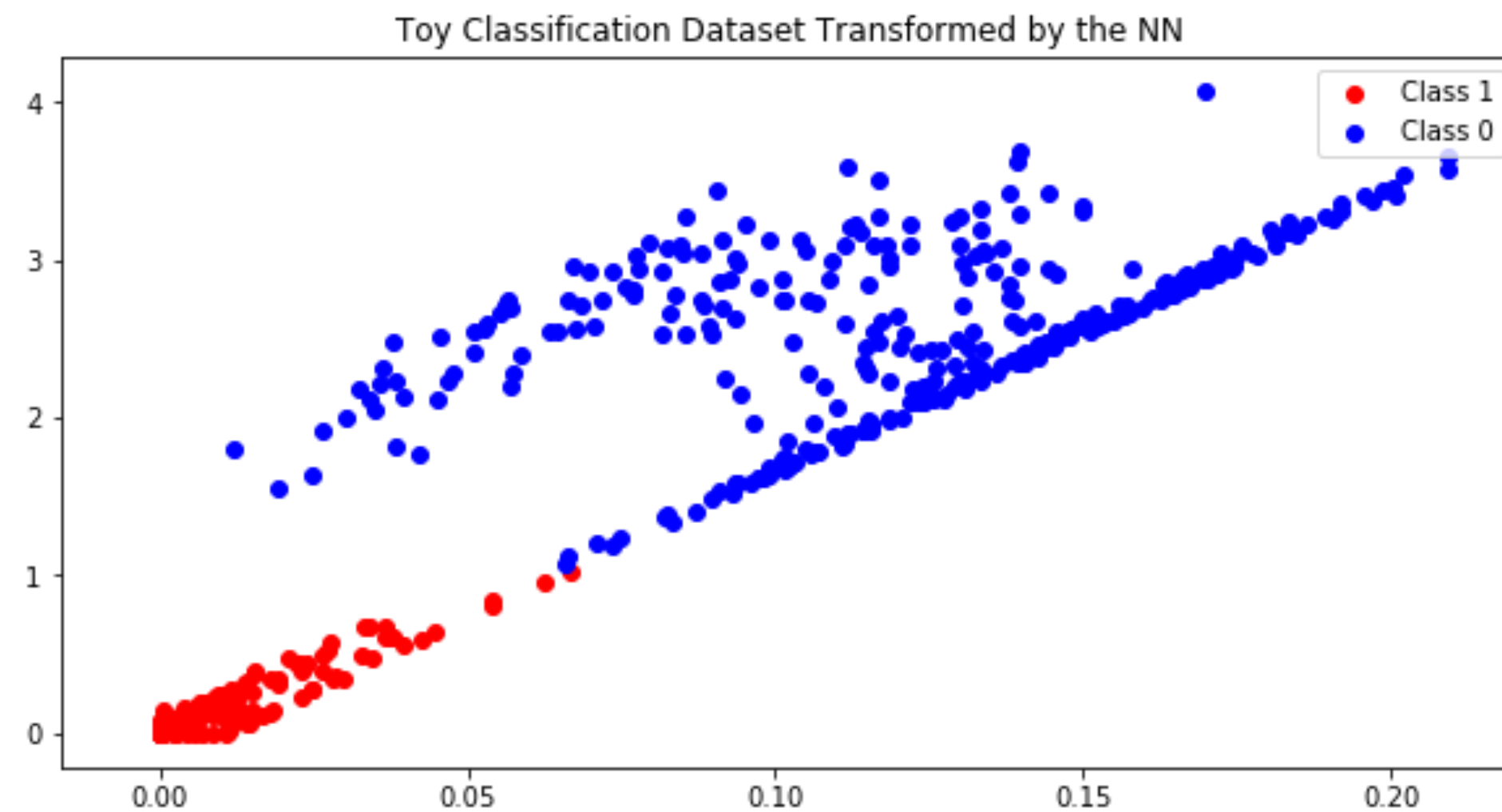
Why is a Neural Network Classifier So Effective?

Visualizing the decision boundary:



Why is a Neural Network Classifier So Effective?

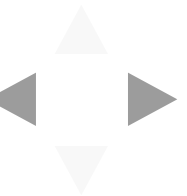
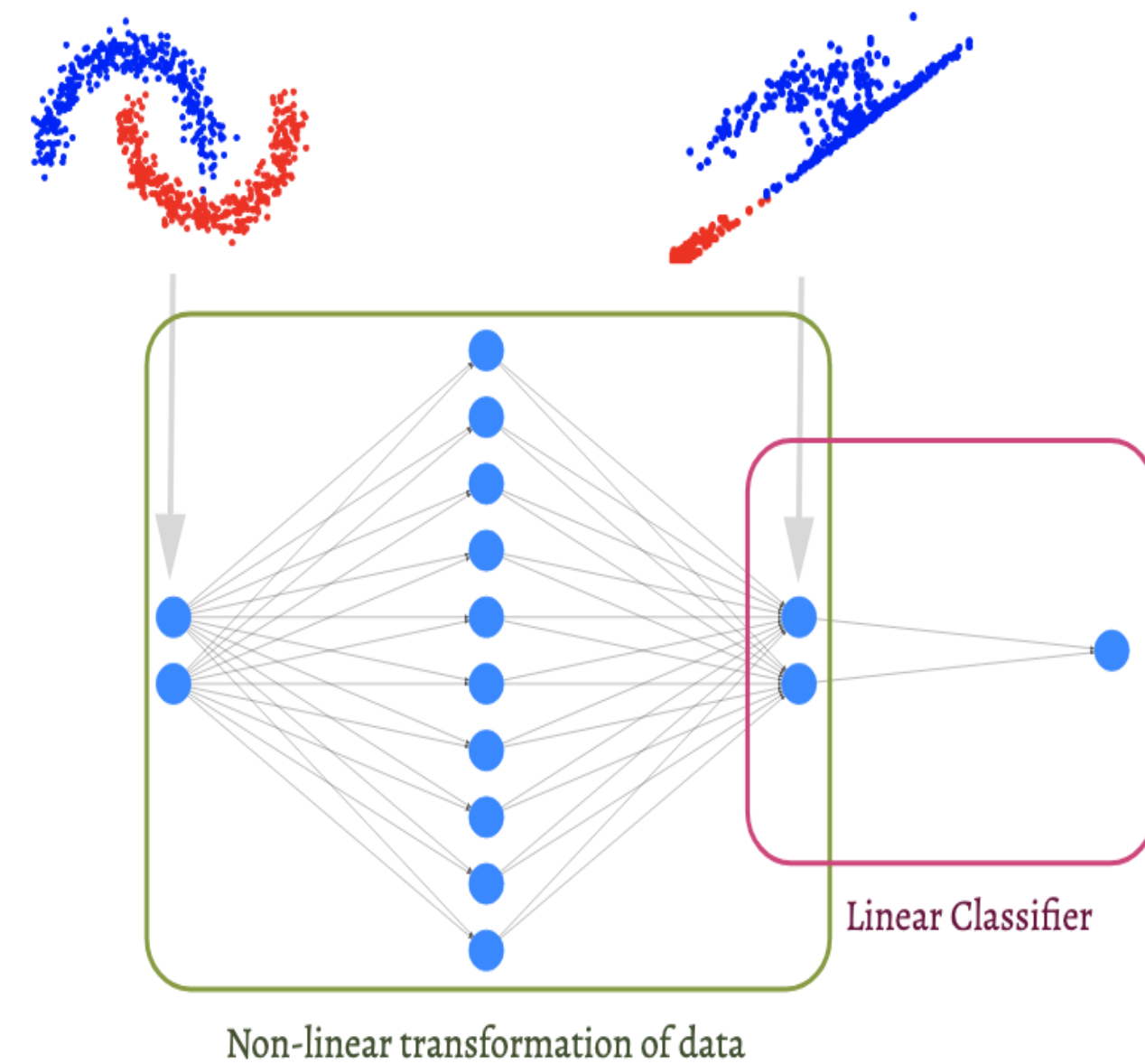
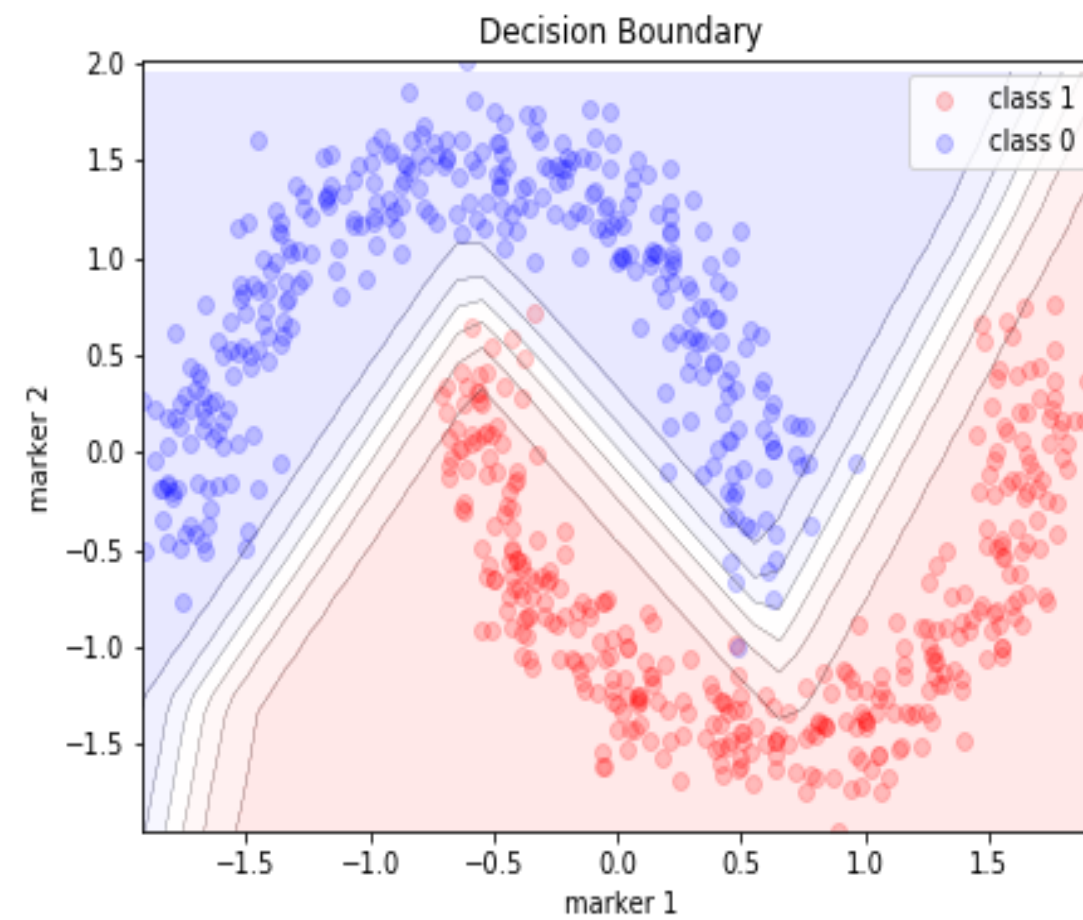
Visualizing the output of the last hidden layer.



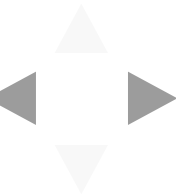
Two Interpretations of a Neural Network Classifier:

A Complex Decision Boundary g

A Transformation g_0 and a linear model g_1



With Great Flexibility Comes with Great Problems

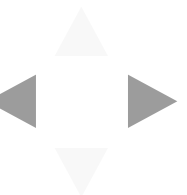
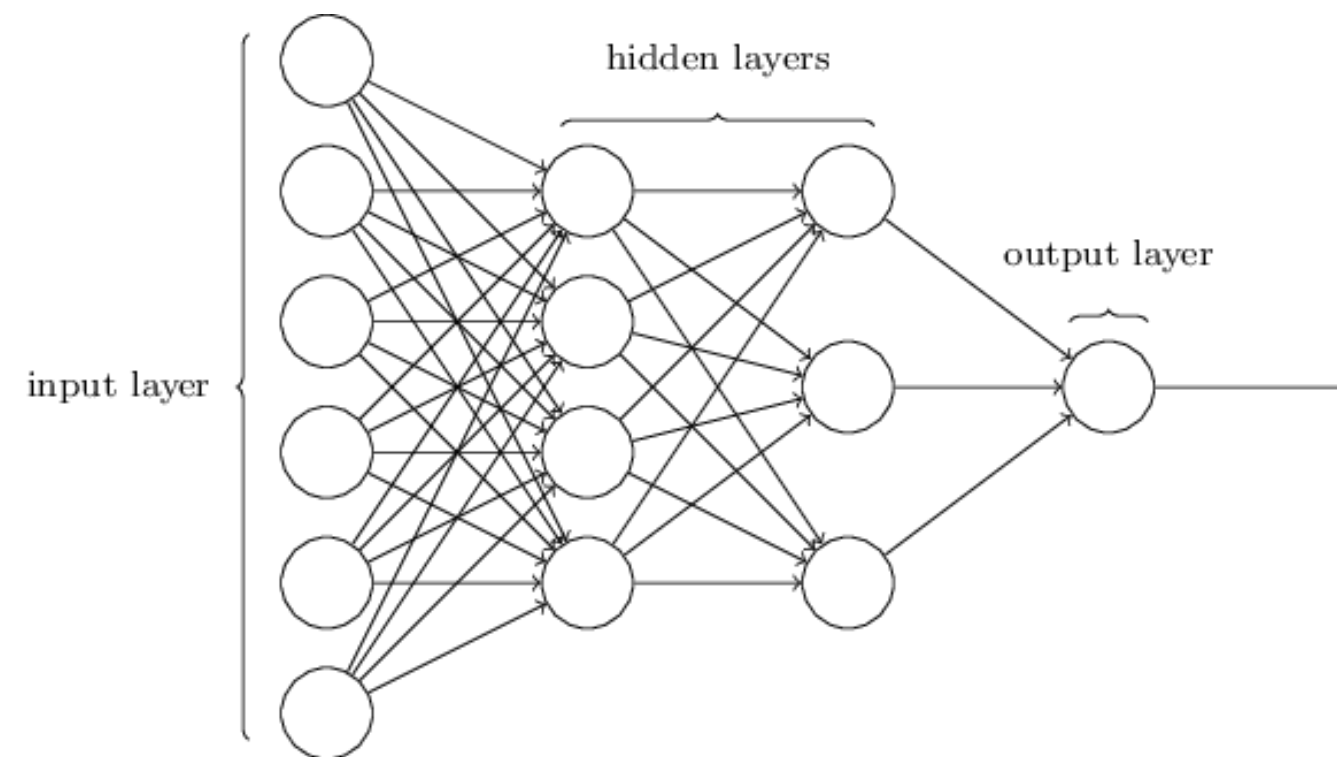


Neural Network Regression vs Linear Regression

Linear models are easy to interpret. Once we've found the MLE of the model parameters, we can formulate scientific hypotheses about the relationship between the outcome Y and the covariates \mathbf{X} :

$$\widehat{\text{income}} = 2 * \text{education (yr)} + 3.1 * \text{married} - 1.5 * \text{gaps in work history}$$

What do the weights of a neural network tell you about the relationship between the covariates and the outcome?



Generalization Error and Bias/Variance

Complex models have **low bias** -- they can model a wide range of functions, given enough samples.

But complex models like neural networks can use their 'extra' capacity to explain non-meaningful features of the training data that are unlikely to appear in the test data (i.e. noise). These models have **high variance** -- they are very sensitive to small changes in the data distribution, leading to drastic performance decrease from train to test settings.

