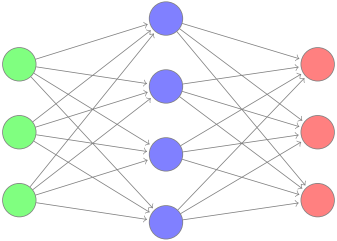


GERMS-ML
Neural Networks I

Paul Villanueva

Iowa State University

April 10, 2019

Outline

1. Background
2. Theory
3. Next time

Section 1

Background

Introduction

- Neural networks are probably the machine learning algorithm you hear the most about.
- Neural networks form the basis for deep learning, which is in a wide variety of fields, such as:
  - Image recognition
  - Speech recognition
  - Video recommendations
- AlphaGo, Google's Go-playing AI that defeated world champion Lee Sedol in 2016, was based on neural networks.

Introduction

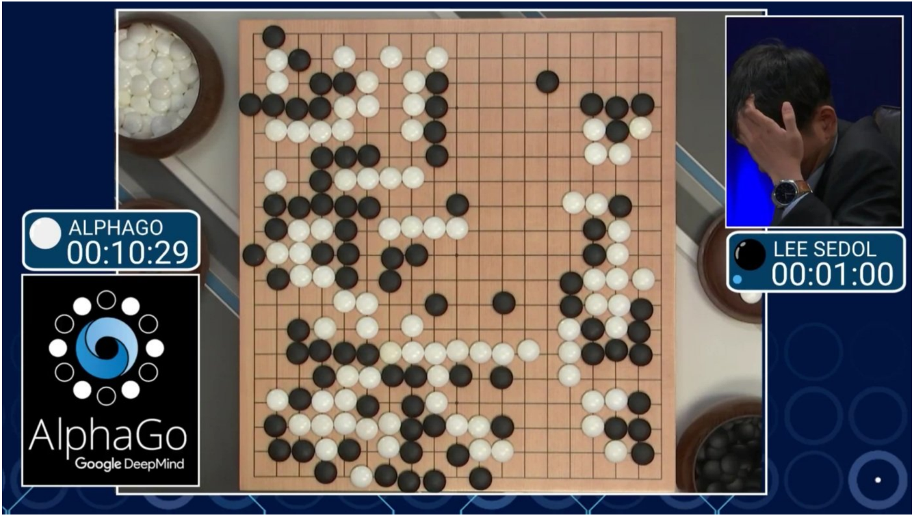


Figure 1: Deep learning network was famously used by AlphaGo to defeat world champion Lee Sedol in Go in 2016 (source).

Introduction

- Even though they've recently seen a surge in popularity, neural networks have been around for a while.
- The first neural network model was conceived by Warren McCulloch and Walter Pitts in their paper *"A Logical Calculus of Ideas Immanent in Nervous Activity"*.
- McCulloch and Pitts presented a simplified version of how biological neurons might work together to perform complex computations.

Introduction

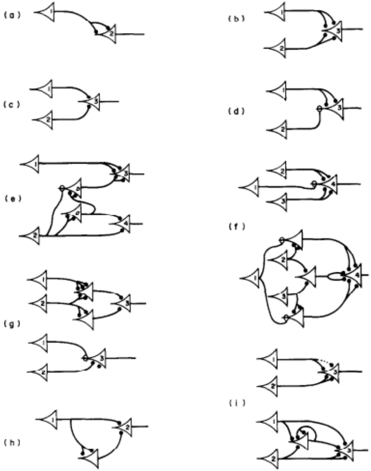


Figure 2: McCulloch and Pitts' propositional logic-based model for how neurons might interact (source).

Introduction

- An early implementation of a neural network was the perceptron, invented by psychologist Frank Rosenblatt in 1958.
- The perceptron was a two layer neural network, having an input layer and an output layer.
- The final prediction was some function of the weighted sum of the inputs.

Introduction

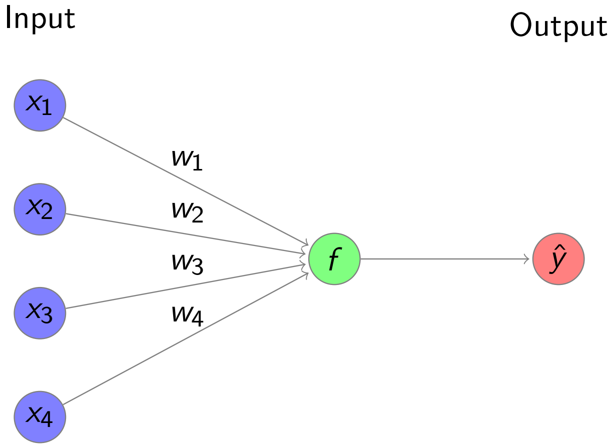


Figure 3: Visualization of what a perceptron might look like. The prediction $\hat{y}$ is some function of the weighted inputs; that is, $\hat{y} = f\big(\sum_i w_i x_i\big)$.

Introduction

- Saw some initial success in simple image recognition tasks, but failed at more sophisticated tasks.
- Fell out of favor for until recently and is enjoying much more success for several reasons:
    - Larger amounts of data available to train on
    - Increases in computational power
    - Algorithms have been improved
    - Sounds cool
- The main reason behind the resurgence of neural networks is the realization that the predictive power of a neural network can be increaesd by adding more layers to it.

Introductions

Pros

- Scales well to larger datasets
- Easy to set up and use

Cons

- Interpretibility
- Requires a lot of data
- Computationally expensive
- How do you determine network structure?

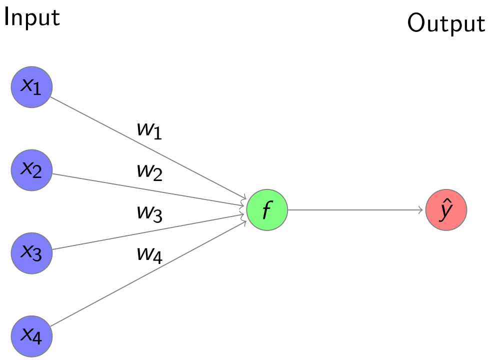
Section 2

Theory

Overview

- Neural networks have a few main components:
    - Input layer
    - Output layer
    - Hidden layers
    - Weights and biases
    - Activation functions
- Another name for neural networks is "multi-layer perceptron"
- We'll start by learning about the perceptron and building up the model from there.

The Perceptron



- Recall that each feature x_i has it's own weight w_i in the final prediction of category y.
- The final prediction $\hat{y}$ of the inputs $\mathbf{x}$ is some function f of the weighted sum of $\mathbf{x}$.
- We call f an *activation* function.

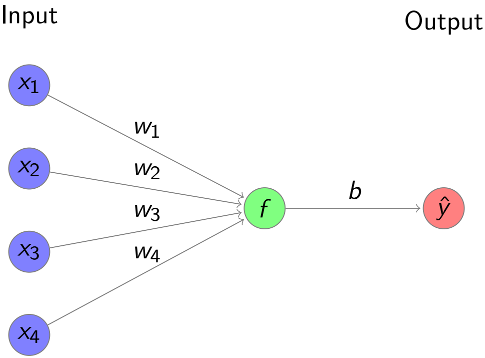
The Step function

- An example of an activation function is the **step** (or Heaviside) function:

$$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x \geq 0 \end{cases}$$

- The idea is to emulate how a neuron activates - once a certain level of stimulus is received, the neuron fires.
- Translated to a learning context, the final predicted class depends on which side of some threshold the weighted sum falls on.

Bias term



- We can introduce a *bias* term to directly influence that threshold.
- Then, the final prediction is calculated via

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$

Activation functions

- The step function isn't used as an activation function anymore for a few reasons:
  - The derivative is 0 everywhere, which is important for model correction (gradient descent)
  - The step function only allows the model to consider linear combinations of input variables
  - Not appropriate for classification of more than 2 classes.

Activation functions

- Functions typically used are:
  - Rectified linear unit (ReLU):

$$f(x) = \max(0, x)$$

  - Logistic (sigmoid)

$$f(x) = \frac{1}{1 + e^{-x}}$$

  - Hyperbolic tangent (tanh):

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

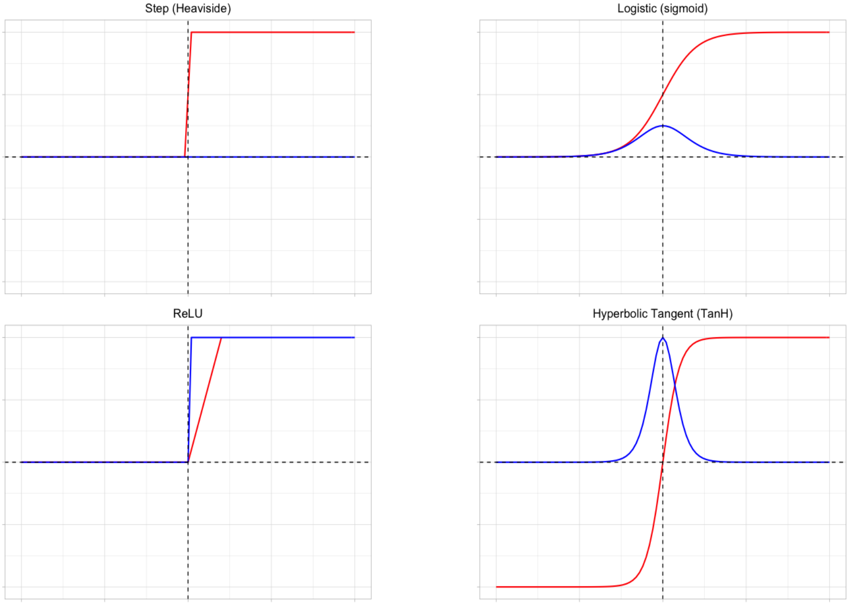
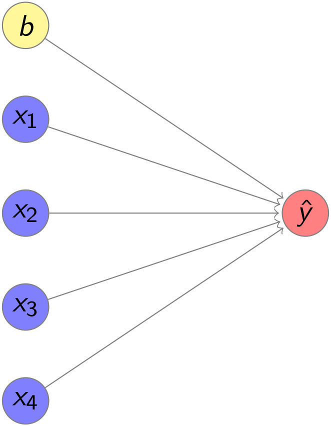


Figure 4: Some activation functions (red) and their derivatives (blue).

The perceptron

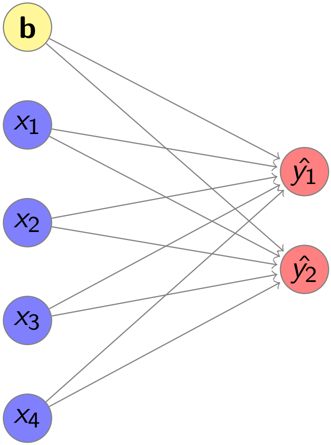


- Let's simplify the diagram a bit. It's understand that there is some activation function acting on inputs from previous layers.
- This is typically how neural networks are represented in the literature.

Increasing model complexity

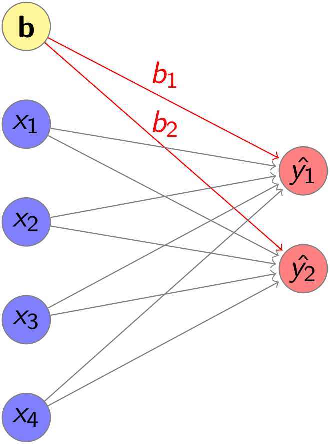
- Neural networks are often used for binary classification:
  - Is this email spam or not?
  - Is this email urgent or not?
  - Is this dollar bill fake or not?
- When the classes are mutually exclusive (such as in the case of the MNIST dataset), we can modify the output layer by adding additional nodes for each of the classes.
- Let's add another output node to our model.

Increasing model complexity



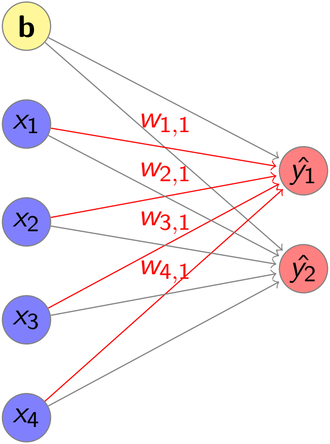
- There are a few things to note.

Increasing model complexity



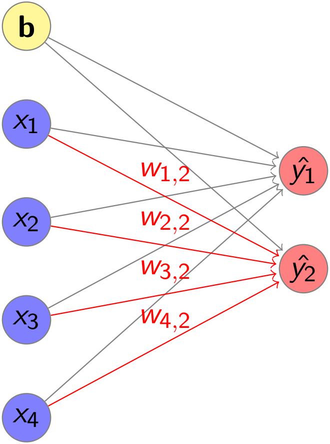
- There is a unique bias for each feature...

Increasing model complexity



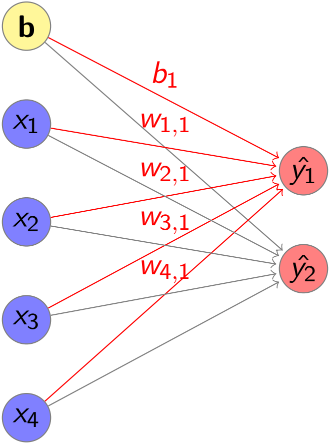
- ...and there are unique weights between each input...

Increasing model complexity



- ...and each output.

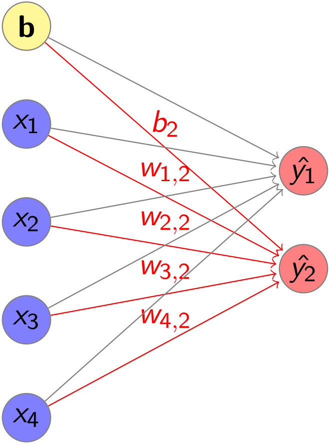
Increasing model complexity



- Thus, the prediction for y_1 is given by

$$\hat{y_1} = f\left(\sum_i w_{i,1}x_i + b_1\right)$$

Increasing model complexity



- Similarly, the prediction for y_2 is given by

$$\hat{y_2} = f\left(\sum_i w_{i,2} x_i + b_2\right)$$

Notation

- Note that $f(\sum w_{i,j}x_j + b_j)$ is a dot product between the inputs and a vector of the weights plus some bias at the end.
- Then, we can arrange everything as

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = f\left(\begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} & w_{4,1} \\ w_{1,2} & w_{2,2} & w_{3,2} & w_{4,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right)$$

Notation

- We can write this more compactly as

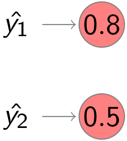
$$\mathbf{Y} = f\Big(\mathbf{WX} + \mathbf{B}\Big),$$

  where

  - $\mathbf{X}$ is the input attributes,
  - $\mathbf{W}$ is the vector of weights,
  - $\mathbf{B}$ is the vector of biases,
  - f is the activation function, and
  - $\mathbf{Y}$ is the vector of categories we're trying to predict

Predictions

- So how does one actually make predictions?
- Suppose we ran our neural network and we got the following class probabilities for $\hat{y}_1$ and $\hat{y}_2$:



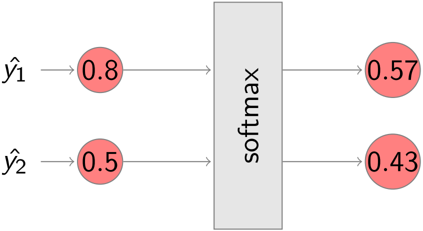
$$\hat{y}_1 \longrightarrow 0.8$$

$$\hat{y}_2 \longrightarrow 0.5$$

The softmax function

- We apply the softmax function to each of our output nodes.
- The softmax function is defined by:

$$\text{softmax}(\hat{\mathbf{Y}})_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

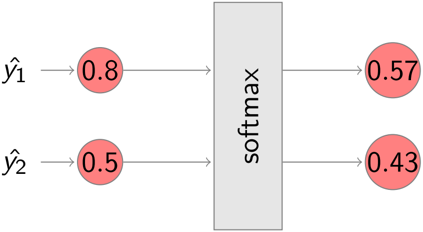


The softmax function

- Applying the softmax function has several advantages.
  - It normalizes the class probabilities so that the values in the output layer sum to 1, and that they are all between 0 and 1. Hence, the softmax transforms the output layer into a probability distribution.
  - The softmax function is differentiable and thus trainable.
  - Most neural nets are trained via maximum likelihood estimation, the calculations of which involve log. The exponential functions in the softmax play nicely with the log-likelihood functions.
  - Several other theoretical advantages.

Predictions

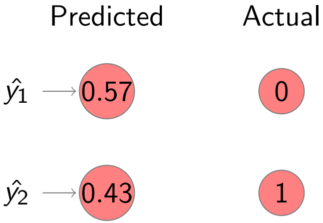
- After applying softmax, we then choose the maximum value to be the predicted class of the input variable.



- Thus, our model predicted that these inputs belong to y_1.

Training

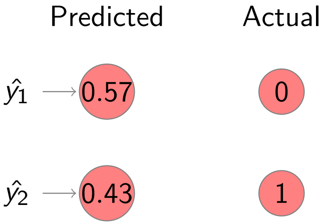
- But what if our model was wrong? What if the input actually belong to class y_2? How can we correct our model?



	Predicted	Actual
$\hat{y}_1 \longrightarrow$	0.57	0
$\hat{y}_2 \longrightarrow$	0.43	1

- We will measure how accurate our model is by calculating a cost function C.
- Lower values of this function mean that our model is doing better.
- We want to minimize this function.

Training



Predicted

Actual

$\hat{y}_1 \longrightarrow$ 0.57 0

$\hat{y}_2 \longrightarrow$ 0.43 1

- For our example, our cost function will be the mean squared error.
- The mean squared error of this data point is

$$(0 - 0.57)^2 + (1 - 0.43)^2 = 0.6498.$$

Training

- We repeat this process for all of our training points and take the mean of these values to arrive at the cost for our training model.
- Our cost function C is a function of all the weights and biases in the neural network. In our little toy example, our cost function depends on 10 variables. The number of variables skyrockets for larger networks.
- The goal is to find the right values for the weights and biases to minimize this cost.
- How do minimize such a high-dimensional function?

Gradient descent

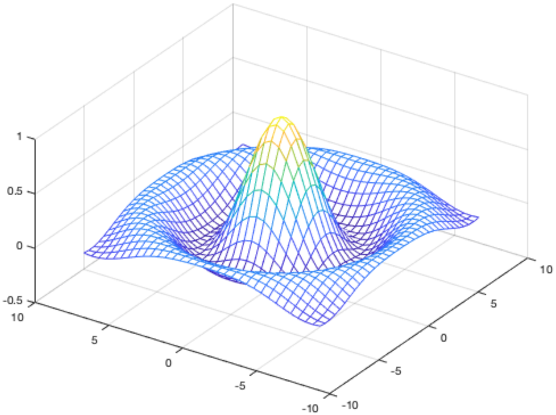


Figure 5: An example of a cost surface. Minimizing a function like this is analogous to finding the coordinates of the global minimum.

Gradient descent and back propagation

- The algorithm used to minimize the cost function is **gradient descent**.
- The algorithm is based on the observation that the negative gradient of a function points in the direction of sharpest decrease.
- If we take a step in the direction of the negative gradient and take note of the coordinates, we will have found a set of variables that has decreased the cost of C.

Gradient descent and back propagation

- The weights in the model are then updated through a process called back propagation.
- We then repeat this process of training, gradient descent, and back propagation until we've optimized our model.
- We will cover gradient descent and back propagation in more detail in later meetings.

Hidden layers

- We have only so far considered a perceptron, also known as a single layer neural network.
- We can increase the complexity of our model by adding additional nodes in between the input and output layers. We call these additional nodes *hidden layers*.

Hidden layers

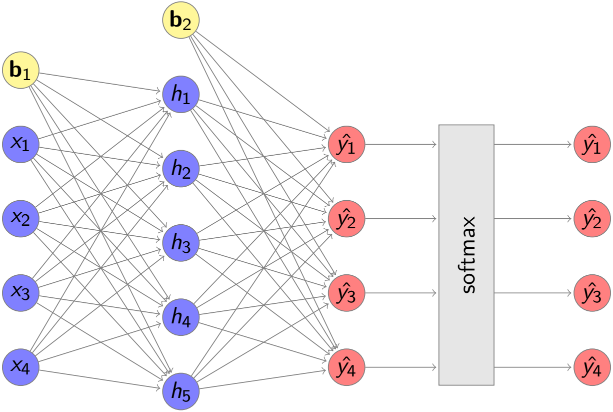


Figure 6: A network with a single hidden layer.

Hidden layers

- By adding hidden layers, the model now has more degrees of freedom to pick up on more sophisticated features of the model.
- The learning happening in the hidden layers is what is known as *deep learning*.
- The mathematics behind this is the obvious extension of the formulae we derived for the perceptron.
-

Section 3

Next time

Next time

In future meetings, we will:

- Revisit the MNIST dataset
- Revisit Jae's dataset
- Talk about genomic prediction
- Talk about convolution neural nets