Machine Learning: Neural Networks

Rodrigo Azuero

IADB

2018

Topics for today:

1. Why do we need neural networks?
2. What are neural networks?
3. How do we estimate neural networks?

Additional (if time permits)

- Derivation of cost function.
- Backpropagation intuition
- Estimation through gradient descent.

It will be progressively difficult.

Outline

1. Why do we need neural networks?

- Let us recall our spam classifier problem:

$$y_i = \begin{cases} 0 \text{ if non-spam} \\ 1 \text{ if spam} \end{cases}$$

- We want to predict probabilities: $P(y_i = 1)$ based on regressors $x_1, \ldots x_p$

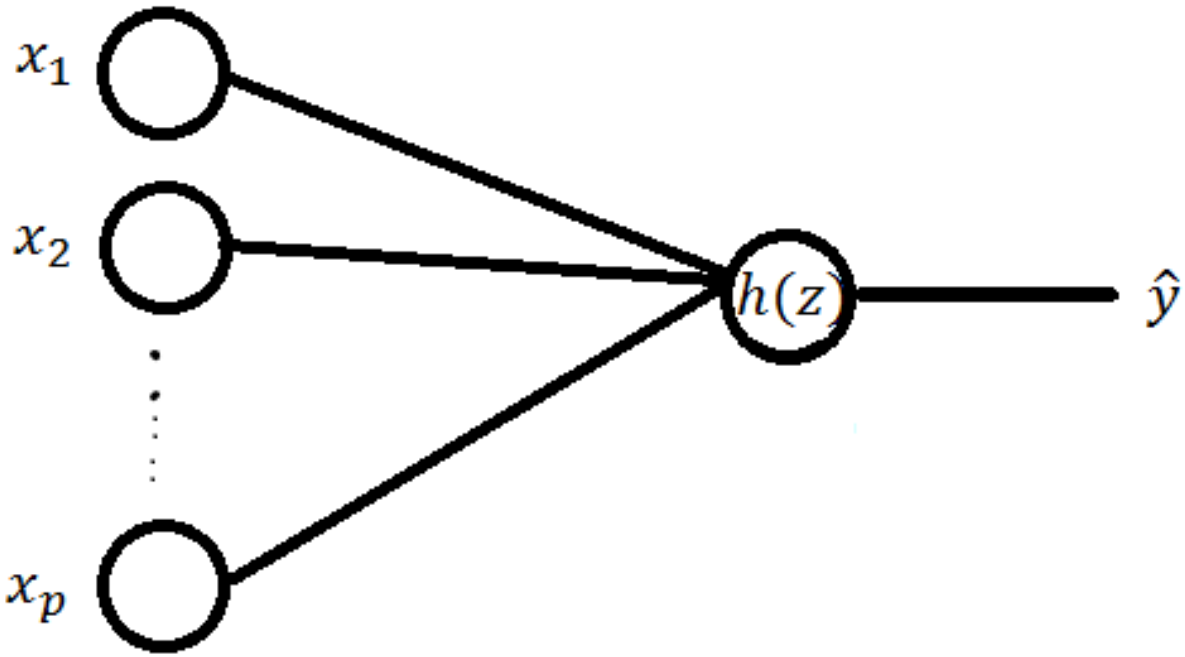
$$P(y_i = 1) = \frac{1}{1 + e^{-(b + w_1 x_{i,1}, \ldots + w_p x_{i,p})}}$$

$$P(y_i = 0) = 1 - P(y_i = 1)$$

$$\hat{y}_i = 1 \text{ if } \hat{P}(y_i = 1) > 0.5$$

- We often refer to $h(z) = \frac{1}{1+e^{-z}}$ as a 'sigmoid' function.

- Logistic regression is a very basic neural network

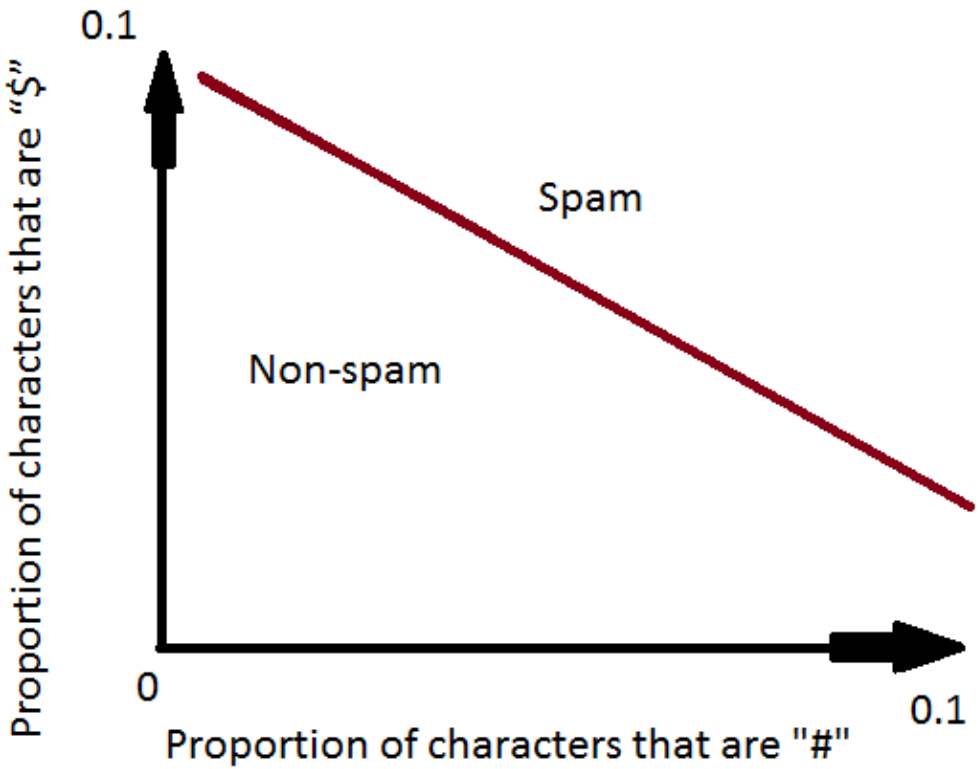


- $x_1, ...x_p$: **input layer** (covariates)
- $z = b + w_1x_1 + w_2x_2 + ...w_px_p$: **linear transformation**.

  b: **bias term** (intercept) ; w_k: **weights**
- $h(z) = \frac{1}{1+e^z}$: **activation function**
- Predictions: **output layer**

I. Why do we need neural networks?

- Spam Classifier

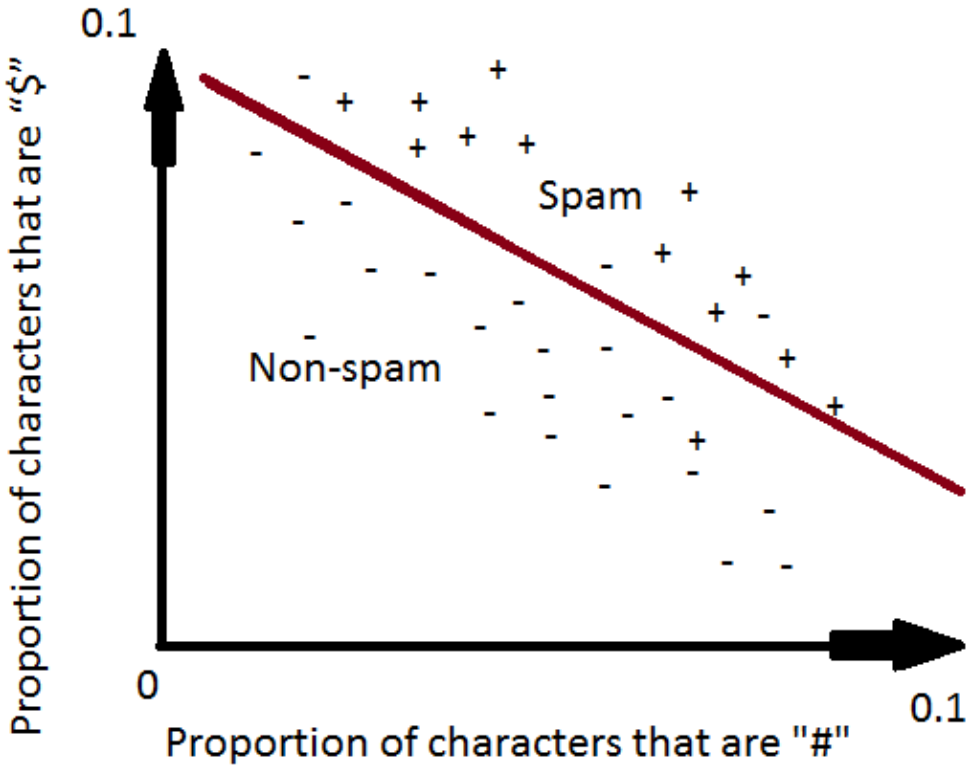


Steps:

- Estimation: $P(Spam_i) = \frac{1}{1+e^{-\left(\beta_0 + x_{i,1}\beta_1, \ldots + x_{i,57}\beta_p\right)}}$
- We know logistic regression performed ok in this problem.

I. Why do we need neural networks?

- Spam Classifier



Steps:

- Estimation: $P(Spam_i) = \frac{1}{1+e^{-\left(\beta_0 + x_{i,1}\beta_1, \ldots + x_{i,57}\beta_p\right)}}$
- We know logistic regression performed ok in this problem.

(Highly non-linear hypothesis)

- Digit recognition

(MNIST dataset)

(Highly non-linear hypothesis)

- Digit recognition

=

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

(MNIST dataset)

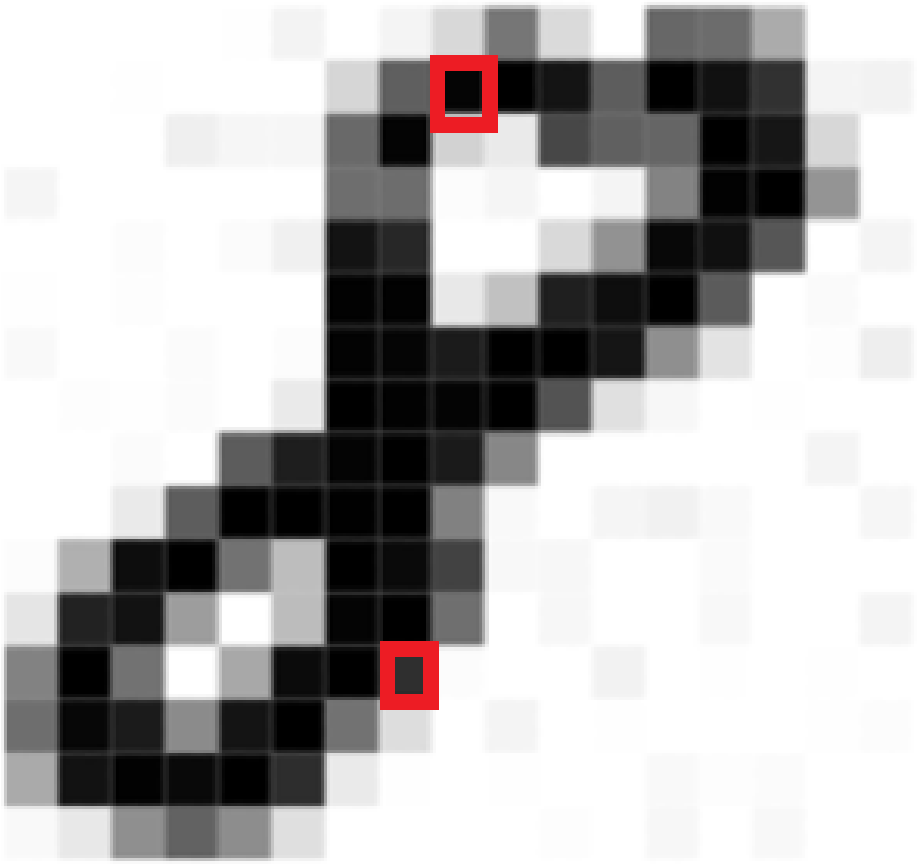
(Highly non-linear hypothesis)

- Digit recognition

(MNIST dataset)

(Highly non-linear hypothesis)

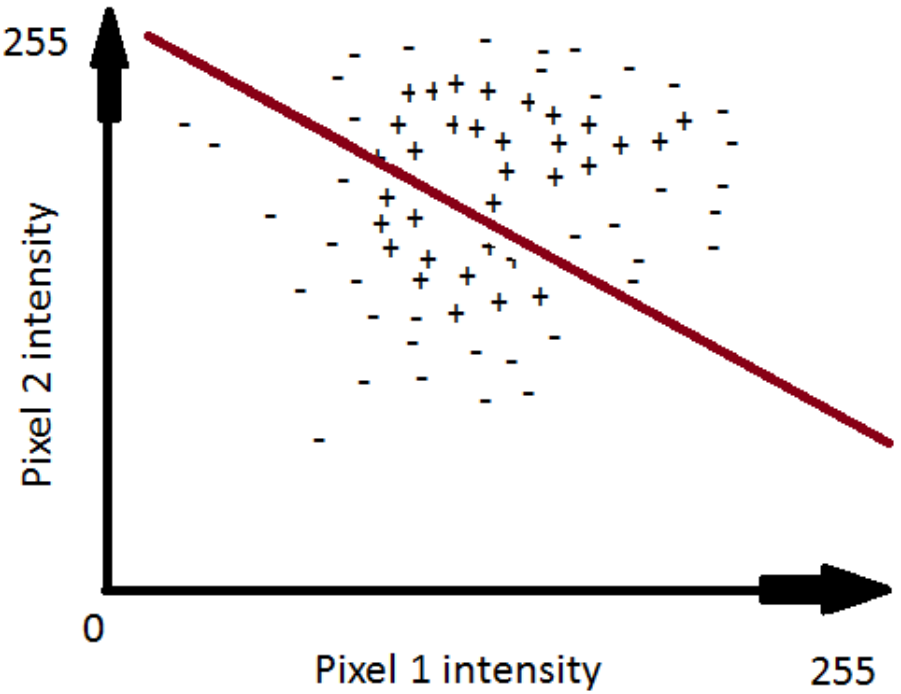
- Digit recognition



(MNIST dataset). Predicting with intensity in two pixels.

(Highly) non-linear hypothesis

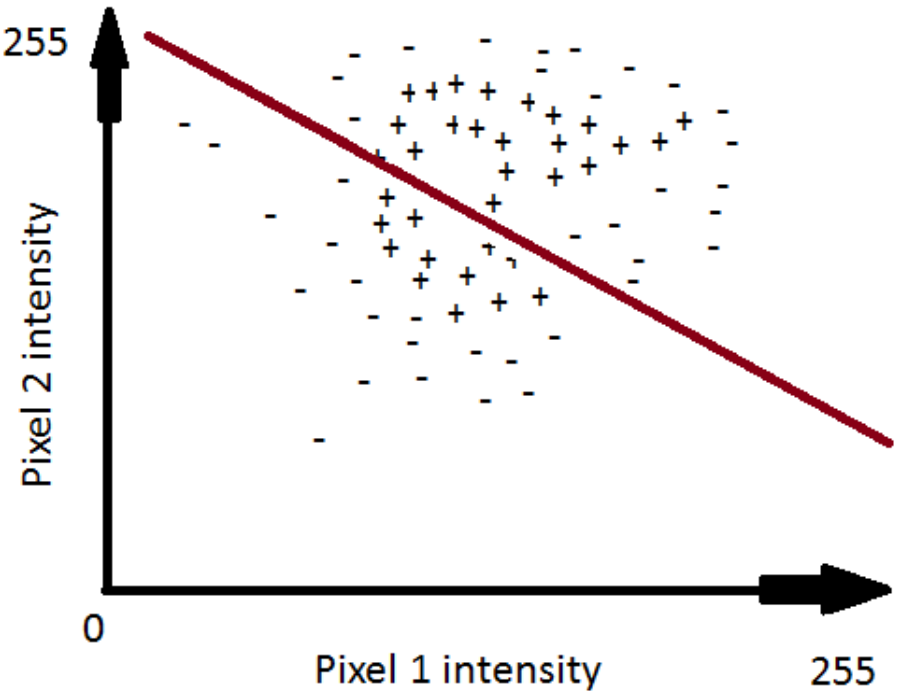
- Digit recognition



Predicting with logistic regression might not be a good idea....

(Highly) non-linear hypothesis

- Digit recognition



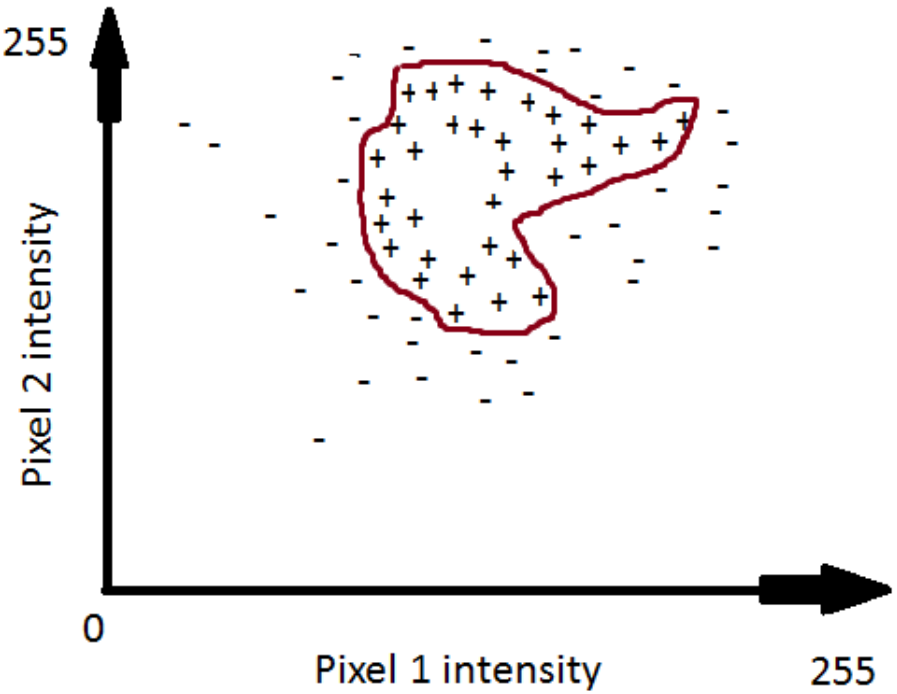
+ : eight

-: no eight

$$P(Digit = 8) = \frac{1}{1+e^{-\left(\beta_0+\beta_1 \text{Pixel}_{i,1}+\beta_2 \text{Pixel}_{i,2}\right)}}$$

(Highly) Non-linear hypothesis

- Digit recognition



We need something highly non-linear...

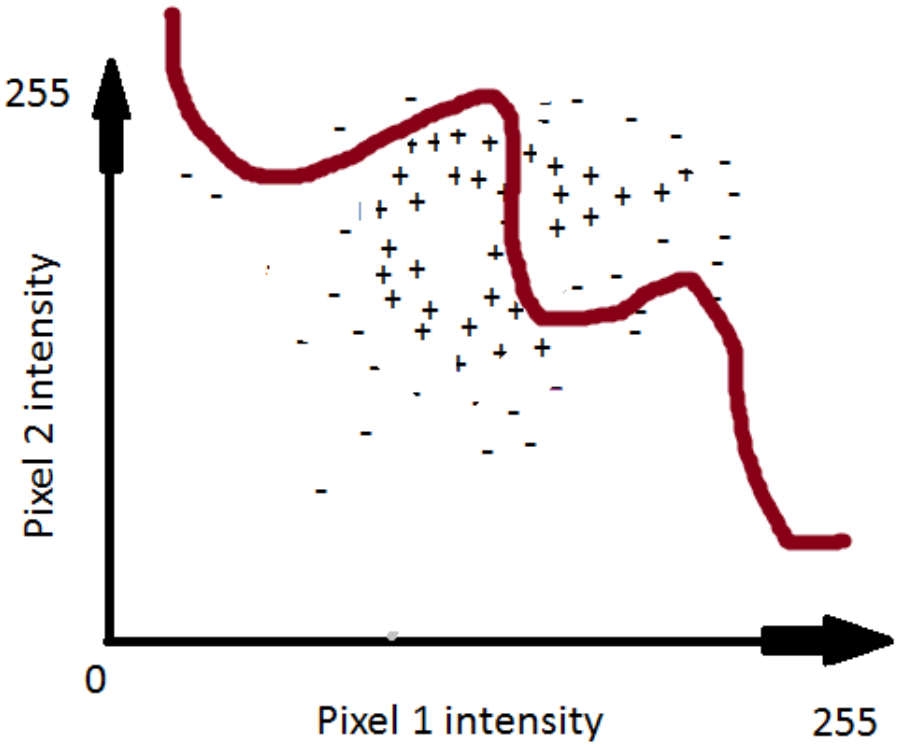
High dimensional problem – highly nonlinear

- Fitting with a polynomial?
- 324 pixels
- Quadratic polynomials: 52,000 terms.
- Quadratic might not be flexible enough.

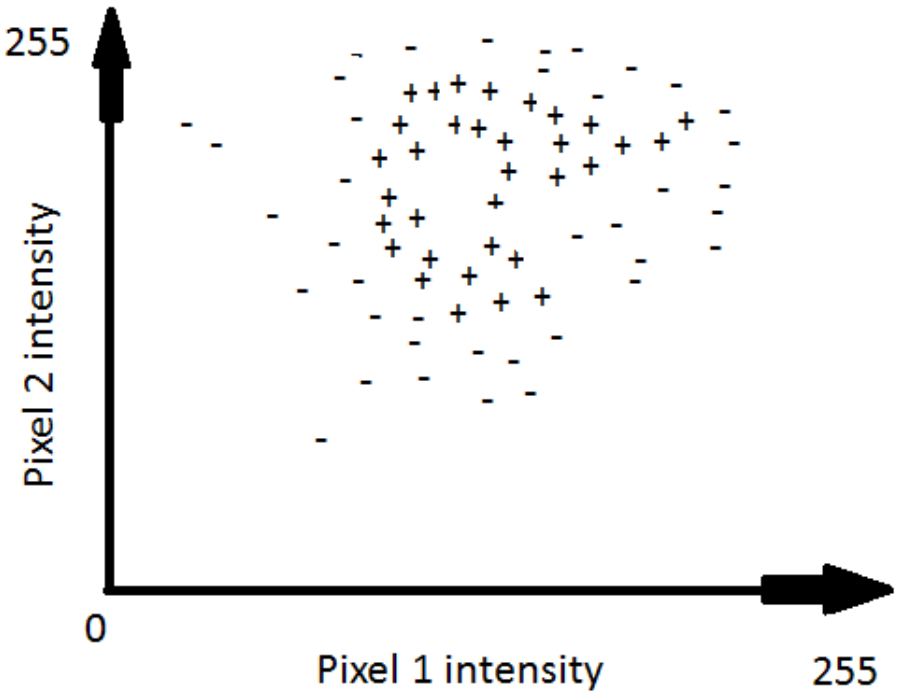
We have a problem that is characterized by...

- High dimensions (324 covariates)
- Highly non-linear (linear, quadratic, logit might not be enough)
- We need it to be computationally feasible

- 6th degree polynomial with only two pixels.
- Even six degrees might not be enough.
- And, again, this is only two out of the 324 pixels.

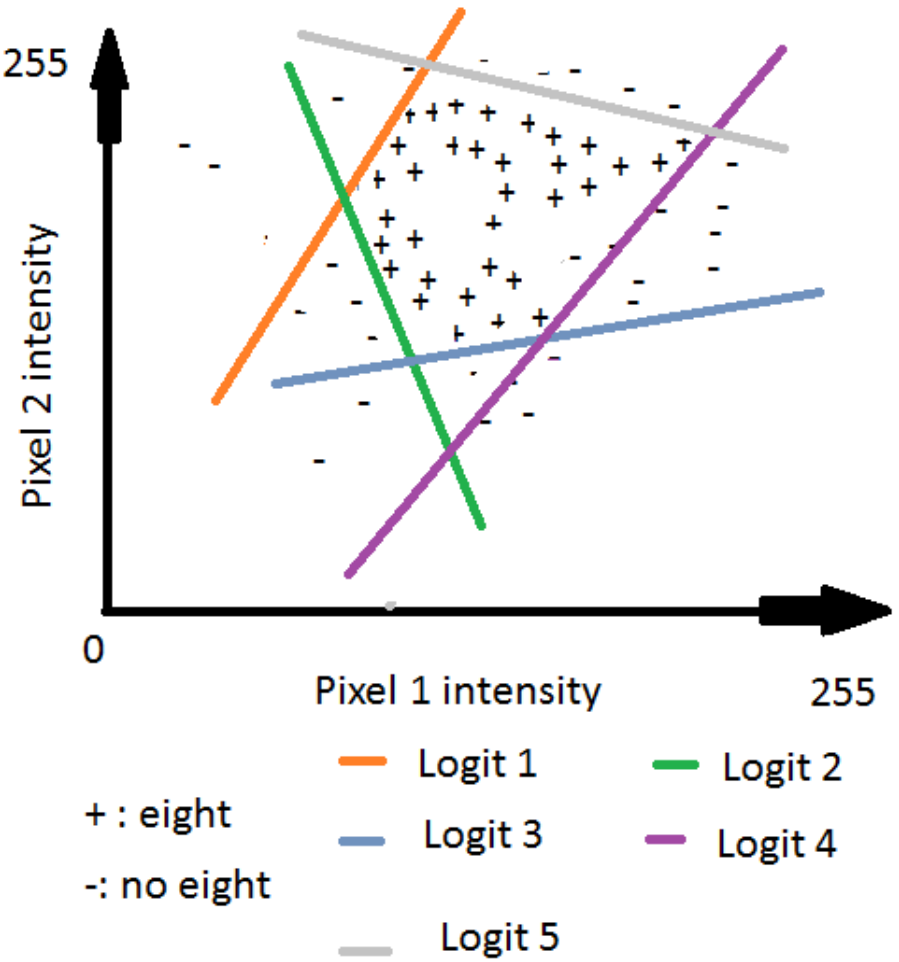


What if we use multiple logistic regressions to come up with a classification rule...

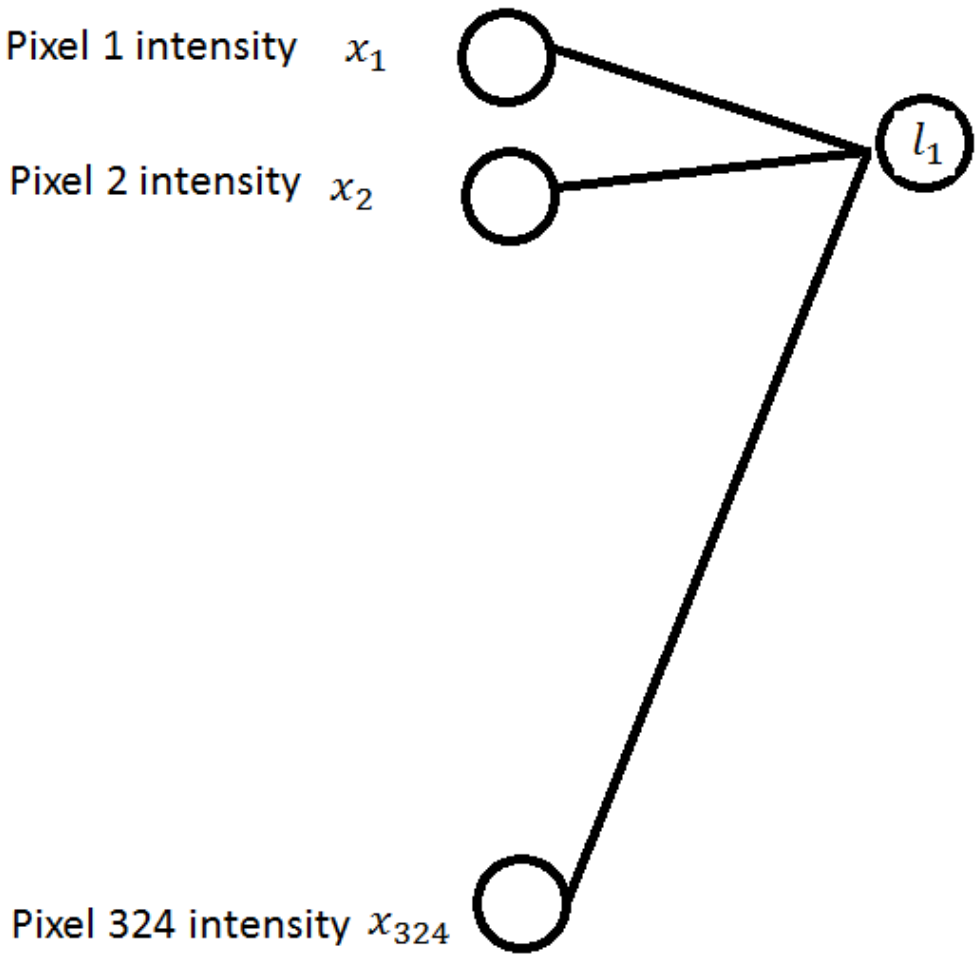


+ : eight

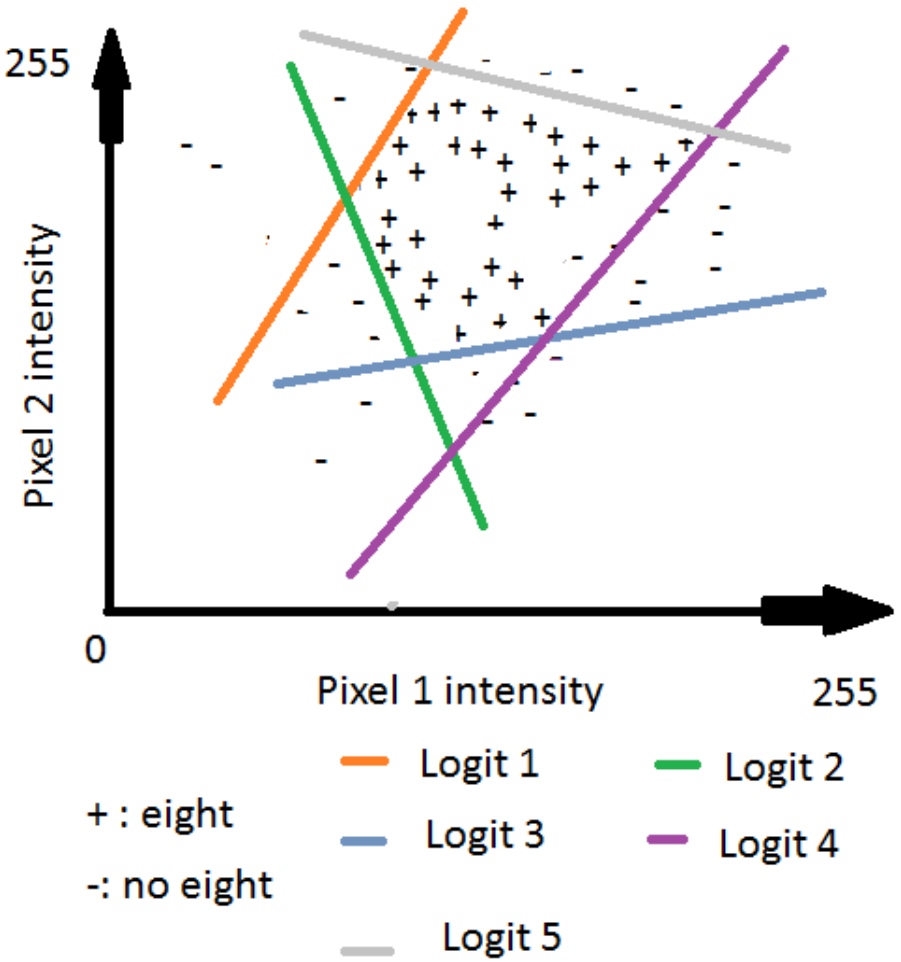
-: no eight

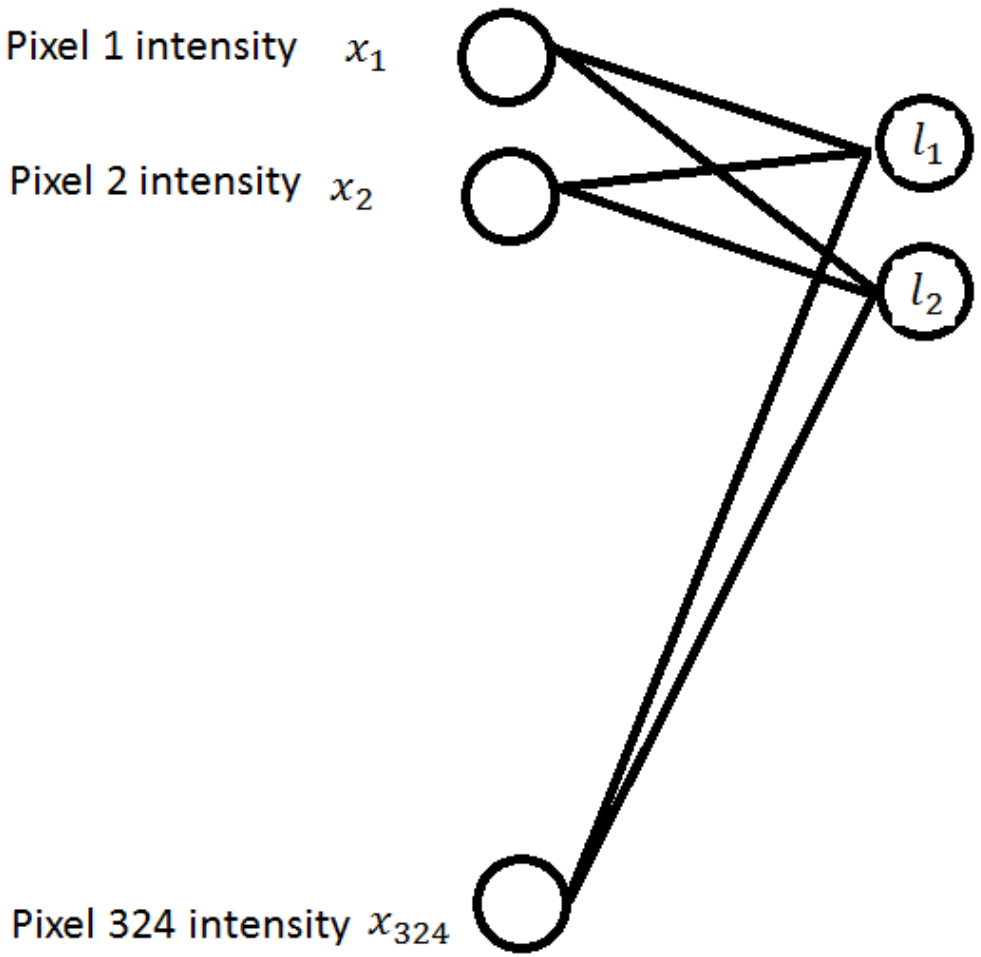


1. Estimate multiple logits: $(l_1, l_2, l_3, l_4, l_5)$

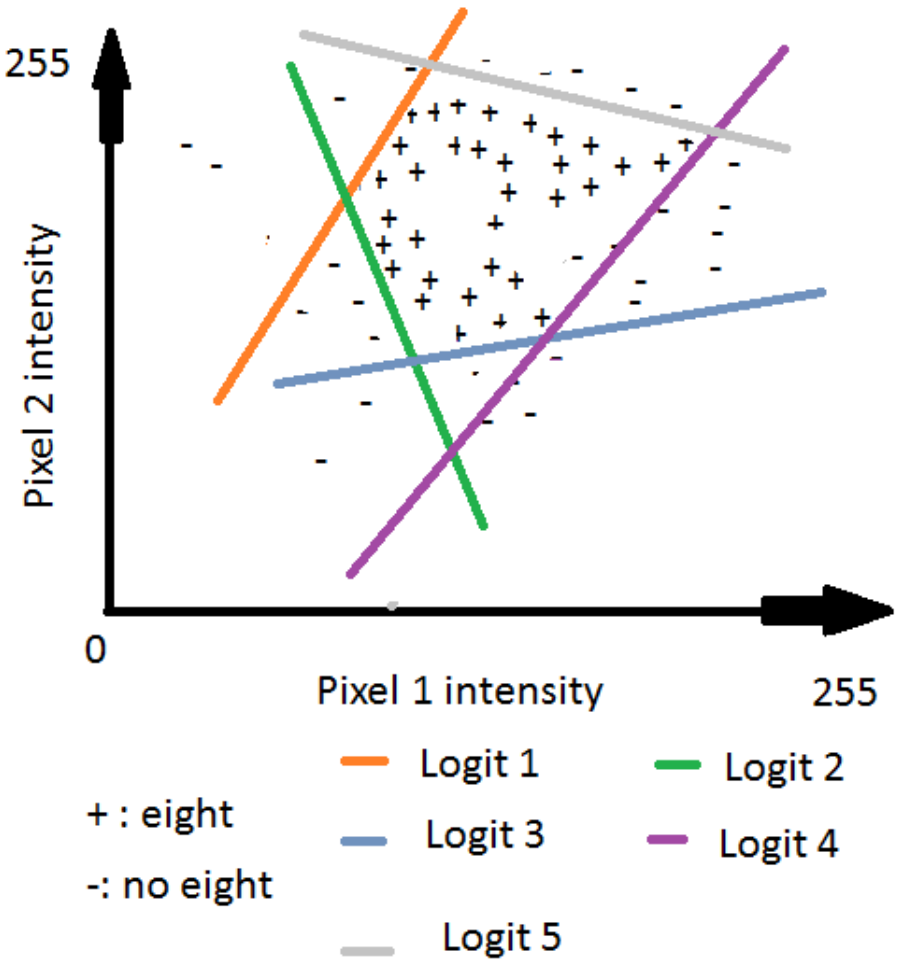


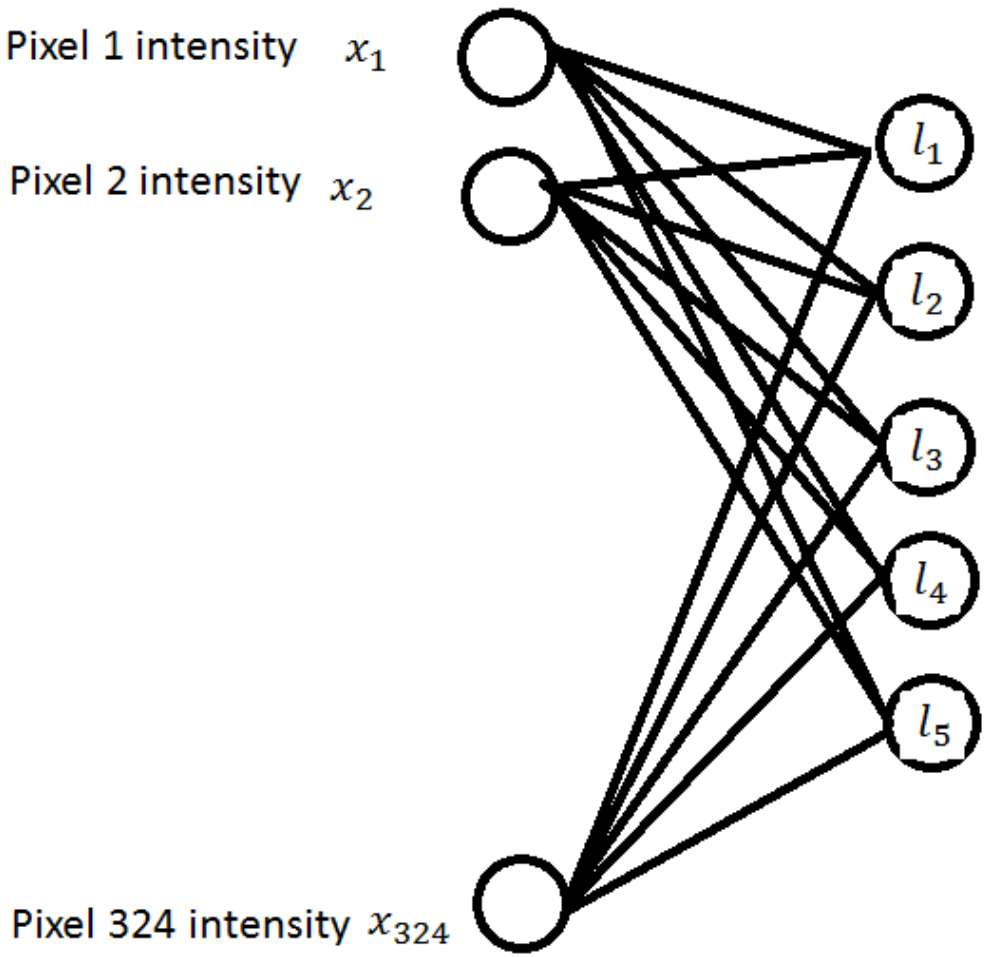
Logistic regression 1: $l_1 = \frac{1}{1+e^{-\left(\beta_{0,1}+\beta_{1,1}\text{Pixel}_{i,1}+\beta_{2,1}\text{Pixel}_{i,2}\right)}}$


255
Pixel 2 intensity
0
Pixel 1 intensity
255
Logit 1
Logit 2
Logit 3
Logit 4
Logit 5
+ : eight
-: no eight



Logistic regression 2: $l_2 = \frac{1}{1+e^{-\left(\beta_{0,2}+\beta_{1,2}\text{Pixel}_{i,2}+\beta_{2,2}\text{Pixel}_{i,2}\right)}}$


255
Pixel 2 intensity
0
Pixel 1 intensity
255
Logit 1
Logit 2
Logit 3
Logit 4
Logit 5
+ : eight
-: no eight



Logistic regressions k=1-5: $l_k = \frac{1}{1+e^{-\left(\beta_{0,k}+\beta_{1,k}\text{Pixel}_{i,2}+\beta_{2,k}\text{Pixel}_{i,2}\right)}}$

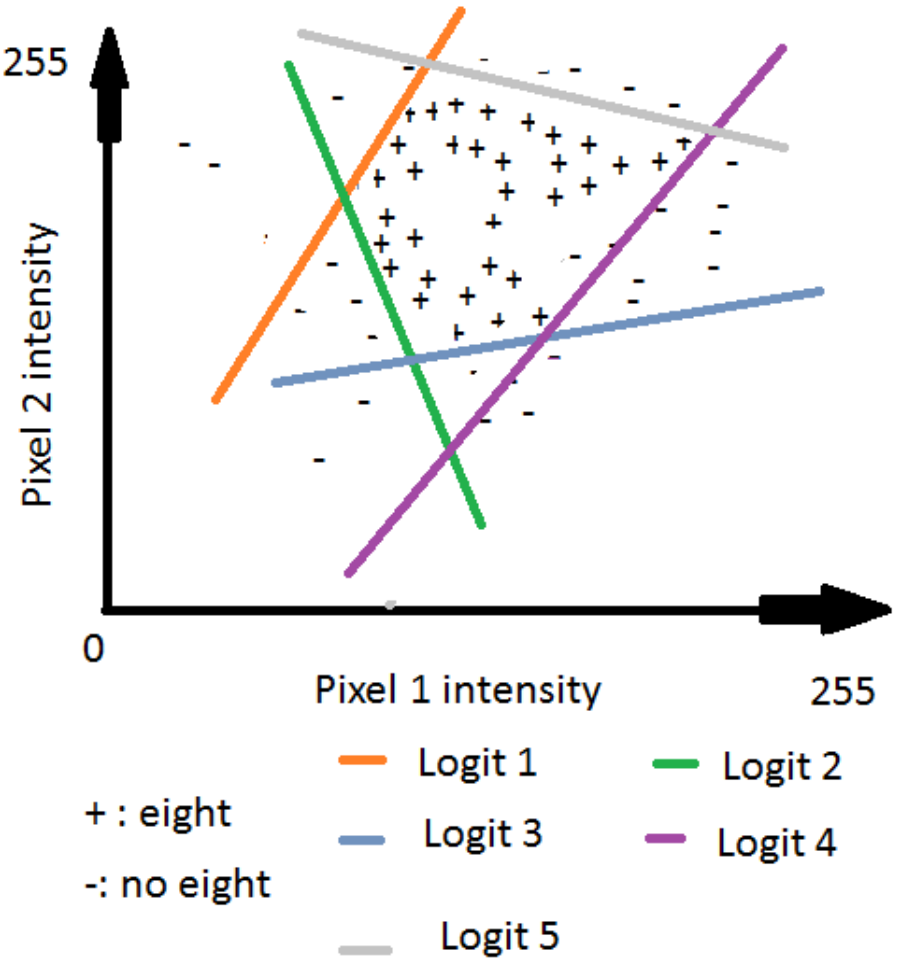
$$I_1 = \frac{1}{1+e^{-\left(\beta_{0,1}+\beta_{1,1}\text{Pixel}_{i,1}+\beta_{2,1}\text{Pixel}_{i,2}\right)}}$$

$$I_2 = \frac{1}{1+e^{-\left(\beta_{0,2}+\beta_{1,2}\text{Pixel}_{i,1}+\beta_{2,2}\text{Pixel}_{i,2}\right)}}$$

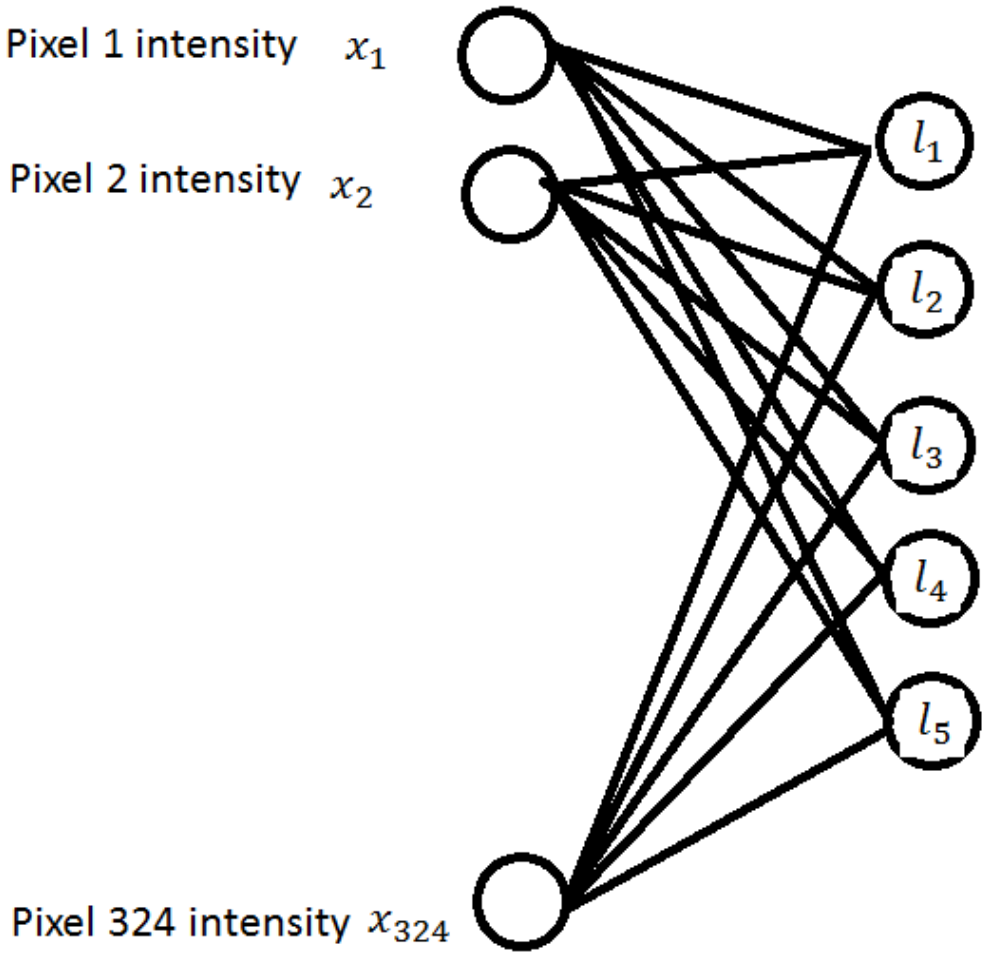
$$I_3 = \frac{1}{1+e^{-\left(\beta_{0,3}+\beta_{1,3}\text{Pixel}_{i,1}+\beta_{2,3}\text{Pixel}_{i,2}\right)}}$$

$$I_4 = \frac{1}{1+e^{-\left(\beta_{0,4}+\beta_{1,4}\text{Pixel}_{i,1}+\beta_{2,4}\text{Pixel}_{i,2}\right)}}$$

$$I_5 = \frac{1}{1+e^{-\left(\beta_{0,5}+\beta_{1,5}\text{Pixel}_{i,1}+\beta_{2,5}\text{Pixel}_{i,2}\right)}}$$

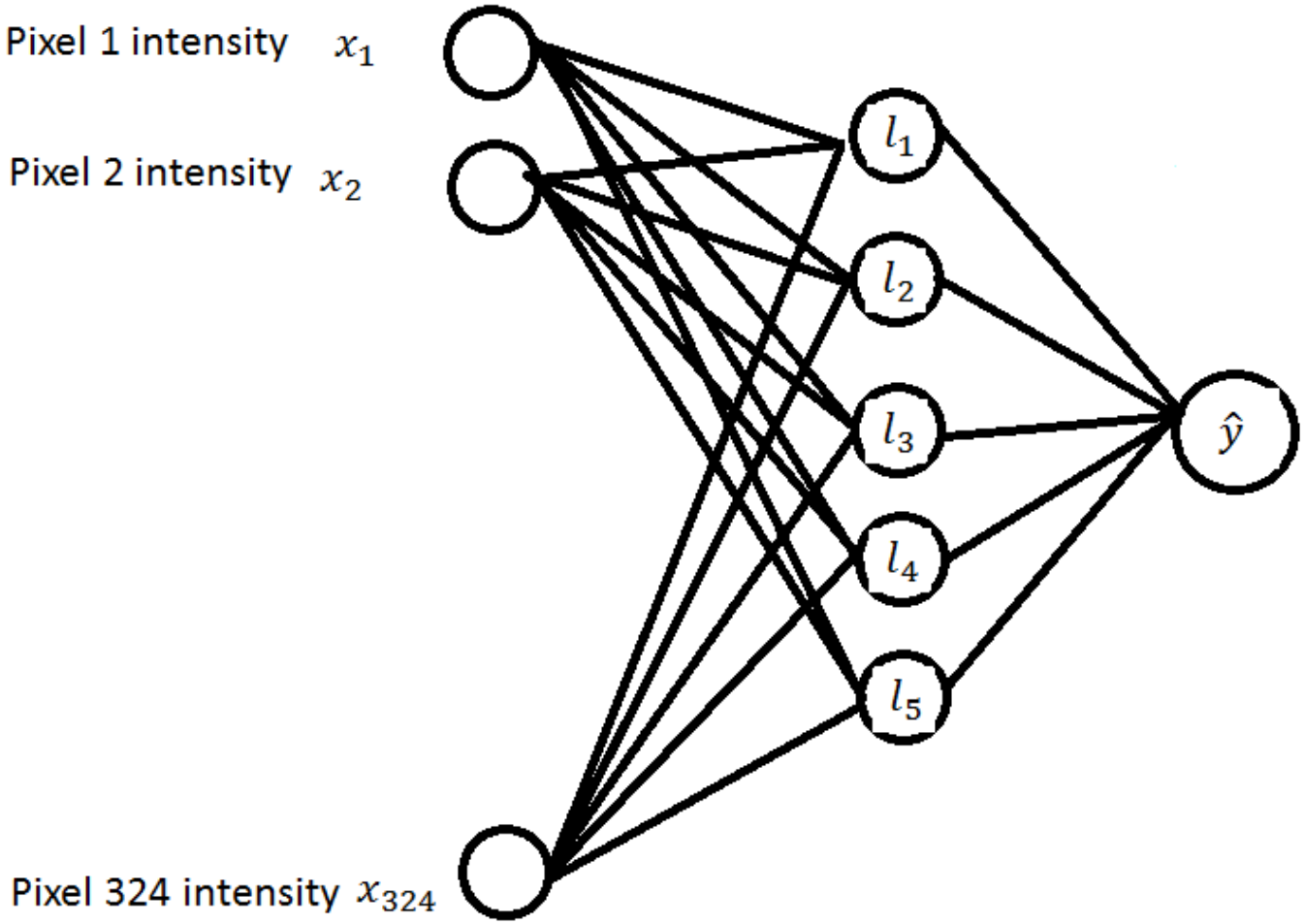


1. Estimate multiple logits: $(l_1, l_2, l_3, l_4, l_5)$
2. Combine them logits in another logistic regression to predict!



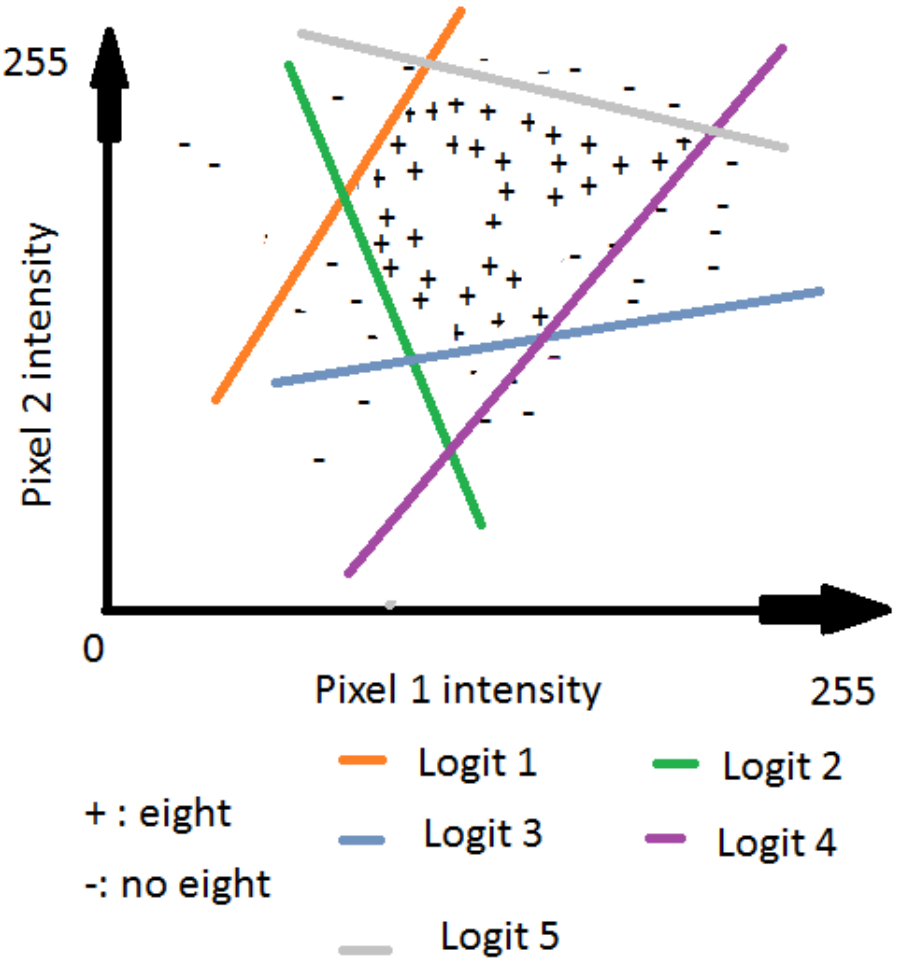
Logistic regressions k=1-5: $l_k = \frac{1}{1+e^{-\left(\beta_{0,k}+\beta_{1,k}\text{Pixel}_{i,2}+\beta_{2,k}\text{Pixel}_{i,2}\right)}}$

Additional layer of logistic regression: Predict 8 if $\frac{1}{1+e^{-(\alpha_0+\alpha_1 l_1+\alpha_2 l_2+\ldots\alpha_5 l_5)}} > 0.5$

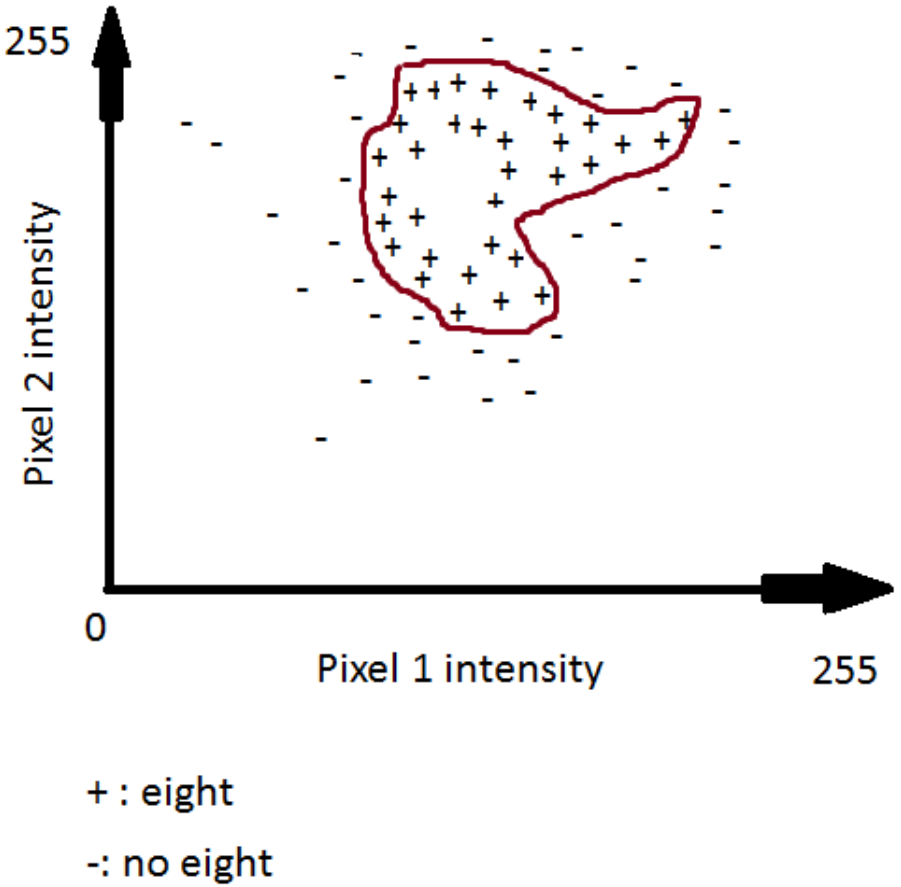


Logistic regressions k=1-5: $l_k = \frac{1}{1+e^{-\left(\beta_{0,k}+\beta_{1,k}\text{Pixel}_{i,2}+\beta_{2,k}\text{Pixel}_{i,2}\right)}}$

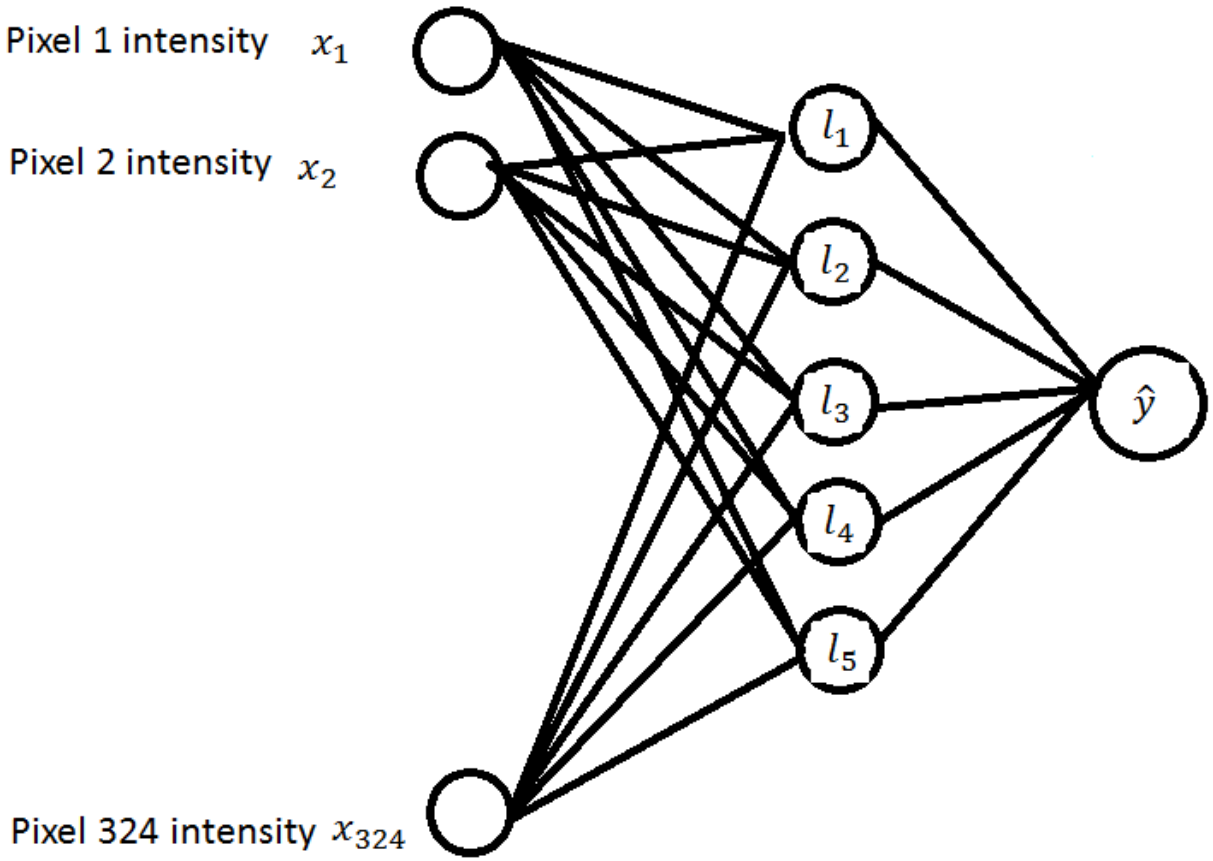
Additional layer of logistic regression $\frac{1}{1+e^{-(\alpha_0+\alpha_1 l_1+\alpha_2 l_2+\ldots\alpha_5 l_5)}} > 0.5$



1. Estimate multiple logits: $(l_1, l_2, l_3, l_4, l_5)$
2. Decision rule: Predict 8 according to a rule of five logits

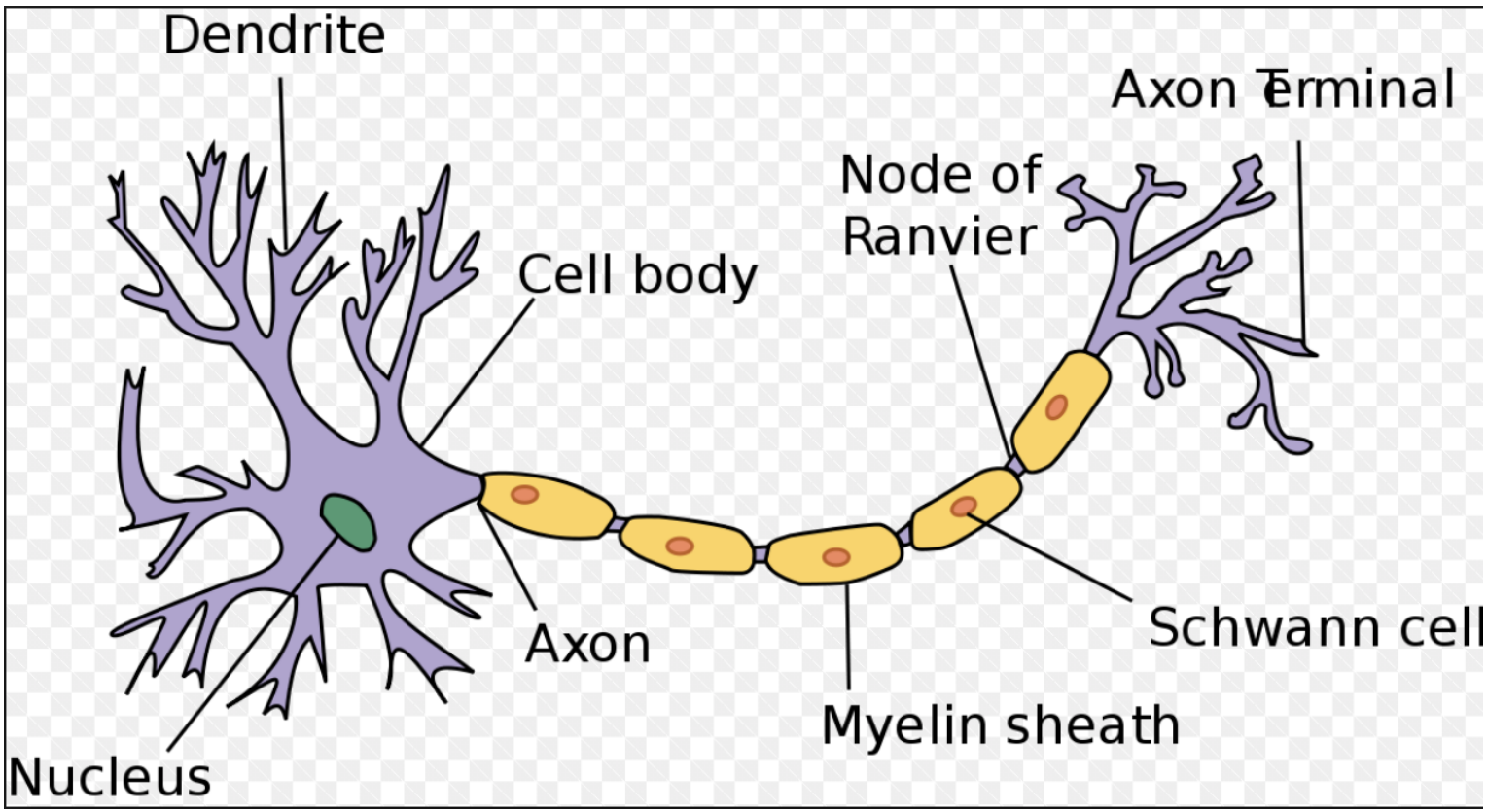


1. Estimate multiple logits: $(l_1, l_2, l_3, l_4, l_5)$
2. Decision rule: Predict 8 according to a rule of five logits



1. **Input layer**: 324 pixels.
2. one **hidden** layer (depth of network)
3. five neurons in hidden layer (width of layer)

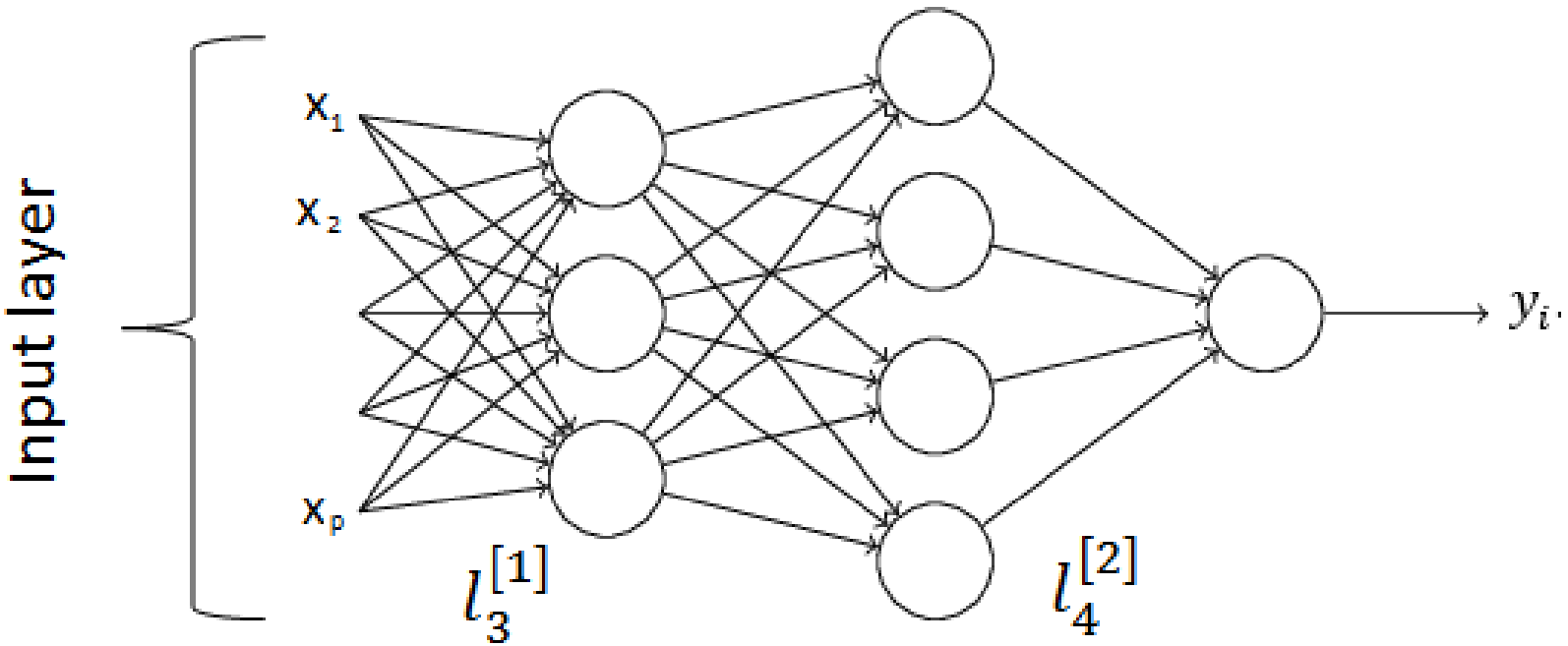
It does look like a neuron



Overly simplistic example.

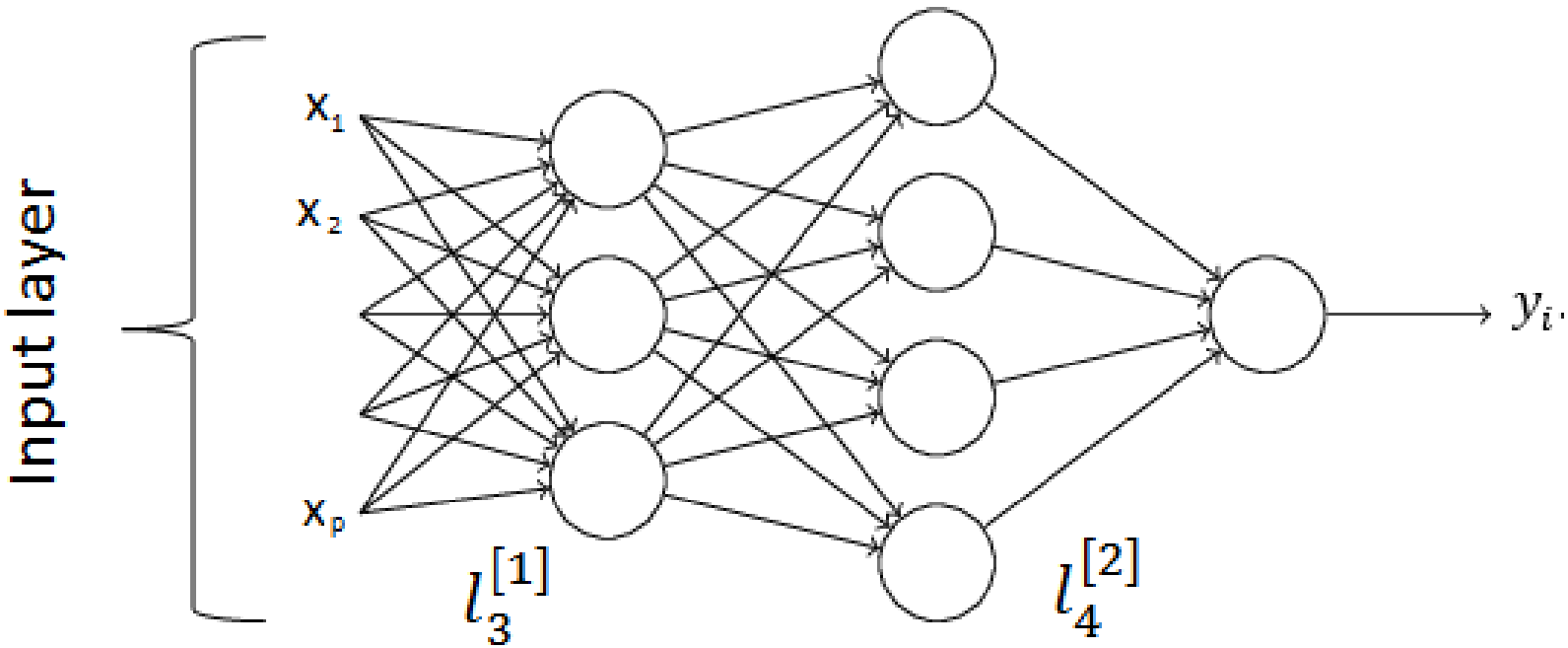
- 324 pixels (not two).
- Face recognition: thousands of pixels+colors (three dimensions in each pixel).
- Highly non-linear. What can you do in such cases?

- One solution: add more layers



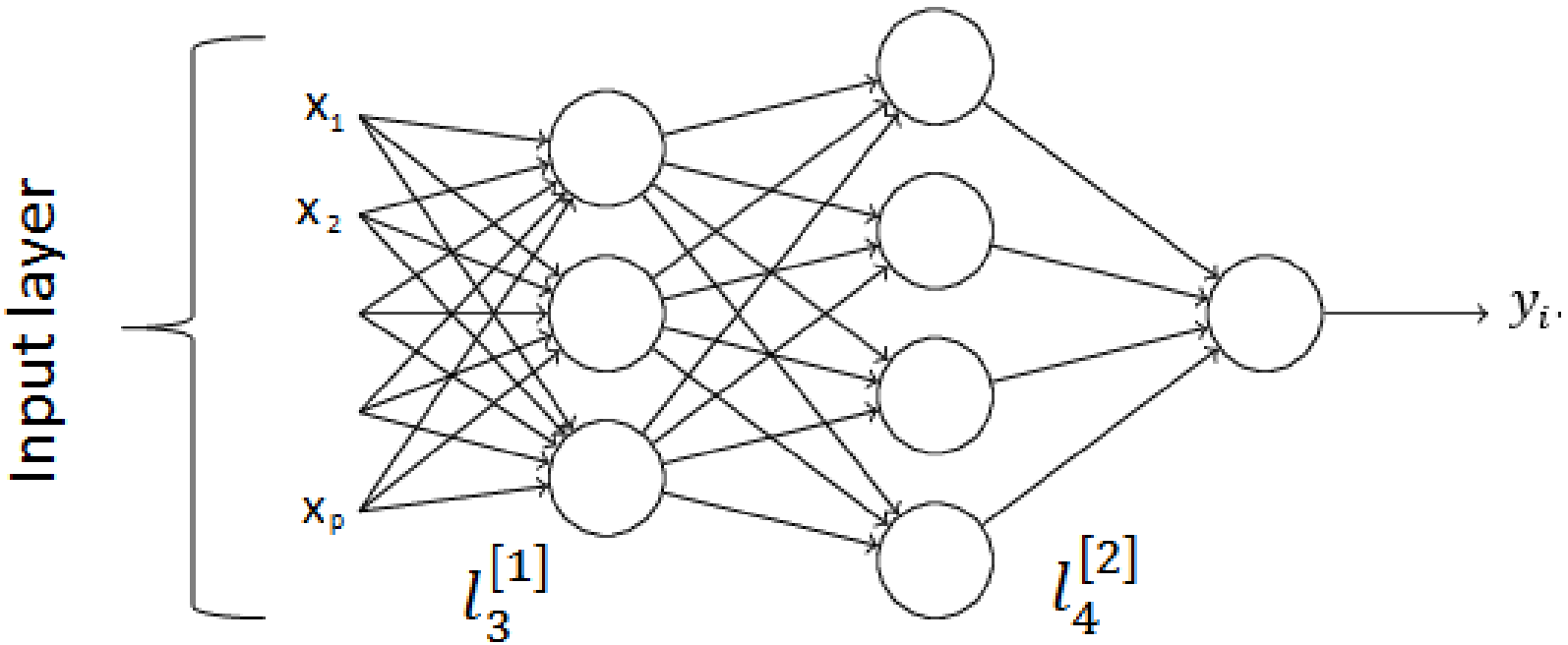
- Input layer: $x_1, ...x_p$
- Hidden layer 1: three logistic $(l_1^{[1]}, l_2^{[1]}, l_3^{[1]})$ regressions.
- $l_3^{[1]} = \frac{1}{1+e^{-\left(\beta_{0,3}^{[1]} + \beta_{1,3}^{[1]} x_{i,1} \cdots + \beta_{p,3}^{[1]} x_{i,p}\right)}}$

- Hidden layer 2



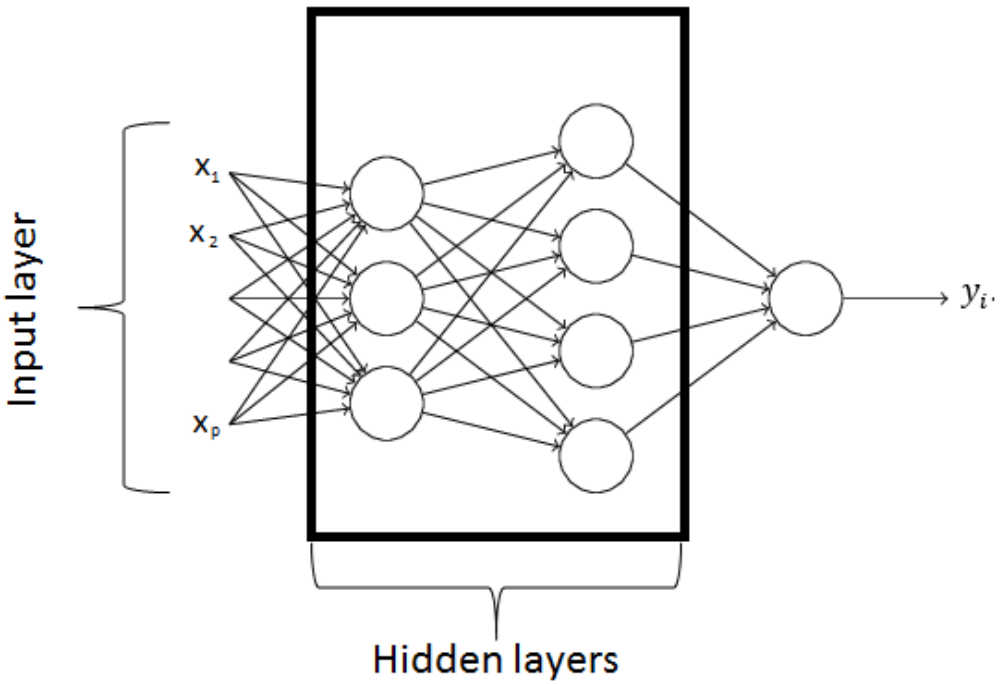
- Hidden layer 2: four logistic ($l_1^{[2]}, l_2^{[2]}, l_3^{[2]}, l_4^{[2]}$) regressions.
- $l_4^{[2]} = \frac{1}{1+e^{-\left(\beta_{0,4}^{[2]}+\beta_{1,4}^{[2]} l_{i,1}^{[1]} \beta_{1,4}^{[2]} l_{i,2}^{[1]}+\beta_{4,4}^{[2]} l_{i,3}^{[1]}\right)}}$

- Output layer

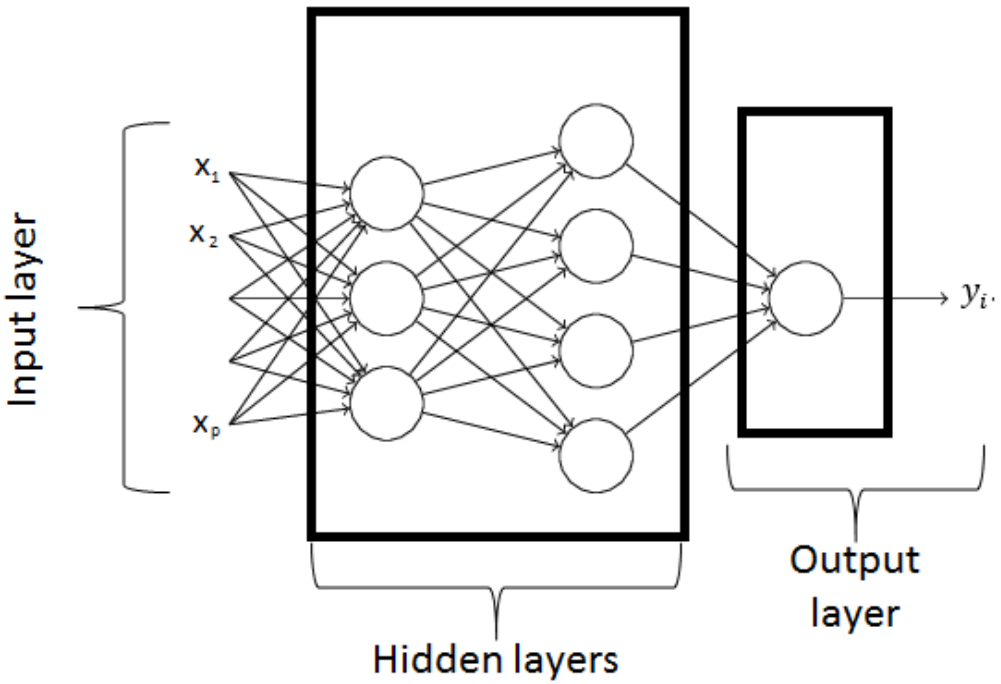


- One logistic regression
- $\hat{P}(y_i) = \frac{1}{1+e^{-\left(\beta_0^{[3]}+\beta_1^{[3]}l_{i,1}^{[2]}+\beta_2^{[3]}l_{i,2}^{[2]}+\beta_3^{[3]}l_{i,3}^{[2]}+\beta_4^{[3]}l_{i,4}^{[2]}\right)}}$

- One solution: add more layers

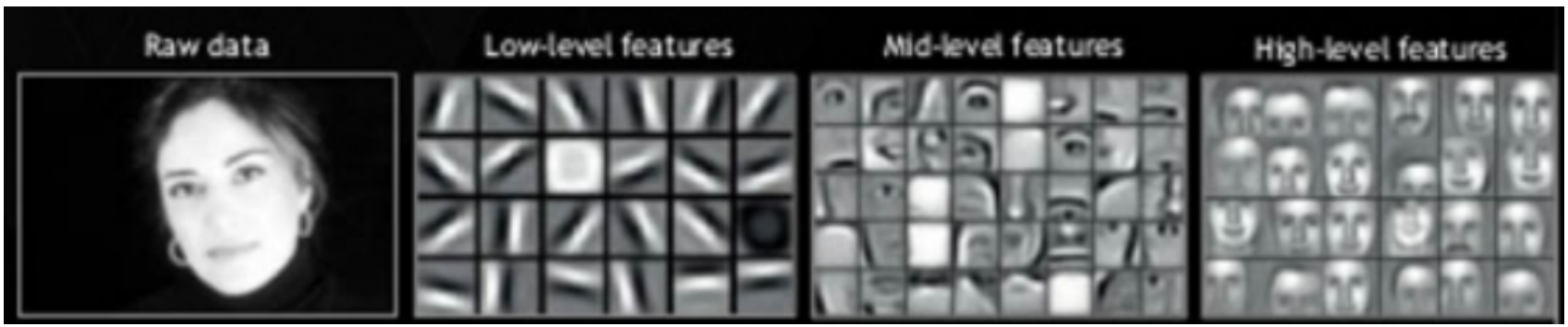


- One solution: add more layers



- Deep learning: traditionally two or more hidden layers

- In general, layers in neural networks are hard to interpret
- Sometimes each layer identifies distinctive features...



(Image belongs to NVIDIA)

- Deep learning: traditionally two or more hidden layers

1. What are neural networks?
   Framework consisting of a collection of neurons in multiple layers

   Neurons in layer $l+1$ use neurons in layer l as input

2. Why do we need them?

   We need them because of their ability to perform well in high-dimensional classification-optimization problems

   Universal Approximation Theorem (Gybenko, 1989) $\rightarrow$ rich enough network could, in principle, "approximate most functions".

   Challenge is in the estimation

Activation functions

- So far we have seen a neural network with the sigmoid (logistic) **activation function**.
- However, neural networks often use other family of functions...
- Different neurons can have different activation functions

Activation functions in neural networks

- Sigmoid function: $f(z) = \frac{1}{1+e^{-z}}$
- Tanh $f(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- ReLu (Rectified Linear Unit): $f(z) = \begin{cases} z \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$
- Leaky ReLu (Rectified Linear Unit):
  $f(z) = \begin{cases} z \text{ if } z \geq 0 \\ \alpha z \text{ otherwise } ; \alpha \in [0, 1] \end{cases}$

- So far, we have seen four activation functions which are, arguably, the most popular in deep learning.
- However, there is a large set of functions used. Check more in this link
- In general, we want activation functions that are differentiable almost everywhere (why? it will be clear in a few slides)

1. What are neural networks?
   Framework consisting of a collection of neurons in multiple layers

   Each neuron is the result of an activation function applied to a linear transformation

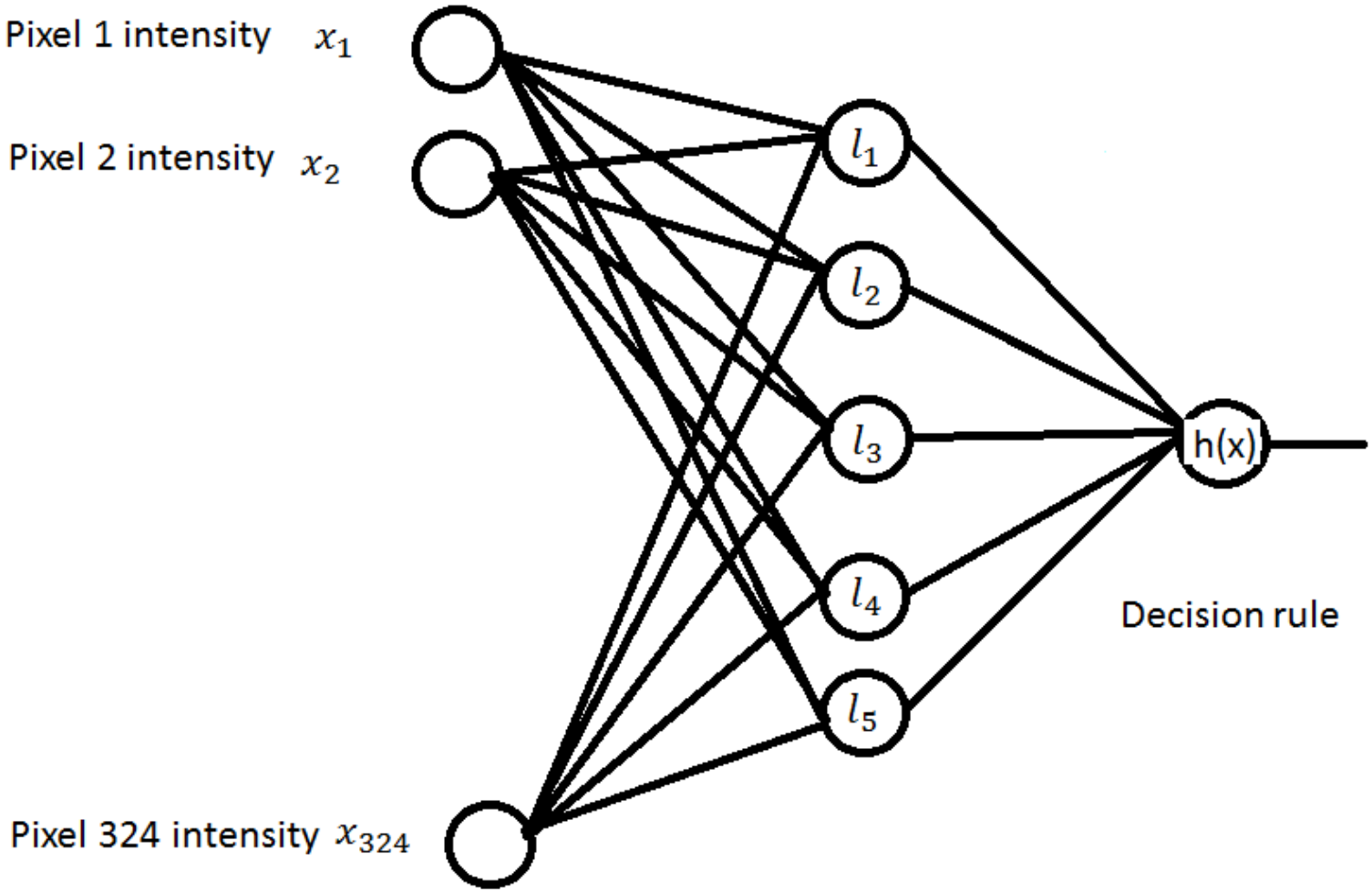
   Neurons in layer $l+1$ use neurons in layer l as input

2. Why do we need them?

   We need them because of their ability to perform well in high-dimensional classification-optimization problems

   Universal Approximation Theorem (Gybenko, 1989) $\rightarrow$ rich enough network could, in principle, "approximate most functions".

   Challenge is in the estimation



1. Estimate $l_k = \frac{1}{1+e^{-\left(\beta_k + \beta_{1,k}\text{Pixels}\right)}}$ for $k = 1...5$

2. Decision rule: Predict 8 if $\frac{1}{1+e^{-(\alpha_0+\alpha_1 l_1+\alpha_2 l_2+\ldots\alpha_5 l_5)}} > 0.5$

Estimation of neural networks

- So far, intuition of neural networks is (hopefully) straightforward
- However, how do we estimate the coefficients of a neural network?
- First, we perform **Feature normalization**:
  - What does it mean that a pixel has intensity of 274 or of 0?
  - We do not care about the actual value but about the relative level...
  - How intense is the first pixel of image 1 with respect to all other images?
  - Normalize pixel 1 for each observation: $\frac{\text{pixel}_{1,i} - \bar{\text{pixel}}_1}{\text{Std. deviation (pixel)}_1}$
  - Normalized pixel 1=-0,1 $\rightarrow$ pixel 1 is 10% of a standard deviation less intense than the average.
  - Alternatively $\frac{\text{pixel}_{1,i} - \min \text{pixel}_1}{\max \text{pixel}_1 - \min \text{pixel}_1} \rightarrow pixel_{1,i} \in [0, 1]$
- ENOUGH! Let us go to R and estimate our first neural network

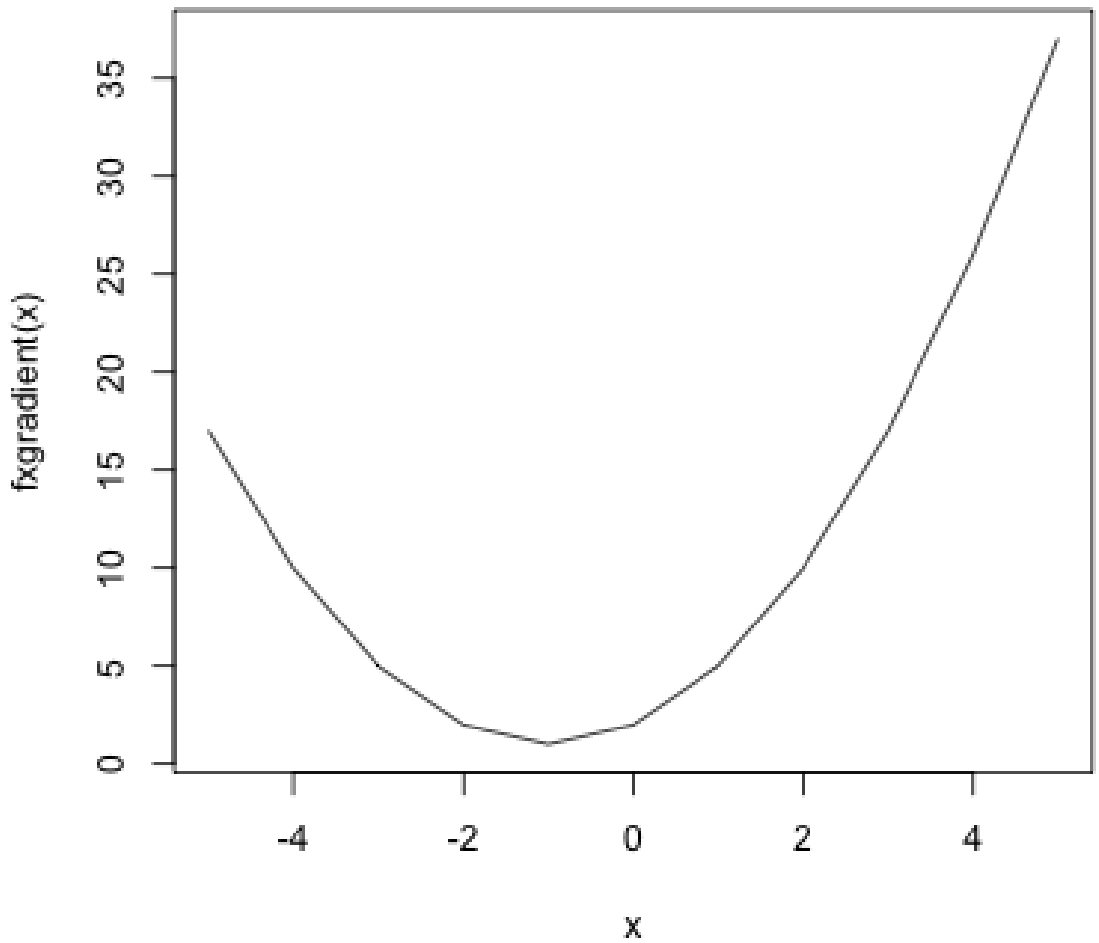
- **In estimation, our goal is to minimize a cost function**
- In eight digit recognition
  - $(324 + 1) \times 5$ coefficients in hidden layer
  - $5 + 1$ coefficients in output layer
  - 1,631 coefficients
- It is not rare to see neural networks where you have to estimate $\geq$5,000 parameters
- Optimization problem in +5,000 dimensions...

- High dimensional problem: how do we overcome overfitting?
  - **We can use a regularization parameter (as in ridge) to overcome overfitting.**
  - Find $(\beta's,)$ that minimize:

    $J(\beta, \alpha) = \text{Cost} + \lambda \sum_{\beta} \beta^2$

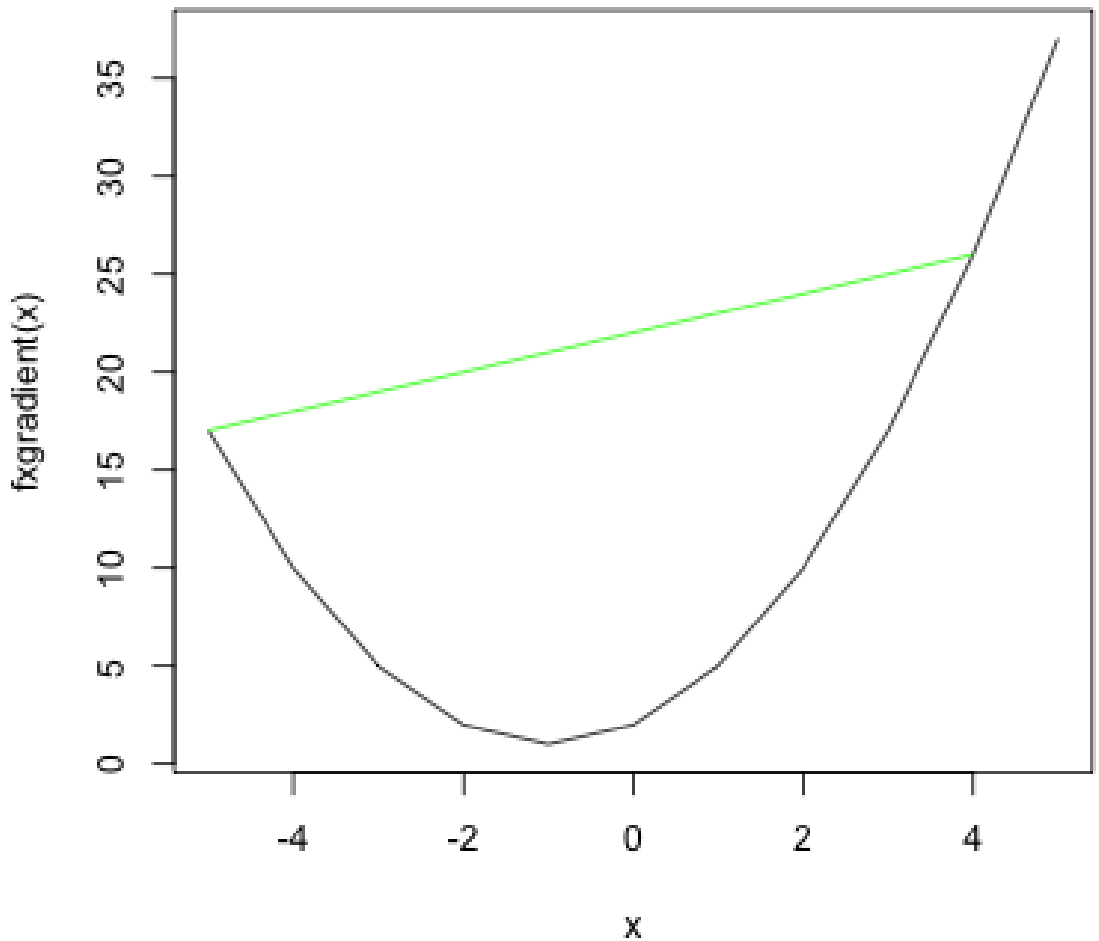
- Estimation problem: minimize a high dimensional cost function
- In eight digit classification problem in 1,631 dimensions.
- How do we know which parameters minimize cost function?
- The most common approach is to use a family of **gradient descent** techniques **backward propagation**

Intuition of gradient descent. Want to minimize function $f(x)$:



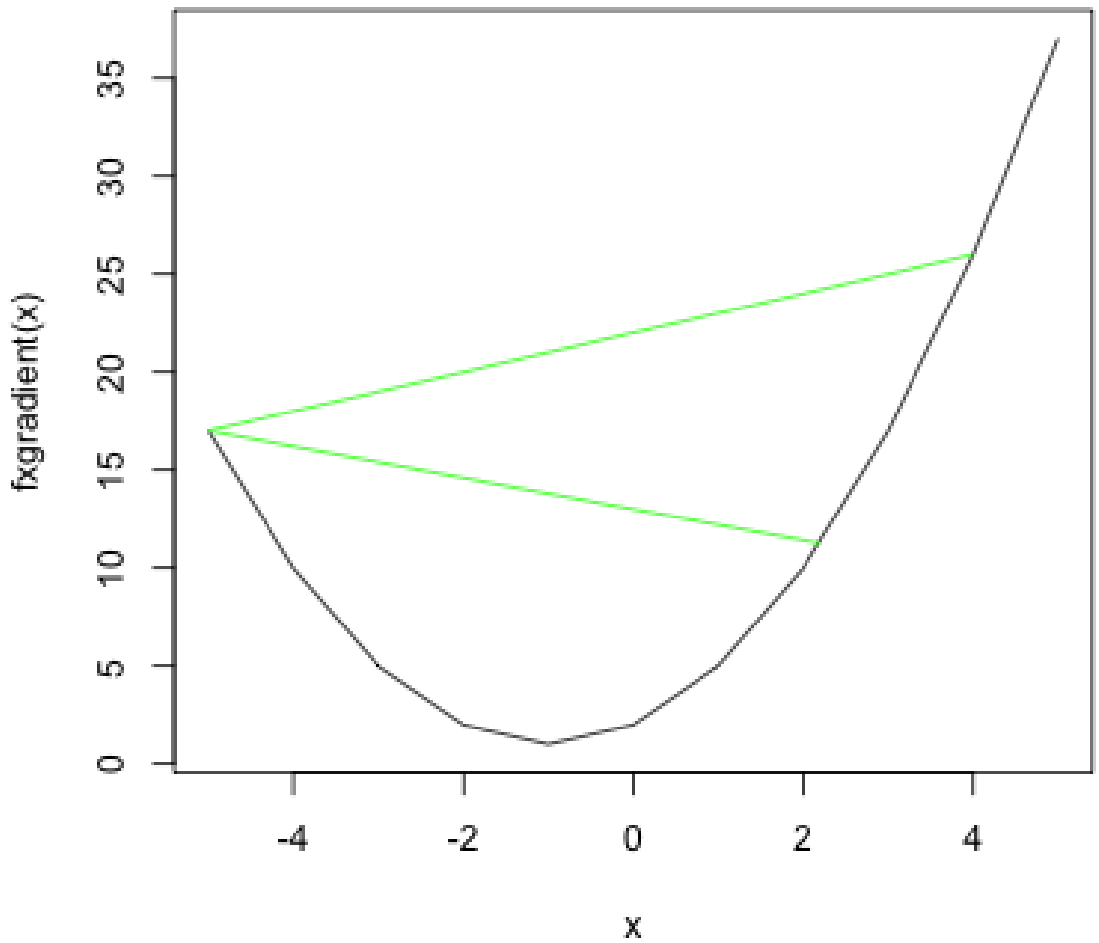
- Obtain derivative (rate of change) with respect to x

Intuition of gradient descent. Want to minimize function $f(x)$:



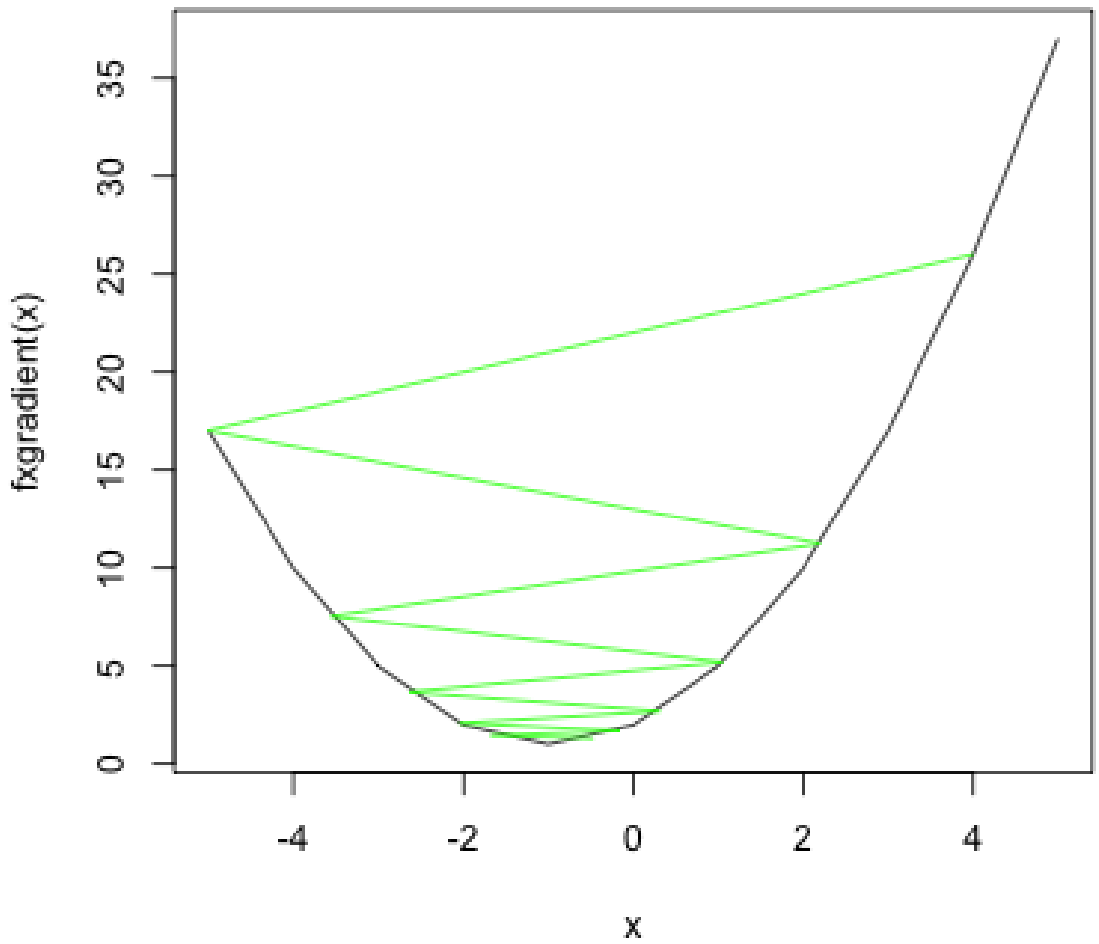
- Derivative is positive $\rightarrow$ decrease x

Intuition of gradient descent. Want to minimize function $f(x)$:



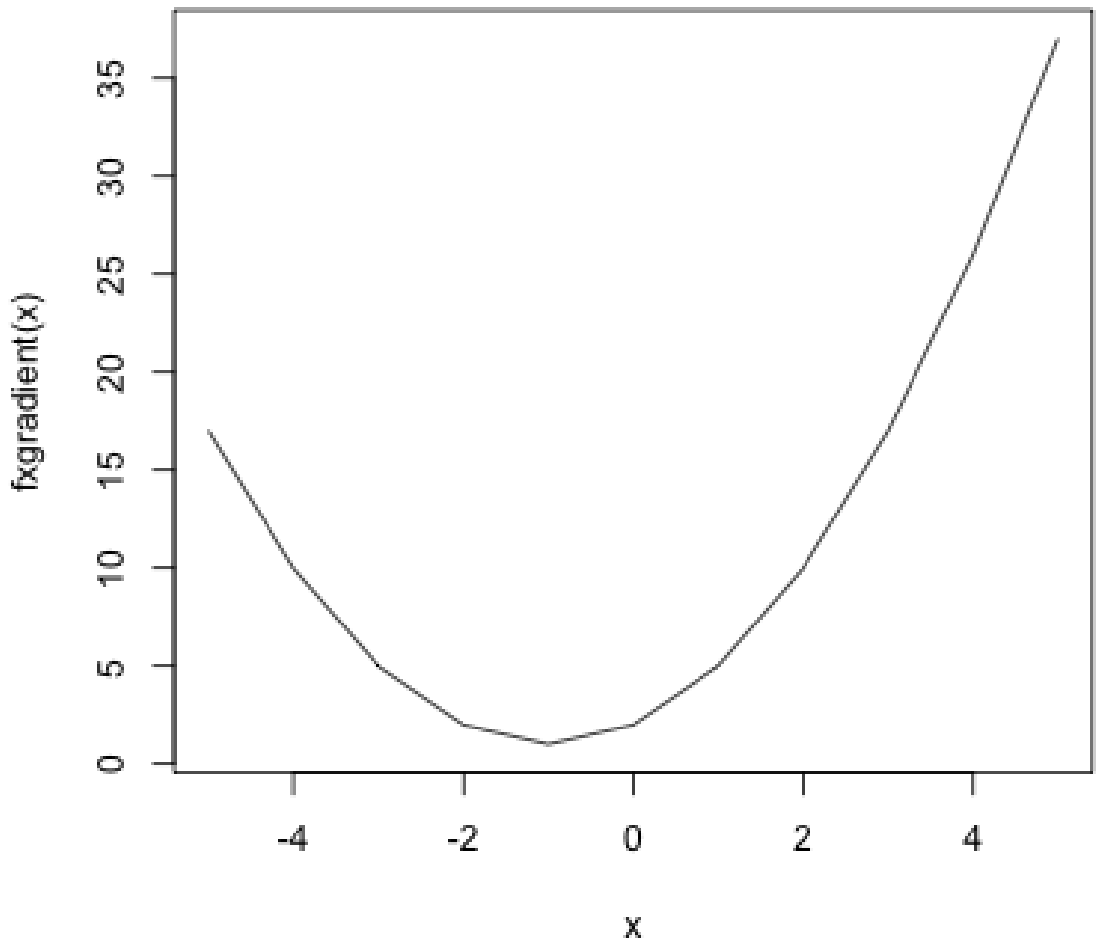
- Derivative is negative $\rightarrow$ increase x

Intuition of gradient descent. Want to minimize function $f(x)$:



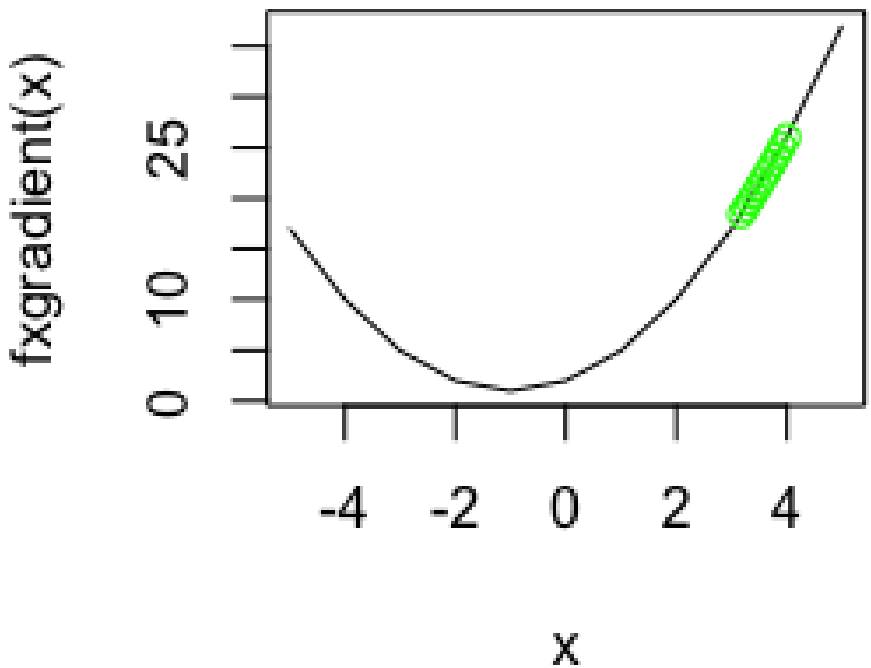
- Repeat until convergence

Intuition of gradient descent. Want to minimize function $f(x)$:



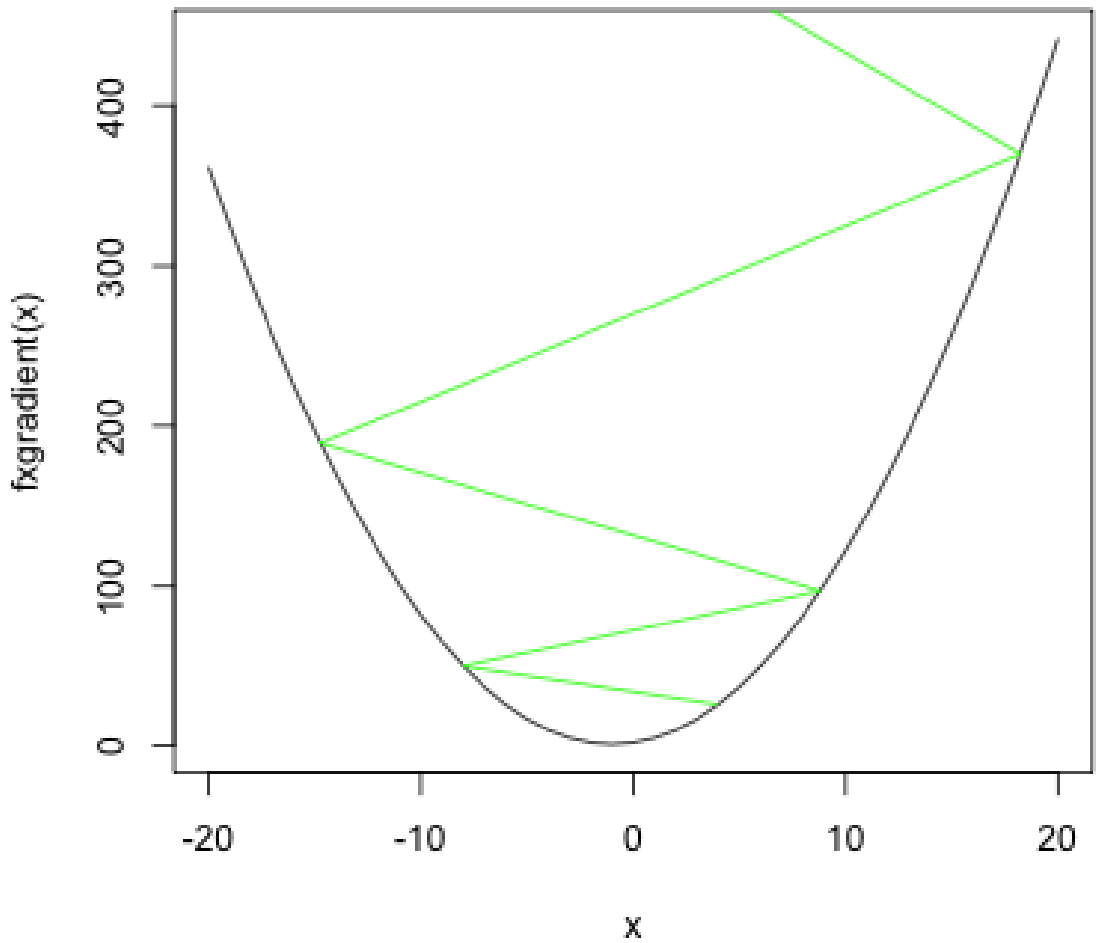
- Learning rate (α) how fast increase/decrease variable

Intuition of gradient descent. Want to minimize function $f(x)$:



- Small (α) $\rightarrow$ slow convergence

Intuition of gradient descent. Want to minimize function $f(x)$:



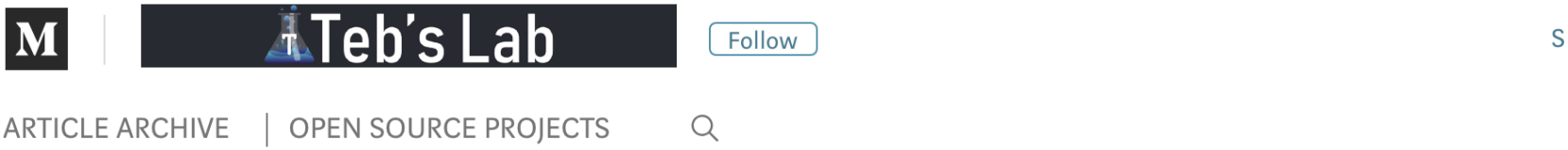
- Large (α) $\rightarrow$ might not convergence

How do we estimate a neural network?:
Our goal is to minimize a cost function.

1. First, we perform **feature normalization** on our data.
2. Define characteristics of the network:
   - hidden layers (depth of network)
   - width of layers (number of neurons)
   - Activation functions (ReLu, Tanh, sigmoid, leaky relu)
3. Define characteristics of optimization algorithm
   - Type of algorithm (gradient descent; backpropagation)
   - Learning rate (learnmax, min, etc.).
4. Initialize parameters or weights (traditionally, random initialization).
5. Initialize optimization routine. Perform forward propagation and backpropagation.
6. Stop optimization given a rule (# of iterations, improvement in cost function, etc).

We still have to solve a high dimenstional problem (often in +5,000 dimensions.)

GPU Parallelization is often useful when estimating neural networks



High Performance Computing is More Parallel Than Ever

The end of Moore's Law and the explosion of machine learning is driving growth in parallel computing, so what is it?

- Moore's law: "number of transistors on a chip doubles each year"
- Computing speed improves by a factor slightly less than 2 every year.
- Moore's law does not (seem) to hold anymore since 2017.
- CPU's are not improving at that speed. Limit is **temperature**.
- Parallel computing is the approach to circumvent the end of Moore's law.

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Traditional approach: sequential:

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Traditional approach: sequential:

$$1.(y_1 - \hat{y}_1)^2 = e_1^2$$

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Traditional approach: sequential:

$$1.(y_1 - \hat{y}_1)^2 = e_1^2$$

$$2.(y_2 - \hat{y}_2)^2 = e_2^2$$

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Traditional approach: sequential:

$$1.(y_1 - \hat{y}_1)^2 = e_1^2$$

$$2.(y_2 - \hat{y}_2)^2 = e_2^2$$

$$\vdots$$

$$n.(y_n - \hat{y}_n)^2 = e_n^2$$

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- CPU parallelization. Divide tasks by # of cores (nc) in your PC.

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- CPU parallelization. Divide tasks by # of cores (nc) in your PC.

$$1.(y_1 - \hat{y}_1)^2 = e_1^2$$

$$nc.(y_2 - \hat{y}_2)^2 = e_{nc}^2$$

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- CPU parallelization. Divide tasks by # of cores (nc) in your PC.

$$1.(y_1 - \hat{y}_1)^2 = e_1^2$$

$$nc.(y_2 - \hat{y}_2)^2 = e_{nc}^2$$

$$nc + 1.(y_n - \hat{y}_n)^2 = e_{nc+1}^2$$
$$n.(y_n - \hat{y}_n)^2 = e_n^2$$

- What is parallelization?
- Suppose I want to compute Sum of Squared Errors

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- GPU parallelization. Divide tasks by # of cores (nc) in your GPU.

$$1.(y_1 - \hat{y}_1)^2 = e_1^2$$

$$nc.(y_2 - \hat{y}_2)^2 = e_{nc}^2$$

$$nc + 1.(y_n - \hat{y}_n)^2 = e_{nc+1}^2$$
$$n.(y_n - \hat{y}_n)^2 = e_n^2$$

- GPU parallelization: standard approach to estimate Neural Network.
- Neural Networks are "Embarassingly Parallel"
- Problem with GPU parallelization.... it is costly to code it (CUDA, a C-family language).
- However, several packages with GPU compatibility (Pytorch, tensorflow).

- Wages as a function of age and education.
  In R: lm(wage∼age+schooling,data=WageCSV)

- Wages as a function of age and education.
  In Cuda: over 100 lines of code.

```
//  Created by Rodrigo Azuero on 2/24/16.
//  Copyright (c) 2016 Rodrigo Azuero Melo. All right
//


#include <algorithm>
#include <ctime>
#include <cmath>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <math.h>
#include <sstream>
#include <string>
#include <stdlib.h>
#include <stdio.h>
#include <vector>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <gsl/gsl_sf_bessel.h>
#include <gsl/gsl_roots.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_math.h>
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/random/linear_congruential_engine.h>
#include <thrust/random/normal_distribution.h>
#include <unistd.h>
#include <nlopt.hpp>
using std::vector;
using namespace std;


__device__ __host__ float normpdfESTANDAR(float D){
        float term=D*D;
        float exp=-term;
        float num=expf(exp);
        float den=powf(2*3.141592,0.5);
        float ans=num/den;
```

```
//----------
//0.0. Wages
//------------
//=========================================================================================//


__device__ __host__  float F_predwageD(float bbeta0, float bbeta1, float bbeta2, float bbeta3,  float Schooling, float Age){
    //1. Defining the output
    float wage=bbeta0+bbeta1*Schooling+bbeta2*Age+bbeta3*Age*Age;
    return(wage);
}


//=========================================================================
//1.0. Likelihood of Wages
//=========================================================================
__host__ __device__ float F_likelihood_wageD(float bbeta0, float bbeta1, float bbeta2, float bbeta3, float stdwage, float Schooling, float Age, float Wage){

    //1. Obtaining the predicted Wage:
    float predwage=F_predwageD(bbeta0,bbeta1,bbeta2,bbeta3,Schooling,Age);
    float likelihood_wage=Wage-predwage;
    likelihood_wage=likelihood_wage/stdwage;
    likelihood_wage=normpdfESTANDAR(likelihood_wage);

    likelihood_wage=likelihood_wage/stdwage;
    if(likelihood_wage==0){
        likelihood_wage=0.000001;
    }
    likelihood_wage=logf(likelihood_wage);
    return(likelihood_wage);
}
```

```
//-----------------------------------------------------
//2. Block of defining likelihood for all the dataset
//-----------------------------------------------------

__device__ __host__ float F_likelihoodSCALARD(const float* par_likelihood,
                                              float  Schooling,
                                              float Age,
                                              float Wage){


     //float aalpha1m=FT_expD(par_likelihood[0]);
    float bbeta0=(par_likelihood[0]);
    float bbeta1=(par_likelihood[1]);
    float bbeta2=(par_likelihood[2]);
    float bbeta3=(par_likelihood[3]);
    float std=exp(par_likelihood[4]);

    float iilike=F_likelihood_wageD(bbeta0, bbeta1,  bbeta2,  bbeta3,  std,  Schooling,  Age,  Wage);
    return(iilike);
}


//Functions of loading all the likelihoods. Looping over individuals for the function F_likelihoodSCALARD
__global__ void KernelLikelihood(
    const float* par_likelihood,
    const float *V_Schooling,
    const float *V_Age,
    const float * V_Wage, float *output){

    //Indexing
    int ii=threadIdx.x+blockIdx.x*blockDim.x;
    if(ii<1000){
        //Loading
        float Schooling=V_Schooling[ii];
        float Age=V_Age[ii];
        float Wage=V_Wage[ii];
        float iilik=F_likelihoodSCALARD(par_likelihood,Schooling,Age,Wage);
        output[ii]=iilik;
    }
    if(ii>=1000){
        output[ii]=0;
    }
}
```

```
//Attempt of likelihood
float F_likelihood_FIN(const double *PARDOUBLE){ //In this function I will call the kernel
    //0. Define the current directory


    int THREADSPERBLOCK=200;
    int TOTALTHREADS=800;
    int SIZEOBS=800;
    int NVAR=3;
    int NPAR=5;
    //Passing copies of the parameters to the device
    float *dSchooling, *dAge, *dWage, *doutput, *dPAR;
    float *d_Schooling, *d_Age, *d_Wage, *d_output, *d_PAR;

    size_t sizePARFLOAT=NPAR*sizeof(float);
    size_t dimobsFLOAT=TOTALTHREADS*sizeof(float);


    //Allocate space for device copies of Parameters
    cudaMalloc((void **)&d_PAR,sizePARFLOAT);
    cudaMalloc((void **)&d_Schooling,dimobsFLOAT);
    cudaMalloc((void **)&d_Age,dimobsFLOAT);
    cudaMalloc((void **)&d_Wage,dimobsFLOAT);
    cudaMalloc((void **)&d_output,dimobsFLOAT);

    //*Cliveswithfather12, *Cliveswithmother12, *Hhchores12, *Meffort12, *Effort12, *Mage12, *Fage12, *Cedad_meses12, *Ctestsfactorsss2012, *Ctestsfac
        *Fyrschool12, *Ffraclabor12, *Mfraclabor12, *Mwage12, *Fwage12, *Mnly12, *Fnly12, *Cfactorinv12, *Cchildcare12, *Ccareskills12, *Hbarg, *Effor
        *Cbirthfactor, *Mwage10, *Fwage10, *Mnly10, *Fnly10, *Cedad_meses10, *Cliveswithfather10, *Cliveswithmother10, *Mage10, *Fage10, *Mfraclabor10,
        *Hchildcareobs, *PG, *MTJH,*MRATIO,*Unemployment,
        *Wageratio,*Distance,*Magegroup10,*Magegroup12,*Hmemberstotal10,*Hmemberstotal12,*CSTDtepsi_pb_coo10,*CSTDtepsi_pb_len10,*CSTDtepsi_pb_mot10,*CS
        STDcbcl1_pb_210,*CSTDcbcl1_pb_310,*CSTDcbcl1_pb_410,*CSTDcbcl1_pb_510,*CSTDcbcl1_pb_610,*CSTDcbcl1_pb_710,*CSTDtadi_pb_cog12, *CSTDtadi_pb_mot1
        *CSTDtadi_pb_se12, *CSTDbt_112, *CSTDbt_212, *CSTDbt_312, *CSTDbt_412, *CSTDbt_512, *CSTDbt_t12,  *CSTDhtks_st12, *CSTDbdst_st12, *CSTDppvt_t12,
        *Ccondpregg9, *Ccondpregg11b, *Ccondpregg24, *Ccondpregg23,*Cwais_pb_num, *Cwais_pb_vo, *Cbfi_pb_ama,  *Cbfi_pb_ape, *Cbfi_pb_ext, *Cbfi_pb_neu,
        *Cpsi_pb_total,*Hbargg2a, *Hbargg2b, *Hbargg2c, *Hbargg2d, *Hbargg2e, *Hbargg2f, *Hbargg2g, *Hbargg2h, *Hbargg21, *Hbargg2j, *Hcaresacwom, *Hca
        *Cinvf11c, *Cinvf11d, *Cinvf11e, *Cinvf11f, *Cinvf11g, *Cinvf11h, *Cinvf11i, *Cinvf11j, *Cinvf11k,*Csharesbedroomhowmany12,  *Csharesbedhowmany
        *g42_d2, *g42_e2, *g42_f2,*g42_a1, *g42_b1, *g42_c1, *g42_d1, *g42_e1, *g42_f1,*f21a_p_t, *f21b_p_t, *f21c_p_t, *f21d_p_t, *f21e_p_t, *f21f_p_t,
        *f21i_p_t, *f21j_p_t, *f21k_p_t, *f21l_p_t, *f21m_p_t, *f21n_p_t,*f21a_m_t, *f21b_m_t, *f21c_m_t, *f21d_m_t, *f21e_m_t, *f21f_m_t, *f21g_m_t, *
        *f21k_m_t, *f21l_m_t, *f21m_m_t, *f21n_m_t;


    Schooling=(float *)malloc(dimobsFLOAT);
    Age=(float *)malloc(dimobsFLOAT);
    Wage=(float *)malloc(dimobsFLOAT);
    output=(float *)malloc(dimobsFLOAT);
    PAR=(float *)malloc(sizePARFLOAT);
```

1. What are neural networks?
   Framework consisting of a collection of neurons in multiple layers

   Each neuron is the result of an activation function applied to a linear transformation

   Neurons in layer $l+1$ use neurons in layer l as input

2. Why do we need them?

   We need them because of their ability to perform well in high-dimensional classification-optimization problems

   Universal Approximation Theorem (Gybenko, 1989) $\rightarrow$ rich enough network could, in principle, "approximate most functions".

   Challenge is in the estimation

How do we estimate a neural network?

- First, we perform **feature normalization** on our data.
- Our goal is to minimize a **cost function**
  (SSR, cross entropy cost; -log-likelihood function)
- Traditionally, we use a family of **gradient descent** algorithms to minimize the cost function
- For this, we need to chose a **learning rate**
  - High learning rate: minimization might not converge
  - Low learning rate: convergence might be too slow
- We also chose a **regularization** parameter to avoid overfitting
- Deep neural networks are often estimated with **GPU parallelization**

Other questions

When is it ok to estimate a Neural Network?

- Only you can know the answer to the question
- Recall answer to the question: why do we need neural networks? How complex is your data? Is it linearly separable?
- Usually, sample of 1,000 is enough to train a shallow neural network (Ciresan, Meier & Schidhuber 2012).

Building your own neural network. How many layers and how many neurons in each layer?
(Taken from: Heaton, J. "Intro to neural networks with Java")

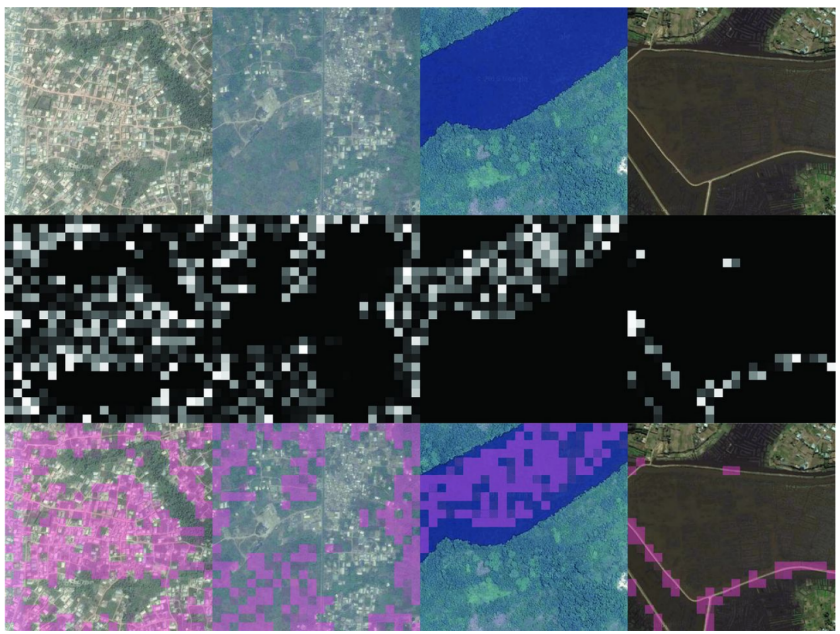
- Problems that require two or more hidden layers are rarely encountered.
- No theoretical reason to use more than two hidden layers
- Bengio, Y.1993 "Just keep adding layers until the test error does not improve anymore."
- How many neurons? Multiple rules of thumb:
  - should be between the size of the input layer and the size of the output layer
  - The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
  - The number of hidden neurons should be less than twice the size of the input layer.
  - Masters (1993) suggests $\sqrt{n \times m}$; n: input layer. m output layer.
  - Ultimately, it is up to you. Trial and error.

Applications of neural networks in economics:

- **Convolutional Neural Networks (CNN)**: are the state of the art approach for image classification.

  CNN Start by matching general features of images in various places of the image. → Convolution.

  Example: Combining satellite imagery and machine learning to predict poverty levels in Rwanda. Going beyond the nightlight approach . (Jean et al., 2016.)



- **Recurrent Neural Network**. Idea is to use previous predictions to make actual predictions.
  - Speech and handwritten recognition.
    Given prediction "Hello, how are ——". Use not only sound (image) in X but also **previous predicted** words to predict **you**. Google search.
  - Time series forecasting. Given that we predict demand has increased during the last 30 hours, what will happen next hour?
    Azure, Microsoft. (Falat & Pancikova, 2015.)
    Airline demand estimation (Mottini & Acuna, 2017)
  - How much memory should a neural network have? Long short-term memory (LSTM).

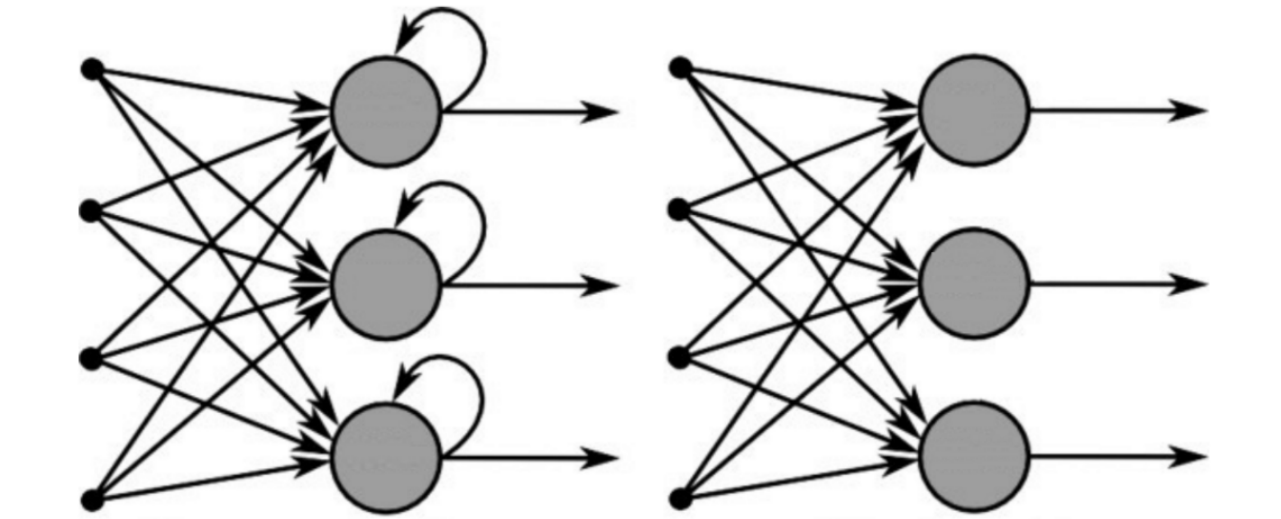


Image taken from: Towards data science.
https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

- Solving non-linear continuous time macroeconomic models (Duarte, 2018)
- General **prediction** problems (Fescioglu & Tanyeri, 2013)

- R has some support for estimation of neural networks. (h20, keras, neuralnet)
- Python, C++ (scikit-learn, tensorflow, pytorch)
- If you want to learn more NN, google: tensorlow scikit-learn. You will find plenty of examples
- See how to use scikit-learn in this link
- How to learn more: Practice! (Kaggle) Coursera. Open courses (MIT, Stanford, etc.).
- Implementing a neural network from scratch: see this link
- if you want an elaborate example of estimating an actual neural network: email me
- If you want an example of the cuda parallelization: email me.

ENOUGH! Let us go to R....

Neural Networks: advanced topics

- Review of probability and statistics
- Cost function. Cross entropy - log-likelihood.
- Back propagation in detail
- Gradient descent in detail

Review of probability and statistics: I

- If X is a discrete random variable, with probability mass function *pmf* defined by $p()$, then:
  - $0 \leq p(x) \leq 1$ for every x.
  - $\sum_x p(x) = 1$
  - $p(\tilde{x}) = P(X = \tilde{x})$: Probability that realization of random variable X takes value $\tilde{x}$

Review of probability and statistics: II

- If W is a continuous random variable with probability density function *pdf* defined by $f()$ then:
  - $f(x) \geq 0$ for every x
  - $\int_{-\infty}^{\infty} f(x) = 1$

Review of probability and statistics: III

- For two random variables X, Y, their joint pdf (pmf) $f(x, y)$ we have:
- $f(x, y) = f(x|y) \times f(y)$
- We say x and y are independent if and only if $f(x, y) = f(x) \times f(y)$
- Bayes rule: $f(y|x) = \frac{f(x,y)}{f(x)} = \frac{f(x|y) \times f(y)}{f(x)}$

Review of probability and statistics: IV

- Bernoulli random variable. y can take two outcomes, 0 or 1.
- $p(x;\theta) = \begin{cases} \theta \text{ if } x = 1 \\ (1-\theta) \text{ if } x = 0 \end{cases}$
- $p(x;\theta) = \theta^x \times (1-\theta)^{(1-x)}$
- Given parameter θ, what is the probability of observing realization x of random variable X.

Review of probability and statistics: IV

- Bernoulli random variable. y can take two outcomes, 0 or 1.
- $p(x;\theta) = \begin{cases} \theta \text{ if } x = 1 \\ (1-\theta) \text{ if } x = 0 \end{cases}$
- $p(x;\theta) = \theta^x \times (1-\theta)^{(1-x)}$
- Given parameter θ, what is the probability of observing realization x of random variable X.
- Flip a coin. $x = 1$ heads, $x = 0$ tails. $p = 0.5$.
- Probability of tails: $p(0) = 0.5^0 \times 0.5^1 = 0.5$

MLE in logistic regression

- In logistic regression, we model $P(y_i = 1) = \frac{1}{1+e^{-x_i\beta}}$
- Recall spam problem...

$$g(y_i; \beta) = \left(\frac{1}{1 + e^{-x_i\beta}}\right)^{y_i} \times \left(1 - \frac{1}{1 + e^{-x_i\beta}}\right)^{1-y_i}$$

MLE in logistic regression

- In logistic regression, we model $P(y_i = 1) = \frac{1}{1+e^{-x_i\beta}}$
- Recall spam problem...

$$g(y_i; \beta) = \left(\frac{1}{1 + e^{-x_i\beta}}\right)^{y_i} \times \left(1 - \frac{1}{1 + e^{-x_i\beta}}\right)^{1-y_i}$$

- What happens to the $g()$ function $y_i = 1$ (spam):

MLE in logistic regression

- In logistic regression, we model $P(y_i = 1) = \frac{1}{1+e^{-x_i\beta}}$
- Recall spam problem...

$$g(y_i; \beta) = \left(\frac{1}{1+e^{-x_i\beta}}\right)^{y_i} \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-y_i}$$

- What happens to the $g()$ function $y_i = 1$ (spam):

$$\begin{aligned} g(1; \beta) &= \left(\frac{1}{1+e^{-x_i\beta}}\right)^{1} \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-1} = \\ &= \left(\frac{1}{1+e^{-x_i\beta}}\right)^{1} \times 1 \\ &= \left(\frac{1}{1+e^{-x_i\beta}}\right) \\ &= P(y_i = 1) \rightarrow \text{ Probability of being spam} \end{aligned} \quad (1)$$

MLE in logistic regression

If our email is non-spam, our $g()$ function becomes:

$$g(0;\beta) = \left(\frac{1}{1+e^{-x_i\beta}}\right)^0 \times \left(1-\frac{1}{1+e^{-x_i\beta}}\right)^{1-0} =$$

$$= \times \left(1-\frac{1}{1+e^{-x_i\beta}}\right)^{1-0} =$$

$$= \left(1-\frac{1}{1+e^{-x_i\beta}}\right)$$

$$= P(y_i = 0) \rightarrow \text{ Probability of being non-spam}$$

MLE in logistic regression

- We can interpret function $g(y_i; \beta)$ as the probability of observation i of being y_i.
- We want the function $g(y_i; \beta)$ to be as large as possible (close to one).
- For the whole set of $i = 1, ...n$ observations, the joint probability mass function is:

$$f(y_1, ...y_n; \beta) = \prod_{i=1}^{n} g(y_i; \beta)$$

MLE in logistic regression

- Joint probability mass function of $y_1, ...y_n$ given parameters β :

$$f(y_1, ...y_n; \beta) = \prod_{i=1}^{n} g(y_i; \beta)$$

- "What is the probability of observing outcomes $y_1, ...y_n$ given parameters β"
- Likelihood function:

$$\mathcal{L}(\beta | y_1, ...y_n) = \prod_{i=1}^{n} g(y_i; \beta) \tag{2}$$

- "Given observations $y_1....y_n$, how plausible is this data generated from our model with parameters β?"
- Maximum likelihood estimation $\rightarrow$ maximize the likelihood function.

MLE in logistic regression

- If we want $f(y_1, ...y_n; \beta)$ to be as large as possible, we want $\ln(f(y_1, ...y_n; \beta))$ to be as large as possible.

$$\ln(f(y_1, ...y_n; \beta)) = \ln\left(\prod_{i=1}^{n} g(y_i; \beta)\right)$$

$$= \sum_{i=1}^{n} \ln g(y_i; \beta)$$

$$= \sum_{i=1}^{n} \ln\left(\left(\frac{1}{1+e^{-x_i\beta}}\right)^{y_i} \times \left(1 - \frac{1}{1+e^{-x_i\beta}}\right)^{1-y_i}\right)$$

$$= \sum_{i=1}^{n} \left(y_i \ln\left(\frac{1}{1+e^{-x_i\beta}}\right) + (1-y_i)\ln\left(1 - \frac{1}{1+e^{-x_i\beta}}\right)\right)$$

MLE in logistic regression

$$= \sum_{i=1}^{n} \left(y_i \ln \left(\frac{1}{1 + e^{-x_i \beta}} \right) + (1 - y_i) \ln \left(1 - \frac{1}{1 + e^{-x_i \beta}} \right) \right)$$

- We want this function to be as large as possible.
- Alternatively, we can define our cost function as the negative, and minimize it.
- Find β^* to minimize:

$$J(\beta; y_1, ...j_n) = -\frac{1}{m} \sum_{i=1}^{n} \left(y_i \ln \left(\frac{1}{1 + e^{-x_i \beta}} \right) + (1 - y_i) \ln \left(1 - \frac{1}{1 + e^{-x_i \beta}} \right) \right)$$

MLE in logistic regression

- Cross entropy cost:

$$J(\beta; y_1, ...j_n) = -\frac{1}{m}\sum_{i=1}^{n}\left(y_i \ln\left(\frac{1}{1+e^{-x_i\beta}}\right) + (1-y_i)\ln\left(1-\frac{1}{1+e^{-x_i\beta}}\right)\right)$$

- (-log-likelihood function)
- Find β that minimizes the cros entropy cost = maximizes (log)-likelihood function
- We can go to R and do this with our spam classification problem.

Neural network cost function

- In logistic regression, our probability is modeled as $P(y_i = 1) = \frac{1}{1+e^{-x_i\beta}}$
- In general neural networks for classification, the probability is modeled as: $p(y_i = 1) = h(x_i) = a^{[L](i)}$

$$J(W, b; y_1, ...j_n) = -\frac{1}{m}\sum_{i=1}^{n}\left(y_i \ln\left(a^{[L](i)}\right) + (1 - y_i)\ln\left(1 - a^{[L](i)}\right)\right)$$

where W stores all the $w_{p,k}^{[l]}$ terms and b contains all the bias terms.

Formally, estimating a neural network consists of finding the elements W, b that minimize the cross entropy cost:

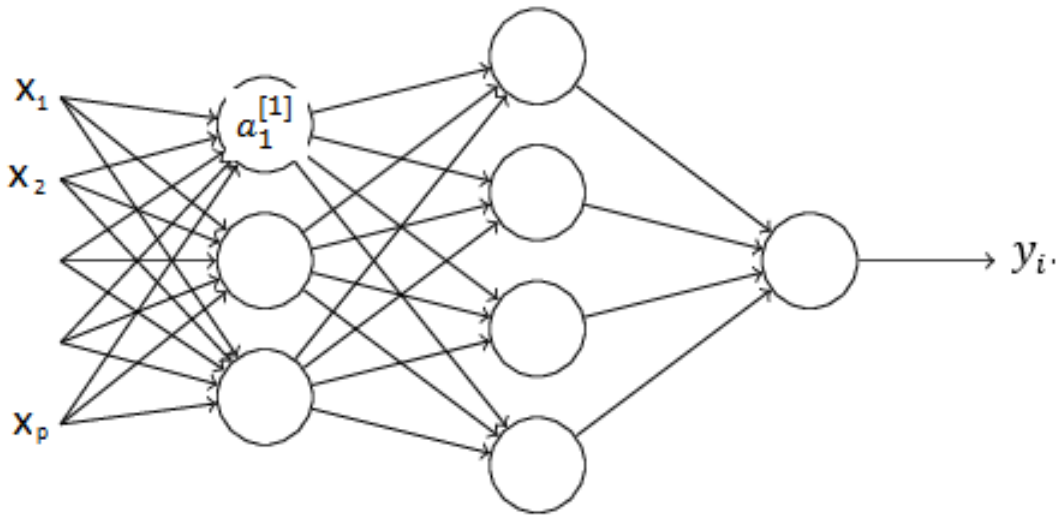
$$J(W, b; y_1, ...j_n) = -\frac{1}{m}\sum_{i=1}^{n}\left(y_i \ln\left(a^{[L](i)}\right) + (1 - y_i)\ln\left(1 - a^{[L](i)}\right)\right)$$

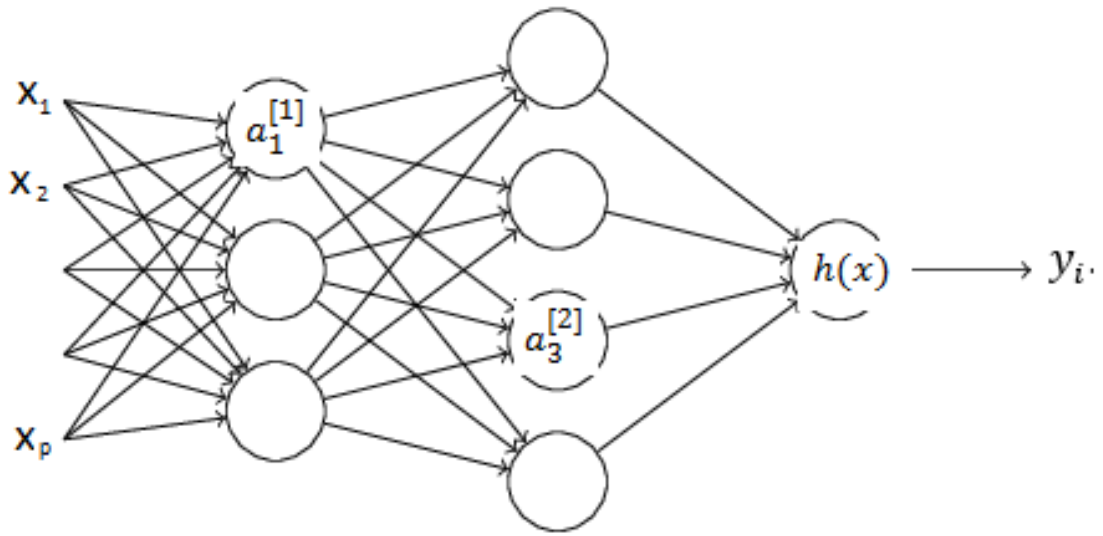
$$-2\frac{\lambda}{m}\sum_{p}\sum_{k}\sum_{l}\left(w_{p,k}^{[l]}\right)^2$$

We often include a regularization term, λ

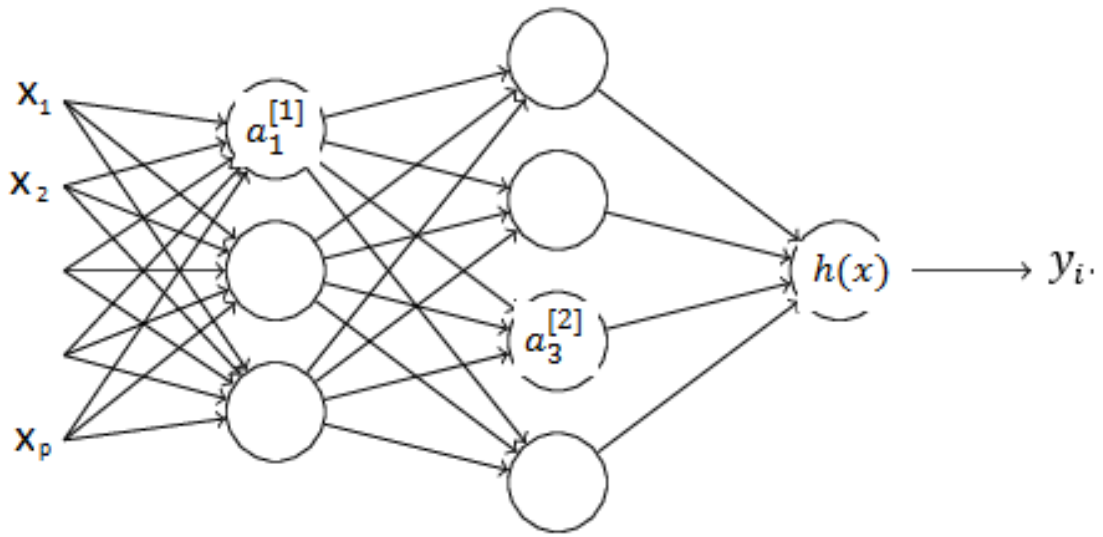
Why backward propagation to obtain gradient?

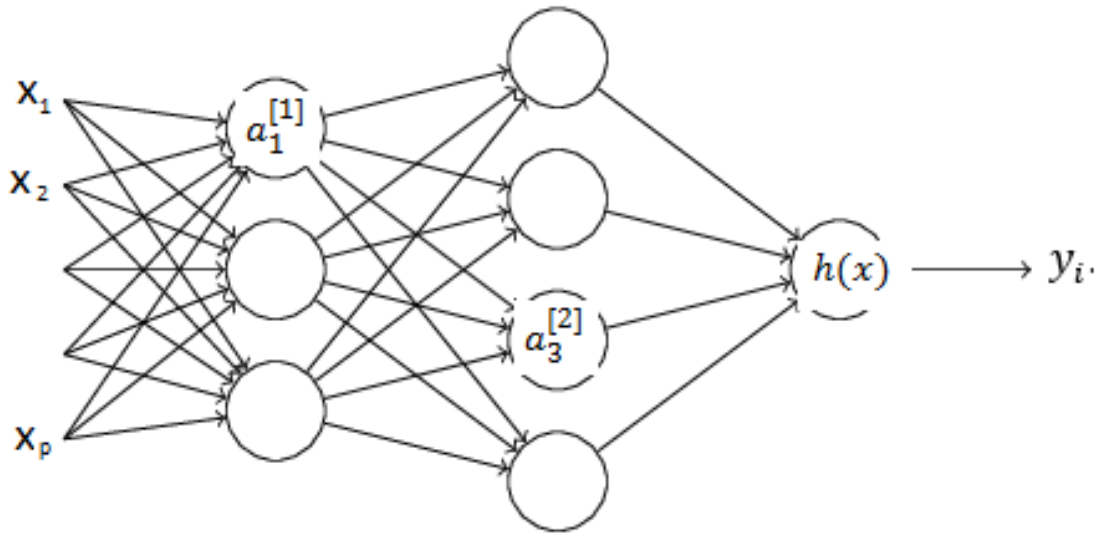
- Chain Rule


x_1
x_2
x_p
$a_1^{[1]}$
y_i

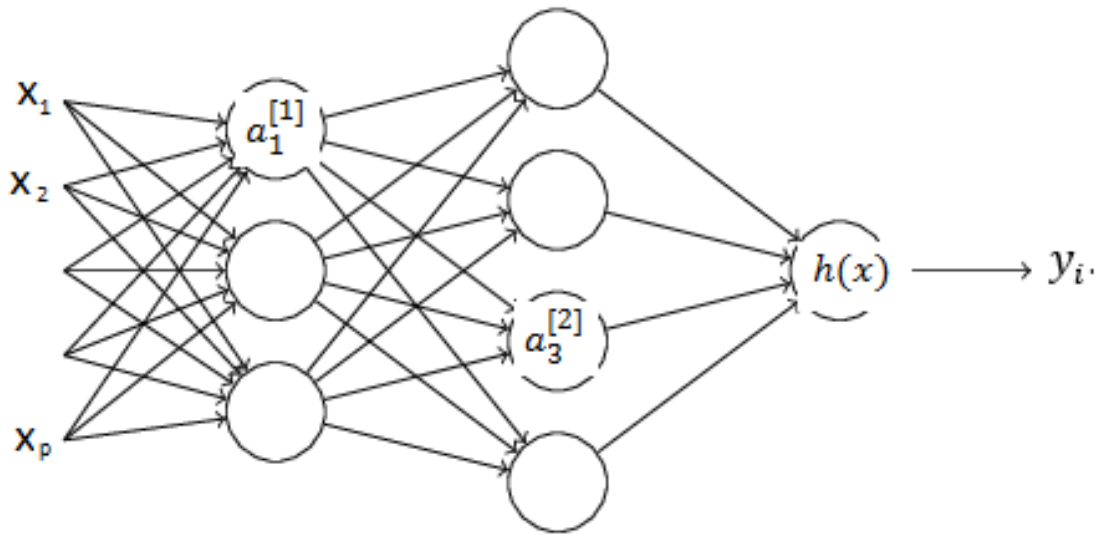


- $a_n^{[l](i)}$: layer l, observation i, neuron n
- Output layer: $h(x)$
- Input layer: usually l=0.

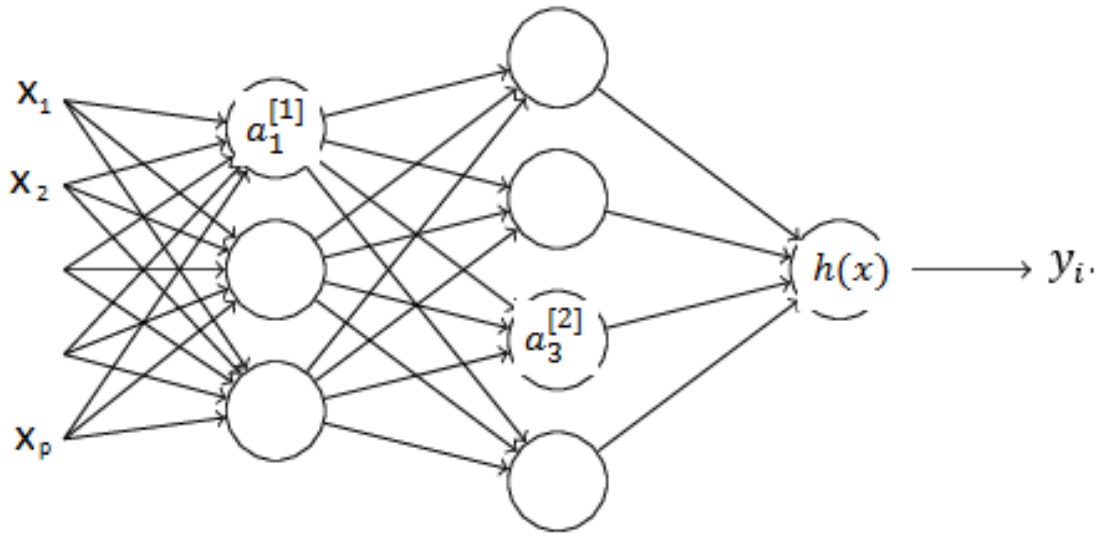

x_1
x_2
x_p
$a_1^{[1]}$
$a_3^{[2]}$
$h(x)$
y_i



- $a_1^{[1](i)} = f_1^{[1]}\left(b_1^{[1]} + w_{1,1}^{[1]}x_1^{(i)} + w_{1,2}^{[1]}x_2^{(i)} + ... + w_{1,p}^{[1]}x_p^{(i)}\right) = f_1^{[1]}\left(z_1^{[1](i)}\right)$



- $a_1^{[1](i)} = f_1^{[1]}\left(b_1^{[1]} + w_{1,1}^{[1]}x_1^{(i)} + w_{1,2}^{[1]}x_2^{(i)} + ... + w_{1,p}^{[1]}x_p^{(i)}\right) = f_1^{[1]}\left(z_1^{[1](i)}\right)$
- $a_3^{[2](i)} = f_3^{[2]}\left(b_3^{[2]} + w_{3,1}^{[2]}a_1^{[1](i)} + w_{3,2}^{[2]}a_2^{[1](i)} + w_{3,3}^{[2]}a_3^{[2](i)}\right) = f_3^{[2]}(z_3^{[2](i)})$



- $a_1^{[1](i)} = f_1^{[1]}\left(b_1^{[1]} + w_{1,1}^{[1]}x_1^{(i)} + w_{1,2}^{[1]}x_2^{(i)} + ... + w_{1,p}^{[1]}x_p^{(i)}\right) = f_1^{[1]}\left(z_1^{[1](i)}\right)$
- $a_3^{[2](i)} = f_3^{[2]}\left(b_3^{[2]} + w_{3,1}^{[2]}a_1^{[1](i)} + w_{3,2}^{[2]}a_2^{[1](i)} + w_{3,3}^{[2]}a_3^{[2](i)}\right) = f_3^{[2]}(z_3^{[2](i)})$
- $a^{[3](i)} = h(x_i) = P(y_i = 1) = f^{[3]}\left(b^{[3]} + w_1^{[3]}a_1^{[2](i)} + ...w_4^{[3](i)}a_4^{[2](i)}\right) = f^{[3]}(z^{[3](i)})$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)} \rightarrow$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)} \rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)} \rightarrow$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$linear transformation of 7th neuron, 3rd layer, observation 8.

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$linear transformation of 7th neuron, 3rd layer, observation 8.
- $z_7^{[3](8)} =$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$linear transformation of 7th neuron, 3rd layer, observation 8.
- $z_7^{[3](8)} = b_7^{[3]} + w_{7,1}^{[3](8)} a_1^{[6](8)} + w_{7,2}^{[3](8)} a_2^{[6](8)} + ... + w_{7,p}^{[3](8)} a_p^{[6](8)}$

- In General:
- $a_p^{[0](i)} = x_p^{(i)} \rightarrow$ covariate p, observation i
- $z_n^{[l](i)} = b_n^{[l]} + w_{n,1}^{[l](i)} a_1^{[l-1](i)} + w_{n,2}^{[l](i)} a_2^{[l-1](i)} + .. + w_{n,q}^{[l](i)} a_q^{[l-1](i)}$
- $a_p^{[l](i)} = f_p^l(z_n^{[l](i)})$
- $a^{[L](i)} = h(x_i) = P(y_i = 1)$

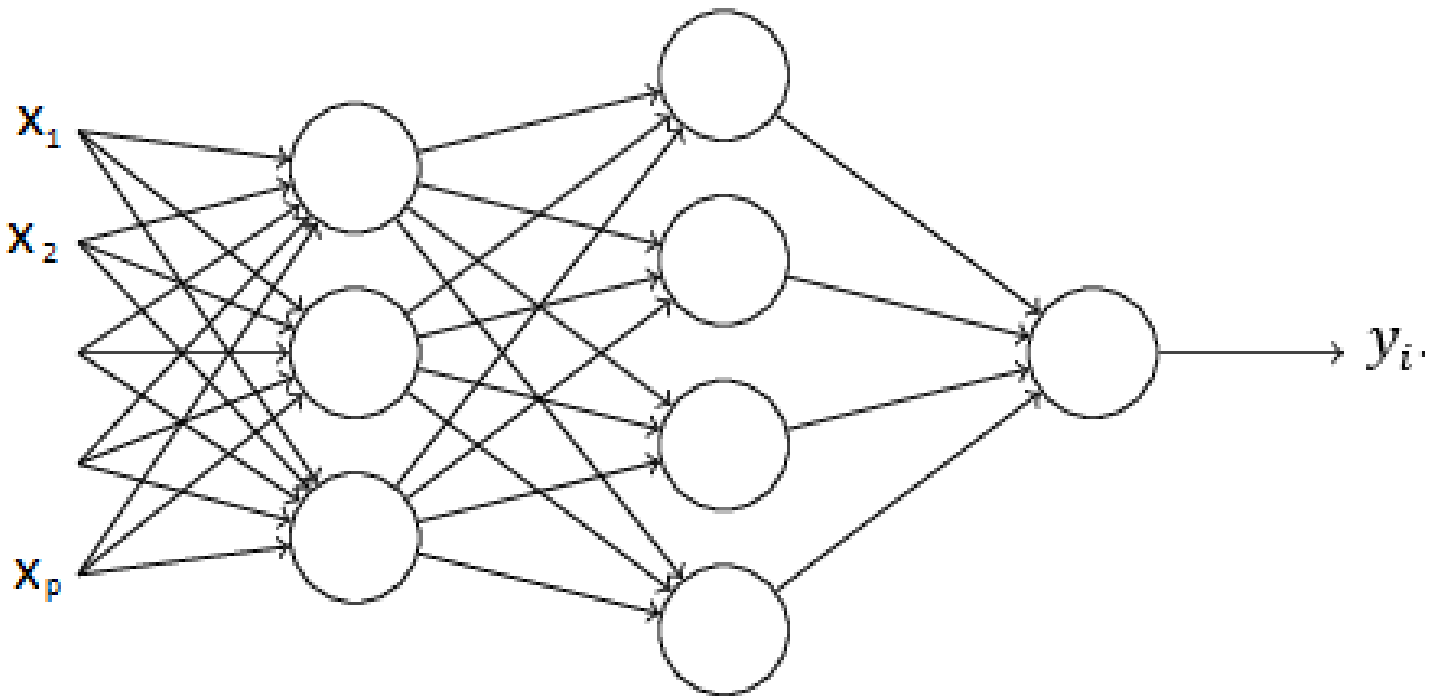
Backward propagation:

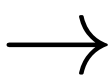
- Cost function $J() = \frac{1}{N}\sum_{i=1}^{N} j_i$ where j_i cost of observation i
- Obtain derivative of cost function $J()$ with respect to $w_{n,1}^{[l]}$ for gradient descent:

$$\frac{\partial J()}{\partial w_{n,1}^{[l]}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial j_i}{\partial a^{[L](i)}} \frac{\partial a^{[L](i)}}{\partial z_n^{[L](i)}} \frac{\partial z_n^{[L](i)}}{\partial a_n^{[L-1](i)}} \cdots \frac{\partial z_n^{[l](i)}}{\partial w_{n,1}^{[l]}}$$

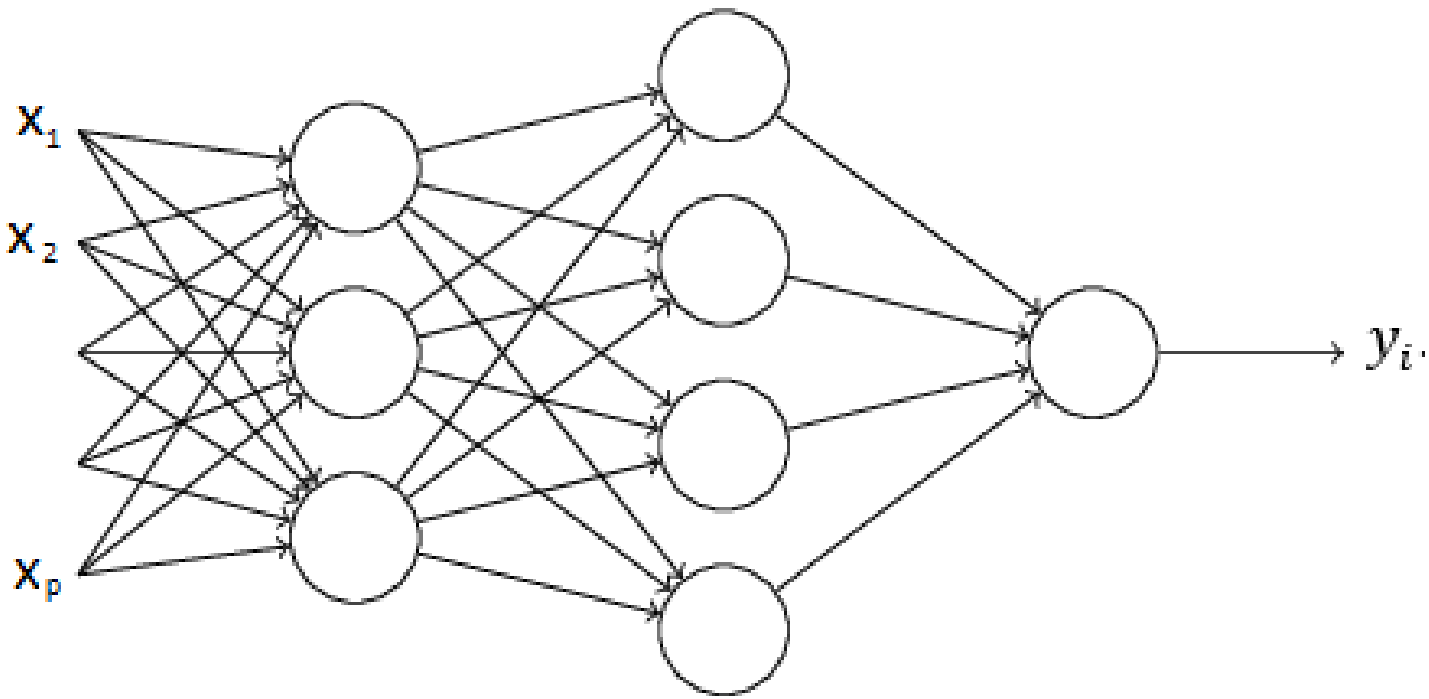
- Gradient descent: If $\frac{\partial J()}{\partial w_{n,1}^{[l]}}$ is positive, decrease your guess of $w_{n,1}$.
- Stochastic gradient descent: rather than using observations $i = 1...N$ to compute gradient, randomly choose a subset of observations $s << N$ to evaluate gradient.

- Forward propagation. Estimate cost.





- Backward propagation. Estimate gradient.



Backward propagation

- Neural networks activation function: relatively inexpensive to compute derivatives
- $\frac{\partial tanh(x)}{\partial x} = 1 - tanh(x)^2$
- $\frac{\partial sigmoid(x)}{\partial x} = sigmoid(x) \times (1 - sigmoid(x))$
- Part of the terms needed to compute derivative already computed in forward propagation
- During the forward propagation, these terms are stored in a "cache"

Backward propagation, gradient descent. Alternatives

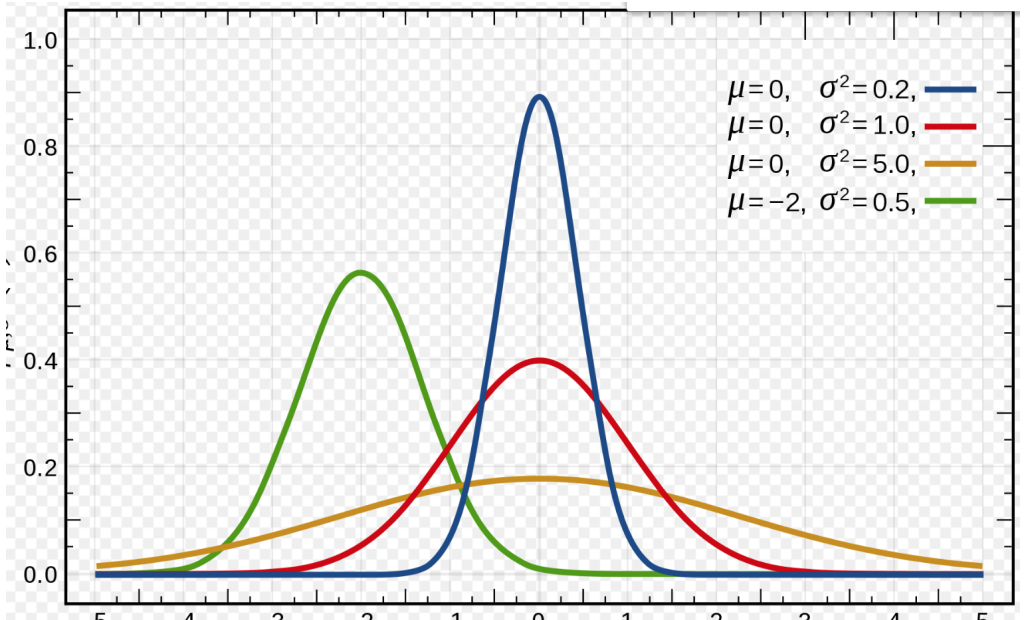
- Stochastic gradient descent. Take randomly one observation to compute gradient.
- Stochastic batch gradient descent. Take a sample $n << N$ to compute the gradient.
- Rprop+. Uses sign of derivative

MLE in regression analysis

- Due to various results in statistics, the Normal distribution is a widely used function.
- We say X follows a normal distribution with mean μ and variance σ^2 if:

$$X \sim N(\mu, \sigma^2)$$

$$f_x(X) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{3}$$



- Let's assume log-wage is given by:

$$\ln(wage_i) = \beta_0 + \beta_1 \text{schooling}_{1,i} + \beta_2 \text{age}_{2,i} + \varepsilon_i \tag{4}$$

but now, we assume that ε_i follows a normal distribution:

$$\varepsilon_i \sim N(0, \sigma^2) \tag{5}$$

If this is the case, then:

$$y_i \sim N(\beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i}, \sigma^2) \tag{6}$$

Probability density function of observation i

$$f(y_i; x_i, \beta, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_i - (\beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i}))^2} \tag{7}$$

Maximum Likelihood Estimation

- The **joint** probability density function of all the data is given by:

$$f(Y; X, \beta, \sigma^2) = f_Y(y_1, ...y_n; X, \beta, \sigma^2)$$

- The Likelihood Function is given by:

$$\mathcal{L}(\beta, \sigma^2 | Y, X) = f_Y(y_1, ...y_n; X, \beta, \sigma^2) \tag{8}$$

- Two random variables are independent iff their joint pdf is the product of their pdf.
- We assume that wages are independent between individuals:

$$f_Y(y_1, ...y_n; X, \beta, \sigma^2) = f_y(y_1; X_1, \beta, \sigma^2) \times f_y(y_2; X_2, \beta, \sigma^2) \times$$

$$... \times f_y(y_n; X_n, \beta, \sigma^2)$$

Maximum Likelihood Estimation

- The independence assumption implies that:

$$\mathcal{L}(\beta, \sigma^2 | Y, X) = \prod_{i=1}^{n} f_y(y_i; X_i, \beta, \sigma^2) \tag{9}$$

- We usually work with the log-likelihood function for various reasons...

$$l(\beta, \sigma^2 | Y, X) = \ln \left(\prod_{i=1}^{n} f_y(y_i; X_i, \beta, \sigma^2) \right) \tag{10}$$

$$= \sum_{i=1}^{n} \ln \left(f_y(y_i; X_i, \beta, \sigma^2) \right)$$

- $\left[\hat{\beta}_{MLE}, \hat{\sigma}^2_{MLE}\right] = \arg\max \ln\left(\prod_{i=1}^{n} f_y(y_i; X_i, \beta, \sigma^2)\right)$
- Let's go to R and do some work.

Neural Networks: basics

- Let us consider the case of a binary outcome

$$y_i = \begin{cases} 0 \text{ if non-spam} \\ 1 \text{ if spam} \end{cases}$$

- We want to predict probabilities: $P(y_i = 1)$ based on regressors $x_1, ...x_p$
- In linear regression, you obtain predicted probabilities less than zero and greater than one
- Classification problems are rarely linear (e.g. image, sound recognition).

$$P(y_i = 1) = \beta_0 + x_{i,1}\beta_1, ... + x_{i,p}\beta_p$$

Logistic regression: basics II

- In Logistic regression, we assume non-linear function bounded between zero and one:

$$P(y_i = 1) = \frac{1}{1 + e^{-(\beta_0 + x_{i,1}\beta_1, \ldots + x_{i,p}\beta_p)}}$$

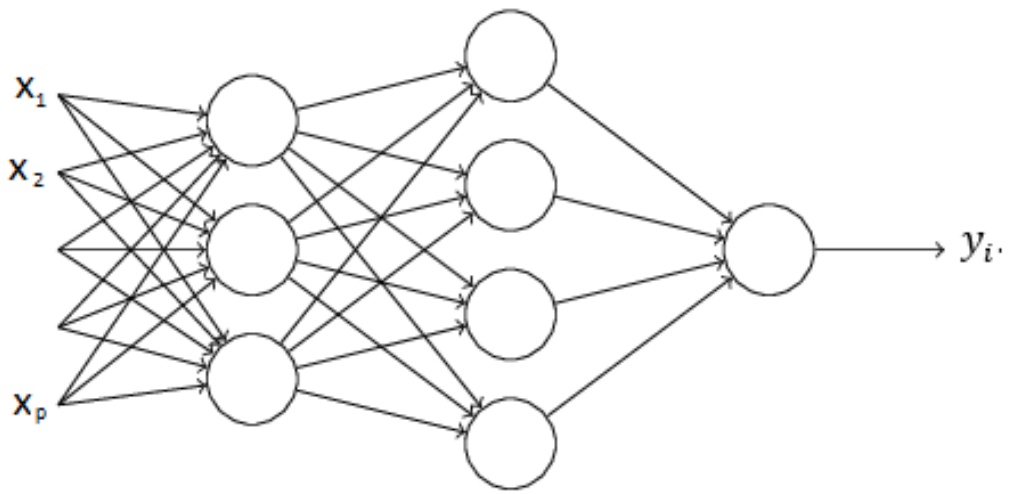
$$P(y_i = 0) = 1 - P(y_i = 1)$$

$$= 1 - \frac{1}{1 + e^{-(\beta_0 + x_{i,1}\beta_1, \ldots + x_{i,p}\beta_p)}}$$

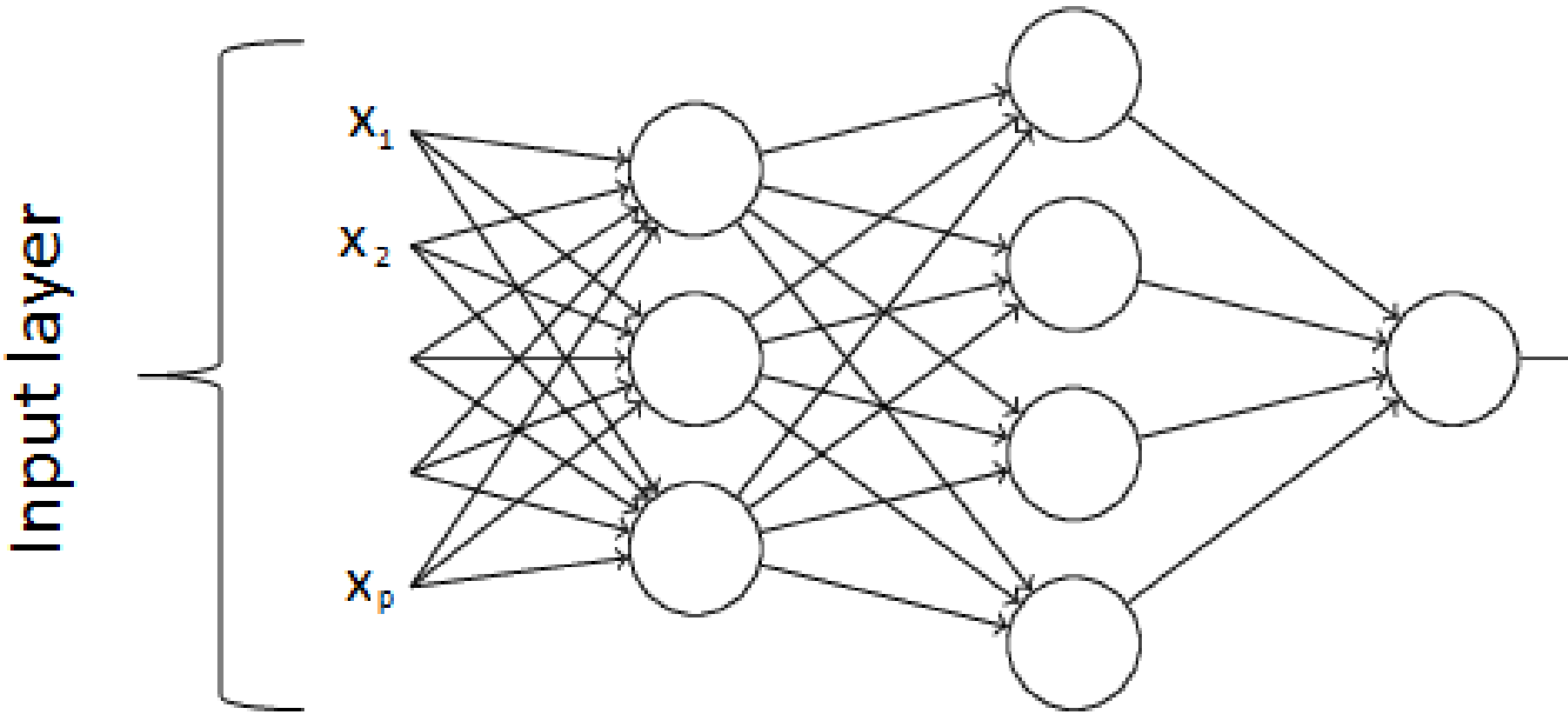
- We often refer to $f(z) = \frac{1}{1+e^{-x}}$ as a 'sigmoid' function.

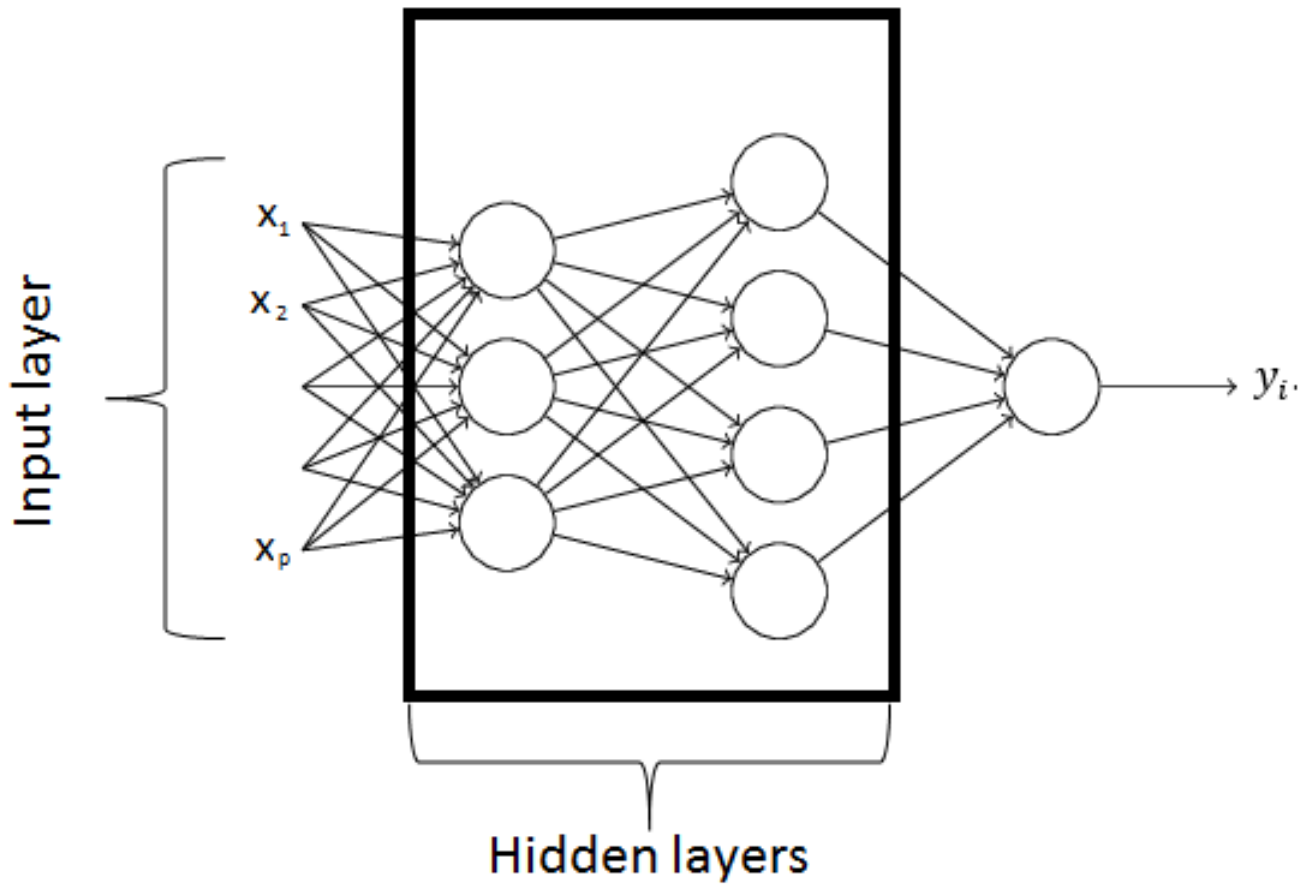
Neural Networks: basics

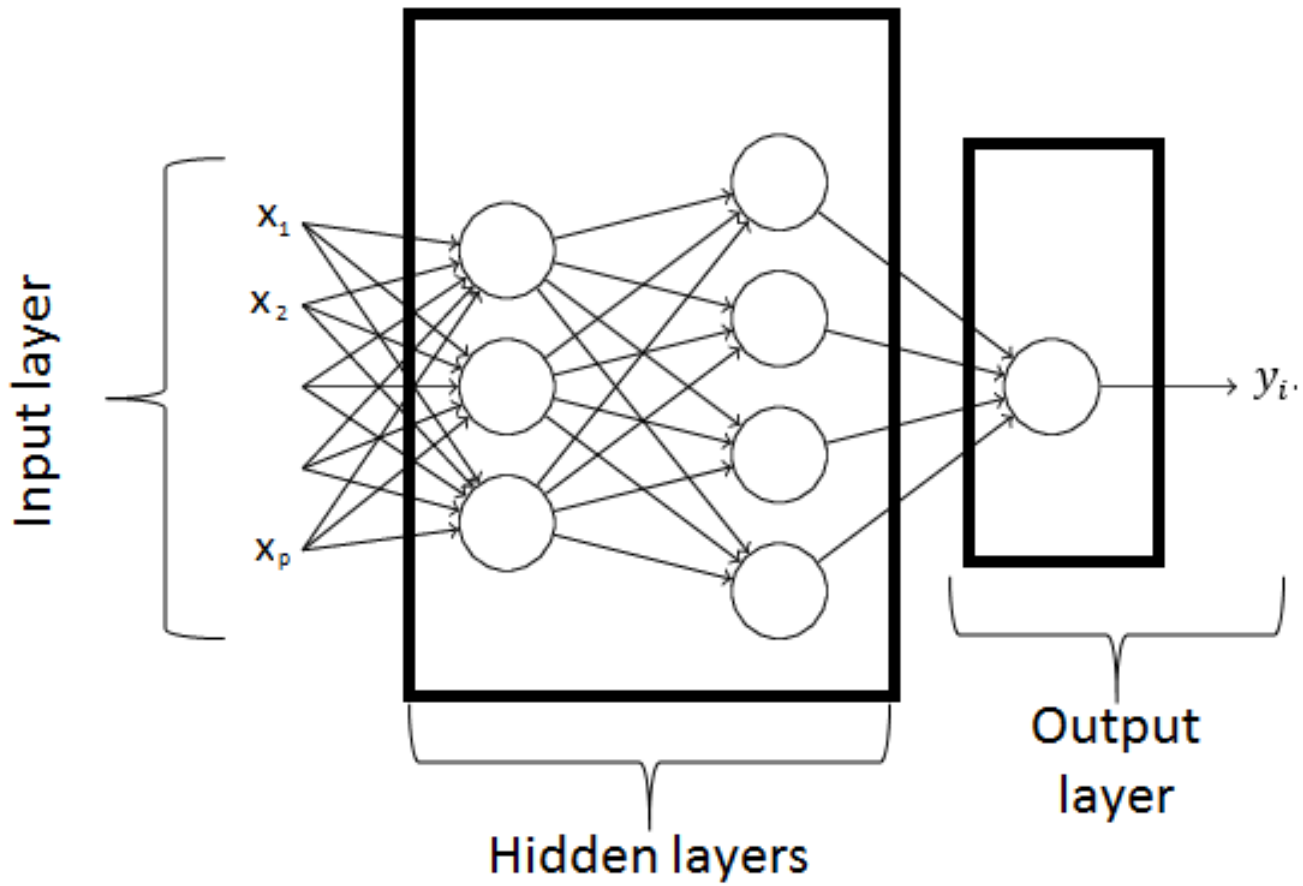
- High dimensional problems are highly non-linear
- Neural network deal with this kind of problems (image and speech recognition)

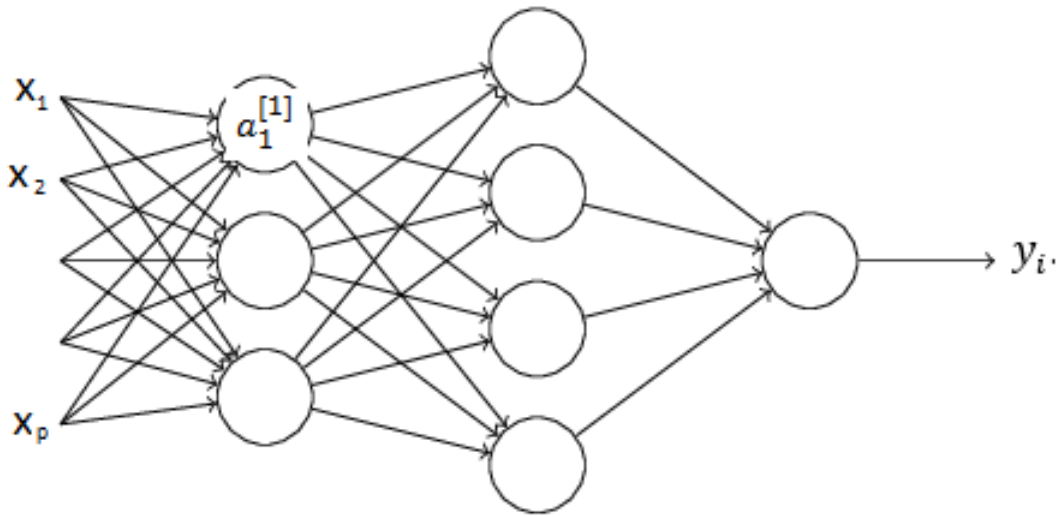


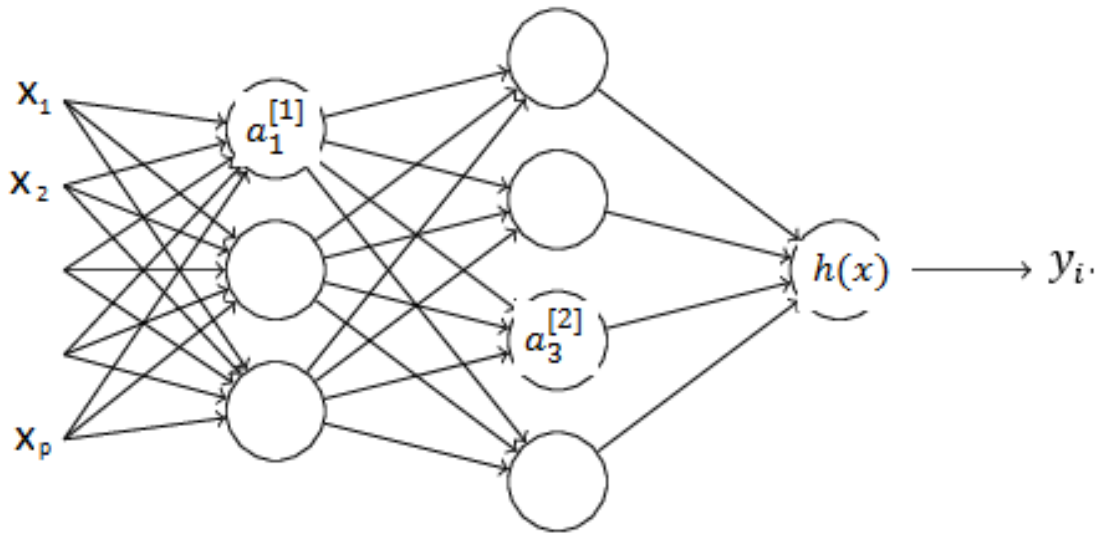
- Composed of various layers.
- Each layer composed of various neurons
- Each neuron is the result of a linear transformation+activation function


Input layer
x_1
x_2
x_p

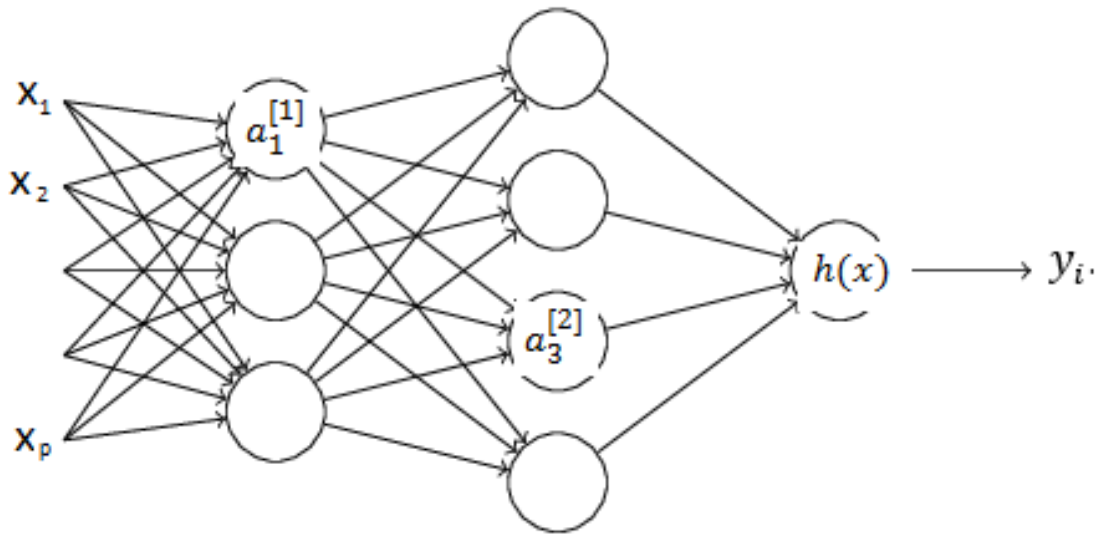

Input layer
x1
x2
xp
yi
Hidden layers

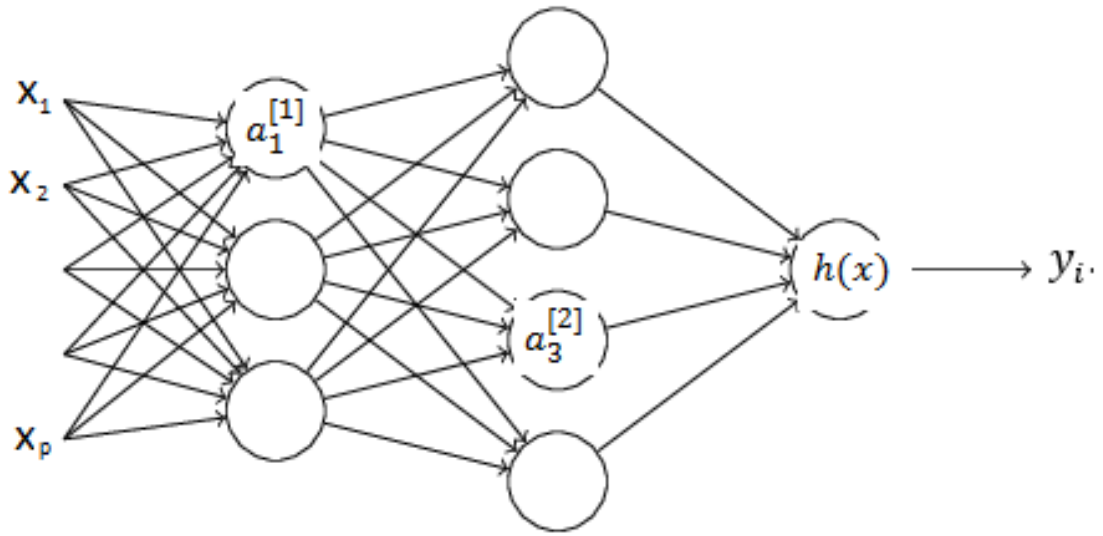

Input layer
x1
x2
xp
yi
Output layer
Hidden layers


x_1
x_2
x_p
$a_1^{[1]}$
y_i

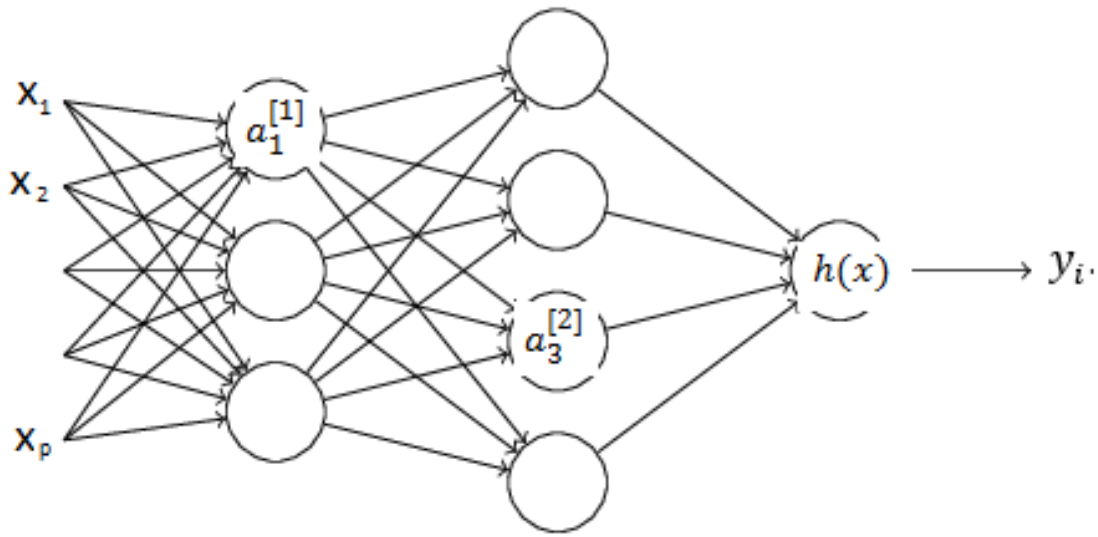


- $a_n^{[l](i)}$: layer l, observation i, neuron n
- Output layer: $h(x)$
- Input layer: usually l=0.

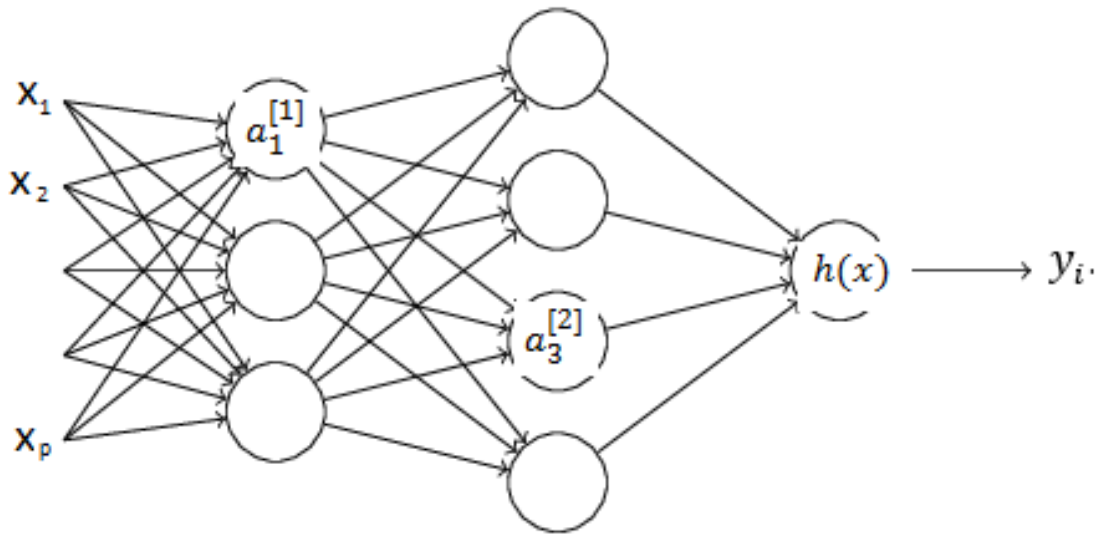

x_1
x_2
x_p
$a_1^{[1]}$
$a_3^{[2]}$
$h(x)$
y_i



- $a_1^{[1](i)} = f_1^{[1]}\left(b_1^{[1]} + w_{1,1}^{[1]}x_1^{(i)} + w_{1,2}^{[1]}x_2^{(i)} + ... + w_{1,p}^{[1]}x_p^{(i)}\right) = f_1^{[1]}\left(z_1^{[1](i)}\right)$



- $a_1^{[1](i)} = f_1^{[1]}\left(b_1^{[1]} + w_{1,1}^{[1]}x_1^{(i)} + w_{1,2}^{[1]}x_2^{(i)} + ... + w_{1,p}^{[1]}x_p^{(i)}\right) = f_1^{[1]}\left(z_1^{[1](i)}\right)$
- $a_3^{[2](i)} = f_3^{[2]}\left(b_3^{[2]} + w_{3,1}^{[2]}a_1^{[1](i)} + w_{3,2}^{[2]}a_2^{[1](i)} + w_{3,3}^{[2]}a_3^{[2](i)}\right) = f_3^{[2]}(z_3^{[2](i)})$



- $a_1^{[1](i)} = f_1^{[1]}\left(b_1^{[1]} + w_{1,1}^{[1]}x_1^{(i)} + w_{1,2}^{[1]}x_2^{(i)} + ... + w_{1,p}^{[1]}x_p^{(i)}\right) = f_1^{[1]}\left(z_1^{[1](i)}\right)$
- $a_3^{[2](i)} = f_3^{[2]}\left(b_3^{[2]} + w_{3,1}^{[2]}a_1^{[1](i)} + w_{3,2}^{[2]}a_2^{[1](i)} + w_{3,3}^{[2]}a_3^{[2](i)}\right) = f_3^{[2]}(z_3^{[2](i)})$
- $a^{[3](i)} = h(x_i) = P(y_i = 1) = f^{[3]}\left(b^{[3]} + w_1^{[3]}a_1^{[2](i)} + ...w_4^{[3](i)}a_4^{[2](i)}\right) = f^{[3]}(z^{[3](i)})$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)} \rightarrow$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$linear transformation of 7th neuron, 3rd layer, observation 8.

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$linear transformation of 7th neuron, 3rd layer, observation 8.
- $z_7^{[3](8)} =$

Neural Networks: basics

- Setup is not too complicated. It might be hard to keep track of indexes.
- Let us see....
- $a_4^{[5](23)}$ $\rightarrow$fourth neuron, fifth layer, observation 23
- $z_7^{[3](8)}$ $\rightarrow$linear transformation of 7th neuron, 3rd layer, observation 8.
- $z_7^{[3](8)} = b_7^{[3]} + w_{7,1}^{[3](8)} a_1^{[6](8)} + w_{7,2}^{[3](8)} a_2^{[6](8)} + ... + w_{7,p}^{[3](8)} a_p^{[6](8)}$

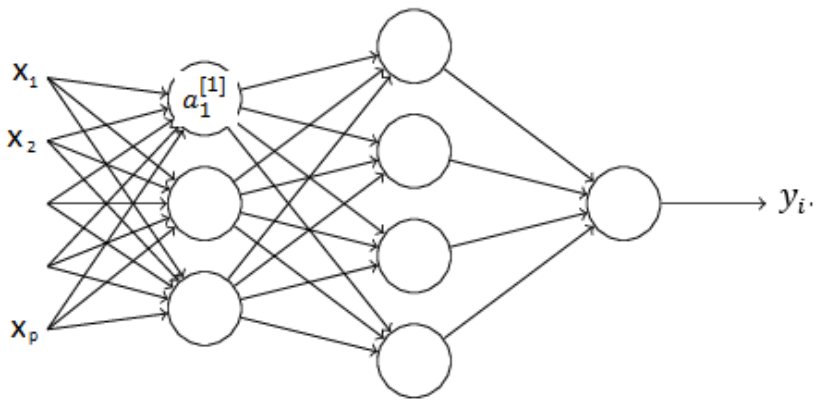
- In General:
- $a_p^{[0](i)} = x_p^{(i)} \rightarrow$ covariate p, observation i
- $z_n^{[l](i)} = b_n^{[l]} + w_{n,1}^{[l](i)} a_1^{[l-1](i)} + w_{n,2}^{[l](i)} a_2^{[l-1](i)} + .. + w_{n,q}^{[l](i)} a_q^{[l-1](i)}$
- $a_p^{[l](i)} = f_p^l(z_n^{[l](i)})$
- $a^{[L](i)} = h(x_i) = P(y_i = 1)$

Neural networks

- Estimating a neural network $\rightarrow$ high dimensional optimization problem

Neural networks

- Estimating a neural network $\rightarrow$ high dimensional optimization problem



- Estimating a neural network $\rightarrow$ high dimensional optimization problem
- Layer 1: $3 + 3 \times p$ parameters to estimate
  - 3 bias terms $(b_1^{[1]}, b_2^{[1]}, b_3^{[1]})$
  - $3 \times p$ slope terms $(w_{n,p}^{[1]})$
- Layer 2 : $4 + 3 \times 4 = 16$parameters to estimate
  - 4 bias terms $b_1^{[2]}, b_2^{[2]}, b_3^{[2]}, b_4^{[2]}$
  - $3 \times 4 = 12$ slope terms $w_{n,p}^{[2]}$
- Output layer
  - 1 bias term
  - 4 slope terms $w_{n,p}^{[3]}$

Estimating a neural network

- It is not uncommon to see neural networks with input layer of size 10,000. 5 hidden layers, and 20 neurons in each layer.
- These are optimization problems in 25,000 dimensions (!)
- Mathematical and computational challenge. What do we do?

Gradient descent

- Gradient descent, and its variants, is the most common approach to estimate a neural network.
- Recall, our goal is to find the parameters W, b that minimize the cross-entropy cost function:

$$(W^*, b^*) \in \arg\min J(W, b; y, x) \tag{11}$$

- Gradient descent uses the derivatives of the cost function $J()$ to find parameters that minimize the cost.

Gradient descent algorithm in one dimension

We want to minimize $f(x)$. We do:
0. Pick initial guess x_0

1. Evaluate $f(x_i)$
2. Update guess $x_{i+1} = x_i - \alpha \frac{\partial f(x_i)}{\partial x}$

Repeat steps 1-2 until some stopping criterion is met.

- α learning rate
- $\frac{\partial f(x_i)}{\partial x}$ partial derivative of $f()$ with respect to x

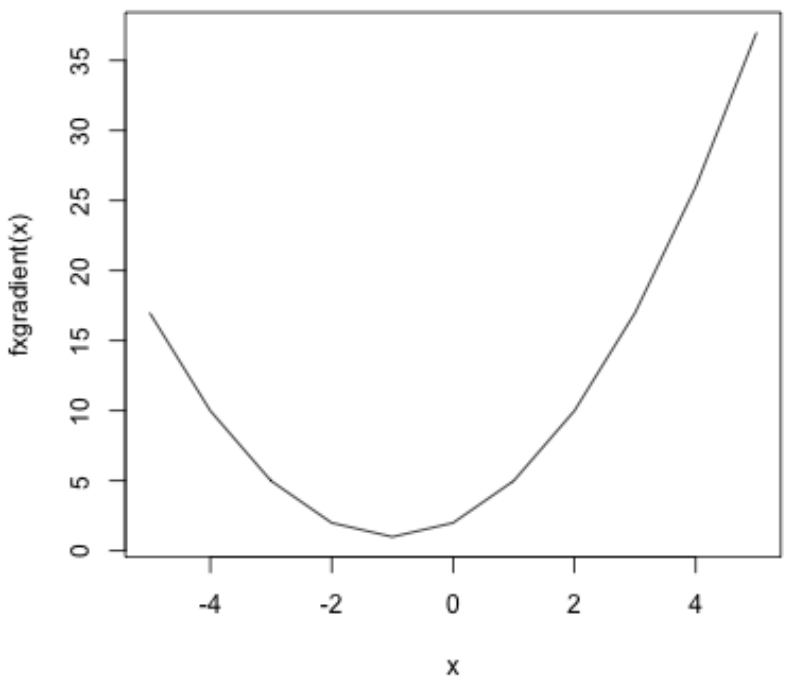
Gradient descent intuition

Let us consider the case of minimizing the function

$f(x) = x^2 + 2x + 2$

$f'(x) = 2x + 2$

$\alpha = 0.1$ learning rate



Gradient descent intuition

Let us consider the case of minimizing the function

$f(x) = x^2 + 2x + 2$

$f'(x) = 2x + 2$

$\alpha = 0.9$ learning rate

Initial guess: $x_0 = 4$.

1. $f(x_0) = f(4) = 26$
2. Update. $f'(x) = f'(4) = 10$. Derivative is positive, we will move to the left in our guess.
   $x_1 = x_0 - \alpha \times f'(x) = 4 - 0.9 \times 10 = -5$

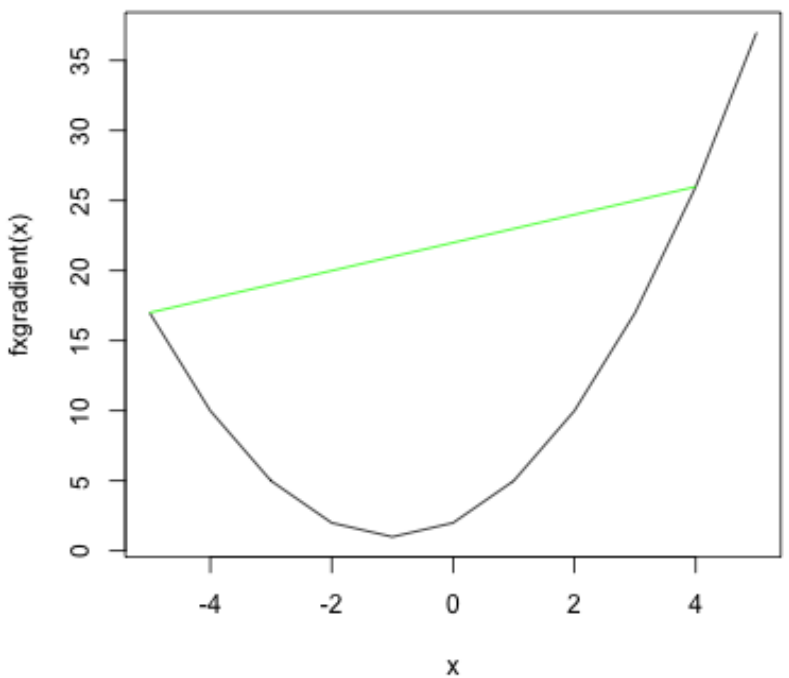
Gradient descent intuition

Let us consider the case of minimizing the function

$f(x) = x^2 + 2x + 2$

$f'(x) = 2x + 2$

$\alpha = 0.9$ learning rate



Gradient descent intuition

Let us consider the case of minimizing the function

$f(x) = x^2 + 2x + 2$

$f'(x) = 2x + 2$

$\alpha = 0.9$ learning rate

1. Evaluate $f(x_1) = f(-5) = 17$
2. Update. $f'(-5) = f'(-5) = -8$. Derivative is negative, we will move to the right in our guess.
   $x_1 = x_0 - \alpha \times f'(x) = 5 - 0.9 \times (-8) = 2.2$

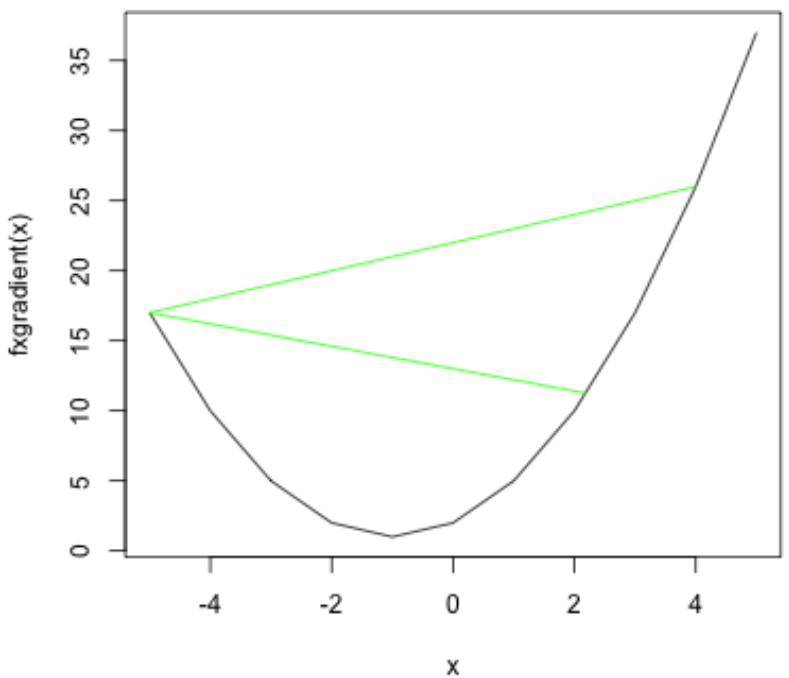
Gradient descent intuition

Let us consider the case of minimizing the function

$f(x) = x^2 + 2x + 2$

$f'(x) = 2x + 2$

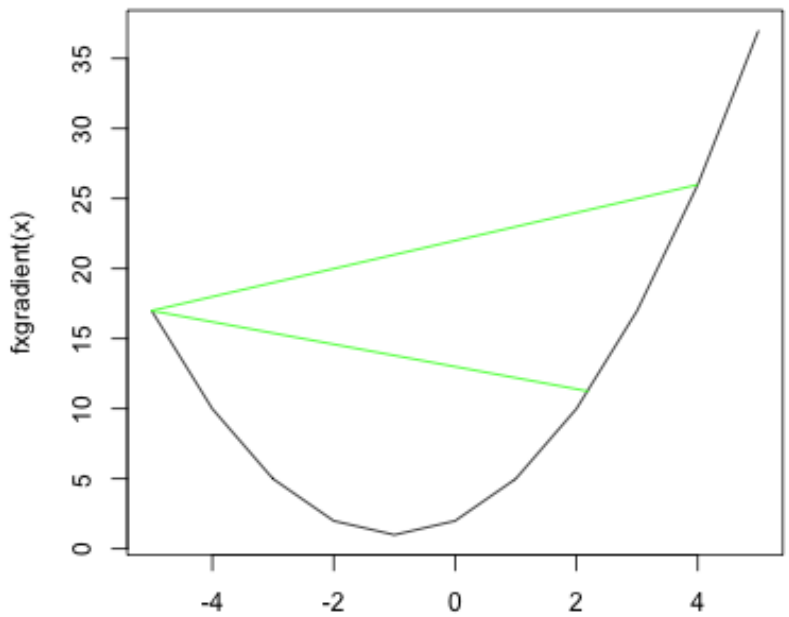
$\alpha = 0.9$ learning rate



Gradient descent intuition

We repeat until a convergence criterion is met.

- 1,000 iterations
- $f(x)$ is approximately 0...
- $f(x)$ did not improve by much
- x_i did not change much compared to x_{i-1}



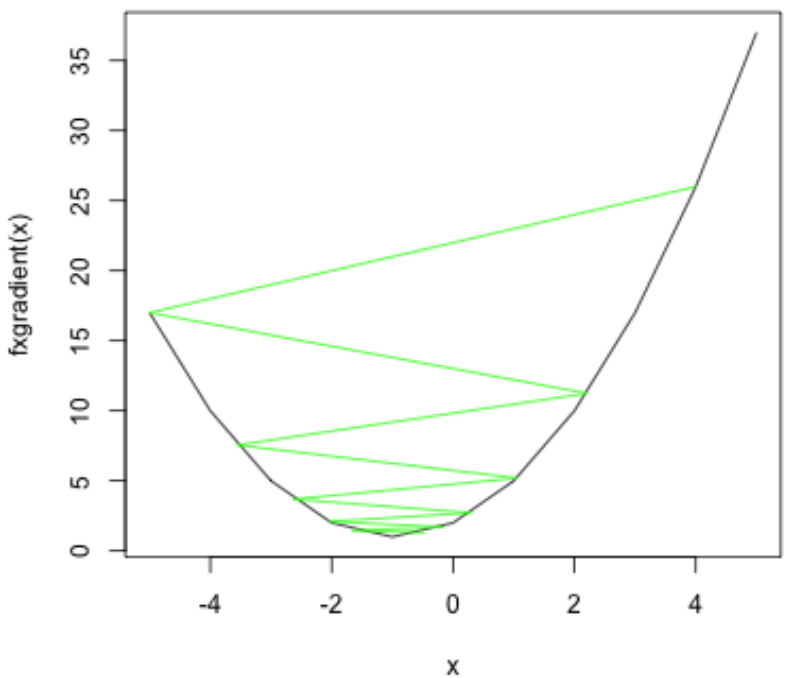
Gradient descent intuition

Let us consider the case of minimizing the function

$f(x) = x^2 + 2x + 2$

$f'(x) = 2x + 2$

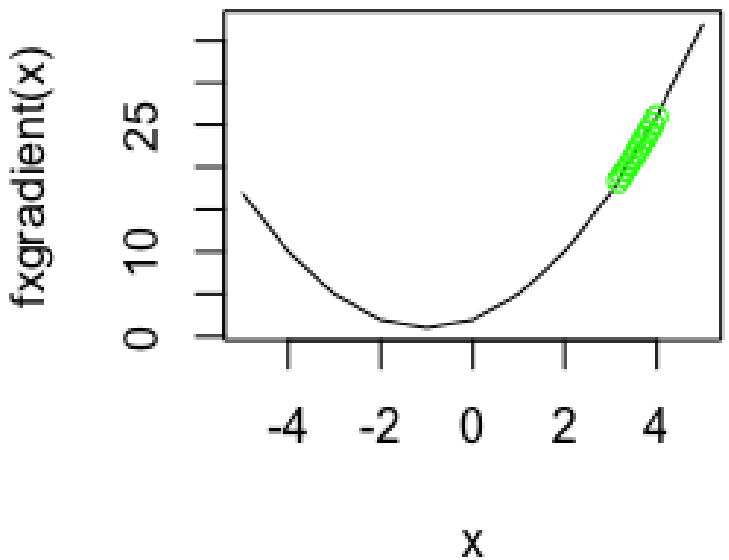
$\alpha = 0.9$ learning rate



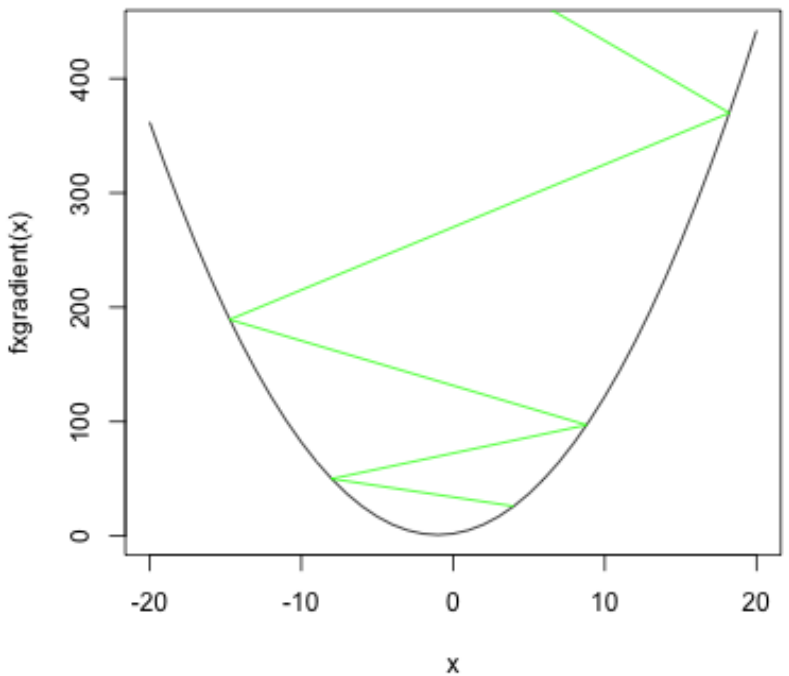
α learning rate

- Low levels of learning rate $\rightarrow$ algorithm becomes very slow
- High levels of $\rightarrow$ alpha: algorithm might not converge

- $\alpha = 0.01$; 100 iterations..



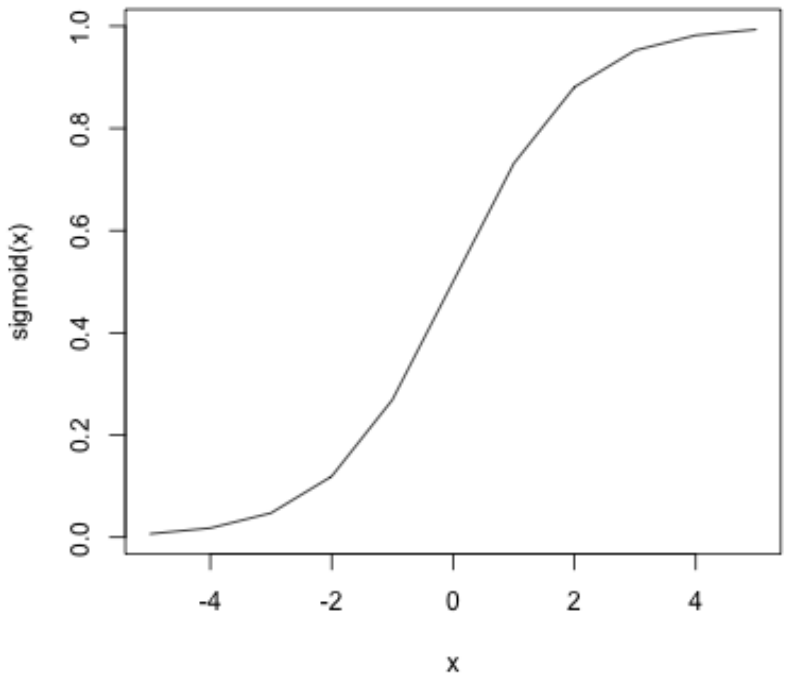
- $\alpha = 1.2$; 5 iterations..



- Cost, as a function of iterations, should go down.
- Print cost every t iterations
- Learning rate usually 0.1

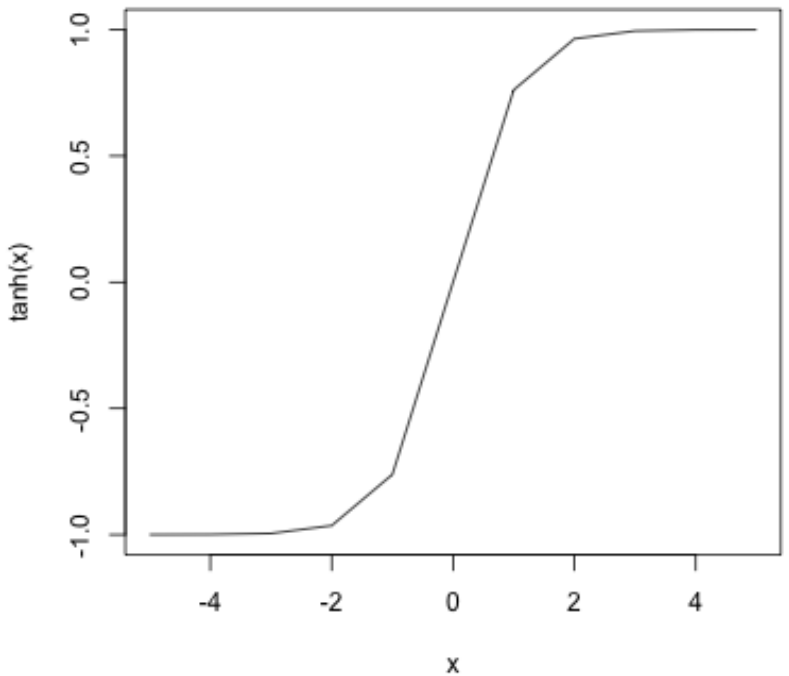
Activation functions in neural networks

- Sigmoid function: $f(z) = \frac{1}{1+e^{-z}}$



Activation functions in neural networks

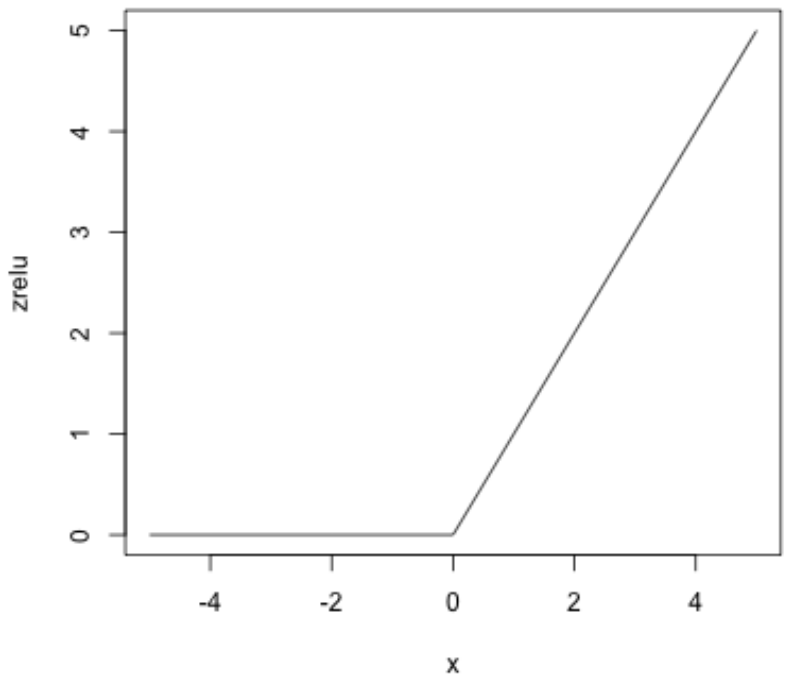
- Tanh $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



Note that $Tanh(z)$ is a transformation of the sigmoid function.
$Tanh(z) = 1 - 2 \times sigmoid(2z)$

Activation functions in neural networks

- ReLu (Rectified Linear Unit): $f(z) = \begin{cases} x \text{ if } x \geq 0 \\ 0 \text{ otherwise} \end{cases}$



Activation functions in neural networks

- Leaky ReLu (Rectified Linear Unit): $f(z) = \begin{cases} x \text{ if } x \geq 0 \\ \alpha x \text{ otherwise} \end{cases}$
- α usually 0.01