

# Neural Networks

Carlos Carvalho, Mladen Kolar and Robert McCulloch

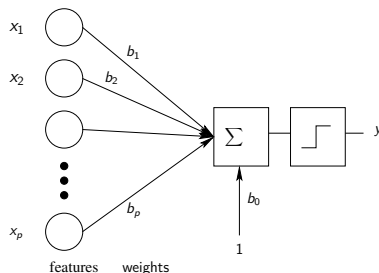
10/21/2015

# Discussion

Perdro Domingo's article

# Linear classification

We started talking about classification using perceptron



$$\text{Predict } \begin{cases} \hat{Y} = 1 & \text{if } b_0 + \sum_{j=1}^p b_j x_j \geq 0 \\ \hat{Y} = 0 & \text{if } b_0 + \sum_{j=1}^p b_j x_j < 0 \end{cases}$$

How do we get probability  $P(Y = 1 \mid x)$  ?

# Logistic regression

Model for conditional distribution of  $Y$  given  $X = x$

$$p_1(x; b) = P(Y = 1 \mid x; b) = \frac{\exp(b_0 + \sum_{j=1}^p b_j \cdot x_j)}{1 + \exp(b_0 + \sum_{j=1}^p b_j \cdot x_j)}$$

$$p_0(x; b) = P(Y = 0 \mid x; b) = \frac{1}{1 + \exp(b_0 + \sum_{j=1}^p b_j \cdot x_j)}$$

The log-odds of class 1 is a linear function of  $X$

$$\log \left( \frac{p_1(x; b)}{p_0(x; b)} \right) = b_0 + \sum_{j=1}^p b_j \cdot x_j$$

## Notation convention

Recall that  $x = (x_1, \dots, x_p)$  is our feature vector.

It is common to denote  $x_0 = 1$  and write  $x = (1, x_1, \dots, x_p)$ .

This allows us to write

$$x^T b = \sum_{j=0}^p x_j \cdot b_j$$

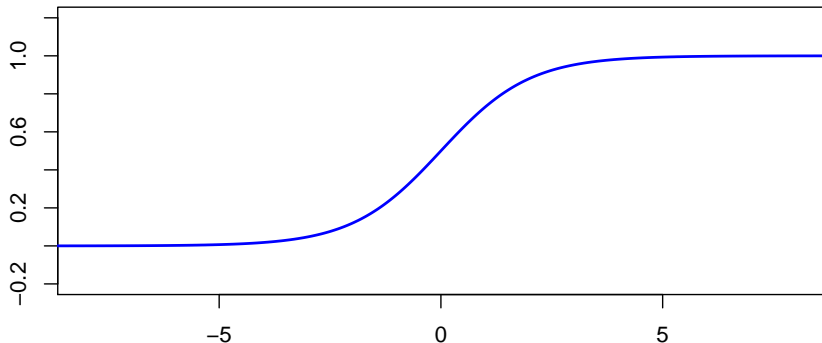
Now we can write

$$P(Y = 1 \mid x; b) = \frac{\exp(x^T b)}{1 + \exp(x^T b)}$$

and we will also be able to present our procedures without doing anything special for our intercept term (also called bias term).

# Representation with logistic regression

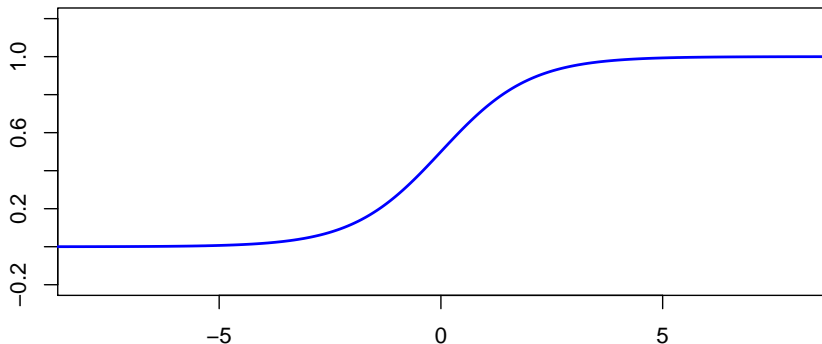
## Logistic sigmoid function



A linear function  $x^T b$  can take values from  $-\infty$  to  $\infty$ . By passing it through sigmoid, we get values in  $[0, 1]$ .

These values also sum to 1 to make a probability.

## Logistic sigmoid function



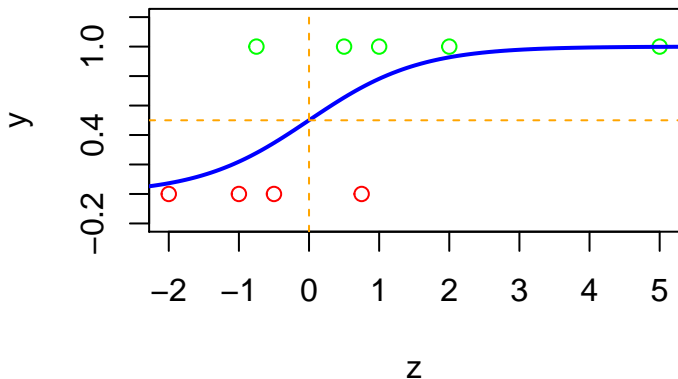
Step 1: Linear combination

$$z = x^T b$$

Step 2: Nonlinear transformation

$$P(Y = 1 \mid x; b) = F(z) = \frac{\exp(z)}{1 + \exp(z)}$$

$z$	$F(z)$
-3	0.05
-2	0.12
-1	0.27
0	0.5
1	0.73
2	0.88
3	0.95



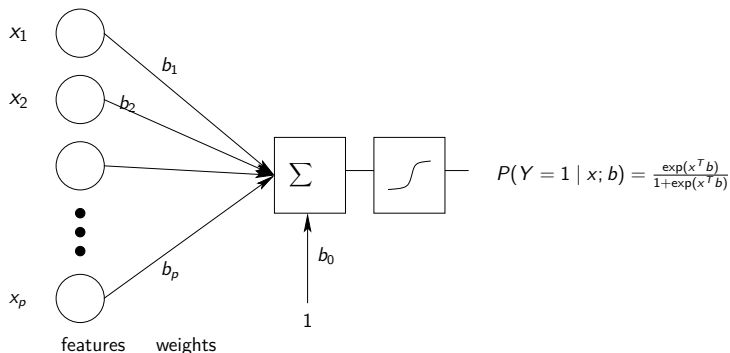
Suppose that  $\hat{Y}_i = 1$  if  $P(Y = 1 | z_i) > 0.5$  and  $\hat{Y}_i = 0$  otherwise.

When do we assign a new observation to a positive class? For what values of  $z$  do we have  $\hat{Y} = 1$ ?

What is the difference compared to perceptron?



# Representation with logistic regression



A close cousin to perceptron. Think about putting a rug over the threshold.

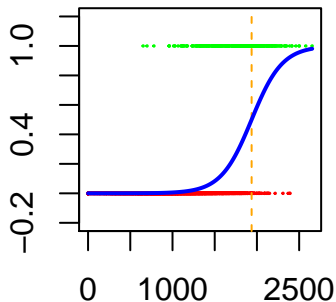
If we classify a new observation  $x$  as positive when  $p_1(x; b) > 0.5$ ,  
then 
$$\begin{cases} \hat{Y} = 1 & \text{if } x^T b > 0 \\ \hat{Y} = 0 & \text{if } x^T b \leq 0 \end{cases}$$

## Example: Predicting default

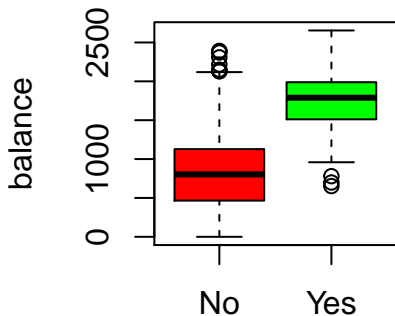
“Default” data set from ISLR package

```
df = Default  
head(df)
```

##	default	student	balance	income
## 1	No	No	729.5265	44361.625
## 2	No	Yes	817.1804	12106.135
## 3	No	No	1073.5492	31767.139
## 4	No	No	529.2506	35704.494
## 5	No	No	785.6559	38463.496
## 6	No	Yes	919.5885	7491.559



balance

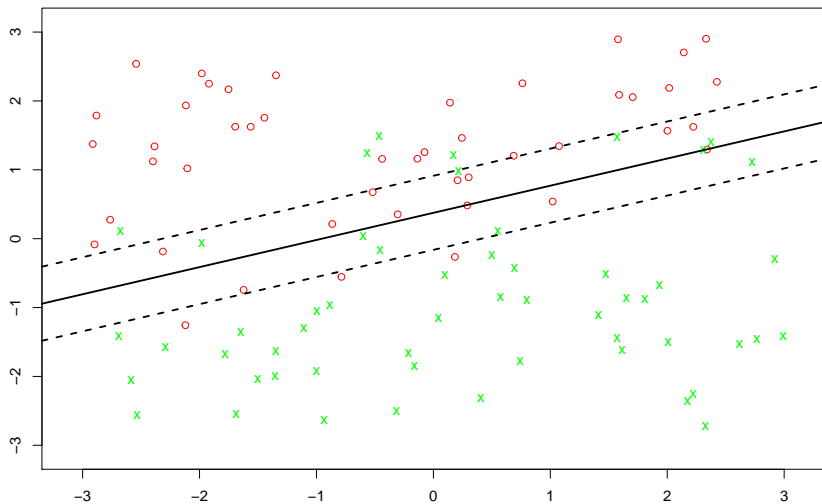


default

```
##      (Intercept)      balance
## -10.651330614    0.005498917
```

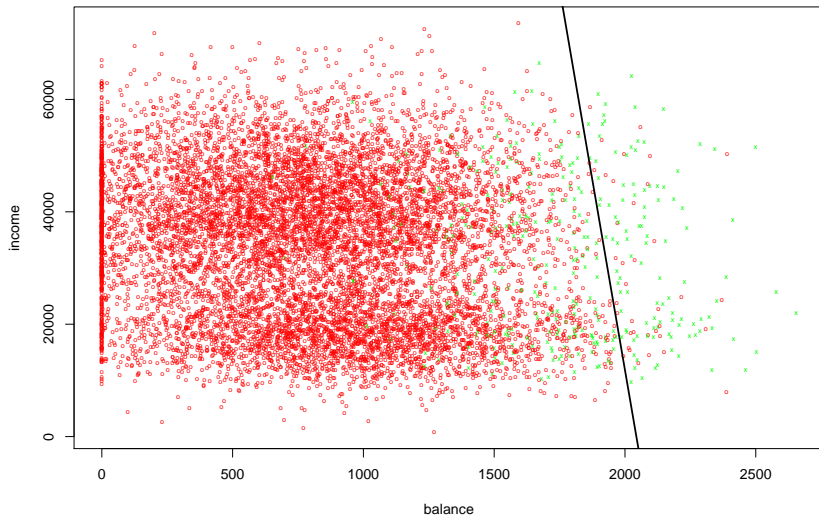
$$\hat{p}_1(x) = \frac{\exp(-10.65 + x \cdot 0.0055)}{1 + \exp(-10.65 + x \cdot 0.0055)}$$

## Simulated example in 2D



Further away an observation is from the decision boundary, more certain its label.

## Example: Predicting default



# How do we learn parameters?

Our model for  $Y$  given  $X = x$  is determined by the vector  $b$ .

Recall that  $P(Y = 1 \mid x; b) = \frac{\exp(x^T b)}{1 + \exp(x^T b)}$ .

How do we find a good  $b$ ?

We will minimize the deviance:

$$\hat{b} = \arg \min \sum_{i=1}^n -2 \times \log P(Y = y_i \mid x_i; b)$$

We need to find  $b$  that minimizes  $\sum_{i=1}^n L(y_i, x_i; b)$ .

## Another way to represent the loss: (Details)

Another common way to write logistic regression objective. . .

Consider an example  $(x_i, y_i)$

- ▶ If  $y_i = 0$ , then deviance is  $-2 \times \log(1 - p_1(x_i; b))$
- ▶ If  $y_i = 1$ , then deviance is  $-2 \times \log(p_1(x_i; b))$

Another way to write loss for this example

$$\begin{aligned} L(y_i, x_i; b) &= -\log(P(Y = y_i \mid x_i; b)) \\ &= -(1 - y_i) \log(1 - p_1(x_i; b)) - y_i \log p_1(x_i; b) \end{aligned}$$

Loss on training data

$$J(b) = \sum_{i=1}^n L(y_i, x_i; b)$$

# Minimization procedures

Find  $b$  that makes the loss  $J(b)$  as small as possible.

Many different numerical procedures – treat them as black boxes.

Still you need to know some high-level details.

Optimization procedures commonly need to compute

- ▶ loss  $J(b)$

- ▶ (partial) derivative of the loss  $\nabla J(b) = \begin{pmatrix} \partial_1 J(b) \\ \vdots \\ \partial_p J(b) \end{pmatrix}$

- ▶ second (partial) derivative

$$\Delta J(b) = \begin{pmatrix} \partial_{11} J(b) & \partial_{12} J(b) & \dots & \partial_{1p} J(b) \\ \partial_{21} J(b) & \partial_{22} J(b) & \dots & \partial_{2p} J(b) \\ \vdots & & \ddots & \vdots \\ \partial_{p1} J(b) & \partial_{p2} J(b) & \dots & \partial_{pp} J(b) \end{pmatrix}$$



# Minimization procedures

## Derivative free minimization

- ▶ require  $J(b)$  only
- ▶ useful when computing  $\nabla J(b)$  is very expensive

## Gradient descent and other first order methods

- ▶ require  $J(b)$  and  $\nabla J(b)$
- ▶ useful when dealing with large amounts of data
- ▶ many iterations to get high accuracy solution, but each iteration is cheap (We do not care about high accuracy. Why?)

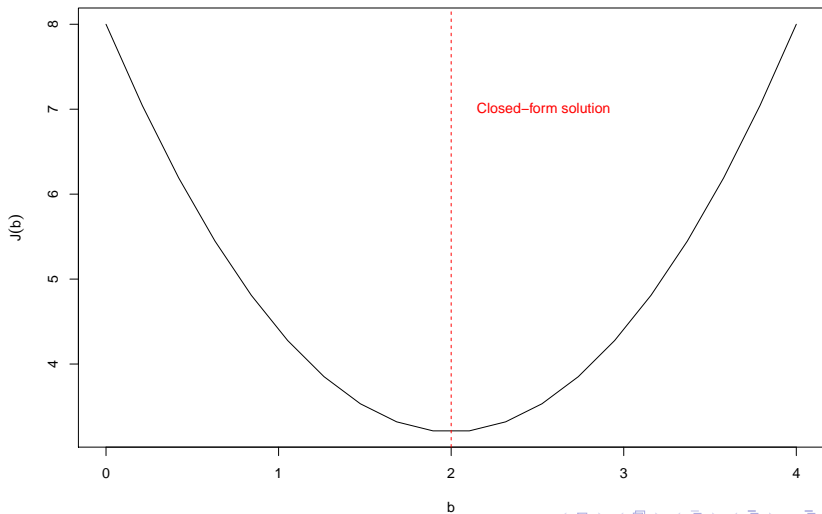
## Second order methods (Newton's method)

- ▶ require  $J(b)$ ,  $\nabla J(b)$ , and  $\Delta J(b)$
- ▶ few iteration to find minimum, but each iteration can be expensive
- ▶ R packages (that I have seen) implement this

## Gradient descent

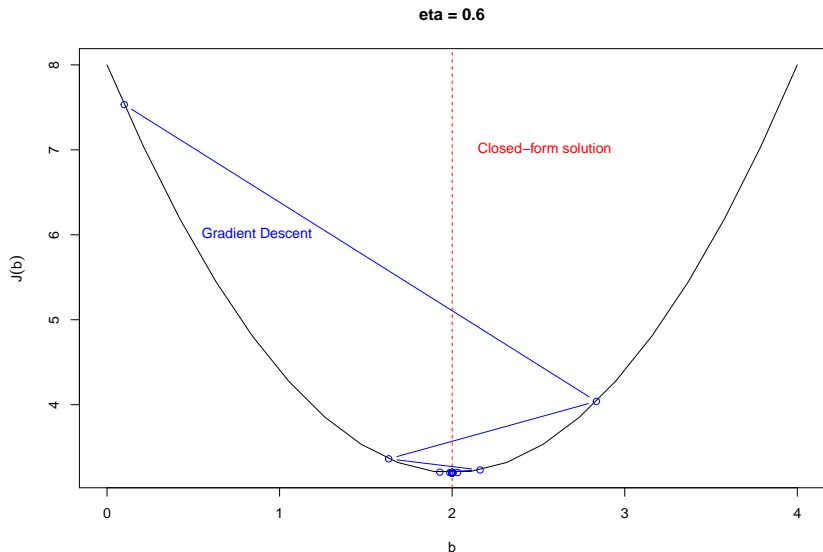
Suppose that we want to find  $b$  that minimizes the following function

$$J(b) = 1.2(b - 2)^2 + 3.2$$

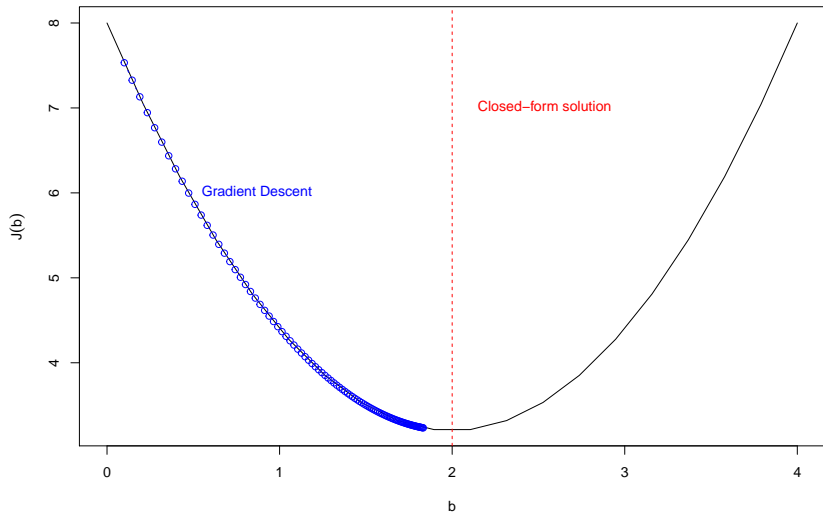


Set  $b^0 = 0$ . Iteratively update  $b^{t+1} = b^t - \eta \partial J(b^t)$ .

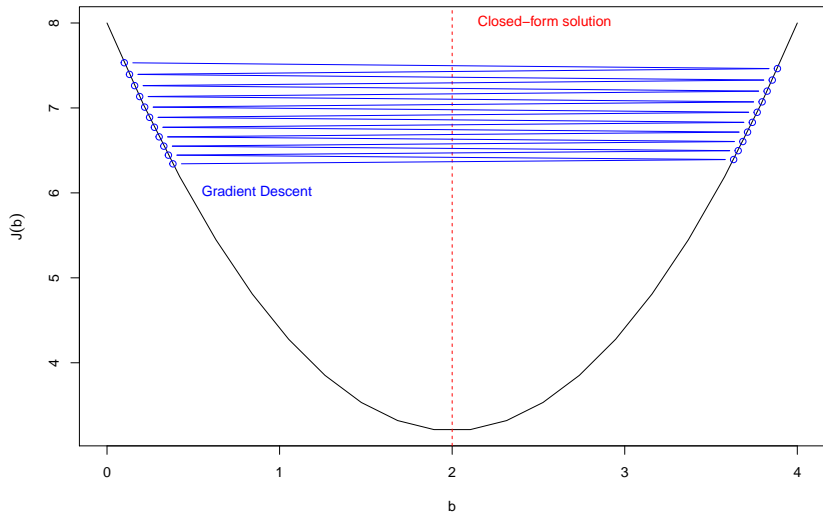
$\eta$  is the step size. Also known as the learning rate.



**eta = 0.01**



**eta = 0.83**



# Gradient descent

The learning rate  $\eta$  plays a crucial role in convergence.

For multivariate problems, where  $b = (b_0, b_1, \dots, b_p)$ , the gradient descent takes the following form:

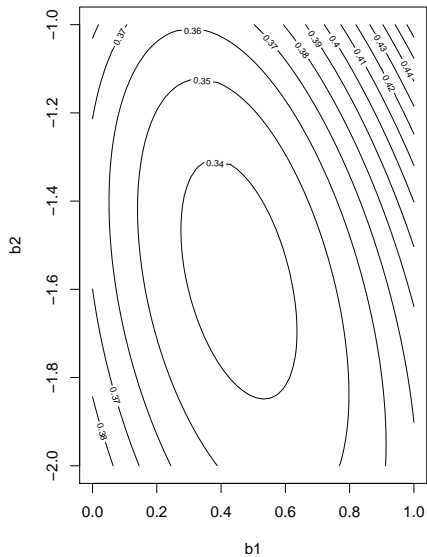
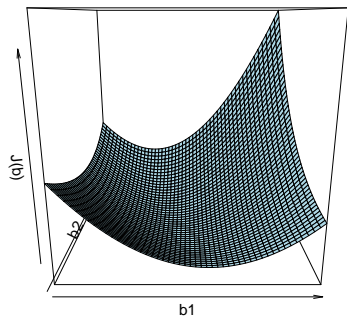
- ▶ Start with  $b^0 = (0, \dots, 0)$ .
- ▶ In each iteration,  $t = 1, 2, \dots$  update

$$b_j^{t+1} = b_j^t - \eta \partial_j J(b^t), \quad \text{for all } j = 0, 1, \dots, p$$

When to stop?

- ▶  $|J(b^t) - J(b^{t-1})|$  is small
- ▶  $|b^t - b^{t-1}|$  is small
- ▶  $\|\nabla J(b^t)\|$  is small
- ▶ ...

# Gradient descent



# Gradient descent



# Gradient descent for logistic regression

Logistic regression loss per example

$$L(y_i, x_i; b) = -(1 - y_i) \log(1 - p_1(x_i; b)) - y_i \log p_1(x_i; b)$$

Gradient of the loss per example

$$\partial_j L(y_i, x_i; b) = (p_1(x_i; b) - y_i) x_{ij}$$

The overall gradient

$$\partial_j J(b) = \sum_{i=1}^n (p_1(x_i; b) - y_i) x_{ij}$$

# Gradient descent for logistic regression

Initialize  $b = (0, \dots, 0)$

**repeat**

    Let  $g = (0, \dots, 0)$  be the gradient vector

**for**  $i = 1:n$  **do**

$p_i = \exp(x_i^T b) / (1 + \exp(x_i^T b))$

$\text{error}_i = y_i - p_i$

$g = g + \text{error}_i \cdot x_i$

**end**

$b = b - \eta \cdot g / n$

**until** *until convergence*;

Note that algorithm uses all observations to compute the gradient.

This approach is called batch gradient descent.

# Stochastic gradient descent for logistic regression

Initialize  $b = (0, \dots, 0)$

**repeat**

    Pick observation  $i$

$$p_i = \exp(x_i^T b) / (1 + \exp(x_i^T b))$$

$$\text{error}_i = y_i - p_i$$

$$b = b - \eta(\text{error}_i \cdot x_i)$$

**until** *until convergence*;

Coefficient  $b$  is updated after each observation.

In practice, mini batch is often used.

- ▶ Compute gradient based on a small subset of observations
- ▶ Make update to coefficient vector

## Extending logistic regression to $K > 2$ classes

Choose class  $K$  to be the reference class

Represent each of the other classes as a logistic function of the odds of class  $k$  versus class  $K$

$$\log \left( \frac{P(Y = 1 | x)}{P(Y = K | x)} \right) = x^T b_1$$

$$\log \left( \frac{P(Y = 2 | x)}{P(Y = K | x)} \right) = x^T b_2$$

$$\vdots$$

$$\log \left( \frac{P(Y = K - 1 | x)}{P(Y = K | x)} \right) = x^T b_{K-1}$$

## Extending logistic regression to $K > 2$ classes

The conditional probability for class  $k \neq K$

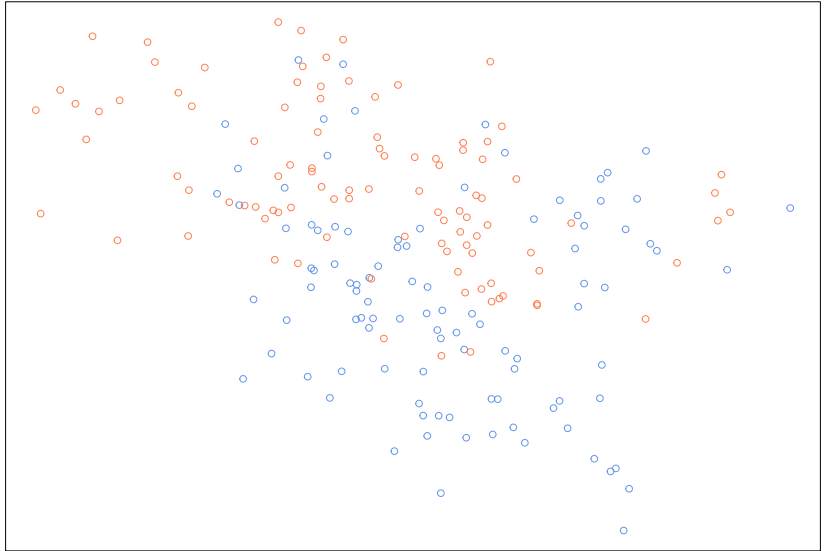
$$P(Y = k | x) = \frac{\exp(x^T b_k)}{1 + \sum_{l=1}^{K-1} \exp(x^T b_l)}$$

The conditional probability for class  $K$

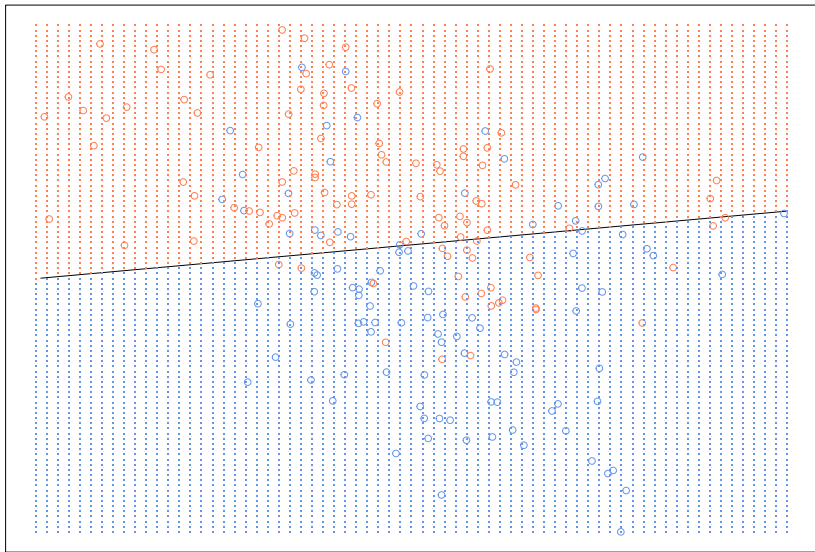
$$P(Y = K | x) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp(x^T b_l)}$$

Fitting can be again done using a number of procedures that will simultaneously obtain coefficients for each class  $b_1, \dots, b_{K-1}$

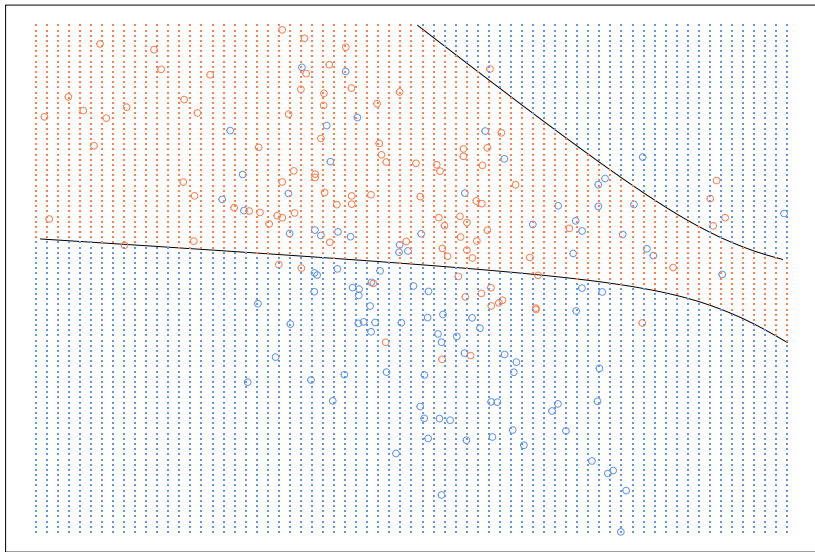
# Avoiding overfitting



## logistic regression

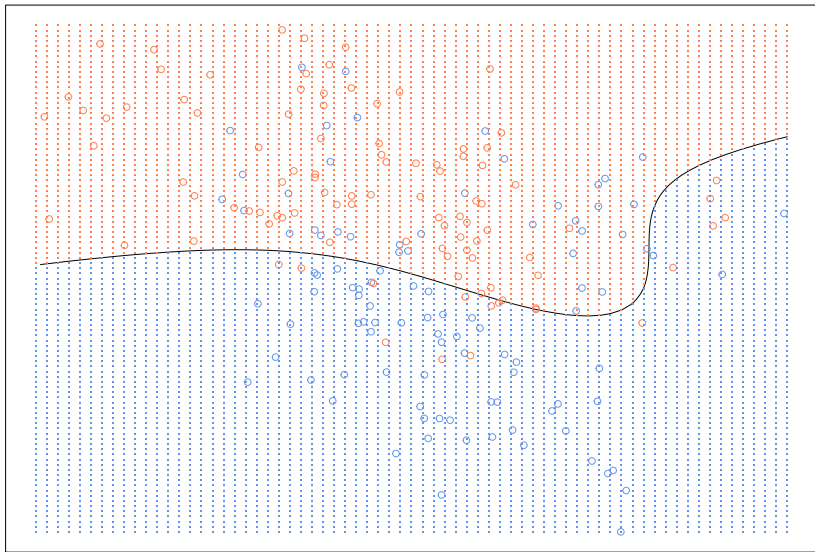


## logistic regression — quadratic with interactions

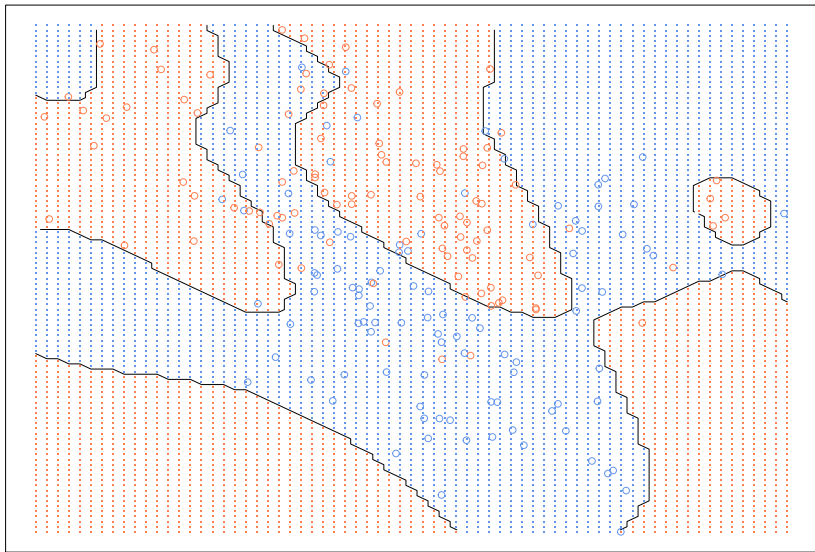




## logistic regression -- cubic with interactions



logistic regression -- 7th degree polynomial



```
lr_fit$coefficients[1:10]
```

```
##      (Intercept)          x.1.0          x.2.0
## -1.100958e+15 -1.733179e+15 -9.387873e+13
##           x.3.0          x.4.0          x.5.0
##  3.202179e+15 -1.736921e+15 -2.390935e+13
##           x.6.0          x.7.0          x.0.1
##  1.699264e+14 -2.496843e+13  1.283042e+15
##           x.1.1
## -7.430262e+15
```

# Avoiding overfitting using L1/L2 penalization

Find  $b$  that minimizes

$$J(b) + \lambda \left( \frac{1-\alpha}{2} \sum_{j=1}^p b_j^2 + \alpha \sum_{j=1}^p |b_j| \right)$$

This is known as an elastic net penalty (see glmnet package)

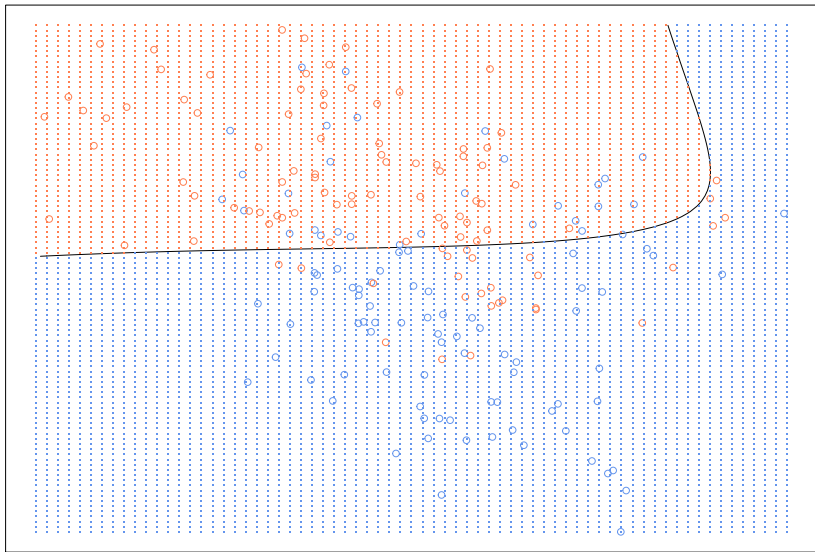
$\lambda$  is the penalty parameter

- ▶ large value: prefer simple models
- ▶ small value: prefer complex models (no regularization)

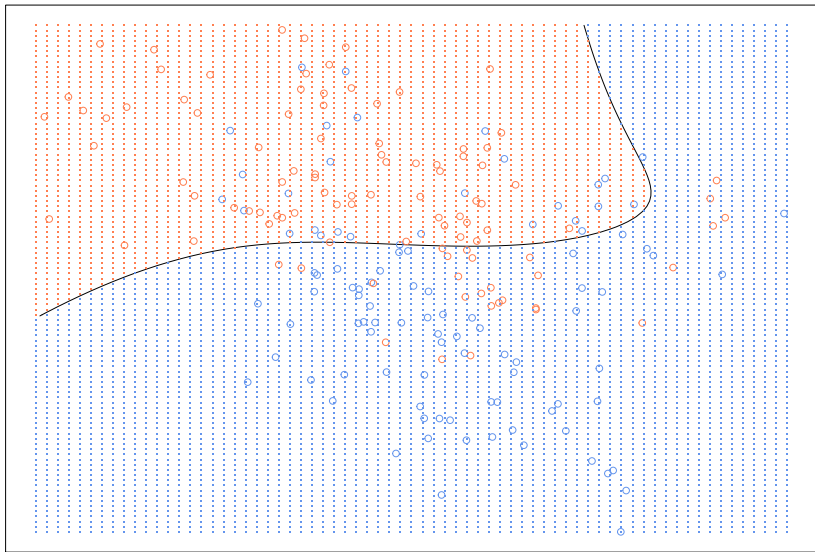
$\alpha$  is the elasticnet mixing parameter

- ▶  $\alpha = 0$ : ridge penalty
- ▶  $\alpha = 1$ : lasso penalty

## logistic regression -- lasso



## logistic regression -- ridge penalty



# Example: Predicting quality of restaurant

Can we predict quality of a restaurant based on a review?

```
# data from textir package  
data(we8there)  
dim(we8thereCounts)
```

```
## [1] 6166 2640
```

Data are very high dimensional.

But fortunately not many words per document.

An example of a positive review

```
we8thereCounts[1, we8thereCounts[1, ]!=0]
```

```
## even though larg portion mouth water
##          1          1          1
##    red sauc    babi back    back rib
##          1          1          1
## chocol mouss veri satisfi
##          1          1
```



An example of a negative review

```
we8thereCounts[6100, we8thereCounts[6100, ]!=0]
```

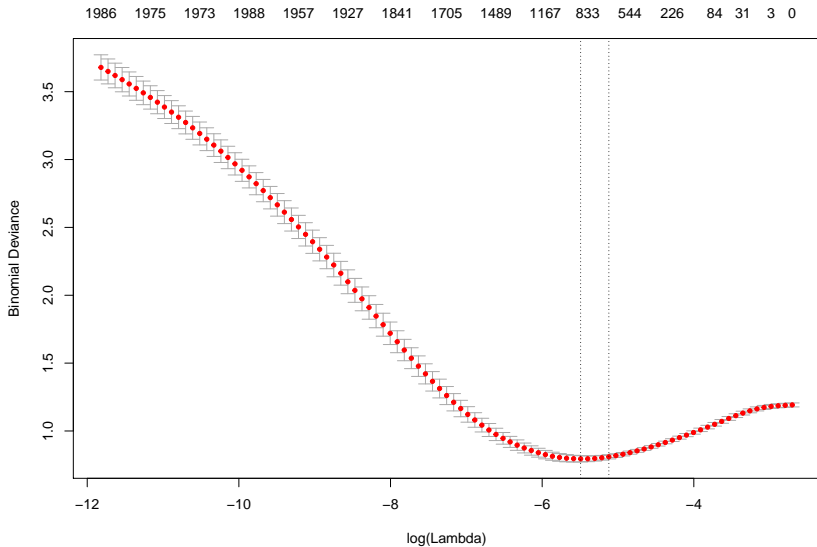
```
## never return      just few      one side
##           1           1           1
##      tri make      go box    order wrong
##           1           1           1
## place clean    order just      seat one
##           1           1           1
##      side build
##           1
```

```
we8thereRatings[1:3,]
```

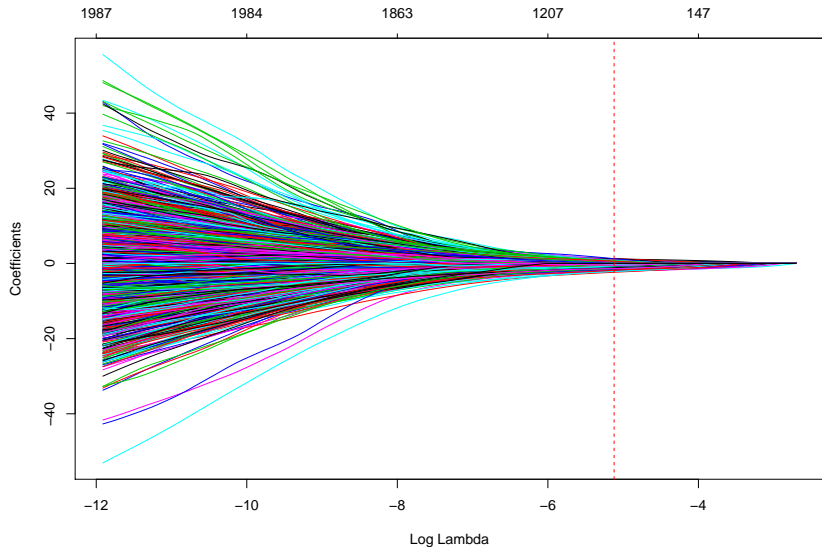
	Food	Service	Value	Atmosphere	Overall
## 1	5	5	5	5	5
## 2	5	5	5	5	5
## 5	5	5	4	4	5

```
# transform rating into {0, 1}
y = ifelse(we8thereRatings$Overall>3, 1, 0)
y = as.factor(y)
glm_fit = cv.glmnet(x = we8thereCounts, y = y,
                    family = "binomial",
                    # lasso - 1, ridge - 0
                    alpha = 1,
                    nfold = 5
                    )
```

```
plot(glm_fit)
```



```
plot(glm_fit$glmnet.fit, xvar = "lambda")  
abline(v = log(glm_fit$lambda.1se), lty=2, col="red")
```



```
glm.coef = coef(glm_fit$glmnet.fit, s=glm_fit$lambda.1se)  
o = order( glm.coef, decreasing = TRUE )
```

```
# positive coefficients
```

```
glm.coef@Dimnames[[1]][o[1:10]]
```

```
## [1] "can wait" "between two"  
## [3] "beef sandwich" "high recommend"  
## [5] "friend help" "best meal"  
## [7] "food delici" "(Intercept)"  
## [9] "melt mouth" "wonder experi"
```

```
glm.coef[o[1:10]]
```

```
## [1] 1.2741278 1.2633252 1.2482930 1.2121773  
## [5] 1.0930038 1.0704775 1.0245261 1.0169857  
## [9] 1.0118636 0.9496681
```

```
# negative coefficients
```

```
glm.coef@Dimnames[[1]][tail(o,10)]
```

```
## [1] "gone down"      "never go"
## [3] "servic terribl"  "food terribl"
## [5] "stay away"       "never return"
## [7] "far better"      "mediocr best"
## [9] "veri rude"       "extrem rude"
```

```
glm.coef[tail(o,10)]
```

```
## [1] -1.842253 -1.875542 -1.892049 -1.939003
## [5] -1.949946 -2.025724 -2.107226 -2.109239
## [9] -2.191691 -2.348292
```

# Avoiding overfitting using early stopping

Common practice is to stop gradient descent before it reaches optimum

One monitors performance of the current solution on the validation data

If the performance starts to deteriorate, stop the descent

Why should this work?

# Summary: Logistic regression

Learns conditional probability distribution

Linear decision boundary. We can craft non-linear features (feature engineering), but the decision is going to be linear in the non-linear features.

Gradient descent starts with an initial vector and modifies it to find best parameter.



# Neural networks

Our learning algorithms so far:

Training data:  $(x_i, y_i)_{i=1}^n \longrightarrow \text{Machine Learning} \longrightarrow y = \hat{f}(x)$

All of the procedures directly work on input features.

What if the input features are not informative?

# Neural networks

Feature engineering — handcrafting transformations

Training data:  $(x_i, y_i)_{i=1}^n \longrightarrow \Phi \longrightarrow (\Phi_x(x_i), \Phi_y(y_i))_{i=1}^n$

Here  $\Phi$  is designed by a human.

$(\Phi_x(x_i), \Phi_y(y_i))_{i=1}^n \longrightarrow \text{Machine Learning} \longrightarrow (\Phi_y(x_i), \Phi_y(y_i))_{i=1}^n$

This process is expensive and time consuming.

## Example: Handwritten Digit Recognition

$$P(y = 2 \mid \text{2}, b)$$

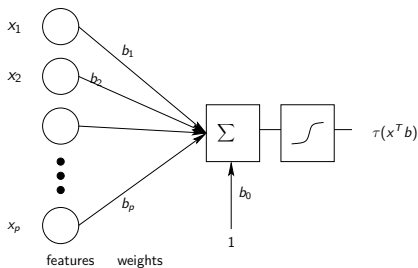
$$P(y = 9 \mid \text{9}, b)$$

How to represent image?

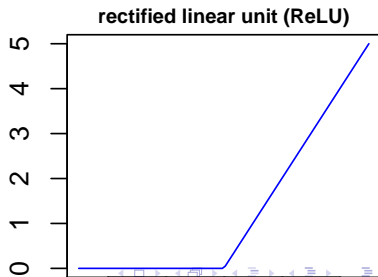
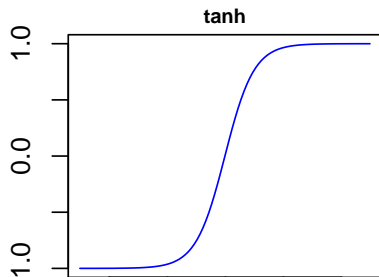
How informative is each pixel?

Logistic regression trained on pixel values gives ~80% accuracy.

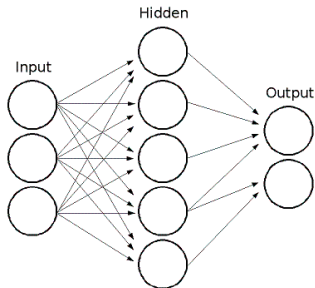
# Model of a neuron



## Other nonlinear activations



# Multilayer Perceptron



## 2 Layers of Neurons

- ▶ 1st layer takes input  $x$
- ▶ 2nd layer takes output of 1st layer
- ▶ The last layer is the output

# Multilayer Perceptron

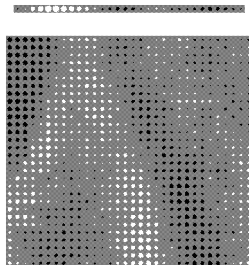
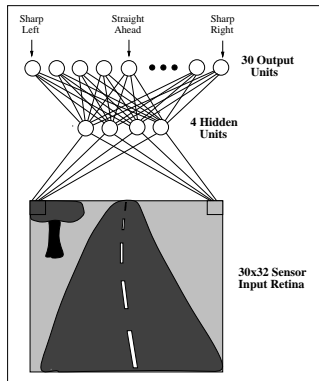
The activities of the neurons in each layer are a non-linear function of the activities in the layer below.

Can approximate arbitrary functions

- ▶ Provided hidden layer is large enough
- ▶ “fat” 2-layer network

# Example: ALVINN

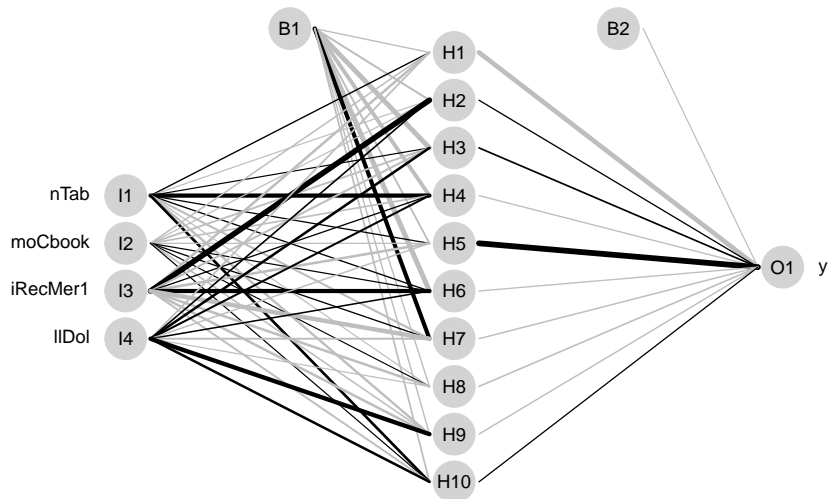
## Autonomous Land Vehicle In a Neural Network



Video: <http://watson.latech.edu/book/intelligence/intelligenceOverview5b4.html>

## Example: Tabloid data

```
plot.nnet(nnetfit)
```

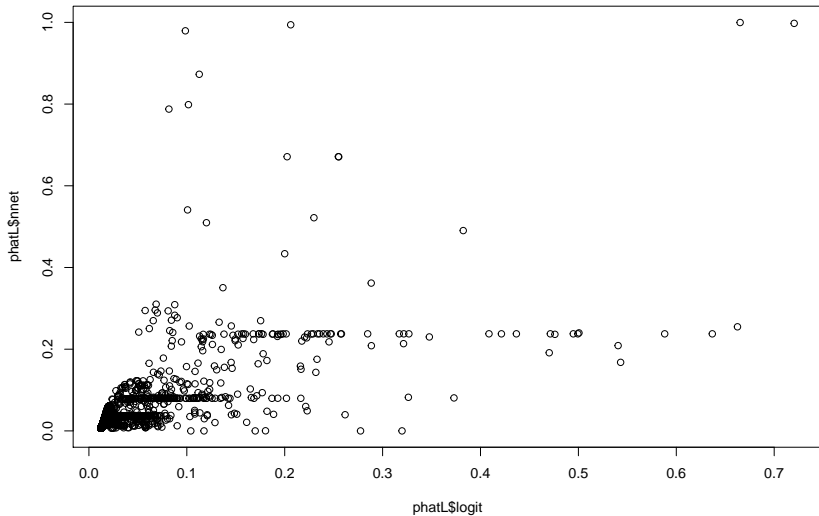


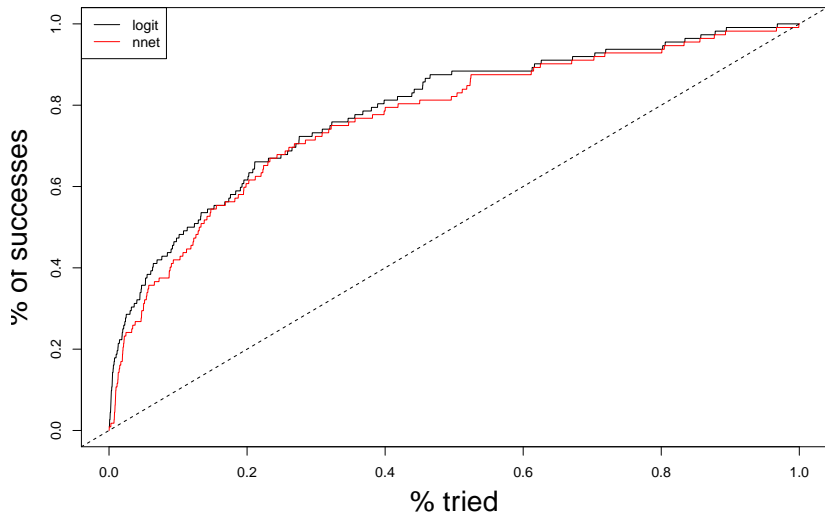


```
print(summary(nnetfit))
```

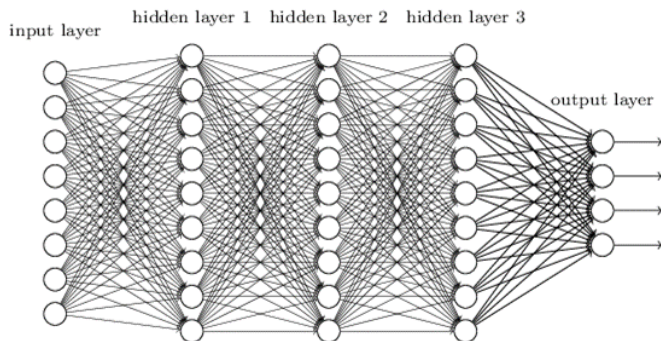
```
a 4-10-1 network with 61 weights
options were - entropy fitting decay=5e-04
b->h1 i1->h1 i2->h1 i3->h1 i4->h1
  0.65  -0.98  -1.31  -0.29  -2.15
b->h2 i1->h2 i2->h2 i3->h2 i4->h2
-13.78  -4.60 -10.68  -9.75  28.54
b->h3 i1->h3 i2->h3 i3->h3 i4->h3
  7.01  -3.10  -4.49  -4.78 -16.22
...
b->h9 i1->h9 i2->h9 i3->h9 i4->h9
 14.92  -7.47  -4.49 -10.16  17.72
b->h10 i1->h10 i2->h10 i3->h10 i4->h10
 14.79   -7.03   17.36   6.31   17.41
b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o h10->o
 9.83  -2.00   3.44 -14.40  10.74  -1.23  -1.90 -11.56 -16.53 -15.03  -9.75
```

```
plot(phatL$logit, phatL$nnet)
```



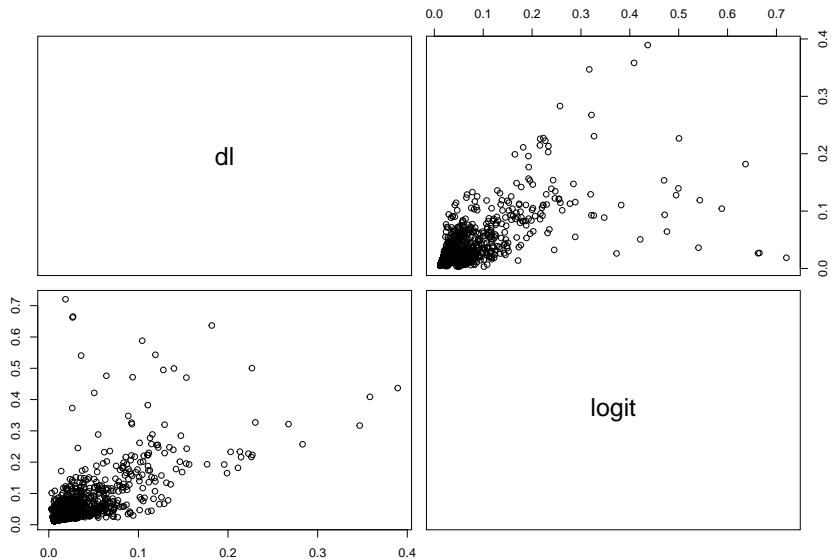


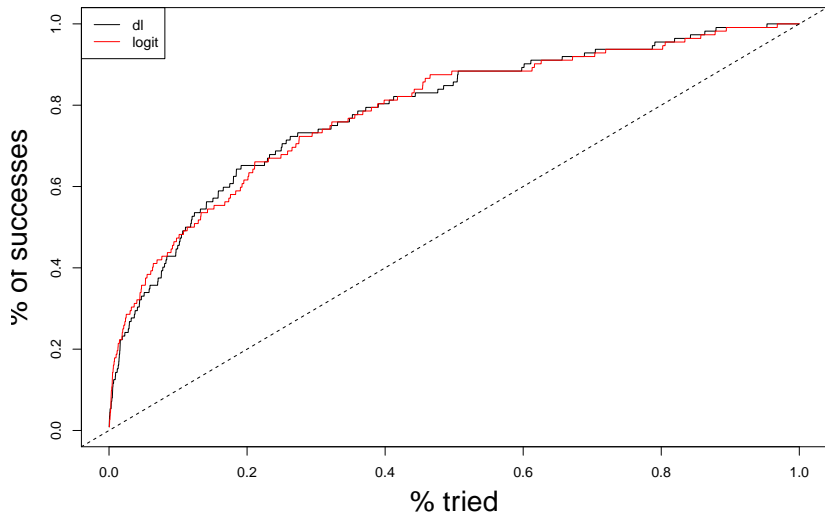
# Deep neural network



If there is more than one hidden layer, networks are called “deep” neural networks.

# Tabloid again





# Fitting neural networks

Gradient descent + chain rule + lot of tricks

- ▶ We will not provide details
- ▶ The procedure is called backpropagation

Difficult to train because there are many local minima

- ▶ Train multiple nets with different initial weights
- ▶ Initialize weights near zero
- ▶ Therefore, initial networks near-linear
- ▶ Increasingly non-linear functions possible as training progresses

# Avoiding overfitting

## Regularization

- ▶ L1 penalty on the parameters
- ▶ L2 penalty on the parameters (weight decay parameter)
- ▶ Early stopping

## Momentum

- ▶  $b^{t+1} = b^t - \eta \cdot \nabla J(b) + \alpha(b^t - b^{t-1})$
- ▶  $\alpha$  is the momentum parameter
- ▶ helps avoiding stuck in a local optimum

## Dropout



# Example: MNIST

Famous data set in machine learning community

- ▶ <http://yann.lecun.com/exdb/mnist/>

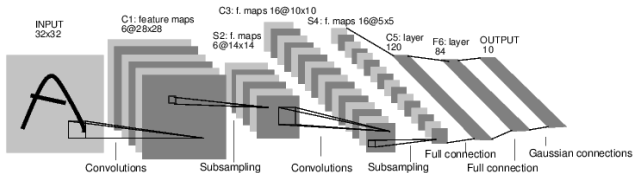
Even today: Kaggle competition

- ▶ <https://www.kaggle.com/c/digit-recognizer>

Online demo

- ▶ <http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>

# LeNet5: convolutional neural network

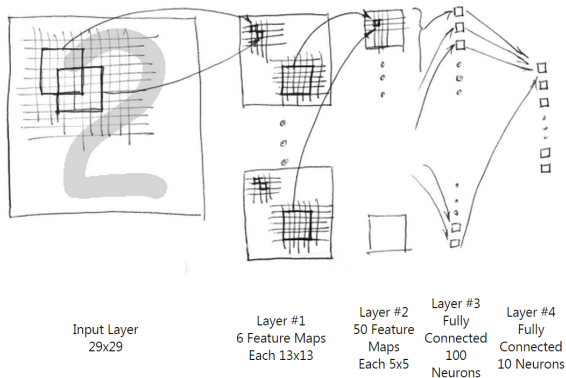


See <http://yann.lecun.com/exdb/lenet/>

# Mistakes made by LeNet5



# A simpler architecture



From: <http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi>

# Practical consideration

Standard trick — expand the set of examples

- ▶ small distortions, scaling, rotation, . . .

What else needs to be done to make system useful?

# Advantages and disadvantages

## Pros:

- ▶ Tolerance to noise
- ▶ Able to capture complex signals
- ▶ In some applications lead to the state-of-the-art performance
- ▶ Fast at test time

## Cons:

- ▶ Very hard/impossible to interpret (black box method)
- ▶ Can easily overfit
- ▶ Need a large amount of data to train
- ▶ Slow to train