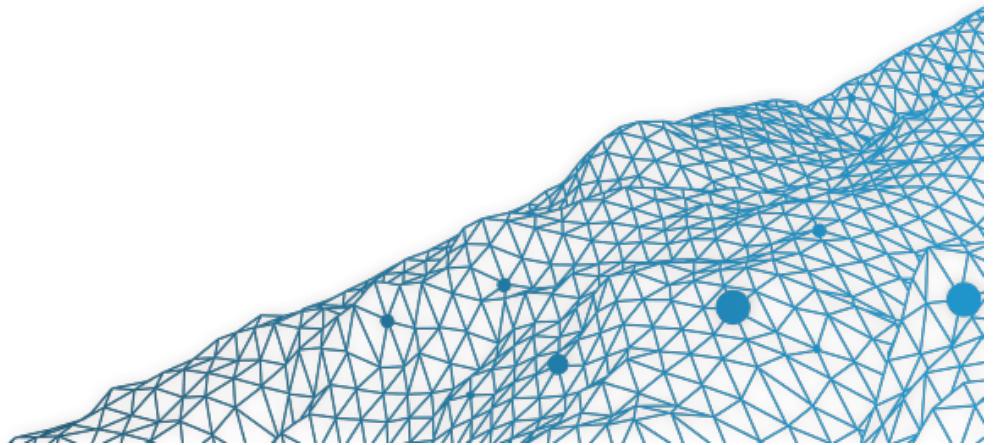


## # 4 Classification & Neural Nets

Arthur Charpentier (Université du Québec à Montréal)

Machine Learning & Econometrics

SIDE Summer School - July 2019



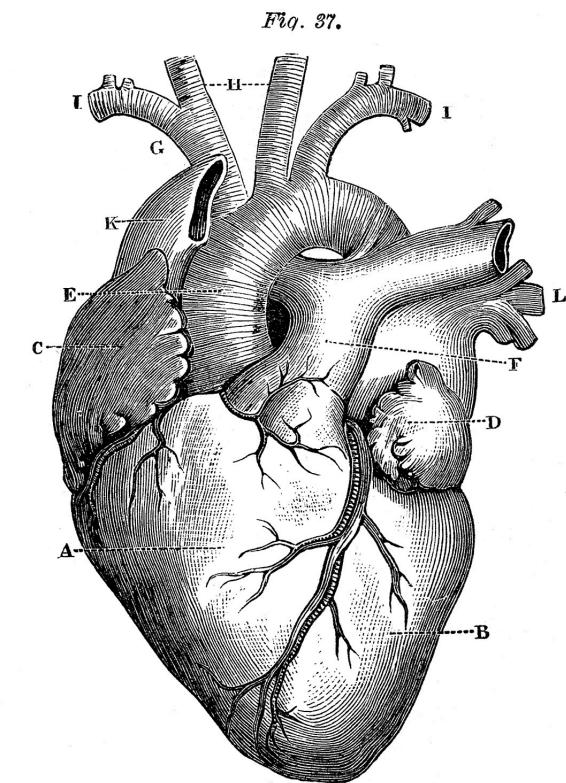
## Modeling a 0/1 random variable

Myocardial infarction of patients admitted in E.R.

```
1 > myocarde=read.table("http://freakonometrics.free.fr/myocarde.csv",
  head=TRUE , sep=";" )
```

dataset  $\{(x_i, y_i)\}$  where  $i = 1, \dots, n$ ,  $n = 73$ .

- |         |                                     |
|---------|-------------------------------------|
| 1 FRCAR | ○ heart rate $x_1$                  |
| 2 INCAR | ○ heart index $x_2$                 |
| 3 INSYS | ○ stroke index $x_3$                |
| 4 PRDIA | ○ diastolic pressure $x_4$          |
| 5 PAPUL | ○ pulmonary arterial pressure $x_5$ |
| 6 PVENT | ○ ventricular pressure $x_6$        |
| 7 REPUL | ○ lung resistance $x_7$             |
| 8 PRONO | ○ death or survival $y$             |



## Classification : (Linear) Fisher's Discrimination

Consider the following naive classification rule

$$m^*(\mathbf{x}) = \operatorname{argmin}_y \{\mathbb{P}[Y = y | \mathbf{X} = \mathbf{x}]\}$$

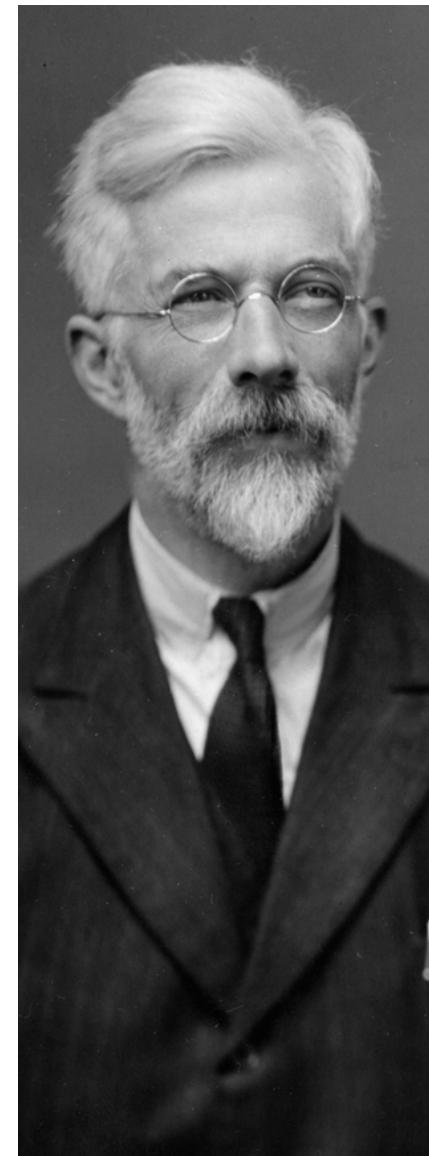
or

$$m^*(\mathbf{x}) = \operatorname{argmin}_y \left\{ \frac{\mathbb{P}[\mathbf{X} = \mathbf{x} | Y = y]}{\mathbb{P}[\mathbf{X} = \mathbf{x}]} \right\}$$

(where  $\mathbb{P}[\mathbf{X} = \mathbf{x}]$  is the density in the continuous case).

In the case where  $y$  takes two values, that will be standard  $\{0, 1\}$  here, one can rewrite the later as

$$m^*(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbb{E}(Y | \mathbf{X} = \mathbf{x}) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$



## Classification : (Linear) Fisher's Discrimination

the set

$$\mathcal{D}_S = \left\{ \mathbf{x}, \mathbb{E}(Y|\mathbf{X} = \mathbf{x}) = \frac{1}{2} \right\}$$

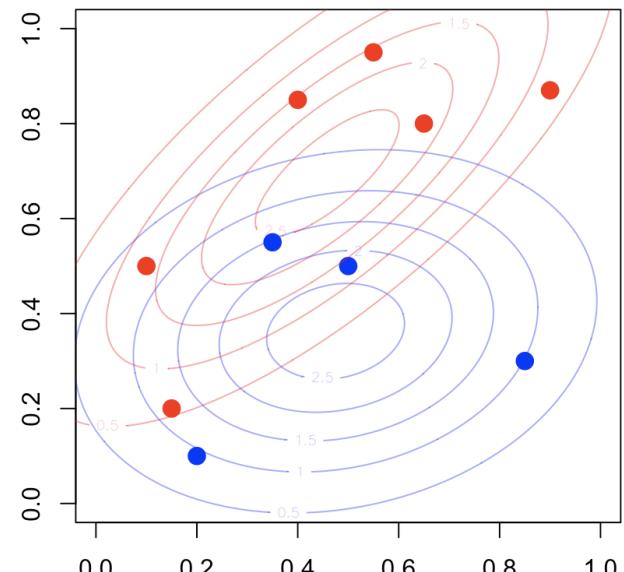
is called the decision boundary.

Assume that  $\mathbf{X}|Y = 0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma})$  and  $\mathbf{X}|Y = 1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ , then explicit expressions can be derived.

$$m^*(\mathbf{x}) = \begin{cases} 1 & \text{if } r_1^2 < r_0^2 + 2\log \frac{\mathbb{P}(Y = 1)}{\mathbb{P}(Y = 0)} + \log \frac{|\boldsymbol{\Sigma}_0|}{|\boldsymbol{\Sigma}_1|} \\ 0 & \text{otherwise} \end{cases}$$

where  $r_y^2$  is the Mahalanobis distance,

$$r_y^2 = [\mathbf{X} - \boldsymbol{\mu}_y]^\top \boldsymbol{\Sigma}_y^{-1} [\mathbf{X} - \boldsymbol{\mu}_y]$$



## Classification : (Linear) Fisher's Discrimination

Let  $\delta_y(\mathbf{x})$  be defined as

$$-\frac{1}{2} \log |\Sigma_y| - \frac{1}{2} [\mathbf{x} - \boldsymbol{\mu}_y]^\top \Sigma_y^{-1} [\mathbf{x} - \boldsymbol{\mu}_y] + \log \mathbb{P}(Y = y)$$

the decision boundary of this classifier is

$$\{\mathbf{x} \text{ such that } \delta_0(\mathbf{x}) = \delta_1(\mathbf{x})\}$$

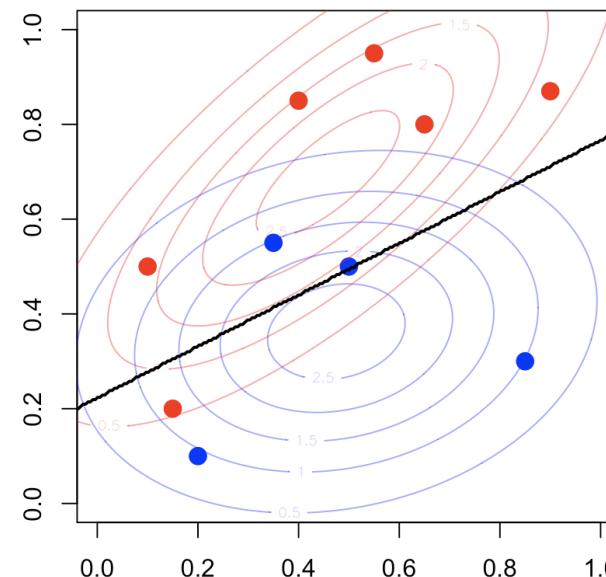
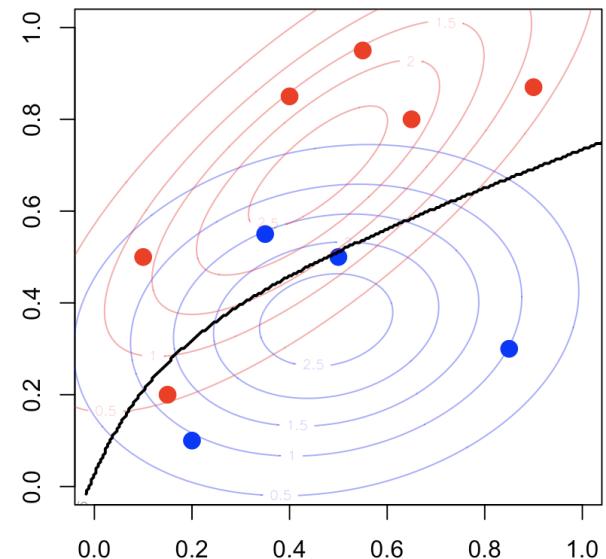
which is quadratic in  $\mathbf{x}$ . This is the quadratic discriminant analysis.

In Fisher (1936, [The use of multile measurements in taxonomic problems](#)), it was assumed that  $\Sigma_0 = \Sigma_1$

In that case, actually,

$$\delta_y(\mathbf{x}) = \mathbf{x}^\top \Sigma^{-1} \boldsymbol{\mu}_y - \frac{1}{2} \boldsymbol{\mu}_y^\top \Sigma^{-1} \boldsymbol{\mu}_y + \log \mathbb{P}(Y = y)$$

and the decision frontier is now linear in  $\mathbf{x}$ .



## Neural Nets & Deep Learning in a Nutshell

Very popular for pictures...

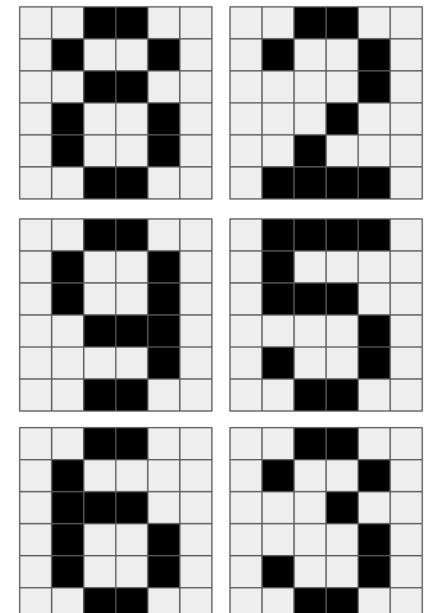
Picture  $x_i$  is

- a  $n \times n$  matrix in  $\{0, 1\}^{n^2}$  for black & white
- a  $n \times n$  matrix in  $[0, 1]^{n^2}$  for grey-scale
- a  $3 \times n \times n$  array in  $([0, 1]^3)^{n^2}$  for color
- a  $T \times 3 \times n \times n$  tensor in  $(([0, 1]^3)^T)^{n^2}$  for video

$y$  here is the label (“8”, “9”, “6”, etc)

Suppose we want to recognize a “6” on a picture

$$m(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{x} \text{ is a “6”} \\ -1 & \text{otherwise} \end{cases}$$



## Neural Nets & Deep Learning in a Nutshell

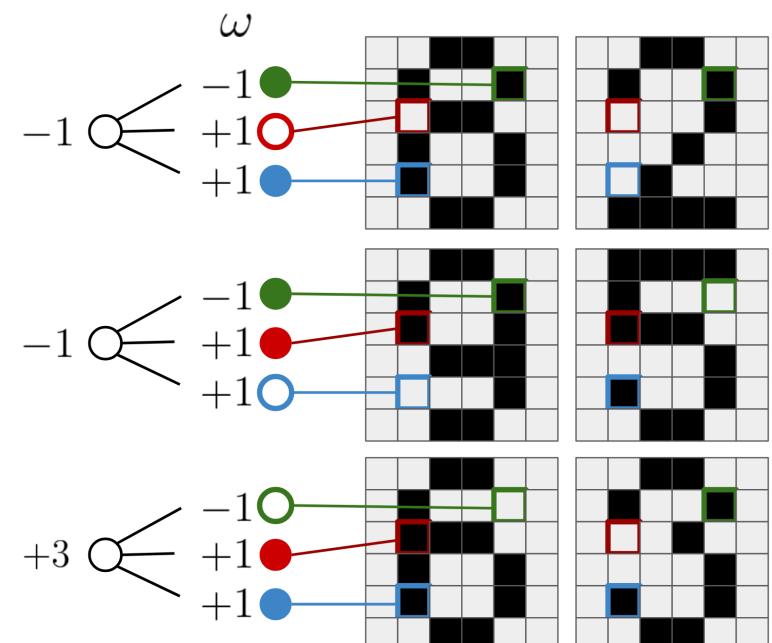
Consider some specifics pixels, and associate weights  $\omega$  such that

$$\hat{m}(\mathbf{x}) = \text{sign} \left( \sum_{i,j} \omega_{i,j} x_{i,j} \right)$$

where

$$x_{i,j} = \begin{cases} +1 & \text{if pixel } x_{i,j} \text{ is black} \quad \blacksquare \\ -1 & \text{if pixel } x_{i,j} \text{ is white} \quad \square \end{cases}$$

for some weights  $\omega_{i,j}$  (that can be negative...)

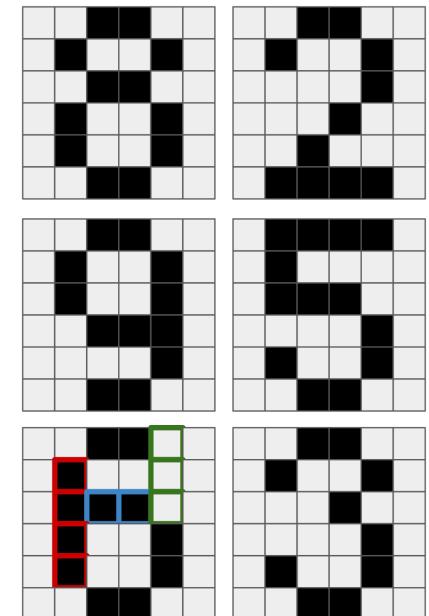
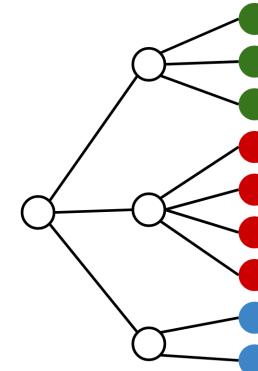


## Neural Nets & Deep Learning in a Nutshell

A deep network is a network with a lot of layers

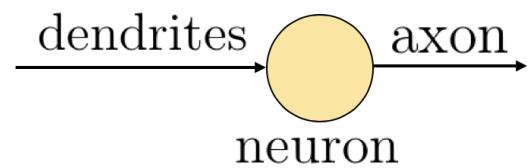
$$\hat{m}(\mathbf{x}) = \text{sign} \left( \sum_i \omega_i \hat{m}_i(\mathbf{x}) \right)$$

where  $\hat{m}_i$ 's are outputs of previous neural nets  
 Those layers can capture shapes in some areas  
 nonlinearities, cross-dependence, etc



## Classification : Neural Nets

A neuron is simply :

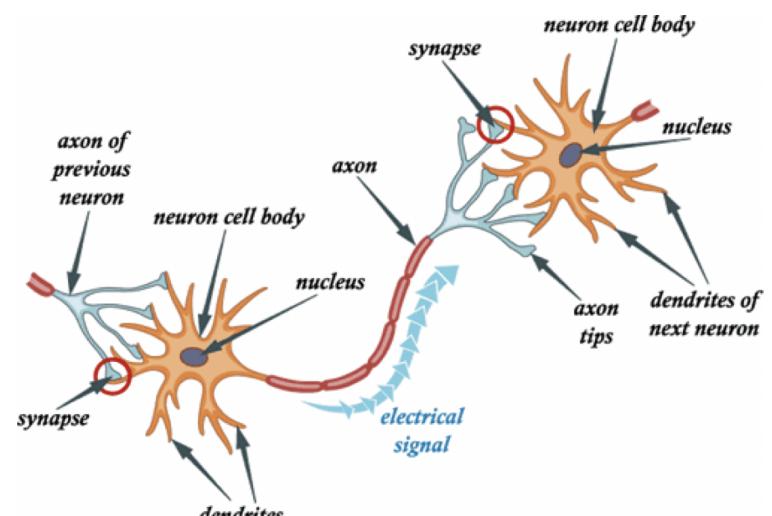


Human body has  $\sim 100$  billion neuron cells

Good at :

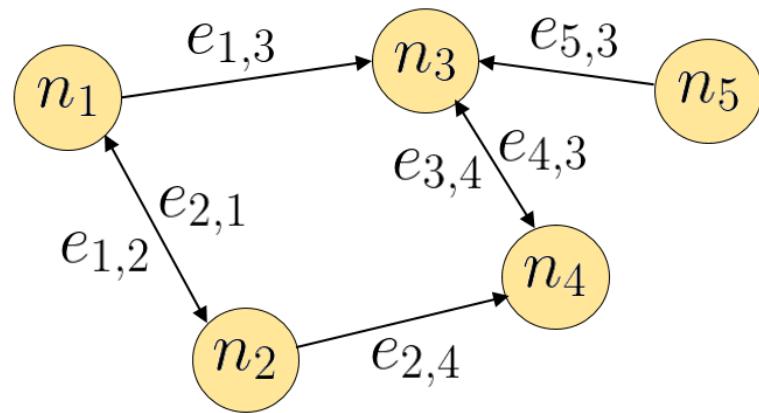
vision, speech, driving, playing games, etc

The most interesting part is not ‘neural’ but ‘network’.



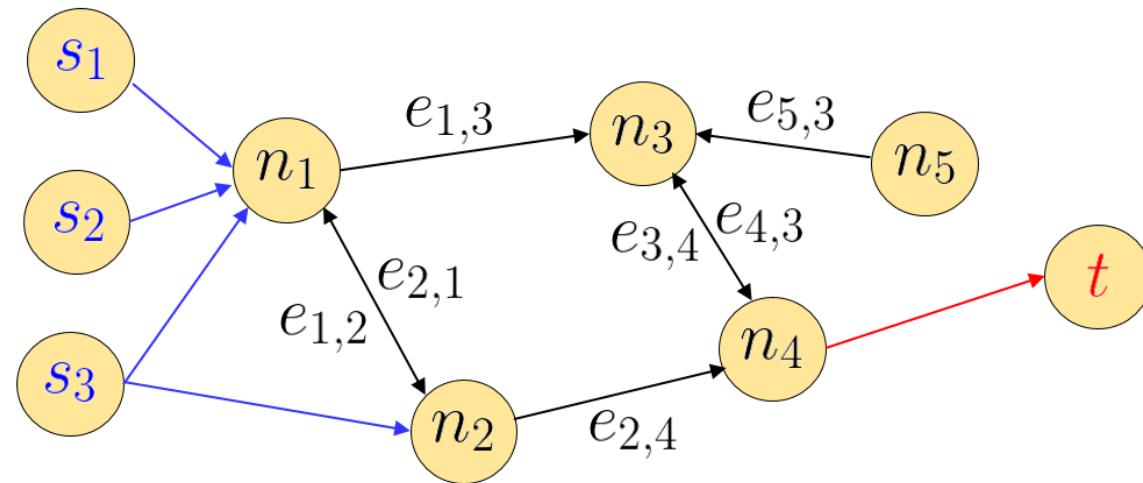
## Classification : Neural Nets

Networks have nodes, and edges (possibly directed) that connect nodes,



or maybe, to more specific some sort of **flow network**,

## Classification : Neural Nets



In such a network, we usually have sources (here multiple) sources (here  $\{s_1, s_2, s_3\}$ ), on the left, on a sink (here  $\{t\}$ ), on the right. To continue with this metaphorical introduction, information from the sources should reach the sink. An usually, sources are explanatory variables,  $\{x_1, \dots, x_p\}$ , and the sink is our variable of interest  $y$ .

The output here is a binary variable  $y \in \{0, 1\}$

## Classification : the Binary Threshold Neuron & the Perceptron

If  $x \in \{0, 1\}^p$ , McCulloch & Pitts (1943, [A logical calculus of the ideas immanent in nervous activity](#)) suggested a simple model, with **threshold**  $b$

$$y_i = f \left( \sum_{j=1}^p x_{j,i} \right) \text{ where } f(x) = \mathbf{1}(x \geq b)$$

where  $\mathbf{1}(x \geq b) = +1$  if  $x \geq b$  and 0 otherwise, or (equivalently)

$$y_i = f \left( \omega + \sum_{j=1}^p x_{j,i} \right) \text{ where } f(x) = \mathbf{1}(x \geq 0)$$

with **weight**  $\omega = -b$ . The trick of adding 1 as an input was very important !

$\omega = -1$  is the **or** logical operator :  $y_i = 1$  if  $\exists j$  such that  $x_{j,i} = 1$

$\omega = -p$ , is the **and** logical operator :  $y_i = 1$  if  $\forall j$ ,  $x_{j,i} = 1$

## Classification : the Binary Threshold Neuron & the Perceptron

but not possible for the **xor** logical operator :  $y_i = 1$  if  $x_{1,i} \neq x_{2,i}$

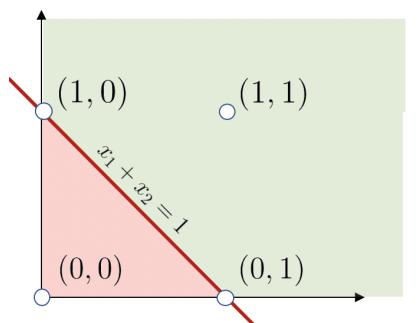
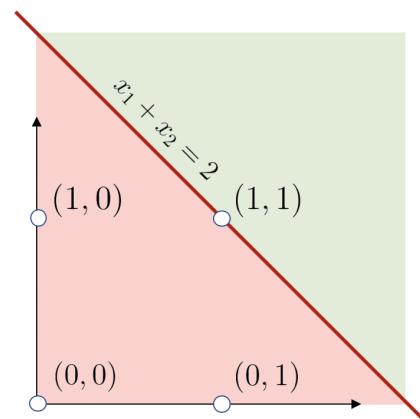
Rosenblatt (1961, **Principles of Neurodynamics: Perceptron & Theory of Brain Mechanism**) considered the extension where  $x$ 's are real-valued, with **weight**  $\omega \in \mathbb{R}^p$

$$y_i = f \left( \sum_{j=1}^p \omega_j x_{j,i} \right) \text{ where } f(x) = \mathbf{1}(x \geq b)$$

where  $\mathbf{1}(x \geq b) = +1$  if  $x \geq b$  and 0 otherwise, or (equivalently)

$$y_i = f \left( \omega_0 + \sum_{j=1}^p \omega_j x_{j,i} \right) \text{ where } f(x) = \mathbf{1}(x \geq 0)$$

with **weights**  $\omega \in \mathbb{R}^{p+1}$



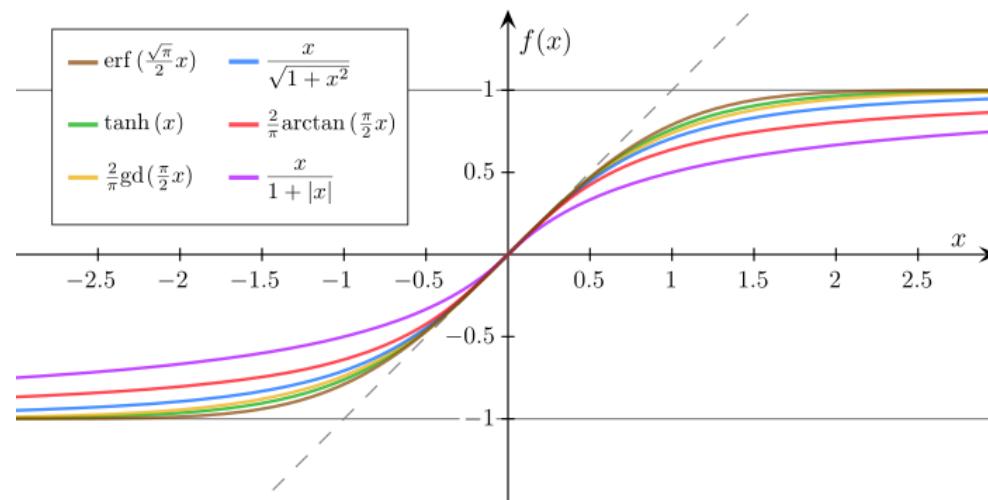
## Classification : the Binary Threshold Neuron & the Perceptron

Minsky & Papert (1969, [Perceptrons: an Introduction to Computational Geometry](#)) proved that perceptron were a linear separator, not very powerful

Define the **sigmoid** function  $f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$  (= **logistic** function).

This function  $f$  is called the **activation function**.

If  $y \in \{-1, +1\}$ , one can consider the hyperbolic tangent  $f(x)) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$  or the inverse tangent function (see [wikipedia](#)).



## Classification : Neural Nets

So here, for a classification problem,

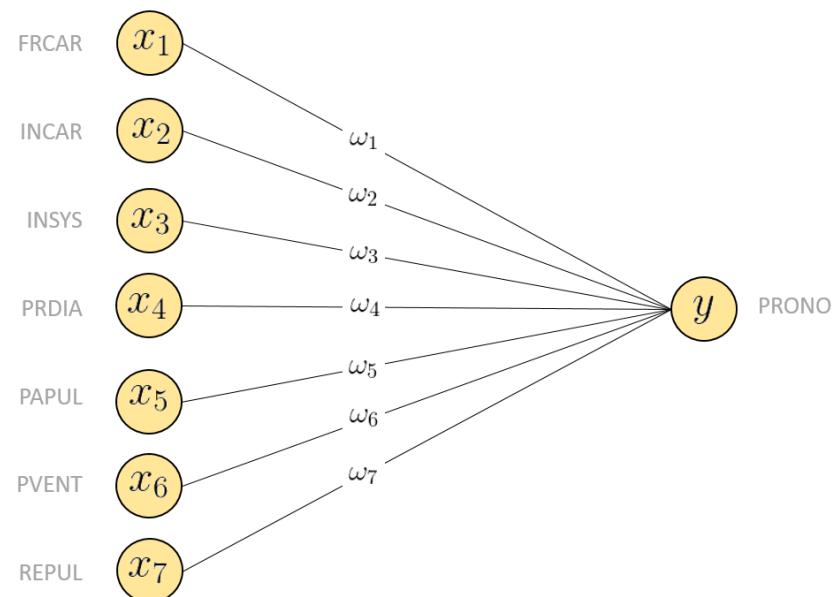
$$y_i = f \left( \omega_0 + \sum_{j=1}^p \omega_j x_{j,i} \right) = p(\mathbf{x}_i)$$

The error of that model can be computed used the quadratic loss function,

$$\sum_{i=1}^n (y_i - p(\mathbf{x}_i))^2$$

or cross-entropy

$$\sum_{i=1}^n (y_i \log p(\mathbf{x}_i) + [1 - y_i] \log[1 - p(\mathbf{x}_i)])$$



## Classification : Neural Nets

If

$$p(\boldsymbol{x}) = f \left( \omega_0 + \sum_{h=1}^3 \omega_h f_h (\omega_{h,0} + \boldsymbol{x}^\top \boldsymbol{\omega}_h) \right)$$

we want to solve, with back propagation,

$$\boldsymbol{\omega}^* = \operatorname{argmin} \left\{ \frac{1}{n} \sum_{i=1}^n \ell(y_i, p(\boldsymbol{x}_i)) \right\}$$

To visualize a neural network, use

```
1 library(devtools)
2 source_url('https://freakonometrics.free.fr/nnet_plot.R')
```

## Classification : Neural Nets

```

1 library(nnet)
2 mix = function(z) (z-min(z))/(max(z)-min(z))
3 myocarde = apply(myocarde, 1, mix)
4 fit_nn = nnet(PRONO~., data=myocarde, size=3)
5 plot.nnet(fit_nn)

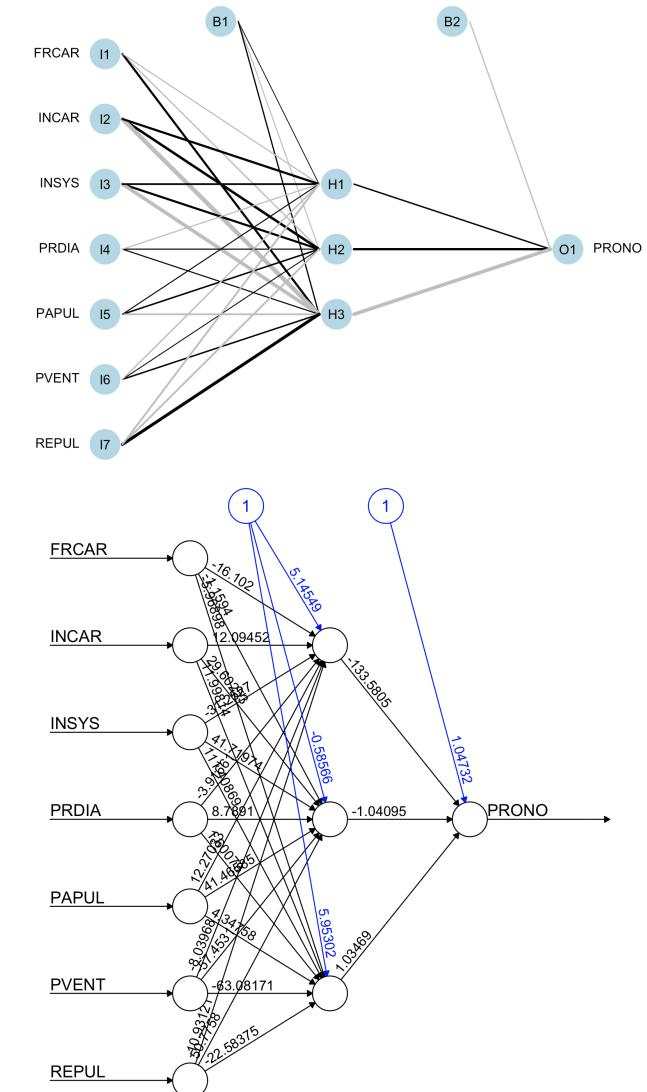
```

or

```

1 library(neuralnet)
2 fit_nn = neuralnet(formula(glm(PRONO~., data =
            myocarde)), myocarde, hidden = 3, act.fct
            = sigmoid)
3 plot(fit_nn)

```



## Classification : Neural Nets

Of course, one can consider a multiple layer network,

$$p(\boldsymbol{x}) = f \left( \omega_0 + \sum_{h=1}^3 \omega_h f_h (\omega_{h,0} + \boldsymbol{z}_h^\top \boldsymbol{\omega}_h) \right)$$

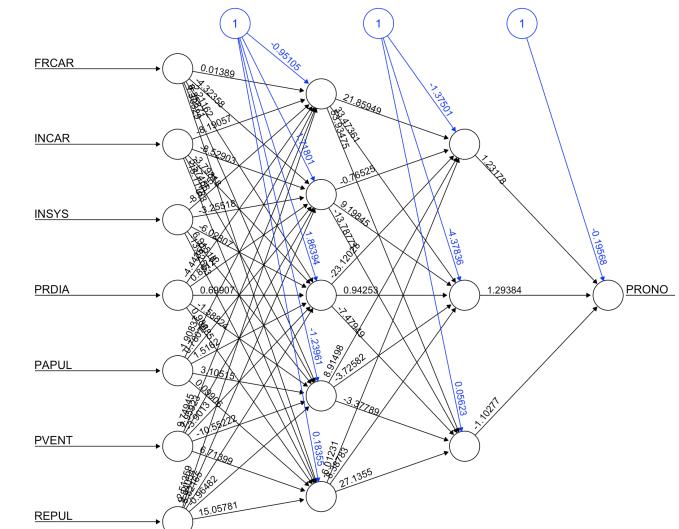
where

$$\boldsymbol{z}_h = f \left( \omega_{h,0} + \sum_{j=1}^{k_h} \omega_{h,j} f_{h,j} (\omega_{h,j,0} + \boldsymbol{x}^\top \boldsymbol{\omega}_{h,j}) \right)$$

```

1 library(neuralnet)
2 model_nnet = neuralnet(formula(glm(PRONO~.,
  data = myocarde)), myocarde, hidden = 3,
  act.fct = sigmoid)
3 plot(model_nnet)

```



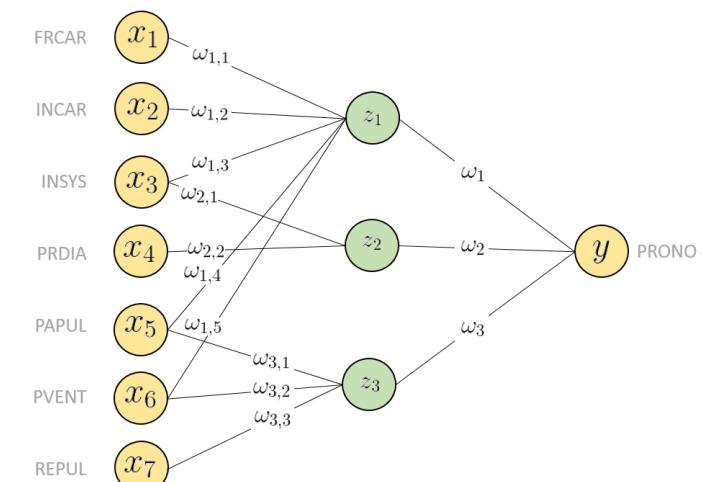
## Classification : Neural Nets

Consider a single hidden layer, with 3 different neurons, so that

$$p(\mathbf{x}) = f \left( \omega_0 + \sum_{h=1}^3 \omega_h f_h \left( \omega_{h,0} + \sum_{j=1}^p \omega_{h,j} x_j \right) \right)$$

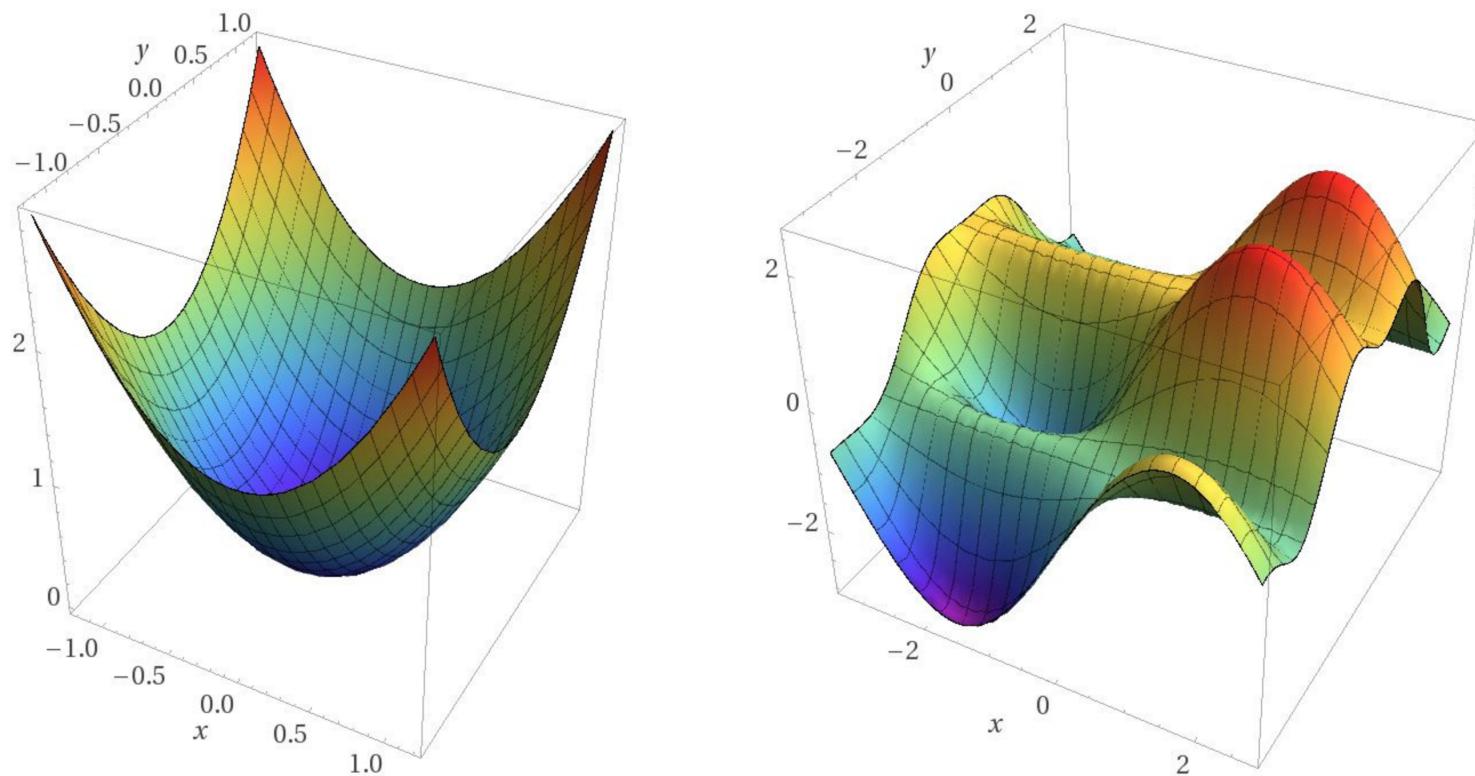
or

$$p(\mathbf{x}) = f \left( \omega_0 + \sum_{h=1}^3 \omega_h f_h (\omega_{h,0} + \mathbf{x}^\top \boldsymbol{\omega}_h) \right)$$



## Optimization Issues

If classical econometrics are based on convex optimization problems (most the time), (deep) Neural Networks are usually not convex



## Optimization Issues

Consider optimization problem

$$\min_{\boldsymbol{x} \in \mathcal{X}} \{f(\boldsymbol{x})\}$$

solved by gradient descent,

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \gamma_n \nabla f(\boldsymbol{x}_n), \quad n \geq 0$$

with starting point  $\boldsymbol{x}_0$

$\gamma_n$  can be call **learning rate**

Sometime the dimension of the dataset can be really big: we can't pass all the data to the computer at once to compute  $f(\boldsymbol{x})$

we need to divide the data into smaller sizes and give it to our computer one by one

## Optimization Issues

### Batches

The Batch size is a hyperparameter that defines the number of sub-samples we use to compute quantities (e.g. the gradient).

### Epoch

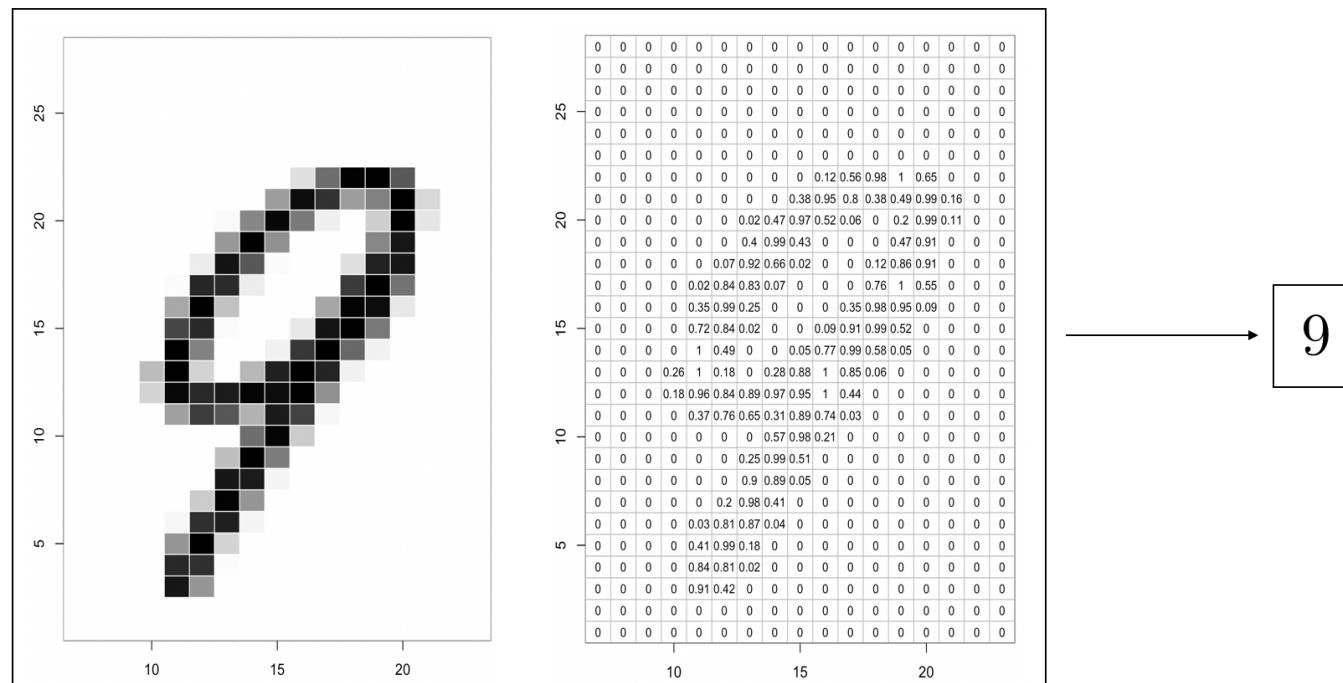
One Epoch is when the dataset is passed forward and backward through the neural network (only once).

We can divide the dataset of 2,000 examples into batches of 400 then it will take 5 iterations to complete 1 epoch.

The number of epochs is the hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

Classification : Image Recognition with Neural Networks

## Classical $\{0, 1, \dots, 9\}$ handwritten digit recognition



$$x_i \in \mathcal{M}_{28,28}$$

$$y_i \in \{0, 1, \dots, 9\}$$

see tensorflow (Google) and keras.

## Classification : Image Recognition with Logistic Regression?

See <https://www.tensorflow.org/install>

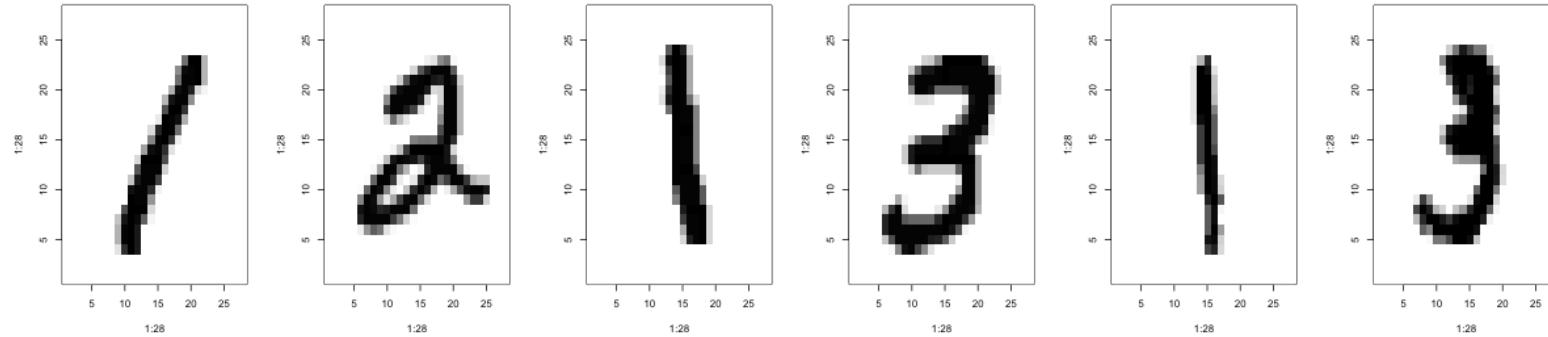
```

1 library(tensorflow)
2 install_tensorflow(method = "conda")
3 library(keras)
4 install_keras(method = "conda")

1 mnist <- dataset_mnist()
2 idx123 = which(mnist$train$y %in% c(1,2,3))
3 V <- mnist$train$x[idx123[1:800], ,]
4 MV = NULL
5 for(i in 1:800) MV=cbind(MV, as.vector(V[i,,]))
6 MV=t(MV)
7 df=data.frame(y=mnist$train$y[idx123][1:800], x=MV)
8 reg=glm((y==1)^., data=df, family=binomial)
```

Here  $x_i \in \mathbb{R}^{784}$  (for a small greyscale picture).

## Classification : Image Recognition and Convolutional Neural Network



Classical strategy : reduce dimension via layers  $z$  (e.g. PCA, principal component analysis)

and then consider a classifier  $z \mapsto y$

### CNN : Convolutional Neural Network

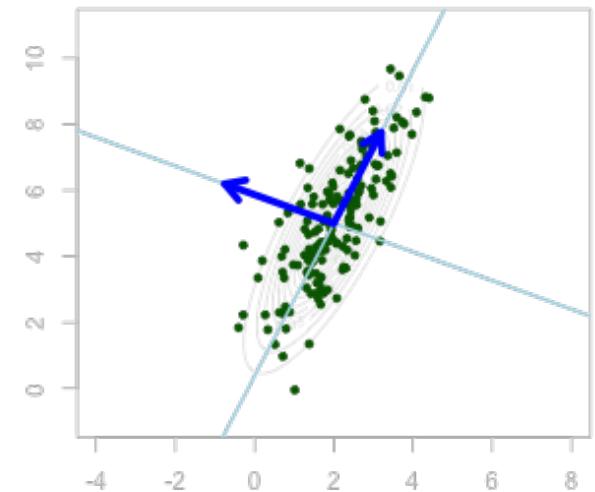
A convolutional neural network is a fully connected networks : each neuron in one layer is connected to all neurons in the next layer.

## Non-supervised : PCA

Projection method, used to reduce dimensionality

### PCA : Principal Component Analysis

find a small number of directions in input space that explain variation in input data represent data by projecting along those directions



Consider (joint)  $n$  observations of  $d$  variables, stored in matrix  $\mathbf{X} \in \mathcal{M}_{n \times d}$

We want to project (multiply by a projection matrix) into (much) lower dimensional space  $k < d$

Search for orthogonal directions in space with highest variance

## Non-supervised : PCA

Information is stored in the covariance matrix  $\Sigma = \mathbf{X}^\top \mathbf{X}$  (variables are centered)

The  $k$  largest eigenvectors of  $\Sigma$  are the **principal components**

Assemble these eigenvectors into a  $d \times k$  matrix  $\tilde{\Sigma}$

Express  $d$ -dimensional vectors  $\mathbf{x}$  by projecting them to  $m$ -dimensional  $\tilde{\mathbf{x}}$ ,  
 $\tilde{\mathbf{x}} = \tilde{\Sigma}^\top \mathbf{x}$

- Heuristics for the *first* principal component

We have data  $\mathbf{X} \in \mathcal{M}_{n \times d}$ , i.e.  $n$  vectors  $\mathbf{x} \in \mathbb{R}^d$ ,

Let  $\omega_1 \in \mathbb{R}^d$  denote weights.

We want to maximize the variance of  $z_1 = \omega_1^\top \mathbf{x}$

$$\max \left\{ \frac{1}{n} \sum_{i=1}^n (\omega_1^\top \mathbf{x}_i - \omega_1^\top \bar{\mathbf{x}})^2 \right\} = \max \{ \omega_1^\top \Sigma \omega_1 \}$$

subject to  $\|\omega_1\| = 1$ .

## Non-supervised : PCA

We can use Lagrange multiplier to solve this constraint optimization problem

$$\max_{\omega_1, \lambda_1} \{ \omega_1^\top \Sigma \omega_1 + \lambda(1 - \omega_1^\top \omega_1) \}$$

If we differentiate

$$\Sigma \omega_1 - \lambda_1 \omega_1 = \mathbf{0}, \text{ i.e. } \Sigma \omega_1 = \lambda_1 \omega_1$$

i.e.  $\omega_1$  is an eigenvector of  $\Sigma$ , with eigenvalue the Lagrange multiplier. It should be the one with the largest eigenvalue.

A nonlinear version can be obtained using reproducing kernel Hilbert spaces, and kernels, see Schölkopf *al.* (1996, [Nonlinear Component Analysis as a Kernel Eigenvalue Problem](#)).

For the second one, we should solve a similar problem, with  $\omega_2 \perp \omega_1$ , i.e.  
 $\omega_2^\top \omega_1 = 0$

## Non-supervised : PCA

- Heuristics for the *second* principal component

$$\begin{aligned} & \max \{ \boldsymbol{\omega}_2^\top \boldsymbol{\Sigma} \boldsymbol{\omega}_2 \} \\ & \text{subject to } \|\boldsymbol{\omega}_2\| = 1 \\ & \quad \boldsymbol{\omega}_2^\top \boldsymbol{\omega}_1 = 0 \end{aligned}$$

The Lagragian is

$$\boldsymbol{\omega}_2^\top \boldsymbol{\Sigma} \boldsymbol{\omega}_2 + \lambda_2(1 - \boldsymbol{\omega}_2^\top \boldsymbol{\omega}_2) - \lambda_1 \boldsymbol{\omega}_2^\top \boldsymbol{\omega}_1$$

when differentiation with respect to  $\boldsymbol{\omega}_2$ , we get  $\boldsymbol{\Sigma} \boldsymbol{\omega}_2 = \lambda_2 \boldsymbol{\omega}_2$

hence,  $\lambda_2$  is the second largest eigenvalue, etc.

## Non-supervised : Principal Components on Handwritten Pictures

Consider some handwritten pictures, from the `mnist` dataset. Here  $\{(y_i, \mathbf{x}_i)\}$  with  $y_i = "3"$  and  $\mathbf{x}_i \in [0, 1]^{28 \times 28}$

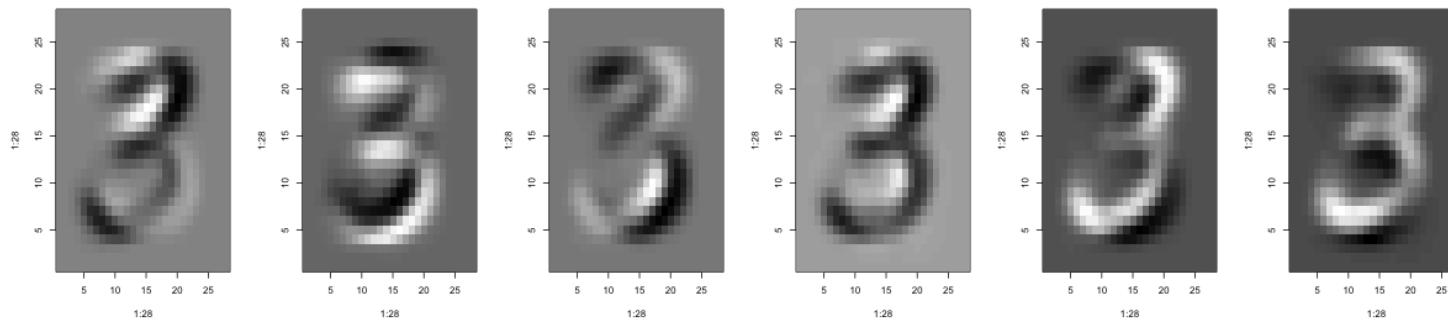
A *variable* is a pixel here...

Matrix  $\mathbf{X}$  is a  $n \times 784$  matrix, where  $n$  is the number of pictures in the dataset.

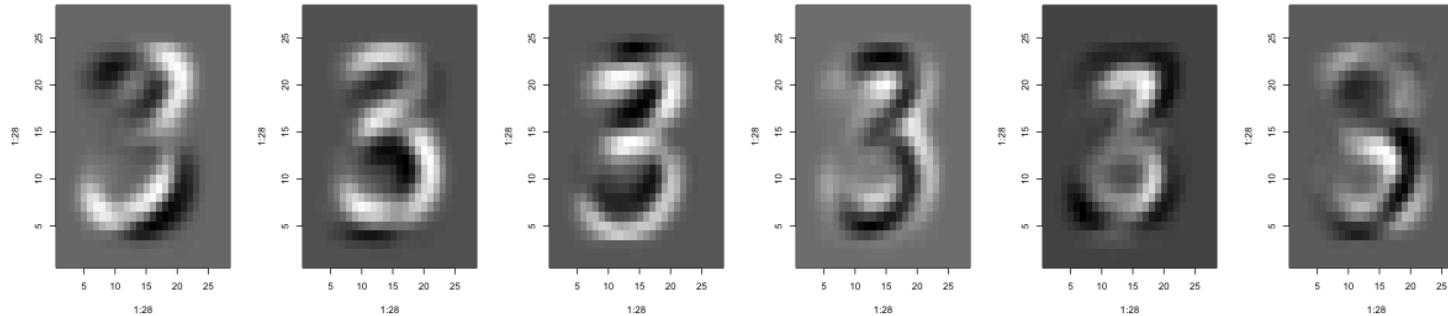
## Non-supervised : PCA on Handwritten Pictures

Consider the PCA of matrix  $\mathbf{X}$ , then  $\omega_j \in \mathbb{R}^{784}$  can be interpreted as a picture (after rescaling).

We can also consider the approximation of  $x_i$  using only the first  $k$  principal components (here  $k = 3$ )



## Non-supervised : PCA on Tensor Data



1D Tensor (vector)

$$\mathbf{A} = [A_i] \in \mathbb{R}^d - i = \text{individual}$$

2D Tensor (matrix)

$$\mathbf{A} = [A_{i,j}] \in \mathcal{M}_{d_1, d_2}, j = \text{feature}$$

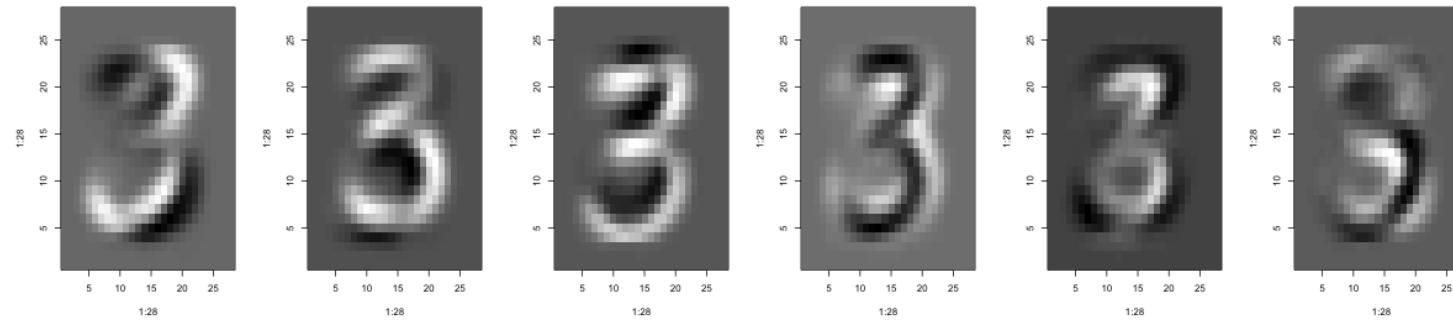
3D Tensor (array)

$$\begin{aligned}\mathbf{A} &= [A_{i,j,k}] = \mathbf{A}_i \text{'s} \\ \mathbf{A}_i &\in \mathcal{M}_{d_1, d_2} \\ \text{e.g. (black \& white) picture}\end{aligned}$$

4D Tensor (arrays)

$$\begin{aligned}\mathbf{A} &= [A_{i,j,k,l}] \in \mathcal{M}_{d_1, d_2} \\ \text{e.g. color picture,} \\ l &\in \{\text{red, green, blue}\}\end{aligned}$$

## Non-supervised : PCA on Tensor Data

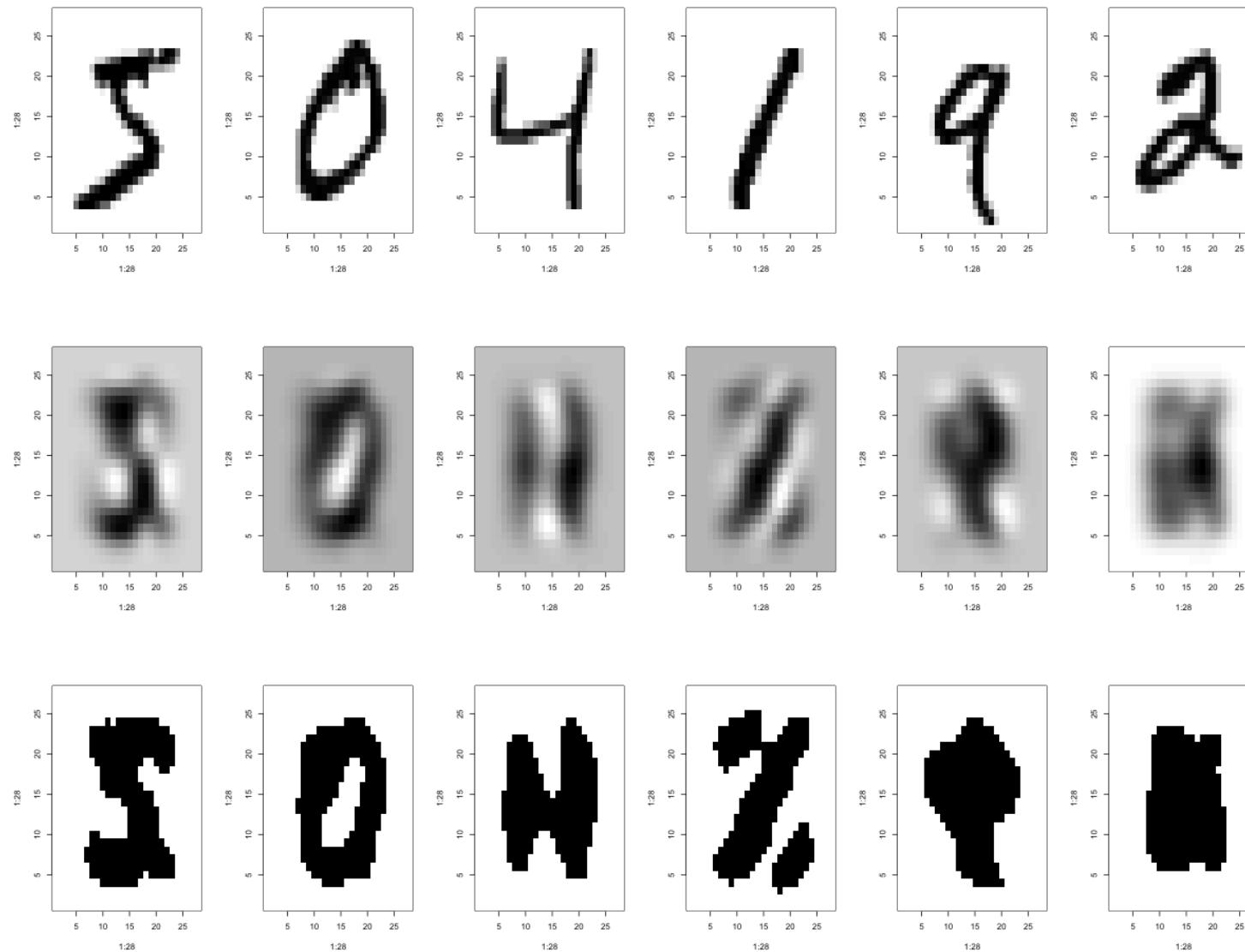


One can define a **multilinear principal component analysis**

see also Tucker decomposition, from Hitchcock (1927, **The expression of a tensor or a polyadic as a sum of products**)

We can use those decompositions to derive a simple classifier.

## Non-supervised : Principal Components on Handwritten Pictures



## Non-supervised : Classification of Handwritten Pictures

Consider a dataset with only three labels,

$y_i \in \{\text{"1"}, \text{"2"}, \text{"3"}\}$  and the  $k$  principal components obtained by PCA,

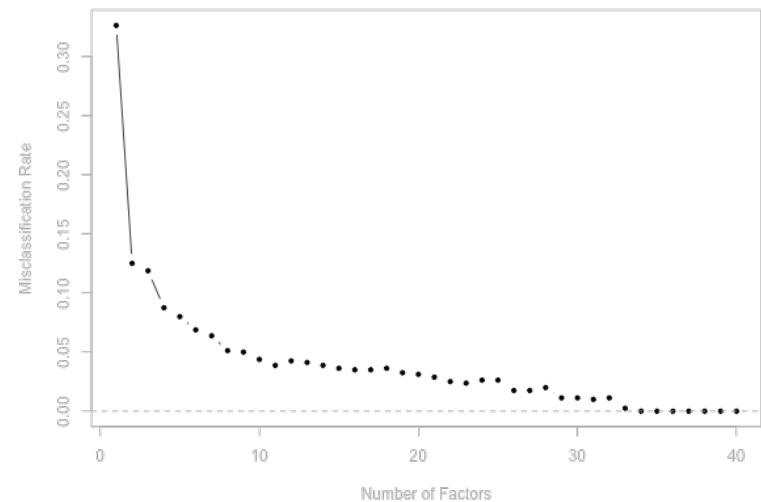
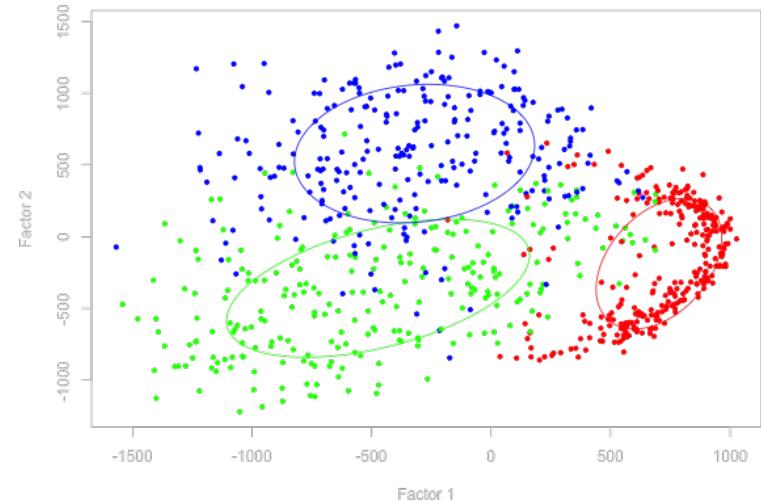
$$\{x_1, \dots, x_{784}\} \rightarrow \{\tilde{x}_1, \dots, \tilde{x}_k\}$$

E.g. scatterplot  $\{\tilde{x}_{1,i}, \tilde{x}_{2,i}\}$

with ● when  $y_i = \text{"1"}$ , ●  $y_i = \text{"2"}$  and ●  $y_i = \text{"3"}$ ,

Let  $m_k$  denote the multinomial logistic regression on  $\tilde{x}_1, \dots, \tilde{x}_k$  and visualize the misclassification rate

See Nielsen (2018, **Neural Networks and Deep Learning**)



## Non-supervised : Autoencoders

PCA is closely related to a neural network

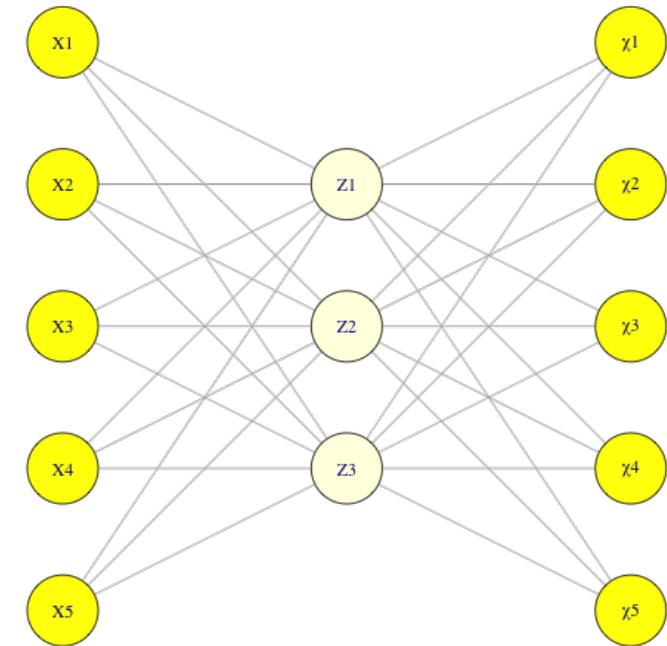
### Autoencoder

An autoencoder is a neural network whose outputs are its own inputs

Here  $\mathbf{z} = f(W\mathbf{x})$  and  $\mathbf{\chi} = g(M\mathbf{z})$ .

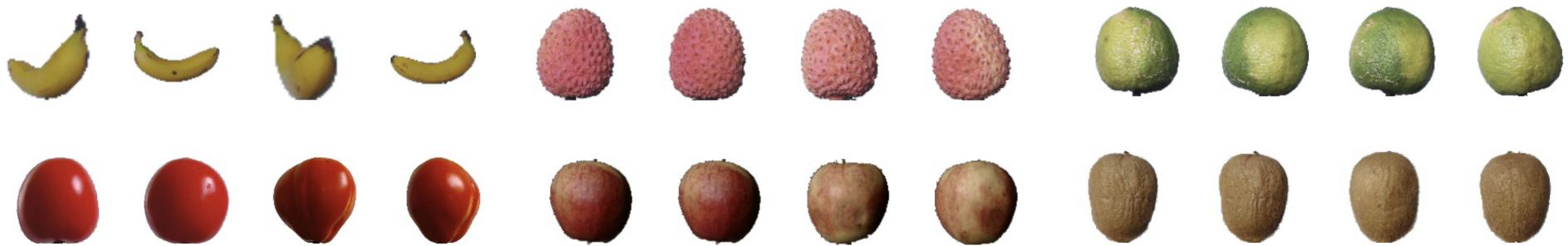
Given  $f$  and  $g$  solve

$$\min_{W,M} \left\{ \sum_{i=1}^n \ell(x_i, \chi_i) \right\} = \min_{W,M} \left\{ \sum_{i=1}^n \ell(x_i, MWx_i) \right\} \text{ if } f \text{ and } g \text{ are linear}$$

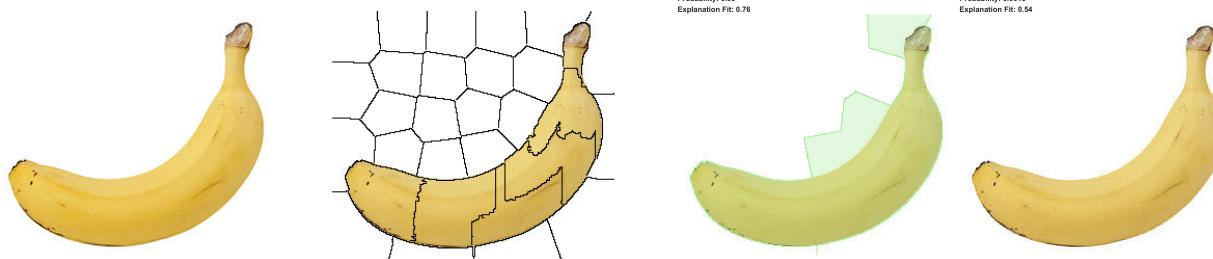


## Neural Networks for More Complex Pictures

The algorithm is trained on tagged picture, i.e.  $(\mathbf{x}_i, y_i)$  where here  $y_i$  is some class of fruits (banana, apple, kiwi, tomatoe, lime, cherry, pear, lychee, papaya, etc)



The challenge is to provide a new picture ( $\mathbf{x}_{n+1}$  and to see the prediction  $\hat{y}_{n+1}$ )

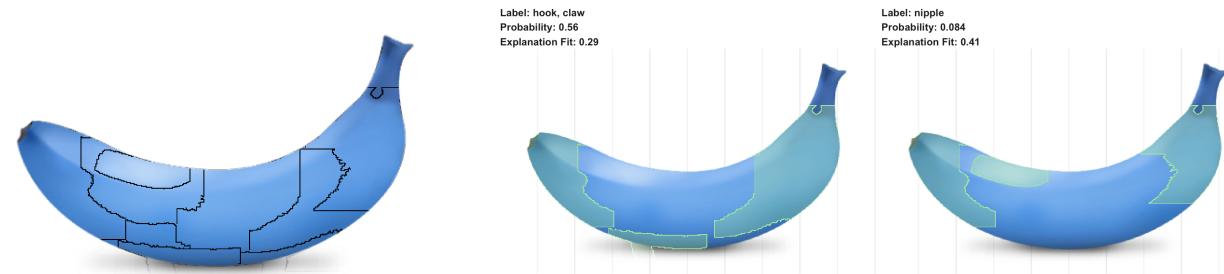


## Neural Networks for More Complex Pictures

Note that ( $x_{n+1}$  can be something that was not in the training dataset (here yellow zucchini)



or some invented one (here a blue banana)



see also pyimagesearch for food/no food classifier

## Neural Networks for Text and Go (Alpha Go)

Neural nets are also popular for text classification...

**Neural network architecture.** The input to the policy network is a  $19 \times 19 \times 48$  image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a  $23 \times 23$  image, then convolves  $k$  filters of kernel size  $5 \times 5$  with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a  $21 \times 21$  image, then convolves  $k$  filters of kernel size  $3 \times 3$  with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size  $1 \times 1$  with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used  $k = 192$  filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with  $k = 128, 256$  and  $384$  filters.

## RNN : Recurrent Neural Networks

Introduced in Williams, Hinton & Rumelhart (1986, [Learning representations by back-propagating errors](#))

### RNN : Recurrent Neural Networks

NN where connections between nodes form a directed graph along a temporal sequence

Allows it to exhibit temporal dynamic behavior.

They will be discussed in the part on time series and dynamic modeling.

## Neural Networks for Pictures Classification

See <http://junma5.weebly.com/> for more advanced models

We should preprocess the data by removing near zero variance columns (and scaling by max)

```

1 train = xxxxxxxx
2 nzv <- nearZeroVar(train)
3 nzv.nolabel <- nzv-1

```

The data is also splitted into two for cross validation

```

1 inTrain <- createDataPartition(y=train$label, p=0.7, list=FALSE)
2 training <- train[inTrain, ]
3 CV <- train[-inTrain, ]
4
5 X <- as.matrix(training[, -1]) # data matrix (each row = single
   example)
6 N <- nrow(X) # number of examples
7 y <- training[, 1] # class labels

```

```

8
9 K <- length(unique(y)) # number of classes
10 X.proc <- X[, -nzv.nolabel]/max(X) # scale
11 D <- ncol(X.proc) # dimensionality
12
13 Xcv <- as.matrix(CV[, -1]) # data matrix (each row = single example)
14 ycv <- CV[, 1] # class labels
15 Xcv.proc <- Xcv[, -nzv.nolabel]/max(X) # scale CV data
16
17 Y <- matrix(0, N, K)
18
19 for (i in 1:N){
20   Y[i, y[i]+1] <- 1
21 }

```

We can train the model

```

1 nnet.mnist <- nnet(X.proc, Y, step_size = 0.3,
2                      reg = 0.0001, niteration = 3500)

```

```

3 ## [1] "iteration 0 : loss 2.30265553844748"
4 ## [1] "iteration 1000 : loss 0.303718250939774"
5 ## [1] "iteration 2000 : loss 0.271780096710725"
6 ## [1] "iteration 3000 : loss 0.252415244824614"
7 ## [1] "iteration 3500 : loss 0.250350279456443"
8 predicted_class <- nnetPred(X.proc, nnet.mnist)
9 print(paste('training set accuracy: ',
10             mean(predicted_class == (y+1))))
11 ## [1] "training set accuracy: 0.93089140563888"
12 predicted_class <- nnetPred(Xcv.proc, nnet.mnist)
13 print(paste('CV accuracy: ',
14             mean(predicted_class == (ycv+1))))
15 ## [1] "CV accuracy: 0.912360085734699"

```

test  $\leftarrow Xcv[sample(1:nrow(Xcv), 1), ]$  Xtest.proc  $\leftarrow$  as.matrix(Xtest[-nzv.nolabel],  
nrow = 1) predicted<sub>test</sub>  $<-$  nnetPred(t(Xtest.proc), nnet.mnist) print(paste('The predicted digit is :', predicted<sub>test</sub> - 1)) [1]"The predicted digit is : 3" displayDigit(Xtest)