Neural Networks

Tadao Hoshino （星野匡郎）

Econometrics II: ver. 2019 Spring Semester

Introduction

Introduction

- Neural Networks is a machine learning method that is inspired by the human brains (*neuron* and *synapse*).

- In the language of statistics, a neural network is just a type of additive single index regression model with a hierarchical structure (layers):

$$Y = f\left(\cdots g\left(\sum_{m=1}^{M} h_m(\mathbf{X}^\top \alpha_m)\beta_m\right)\cdots\right) + error$$

Introduction

- As the number of layers increases (= "deep" neural networks), the number of parameters needed to be learned will increase rapidly; computationally extremely burdensome and thought to be intractable in practice, until recently.
- Deep Learning: set of techniques for learning deep neural networks.
- The advance of computational power and the development of deep learning made the use of deep neural networks practically possible.
- Currently, neural networks and deep learning provide the highest performances in many applications, such as image recognition and language processing.

Neural Networks

Neural Networks

- A single-layer neural network can be viewed as a two-stage regression model.
- Response variable: Y (either continuous or binary).
- Input variables: $\mathbf{X} = (X_1, \ldots, X_p)^\top$.
- Predicted value of Y given $\mathbf{X}$: $\widetilde{Y}(\mathbf{X}; \theta)$, where

$$\begin{aligned} \widetilde{Y}(\mathbf{X}; \theta) &= g(T) \\ T &= \beta_0 + \sum_{m=1}^{M} Z_m \beta_m \\ Z_m &= \sigma\left(\alpha_{m,0} + \sum_{j=1}^{p} X_j \alpha_{m,j}\right) \end{aligned} \tag{1}$$

$\theta = (\alpha, \beta)$, $\alpha = (\alpha_{1,0}, \alpha_{1,1}, \ldots, \alpha_{M,p})$ and $\beta = (\beta_0, \beta_1, \ldots, \beta_M)$.

Neural Networks

- (1) can be written in a single nonlinear regression model:

$$Y = \underbrace{g\left(\beta_0 + \sum_{m=1}^{M} \sigma\left(\alpha_{m,0} + \sum_{j=1}^{p} X_j \alpha_{m,j}\right) \beta_m\right)}_{=\widetilde{Y}(\mathbf{X};\theta)} + error.$$

- The function $\sigma(\cdot)$ is called the activation function. The choice of activation function significantly influences the performance of neural networks. (This issue will be discussed later.)
- The output function $g(\cdot)$ allows a final transformation of the intermediate output T. For example,
  - Regression: $g(T) = T$.
  - Binary classification: $g(T) = \exp(T)/[1 + \exp(T)]$.

Neural Networks

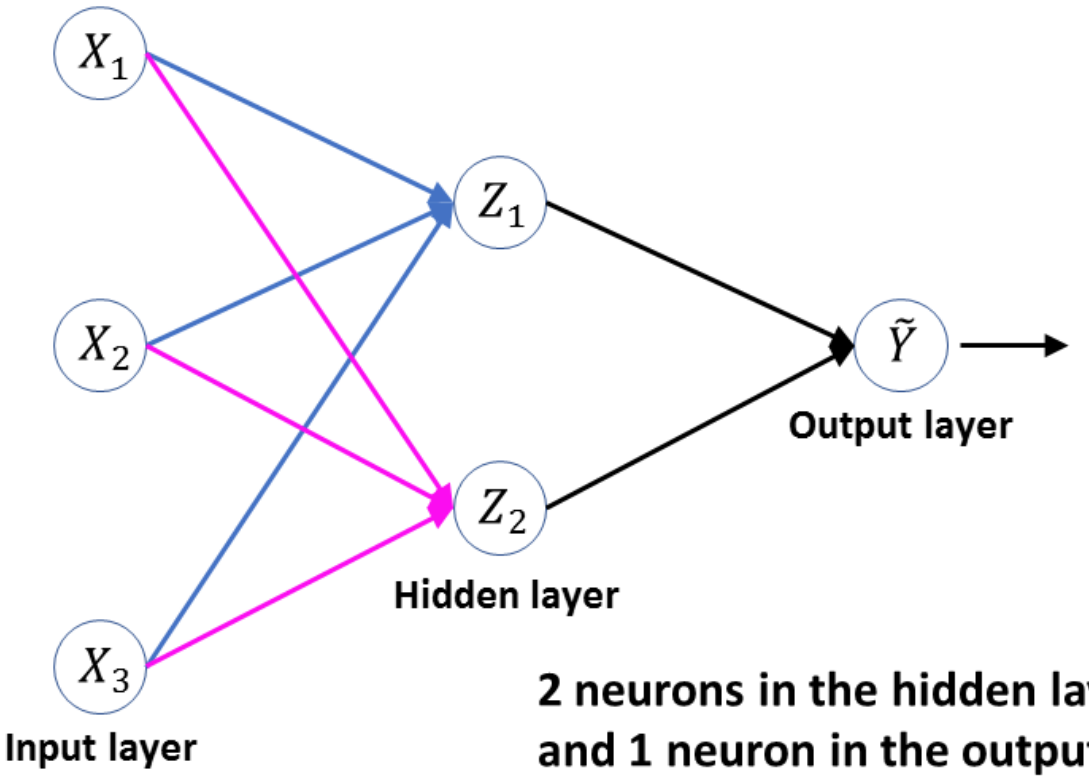
- The units in the middle of the network,

$$\{Z_m : 1 \leq m \leq M\}, \quad Z_m = \sigma\left(\alpha_{m,0} + \sum_{j=1}^{p} X_j \alpha_{m,j}\right)$$

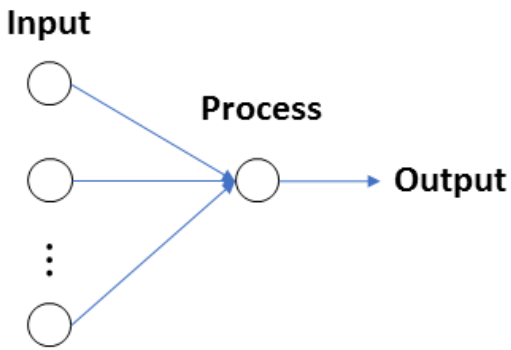
  are called hidden units because the values of Z_m's are not directly observed.
- We can think of Z_m as a "generated" regressor that is constructed from the weighted average (linear combination) of the original input $\mathbf{X}$.
- The above single layer neural network (1) can be represented by the following *network diagram*.

Neural Networks

3 inputs, single hidden layer with M = 2, scalar-valued response variable.



2 neurons in the hidden layer, and 1 neuron in the output.



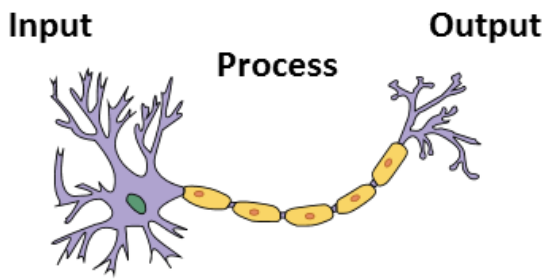
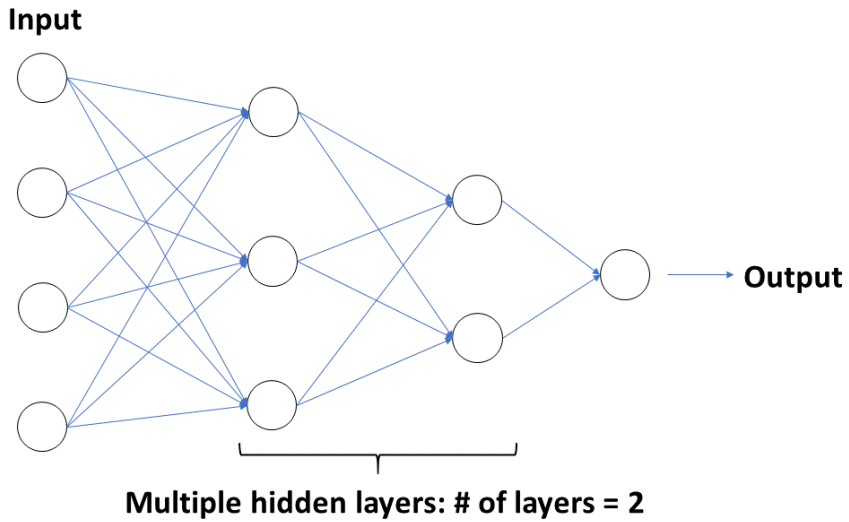


Diagram of a neuron
(from *Wikimedia Commons*)

Neural Networks

- In general, there can be more than one hidden layer: multi-layer neural networks (deep neural networks).



- As the number of hidden layers increases, deeper neural networks are formed, and the capacity of neural networks to learn increasingly more complex information increases.
  - Deep learning is a set of machine learning techniques that can be used to optimize such deep neural networks.

Activation Functions

Activation Functions

- The purpose of the activation function $\sigma(\cdot)$ is to introduce a "nonlinearity" into the neural network.
- If $\sigma(\cdot)$ and $g(\cdot)$ are linear, the neural network collapses to a linear regression model; in the special case when both functions are identity,

$$
\begin{aligned}
Y &= g\left(\beta_0 + \sum_{m=1}^{M} \sigma\left(\alpha_{m,0} + \sum_{j=1}^{p} X_j \alpha_{m,j}\right)\beta_m\right) + \ error \\
&= \beta_0 + \sum_{m=1}^{M}\left(\alpha_{m,0} + \sum_{j=1}^{p} X_j \alpha_{m,j}\right)\beta_m + \ error \\
&= c_0 + \sum_{j=1}^{p} X_j c_j + \ error
\end{aligned}
$$

where $c_0 = \beta_0 + \sum_{m=1}^{M} \alpha_{m,0}\beta_m$, and $c_j = \sum_{m=1}^{M} \alpha_{m,j}\beta_m$.

Activation Functions

Commonly used activation functions are: letting $\mathbf{X}_m^* = \sum_{j=1}^{p} X_j \alpha_{m,j}$,

- Logistic function:

$$\sigma\left(\alpha_{m,0} + \mathbf{X}_m^*\right) = \frac{\exp(\alpha_{m,0} + \mathbf{X}_m^*)}{1 + \exp(\alpha_{m,0} + \mathbf{X}_m^*)}.$$

- Hyperbolic tangent:

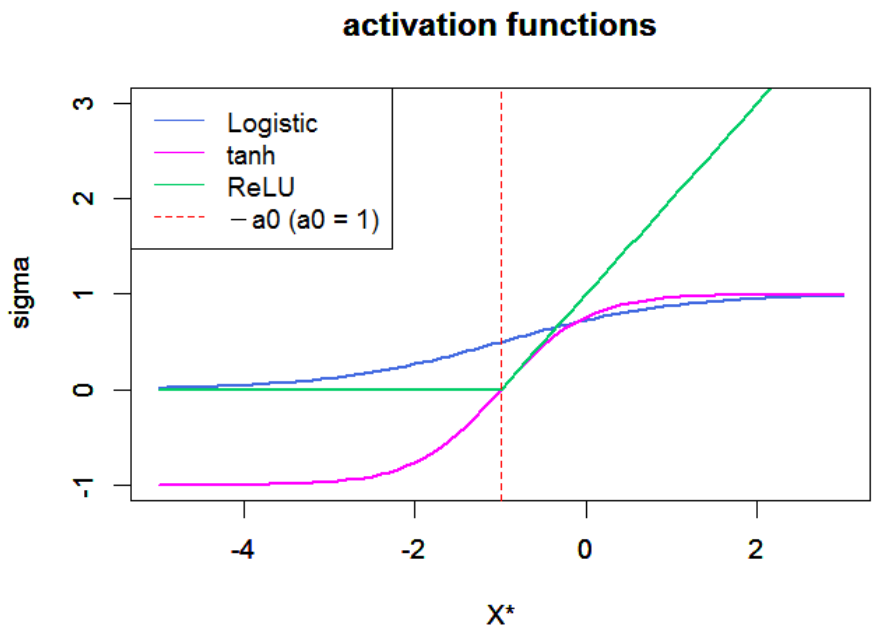
$$\sigma\left(\alpha_{m,0} + \mathbf{X}_m^*\right) = \tanh(\alpha_{m,0} + \mathbf{X}_m^*).$$

Activation Functions

- ReLU (Rectified Linear Unit):

$$\sigma\left(\alpha_{m,0} + \mathbf{X}_m^*\right) = \max\{0, \alpha_{m,0} + \mathbf{X}_m^*\}.$$

Currently, ReLU is the most popularly used for deep neural networks.



Fitting Neural Networks

Gradient Descent Method

- Suppose that we have training data of size N: $\{(Y_i, \mathbf{X}_i) : 1 \leq i \leq N\}$.
- In the neural network

$$\widetilde{Y}(\mathbf{X}; \theta) = g\left(\beta_0 + \sum_{m=1}^{M} \sigma\left(\alpha_{m,0} + \sum_{j=1}^{p} X_j \alpha_{m,j}\right) \beta_m\right),$$

  there are $M(p+2)+1$ unknown parameters:

$$\{(\alpha_{m,0}, \alpha_{m,1}, \ldots, \alpha_{m,p}) : 1 \leq m \leq M\} : M(p+1),$$
$$\{\beta_0, \beta_1, \ldots, \beta_M\} : M+1.$$

- To learn these parameters from the training dataset, we need to set an objective function (i.e., loss function).

Gradient Descent Method

- The goal of learning is to obtain

$$\widehat{\theta}_N = \underset{\theta}{\operatorname{argmin}}\, \ell_N(\theta)$$

  for some loss function $\ell_N(\cdot)$.
- Regression: Y = continuous and $g(T) = T$,

$$\ell_N(\theta) = \underbrace{\sum_{i=1}^{N} s_i(\theta)}_{\text{sum of squared errors}}, \text{ where } s_i(\theta) = (Y_i - \widetilde{Y}(\mathbf{X}_i;\theta))^2.$$

- Binary classification: Y = dummy and $g(T) = \frac{\exp(T)}{1+\exp(T)}$,

$$\ell_N(\theta) = \underbrace{\sum_{i=1}^{N} s_i(\theta)}_{\text{negative log-likelihood}}, \text{ where}$$

$$s_i(\theta) = -\left[Y_i \ln \widetilde{Y}(\mathbf{X}_i;\theta) + (1 - Y_i) \ln(1 - \widetilde{Y}(\mathbf{X}_i;\theta))\right].$$

Gradient Descent Method

- Since $\ell_N(\theta)$ is not convex in general, it is impossible to directly find a "global" minimizer.
- Then, we instead consider to find a "local" minimizer of $\ell_N(\theta)$.
- The generic approach to (locally) minimizing $\ell_N(\theta)$ is by a Gradient Descent Method.
- Here, the gradient means the gradient vector defined by

$$\nabla \ell_N(\theta) = \underset{(M(p+2)+1)\times 1}{\left[\frac{\partial \ell_N(\theta)}{\partial \alpha_{1,0}}, \ldots, \frac{\partial \ell_N(\theta)}{\partial \beta_M}\right]^\top}.$$

(Recall: $\theta = (\alpha, \beta) = (\alpha_{1,0}, \ldots, \alpha_{M,p}, \beta_0, \ldots, \beta_M)$)

Gradient Descent Method

- Set an initial candidate value of θ as $\theta^{(0)}$.
- The gradient descent method generates a new guess $\theta^{(t+1)}$ by moving in the negative gradient direction:

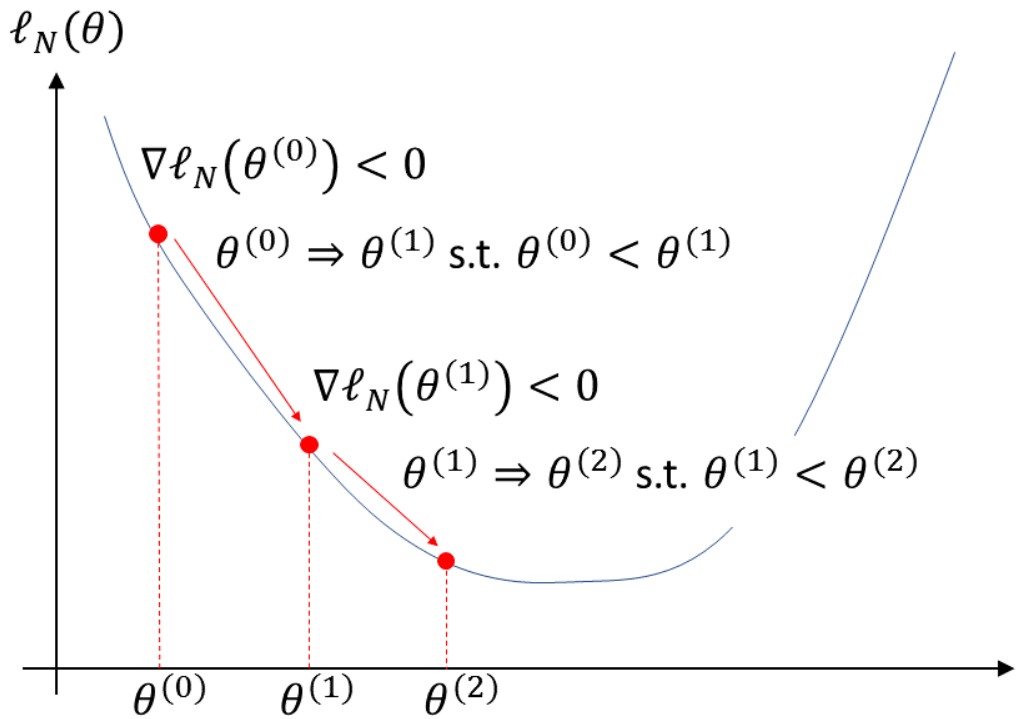
$$\theta^{(t+1)} = \theta^{(t)} - \gamma_t \nabla \ell_N(\theta^{(t)}) \ \ (t = 0, 1, \ldots),$$

where γ_t is the parameter that determines the amount of update at the t-th iteration, which is called the learning rate.

Gradient Descent Method

- If the learning rate is sufficiently small, it must hold that

$$\ell_N(\theta^{(t)}) \geq \ell_N(\theta^{(t+1)}) \geq \ell_N(\theta^{(t+2)}) \geq \cdots$$



Gradient Descent Method

- Thus, for sufficiently large t and small γ_t, we will obtain

$$\nabla \ell_N(\theta^{(t)}) \approx 0,$$

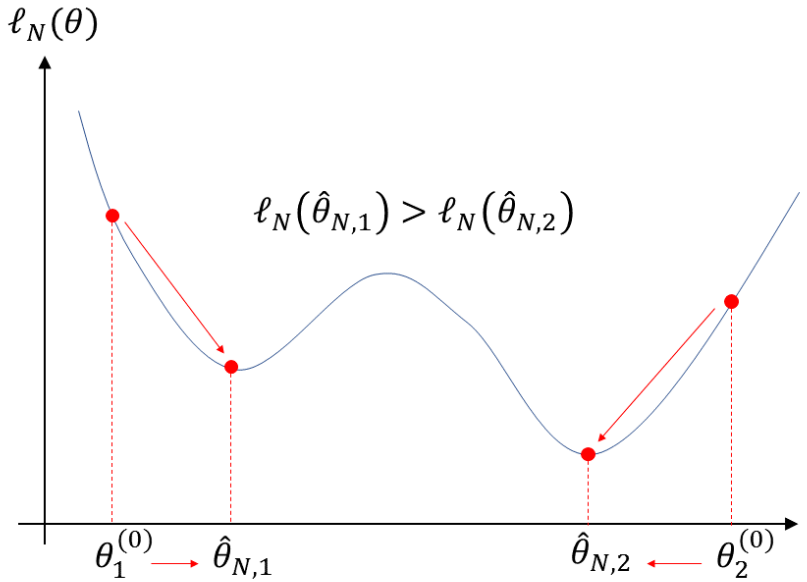
  and then the iteration stops, and define $\widehat{\theta}_N = \theta^{(t)}$ for such t.[1]
- A practical difficulty in directly implementing the above method: the partial derivatives of $\ell_N(\theta)$ w.r.t. the parameters in a layer closer to the input layer have more complicated form to calculate.
  - This can be a serious problem particularly when using deeper neural networks.
  - Several computationally easy-to-implement algorithms to perform gradient descent have been proposed; e.g., back-propagation.

[1] Obviously, how to choose an optimal learning rate γ_t in practice is a critical problem determining the quality of the learning. There is a huge literature on this issue; however, beyond the scope of this lecture.

Some Issues in Training Neural Networks

Starting Values

- One should try several starting values to get the best solution.[2]



- The starting values $\theta^{(0)}$ are usually chosen to be random values near zero.
- Note that using exact zero weights leads to perfect symmetry among the units in the same layer, and the learning fails.

[2] Another approach is *Bagging*: take the average of the predictions from several different neural networks.

Scaling of the Inputs

- The scaling of the inputs determines the effective scaling of the weights in the bottom layer, and it can have a large effect on the quality of the solution.
- It is common to standardize the input variables (subtract the mean and divide by the standard deviation) before running the learning algorithm.

Number of Hidden Units and Layers

- Generally speaking it is better to have too many hidden units (M) than too few.
- It is most common to put down a reasonably large number of units and train them with regularization (LASSO, etc).
- Choice of the number of hidden layers is guided by background knowledge and experimentation (this is still a theoretically unsolved problem).

Neural Networks in R

Spam Mail Detection

Spam Email Data

- Training data: **spam_train.csv**, Test data: **spam_test.csv**
- The data csv files are available from my website or from **Course Navi**.
- In this exercise, we use the **neuralnet** package.

```
install.packages("neuralnet")
library(neuralnet)
```

- Set your working directory appropriately, and import the csv files:

```
setwd("C:/Rdataset")
train <- read.csv("spam_train.csv")
 test <- read.csv("spam_test.csv")
```

Spam Mail Detection

```
>  head(train)
          our       mail       will       free        you       your
1 -0.13713261  2.0171787  0.3925946  0.49788818  0.6803231  0.6081222
2  1.61748092  1.0710834 -0.6286443  1.95103913 -0.4123388  0.4915342
3 -0.01817576 -0.3713240 -0.2804946  0.06194289  0.2579022  1.8489515
4 -0.46426394 -0.1696972  0.1372849 -0.30134485 -0.3391192 -0.1247166
5 -0.46426394 -0.3713240 -0.6286443 -0.30134485  0.4888256 -0.6743457
6 -0.46426394 -0.3713240 -0.6286443 -0.30134485 -0.9361406 -0.6743457
  exclamation     dollar  capitalAve capitalLong capitalTotal    type
1   0.4780467  0.1553156  0.12318162  0.06581726    0.7334251    spam
2   0.3701602  0.5620147 -0.01769697  0.06068619   -0.1159224    spam
3  -0.3298764  0.5498137 -0.10862197 -0.21126026   -0.1621005    spam
4  -0.3298764 -0.3083214  0.01296855  0.02476874    1.4458874 nonspam
5   0.4657869 -0.3083214  0.02154103 -0.03167298   -0.3121794    spam
6  -0.1803064 -0.3083214 -0.09418743 -0.20612919   -0.2528075 nonspam
>  dim(train)
[1] 3601   12
>  dim(test)
[1] 1000   12
```

Spam Mail Detection

Definitions of variables

Response variable

type "spam" or "nonspam"

Input variables

our, ..., your the frequency of the variable name in the email.

exclamation, dollar the frequency of the characters "!" and "$" in the email.

capitalAve, Long, Total the average, longest and total run-length of capital letters.

* All input variables are standardized.

Spam Mail Detection

- We first transform the response variable into a dummy variable.

```
train$type <- train$type == "spam"
 test$type <- test$type == "spam"
```

- Logistic regression (binary logit model):

```
logit <- glm(type ~ ., data = train,
             family = binomial(link = "logit"))
summary(logit)
```

Spam Mail Detection

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	0.06514	0.06733	0.967	0.333310	
our	0.48660	0.05407	9.000	< 2e-16	***
mail	0.18191	0.05221	3.484	0.000493	***
will	-0.18123	0.05647	-3.210	0.001329	**
free	0.87583	0.08552	10.242	< 2e-16	***
you	0.35925	0.04862	7.389	1.48e-13	***
your	0.45690	0.04772	9.575	< 2e-16	***
exclamation	0.66066	0.08776	7.528	5.13e-14	***
dollar	2.39538	0.17056	14.044	< 2e-16	***
capitalAve	-0.47163	0.56789	-0.831	0.406254	
capitalLong	2.77667	0.37104	7.484	7.23e-14	***
capitalTotal	0.25398	0.07718	3.291	0.001000	***

Spam Mail Detection

- Fitting neural networks: use the function `neuralnet()`.
- The basic syntax of `neuralnet()` is as follows:

```
neuralnet(model, data, hidden, act.fct, err.fct,
          linear.output)
```

  - `hidden`: an integer or a vector of integers specifying the number of hidden units in each layer.
  - `act.fct`: activation function: "logistic" or "tanh".
  - `err.fct`: loss function: "sse" (sum of squared errors) or "ce" (cross entropy / negative log-likelihood).
  - `linear.output`: whether the output function is an identity function or not: "TRUE" or "FALSE".

Spam Mail Detection

- Single-layer 2 hidden units:

```
nn1 <- neuralnet(type ~ ., data = train, hidden = 2,
  act.fct = "logistic", err.fct = "ce", linear.output = FALSE)
```

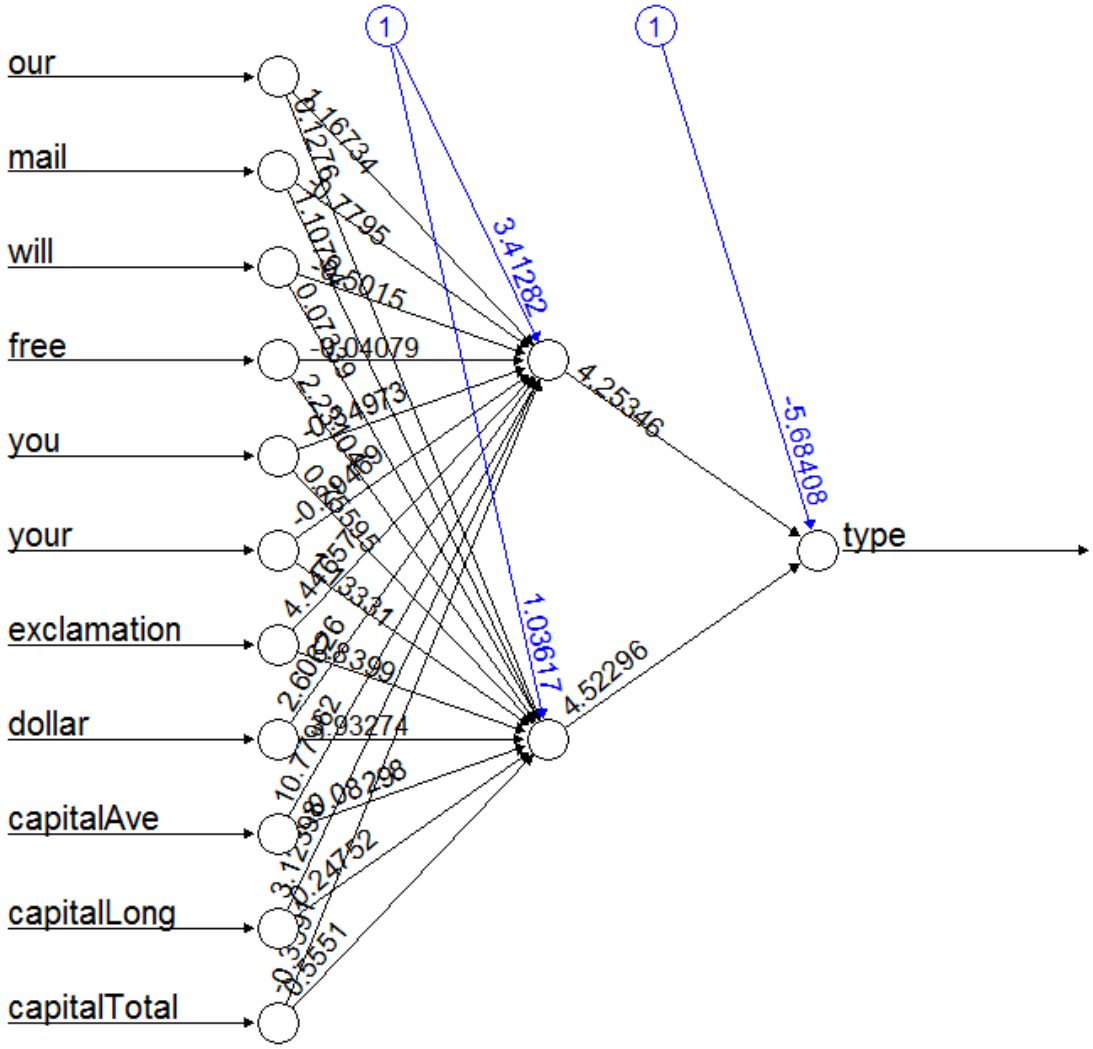
- 2-layer 2 hidden units for each (a deep neural network):[3]

```
nn2 <- neuralnet(type ~ ., data = train, hidden = c(2,2),
  act.fct = "logistic", err.fct = "ce", linear.output = FALSE)
```

[3] The optimization of deep neural networks is much more difficult than that of single-layer neural networks. If the following optimization does not return a solution, try typing `set.seed(10)` before running it.

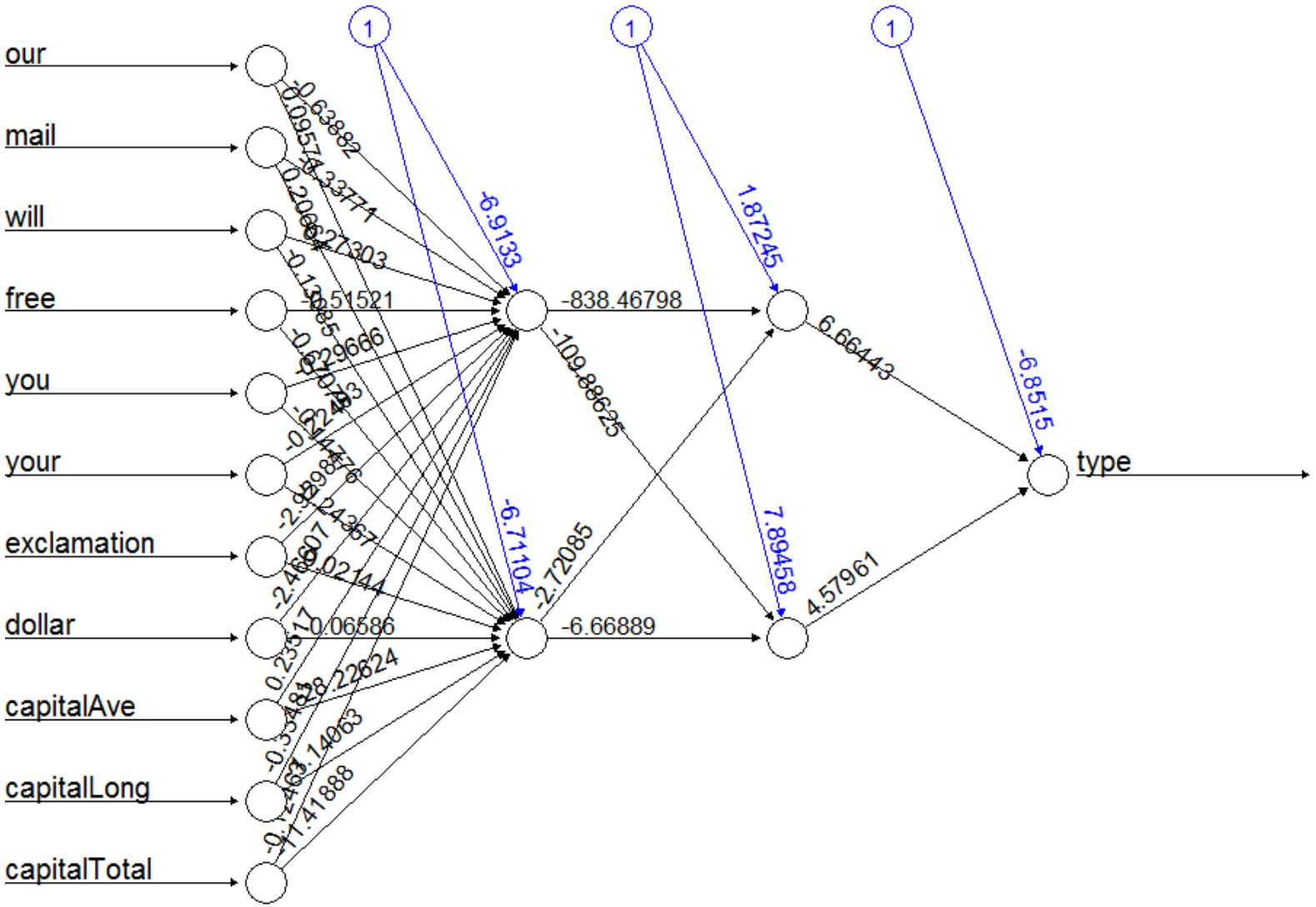
Spam Mail Detection

`plot(nn1)`



Spam Mail Detection

`plot(nn2)`



Spam Mail Detection

- Computation of $\widetilde{Y}(\mathbf{X}^{\text{test}};\theta)$ (= classification score $s(\mathbf{X}^{\text{test}})$, see Lecture Note # 7):

```
s1 <- predict(nn1, newdata = test)
s2 <- predict(nn2, newdata = test)
s.log <- predict(logit, newdata = test)
```

- Load the ROCR package and create "prediction" object:[4]

```
library(ROCR)
pred1 <- ROCR::prediction(s1, test$type)
auc1 <- performance(pred1,"auc")@y.values[[1]]
roc1 <- performance(pred1, "tpr", "fpr")
```

[4] Both packages **ROCR** and **neuralnet** contain `prediction()` function with the same name for different purposes. Here, since we would like to use `prediction()` function from the **ROCR**, we need to specify using the :: double colons.

Spam Mail Detection

- Similarly,

```
pred2 <- ROCR::prediction(s2, test$type)
auc2 <- performance(pred2,"auc")@y.values[[1]]
roc2 <- performance(pred2, "tpr", "fpr")
```

and

```
pred.log <- ROCR::prediction(s.log, test$type)
auc.log <- performance(pred.log,"auc")@y.values[[1]]
roc.log <- performance(pred.log, "tpr", "fpr")
```

Spam Mail Detection

- Check the AUC scores: `c(auc1,auc2,auc.log)`

```
>  c(auc1,auc2,auc.log)
[1] 0.9391928 0.9474122 0.9202770
```

  The deep neural network `nn2` performs the best.
- Plot the ROC curves:

```
plot(roc1, col = "royalblue")
par(new = T)
plot(roc2, col = "magenta")
par(new = T)
plot(roc.log, col = "black")
```

Spam Mail Detection