MSc Big Data for Business - *MAP 534*
Introduction to machine learning

Supervised classification (II)
Logistic regression & feed forward neural networks

Outline

Classification

Setting

→ Historical data about **individuals** $i = 1, \ldots, n$.

→ **Features** vector $X_i \in \mathbb{R}^d$ for each individual i.

→ For each i, the individual **belongs to a group** $(Y_i = 0)$ or not $(Y_i = 1)$.

→ $Y_i \in \{0, 1\}$ is the **label** of i.

Objective

→ Given a new X (with no corresponding label), **predict a label in** $\{0, 1\}$.

→ Use data $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ **to construct a classifier**.

Best Solution

The best solution f^* (which is independent of $\mathcal{D}_n$) is

$$f^* = \arg\min_{f \in \mathcal{F}} R(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}\left[\mathbb{1}_{Y \neq f(X)}\right] = \arg\min_{f \in \mathcal{F}} \mathbb{P}(Y \neq f(X))\,.$$

Bayes Predictor (explicit solution)

$\rightarrow$ **Binary classification** with $0-1$ loss:

$$f^*(\mathbf{X}) = \begin{cases} +1 & \text{if} \quad \mathbb{P}\left\{Y = 1 | \mathbf{X}\right\} \geqslant \mathbb{P}\left\{Y = 0 | \mathbf{X}\right\} \\ & \quad \Leftrightarrow \mathbb{P}\left\{Y = 1 | \mathbf{X}\right\} \geqslant 1/2\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

The explicit solution requires to **know the conditional law of Y given X**...

How to estimate the conditional law of Y?

Fully parametric modeling.

Estimate the law of $(\mathbf{X}, Y)$ and use the **Bayes formula** to deduce an estimate of the conditional law of Y: *LDA/QDA, Naive Bayes...*

Parametric conditional modeling.

Estimate the conditional law of Y by a **parametric** law: *linear regression, logistic regression, Feed Forward Neural Networks...*

Nonparametric conditional modeling.

Estimate the conditional law of Y by a **non parametric** estimate: *kernel methods, nearest neighbors...*

Fully parametric modeling - Discriminant Analysis

The **conditional densities are modeled as multivariate normal**. For all class $k \in \{0, 1\}$, conditionnally on $\{Y = k\}$,

$$X \sim \mathcal{N}(\mu_k, \Sigma_k)\,.$$

Discriminant functions:

$$\psi_k : x \mapsto \ln(g_k(x) + \ln(\mathbb{P}\{Y = k\})\,.$$

In a two-classes problem, the optimal classifier is:

$$\boxed{f^* : x \mapsto \mathbb{1}\{\psi_1(x) > \psi_0(x)\}\,.}$$

QDA (differents Σ_k in each class) and LDA ($\Sigma_k = \Sigma$ for all k)

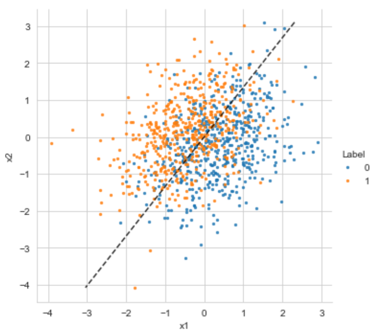
$\longrightarrow$ May lead to poor results is the **model does not describe the data correctly**.

Fully parametric modeling - Discriminant Analysis

In the LDA case, the classification rule is of the form:

$$f^*(x) = 1 \Leftrightarrow \langle w, x \rangle + b \geqslant 0\,,$$

where w **and** b **depends on the model parameters**.



→ How to relax the Gaussian assumption ? (**logistic model**).

→ How to design nonlinear classification rules ? (**neural networks**).

Outline

Semi-parametric modelling - logistic regression

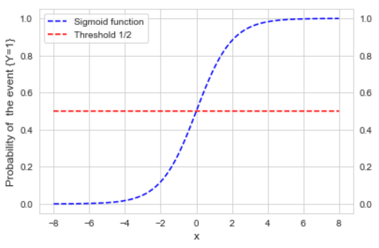
→ The objective is to **predict the label $Y \in \{0,1\}$** based on $X \in \mathbb{R}^d$.

→ Logistic regression **models the distribution of Y given X**.

$$\mathbb{P}(Y = 1|X) = \sigma(\langle w, X\rangle + b)\,,$$

where $w \in \mathbb{R}^d$ is a vector of model **weights** and $b \in \mathbb{R}$ is the **intercept**, and where σ is the **sigmoid** function.

$$\sigma : z \mapsto \frac{1}{1 + e^{-z}}\,.$$



→ The sigmoid function is a **model choice to map $\mathbb{R}$ into $(0,1)$**.

→ Another widespread solution for σ is $\sigma : z \mapsto \mathbb{P}(Z \leqslant z)$ where $Z \sim \mathcal{N}(0,1)$, which leads to a **probit** regression model.

Logistic regression

Log-odd ratio

$$\boxed{\log\Big(\mathbb{P}(Y=1|X)\Big) - \log\Big(\mathbb{P}(Y=0|X)\Big) = \langle w, X\rangle + b\,.}$$

Classification rule

Note that

$$\mathbb{P}(Y=1|X) \geqslant \mathbb{P}(Y=0|X)$$

if and only if

$$\langle w, x\rangle + b \geqslant 0\,.$$

$\longrightarrow$ This is a **linear classification** rule.

$\longrightarrow$ This classifier requires to **estimate w and b**.

Outline

Logistic regression

→ $\{(X_i, Y_i)\}_{1 \leqslant i \leqslant n}$ are **i.i.d. with the same distribution as** (X, Y).

Likelihood

$$\prod_{i=1}^n \mathbb{P}(Y_i|X_i) = \prod_{i=1}^n \sigma(\langle w, X_i\rangle + b)^{Y_i}\left(1 - \sigma(\langle w, X_i\rangle + b)\right)^{1-Y_i},$$
$$= \prod_{i=1}^n \sigma(\langle w, x_i\rangle + b)^{Y_i}\sigma(-\langle w, X_i\rangle - b)^{1-Y_i}$$

and the **normalized negative loglikelihood** is

$$f(w, b) = \frac{1}{n}\sum_{i=1}^n \log(1 + \mathrm{e}^{-Y_i(\langle w, X_i\rangle + b)}) = \frac{1}{n}\sum_{i=1}^n \ell(Y_i, \langle w, X_i\rangle + b)\,.$$

Logistic regression

Compute $\hat{w}_n$ and $\hat{b}_n$ as follows:

$$\boxed{(\hat{w}_n, \hat{b}_n) \in \operatorname{argmin}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \tfrac{1}{n} \textstyle\sum_{i=1}^n \log(1 + e^{-Y_i(\langle w, X_i \rangle + b)})\,.}$$

→ It is an **average of losses**, one for each sample point.

→ It is a **convex and smooth problem**.

Using the **logistic loss** function

$$\ell : (y, y') \mapsto \log(1 + e^{-yy'})$$

yields

$$(\hat{w}_n, \hat{b}_n) \in \operatorname*{argmin}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n \ell(Y_i, \langle w, X_i \rangle + b)\,.$$

Maximum likelihood estimate

Assume for now that the intercept is 0. Then, the likelihood is,

$$L_n(w) = \prod_{i=1}^{n} \left(\frac{e^{X_i^T w}}{1 + e^{X_i^T w}} \right)^{Y_i} \left(\frac{1}{1 + e^{X_i^T w}} \right)^{1-Y_i} = \prod_{i=1}^{n} \left(\frac{e^{X_i^T w Y_i}}{1 + e^{X_i^T w}} \right) .$$

And the **negative** log-**likelihood** is

$$\ell_n(w) = -\log(L_n(w)) = \sum_{i=1}^{n} \left(-Y_i X_i^T w + \log(1 + e^{X_i^T w}) \right) .$$

Derivatives

$$\begin{aligned} \frac{\partial \left(\log(L_n(w))\right)}{\partial w_j} &= \sum_{i=1}^{n} \left(Y_i X_{ij} - \frac{x_{ij} e^{X_i^T w}}{(1 + e^{X_i^T w})} \right) \\ &= \sum_{i=1}^{n} X_{ij} \left(Y_i - \sigma(\langle w, X_i \rangle) \right) . \end{aligned}$$

$\rightarrow$ **No explicit solution** for the maximizer of the loglikelihood... Parameter estimate obtained using **gradient based optimization** (see next lesson).

Maximum likelihood estimate

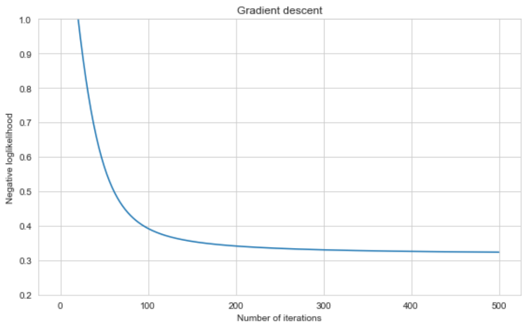
The negative loglikelihood

$$\ell_n(w) = -\log(L_n(w)) = \sum_{i=1}^n \left(-Y_i X_i^T w + \log(1 + e^{X_i^T w})\right) .$$

is minimized using a gradient descent algorithm.

Starting with an **initial estimate** $w^{(0)}$, for all $k \geqslant 1$, set

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla \ell_n(w^{(k-1)}) .$$

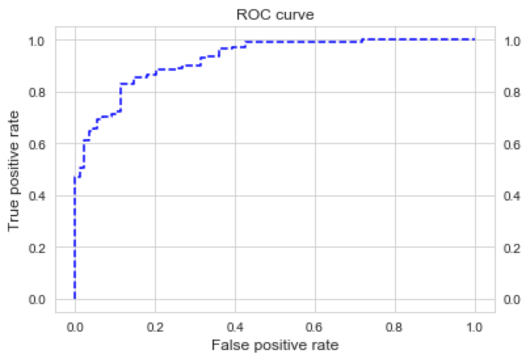


Maximum likelihood estimate

Let (w^*, b^*) be the parameter estimates **after the gradient descent algorithm**.

The usual logistic regression classifier is $f^*(X) = 1 \Leftrightarrow \mathbb{P}(Y = 1|X) > 1/2$.

Sensitivity of the classifier to this threshold: for each value $p^* \in (0,1)$ the ROC curve classifies individuals using $f^*(X) = 1 \Leftrightarrow \mathbb{P}(Y = 1|X) > p^*$ and plots the True positive rate as a function of the False positive rate.



The **gradient of the negative loglikelihood** is,

$$\boxed{\nabla \ell_n(w) = -\sum_{i=1}^{n} Y_i X_i + \sum_{i=1}^{n} \frac{\exp(\langle X_i, w \rangle)}{1+\exp(\langle X_i, w \rangle)} X_i \,.}$$

On the other hand, for all $1 \leqslant i \leqslant n$ and all $1 \leqslant j \leqslant d$,

$$\partial_j \left(\frac{\exp(\langle X_i, w \rangle)}{1+\exp(\langle X_i, w \rangle)} X_i \right) = \frac{\exp(\langle X_i, w \rangle)}{(1+\exp(\langle X_i, w \rangle))^2} X_{ij} X_i \,,$$

where X_{ij} is the jth component of X_i.

Then, the **Hessian matrix** is

$$\left(H_n(w) \right)_{\ell j} = \sum_{i=1}^{n} \frac{\exp(\langle X_i, w \rangle)}{(1+\exp(\langle X_i, w \rangle))^2} X_{ij} X_{i\ell} \,,$$

that is,

$$\boxed{H_n(w) = \sum_{i=1}^{n} \frac{\exp(\langle X_i, w \rangle)}{(1+\exp(\langle X_i, w \rangle))^2} X_i X_i^T \,.}$$

$H_n(\beta)$ **is a semi positive definite matrix, which implies that** $\ell_n(\beta)$ **is convex**.

Asymptotic properties

Assumptions

→ $\widehat{w}_n \to w^*$ **almost surely**.

→ There exists a continuous and nonsingular function H such that $n^{-1} H_n(w)$ **converges to** $H(w)$, uniformly in a ball around w^*.

For all $t \in \mathbb{R}^d$, using a Taylor expansion,

$$\mathbb{E}\left[\exp\left(-\frac{1}{\sqrt{n}}\langle t, \nabla \ell_n(w^\star)\rangle\right)\right] \to_{n\to\infty} \exp\left(\frac{1}{2} t^T H(w^\star) t\right).$$

Therefore,

$$\boxed{-\nabla \ell_n(w^\star)/\sqrt{n} \Rightarrow \mathcal{N}(0, H(w^\star))\,.}$$

On the other hand, by **Slutsky lemma**,

$$\boxed{\sqrt{n}(\widehat{w}_n - w^\star) \Rightarrow \mathcal{N}(0, H(w^\star)^{-1})\,.}$$

Confidence interval

$\to \sqrt{n}(\widehat{w}_j - w_j^\star)$ **converges in distribution to a centered Gaussian random variable with variance** $(H(w^\star)^{-1})_{jj}$.

Almost surely,

$$\widehat{\sigma}_{n,j}^2 = (nH_n(\widehat{w}_n)^{-1})_{jj} \to_{n\to\infty} (H(w^\star)^{-1})_{jj} \,.$$

Then,

$$\boxed{\sqrt{\frac{n}{\widehat{\sigma}_{n,j}^2}}(\widehat{w}_{n,j} - \beta_j^\star) \to_{n\to\infty} \mathcal{N}(0,1) \,.}$$

An **asymptotic confidence interval** $\mathcal{I}_{n,\alpha}$ **of level** $1-\alpha$ is then

$$\boxed{\mathcal{I}_{n,\alpha} = \left[\widehat{w}_{n,j} - z_{1-\alpha/2}\sqrt{\frac{\widehat{\sigma}_{n,j}^2}{n}} \,,\; \widehat{\beta}_{n,j} + z_{1-\alpha/2}\sqrt{\frac{\widehat{\sigma}_{n,j}^2}{n}}\right] \,,}$$

where $z_{1-\alpha/2}$ is the quantile of order $1-\alpha/2$ of $\mathcal{N}(0,1)$.

Outline

Outline

Softmax function

$\rightarrow$ The objective is to **predict the label** $Y \in \{1, \dots, M\}$ based on $X \in \mathbb{R}^d$.

$\rightarrow$ Softmax regression **models the distribution of** Y **given** X.

The model

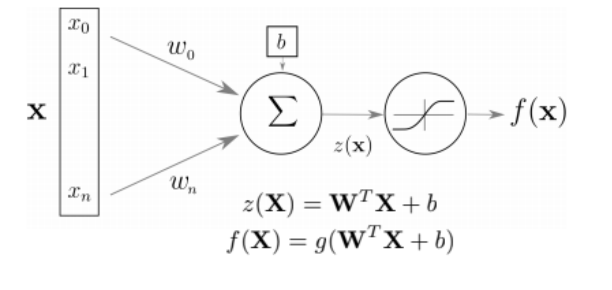
For all $1 \leqslant m \leqslant M$,

$$z_m = \langle w_m, X \rangle + b_m \,,$$
$$\mathbb{P}(Y = m | X) = \mathrm{softmax}(z)_m \,,$$

where $z \in \mathbb{R}^M$, $w_m \in \mathbb{R}^d$ is a vector of model **weights** and $b_m \in \mathbb{R}$ is an **intercept**, and where $\mathrm{softmax}$ is the **softmax** function: for all $1 \leqslant m \leqslant M$,

$$\mathrm{softmax}(z)_m = \frac{\exp(z_m)}{\sum_{j=1}^M \exp(z_j)} \,.$$

A neuron



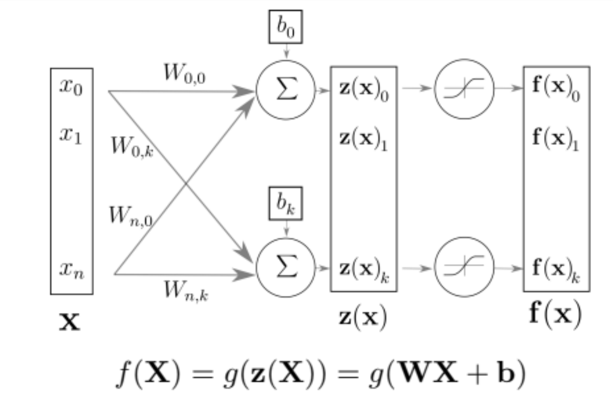
→ X **input in** $\mathbb{R}^d$.

→ $z(X)$ **pre-activation in** $\mathbb{R}^M$, with **weight** $W \in \mathbb{R}^{dxM}$ and **bias** $b \in \mathbb{R}^M$.

→ g **softmax function**.

One neuron is a multi-class extension of the logistic regression model.

Layer of neurons and hidden states



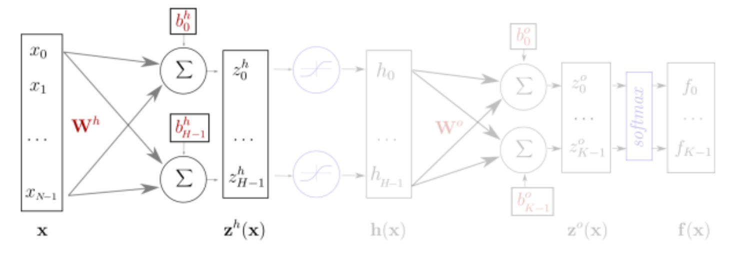
→ X **input in** $\mathbb{R}^d$.

→ $z(X)$ **pre-activation in** $\mathbb{R}^k$, with **weight** $W \in \mathbb{R}^{dxk}$ and **bias** $b \in \mathbb{R}^k$.

→ g **any activation function** (nonlinear & nondecreasing function).

→ $f(X)$ **hidden state** in $\mathbb{R}^k$ which may be used as input of a new neuron...

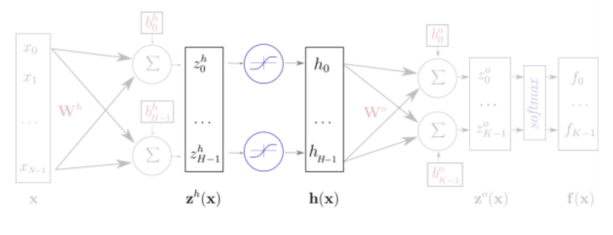
Feed Forward Network



→ X **input in** $\mathbb{R}^d$.

→ $z^h(X)$ **pre-activation in** $\mathbb{R}^H$, with **weight** $W^h \in \mathbb{R}^{dxH}$ and **bias** $b^h \in \mathbb{R}^H$.

Feed Forward Network

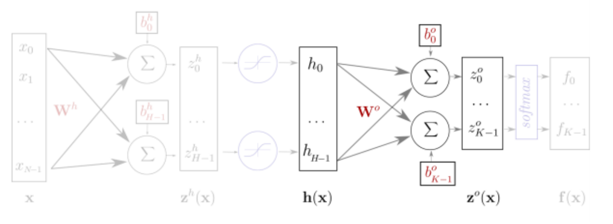


→ X **input in** $\mathbb{R}^d$.

→ $z^h(X)$ **pre-activation in** $\mathbb{R}^H$, with **weight** $W^h \in \mathbb{R}^{dxH}$ and **bias** $b^h \in \mathbb{R}^H$.

→ g **any activation function** to produce $h \in \mathbb{R}^H$.

Feed Forward Network

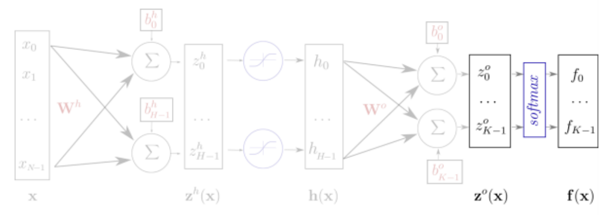


→ X **input in** $\mathbb{R}^d$.

→ $z^h(X)$ **pre-activation in** $\mathbb{R}^H$, with **weight** $W^h \in \mathbb{R}^{dxH}$ and **bias** $b^h \in \mathbb{R}^H$.

→ g **any activation function** to produce $h \in \mathbb{R}^H$.

→ $z^o(X)$ **pre-activation in** $\mathbb{R}^M$, with **weight** $W^o \in \mathbb{R}^{HxM}$ and **bias** $b^o \in \mathbb{R}^M$.



→ X **input in** $\mathbb{R}^d$.

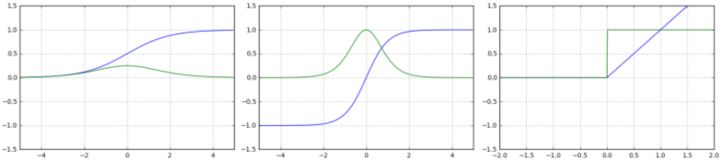
→ $z^h(X)$ **pre-activation in** $\mathbb{R}^H$, with **weight** $W^h \in \mathbb{R}^{dxH}$ and **bias** $b^h \in \mathbb{R}^H$.

→ g **any activation function** to produce $h \in \mathbb{R}^H$.

→ $z^o(X)$ **pre-activation in** $\mathbb{R}^M$, with **weight** $W^o \in \mathbb{R}^{HxM}$ and **bias** $b^o \in \mathbb{R}^M$.

→ Apply the **softmax function to produce the output**, i.e. $\mathbb{P}(Y = m|X)$ for $1 \leqslant m \leqslant M$.

Activation functions



$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad \mathrm{relu}(x) = \max(0,x)$$

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1-\mathrm{sigm}(x)) \qquad \tanh'(x) = 1 - \tanh(x)^2 \qquad \mathrm{relu}'(x) = 1_{x>0}$$

$\longrightarrow$ As there is no modelling assumptions anymore, **virtually any activation function** may be used.

$\longrightarrow$ The rectified linear unit (RELU) activation function $\sigma(x) = \max(0, x)$ and its extensions are the default recommendation in modern implementations (Jarrettet al., 2009; Nair and Hinton, 2010; Glorot et al., 2011a), (Maas et al.,2013), (He et al., 2015). One of the major motivations arise from the **gradient based parameter optimization which is numerically more stable with this choice**.

Outline

MNIST

→ This dataset contains images representing handwritten digits. Each image is made of **28 x 28 pixels**, and each pixel is represented by an integer (gray level). These arrays can be flattened into vectors in $\mathbb{R}^{784}$.

→ The **labels in $\{0, \ldots, 9\}$ are represented using one-hot-encoding** and grayscale of each pixel in $\{0, \ldots, 255\}$ are normalized to be in $(0, 1)$.

```python
from keras.datasets import mnist
# Number of classes
num_classes = 10
# input image dimensions
img_rows, img_cols = 28, 28

# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train     = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test      = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test  = x_test.astype('float32')

print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

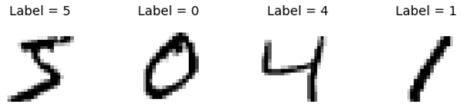
```
x_train shape: (60000, 28, 28, 1)
x_test shape: (10000, 28, 28, 1)
y_train shape: (60000,)
y_test shape: (10000,)
60000 train samples
10000 test samples
```

MNIST

→ This dataset contains images representing handwritten digits. Each image is made of **28 x 28 pixels**, and each pixel is represented by an integer (gray level). These arrays can be flattened into vectors in $\mathbb{R}^{784}$.

→ The **labels in $\{0, \ldots, 9\}$ are represented using one-hot-encoding** and grayscale of each pixel in $\{0, \ldots, 255\}$ are normalized to be in $(0, 1)$.

```
plt.figure(figsize=(8, 2))
for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(x_train[i].reshape(28, 28),
               interpolation="none", cmap="gray_r")
    plt.title('Label = %d' % y_train[i], fontsize=14)
    plt.axis("off")
plt.tight_layout()
```



The model with Keras

```
model_ffnn = Sequential()

model_ffnn.add(Flatten(input_shape=input_shape))

model_ffnn.add(Dense(128, activation='relu'))

model_ffnn.add(Dense(num_classes, activation='softmax'))
```

```
model_ffnn.compile(
    loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adagrad(),
    metrics=['accuracy']
)

model_ffnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Feed Forward Neural network. h_1 is obtained with the **RELU activation function and is in $\mathbb{R}^{128}$**. The last layer is $h_2 \in \mathbb{R}^{10}$ and is **obtained with the softmax activation function** so that each component m models $\mathbb{P}(Y = m|X)$. This neural network with one hidden layer relies on 101.770 parameters.

Inference

$\rightarrow$ This models relies on more than **100.000 unknown parameters** which should be estimated.

$\rightarrow$ As for the logistic regression and the discriminant analysis, a common choice is to **minimize the negative loglikelihood of the data**:

$$\theta \mapsto -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10}\mathbb{1}_{Y_i=k}\log\mathbb{P}_\theta(Y_i = k|X_i)\,.$$

$\rightarrow$ The negative loglikelihood is computed using $n = 60.000$ training samples and **minimized using gradient descent algorithms** - see next lesson.

$\rightarrow$ Then, the performance of the model is assessed using 10.000 new (test) samples: the **accuracy is the frequency of labels which are well predicted by the model with the estimated parameters**.

```python
batch_size = 32
epochs = 8

# Run the train
history = model_ffnn.fit(x_train, y_train,
                         batch_size=batch_size,
                         epochs=epochs,
                         verbose=1,
                         validation_data=(x_test, y_test))
score = model_ffnn.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```python
plt.figure(figsize=(5, 4))
plt.plot(history.epoch, history.history['acc'], lw = 1, label='Training')
plt.plot(history.epoch, history.history['val_acc'], lw = 1, label='Testing')
plt.legend()
plt.title('Accuracy of softmax regression', fontsize=16)
plt.xlabel('Epoch', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.tick_params(labelright=True)
plt.grid('True')
plt.tight_layout()
```

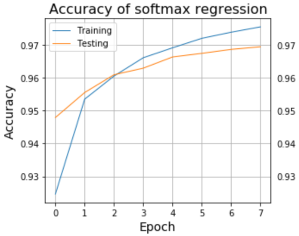


Figure 2: Minimization of the negative liglikelihood using a gradient descent algorithm (here AdaGrad). The gradient is computed using batches of 32 observations and the whole data set is used 8 times.