



Universität Hamburg  
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Fabian Barteld, Benjamin Milde, Prof. Dr.  
Chris Biemann

---

## **TENSORFLOW — REGRESSION MODELS**

# Linear Regression

# Linear Regression

- Given:  $(x_1, y_1), \dots, (x_n, y_n)$
- Goal: find  $w$  and  $b$  such that

$$\hat{y}_i = wx_i + b$$

fits the data, i.e.

$$\arg \min_{w, b} \frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{n}$$

# Define model parameters

Model:  $\hat{y}_i = wx_i + b$

Parameters:  $w$ ,  $b$ , tensors of rank 0

```
w = tf.Variable(tf.ones([]),  
                name="weight")  
b = tf.Variable(tf.zeros([]),  
                name="bias")
```

## Define the model

$$\begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \odot \begin{pmatrix} w \\ \vdots \\ w \end{pmatrix} + \begin{pmatrix} b \\ \vdots \\ b \end{pmatrix}$$

```
yhat = tf.add(tf.multiply(X, w), b)
```

The scalars  $w$  and  $b$  are converted into vectors of the same length as  $X$  (broadcast);

<https://www.tensorflow.org/performance/xla/broadcasting>

## Define the loss

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

```
loss = tf.reduce_mean(tf.square(yhat - Y))
```

# Optimization

```
## Optimizer
optimizer = tf.train.GradientDescentOptimizer(
    0.01 # learning rate
).minimize(loss)

with tf.Session() as sess:
    ## initialize parameters
    sess.run(tf.global_variables_initializer())

    for i in range(20):
        ## run one epoch
        sess.run(optimizer)
```

# Hands on

Do a linear regression to learn  $y = 2x + 1$

```
X_data = np.array([1., 2., 3., 4., 5., 6.],  
                  dtype=np.float32).reshape(6, 1)  
Y_data = 2*X_data + 1
```



# Multiple Linear Regression

# Defining the input

Tensorflow graphs use placeholders for input values

```
input_dim = 13
```

```
X = tf.placeholder(tf.float32, [None, input_dim])  
Y = tf.placeholder(tf.float32, [None, 1])
```

Defines placeholders for two tensors of rank 2,  
the shape is [Number of examples, Dimension]

# Adapting the model

$$\begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix} = \begin{pmatrix} x_{1,1} & \dots & x_{1,input\_dim} \\ \vdots & & \vdots \\ x_{n,1} & \dots & x_{n,input\_dim} \end{pmatrix} \times \begin{pmatrix} w_1 \\ \vdots \\ w_{input\_dim} \end{pmatrix} + \begin{pmatrix} b \\ \vdots \\ b \end{pmatrix}$$

```
w = tf.Variable(tf.ones(input_dim))
yhat = tf.add(tf.matmul(X, w), b)
```

# Getting data into the model

```
## Optimizer
```

```
optimizer = tf.train.GradientDescentOptimizer(  
    0.01 # learning rate  
).minimize(loss)
```

```
with tf.Session() as sess:
```

```
    ## initialize parameters
```

```
    sess.run(tf.global_variables_initializer())
```

```
    for i in range(20):
```

```
        ## run one epoch
```

```
        sess.run(optimizer, {X: data_X, Y: data_Y})
```

# Hands on

Do a multiple linear regression with Boston housing prices

```
from sklearn.datasets import load_boston  
from sklearn.preprocessing import scale
```

```
data_X, data_Y = load_boston(True)  
data_X = scale(data_X)  
data_Y = data_Y.reshape(len(data_Y), 1)
```

# Logistic regression

# Multiple Logistic regression

$$p_i = S(WX_i + b)$$

Loss (Binary-crossentropy):

$$\frac{1}{N} \sum_{i=1}^N (y_i \log p_i + (1 - y_i)(\log 1 - p_i))$$

In tensorflow:

Don't use – numerical problems!

```
p = tf.sigmoid(yhat)
```

```
loss = tf.reduce_mean(y*tf.log(p) + (1-y)*tf.log(1-p))
```

# Multiple Logistic regression

$$p_i = S(WX_i + b)$$

Loss (Binary-crossentropy):

$$\frac{1}{N} \sum_{i=1}^N (y_i \log p_i + (1 - y_i) (\log 1 - p_i))$$

In tensorflow:

**Don't use – numerical problems!**

```
p = tf.sigmoid(yhat)
loss = tf.reduce_mean(y*tf.log(p) + (1-y)*tf.log(1-p))
```

Optimized version (**Use this instead!**)

```
loss = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        labels=y, logits=yhat))
```



# Scaling the input data

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

# Hands on: Binary classification

Dataset: [http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_breast\\_cancer.html](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)

```
from sklearn.datasets import load_breast_cancer  
from sklearn.model_selection import train_test_split
```

```
## load the data
```

```
bc = load_breast_cancer()  
x_data = bc['data']           # shape: (569,30)  
y_data = bc['target'].reshape(  
    len(bc['target']), 1) # shape: (569, 1)
```

```
x_train, x_test, y_train, y_test =  
    train_test_split(x_data, y_data)
```

# One-hot encoding of nominal features

Names dataset <http://www.nltk.org/book/ch06.html>

```
def gender_features(word):  
    return {'last_letter': word[-1]}
```

```
def gender_features(word):  
    return {'suffix1': word[-1:],  
            'suffix2': word[-2:]}
```

# One-hot encoding of nominal features

Names dataset <http://www.nltk.org/book/ch06.html>

```
def gender_features(word):  
    return {'last_letter': word[-1]}
```

```
def gender_features(word):  
    return {'suffix1': word[-1:],  
            'suffix2': word[-2:]}
```

```
from sklearn.feature_extraction import DictVectorizer  
feat_vectorizer = DictVectorizer(  
    dtype=numpy.int32, sparse=False)  
train_X = feat_vectorizer.fit_transform(  
    train_feats)  
test_X = feat_vectorizer.transform(test_feats)
```

# Stochastic gradient descent

```
with tf.Session() as sess:  
    ## initialize parameters  
    sess.run(tf.global_variables_initializer())  
  
    for i in range(20):  
        ## run one epoch  
        ## update for each training example  
        for x, y in zip(x_data, y_data):  
            sess.run(optimizer, {X: x, Y: y})
```

# Stochastic gradient descent

```
with tf.Session() as sess:  
    ## initialize parameters  
    sess.run(tf.global_variables_initializer())  
  
    for i in range(20):  
        ## run one epoch  
        ## update for each training example  
        for x, y in zip(x_data, y_data):  
            sess.run(optimizer, {X: x, Y: y})
```

Usually the data is shuffled and passed in small batches to the optimizer.

# Hands on

Fit a logistic regression model to the names dataset

<http://www.nltk.org/book/ch06.html>

```
import nltk
## names must be installed by running
## nltk.download('names')
from nltk.corpus import names
import random

labeled_names = (
    [(n, 0) for n in names.words('male.txt')] +
    [(n, 1) for n in names.words('female.txt')]
random.shuffle(labeled_names)
```