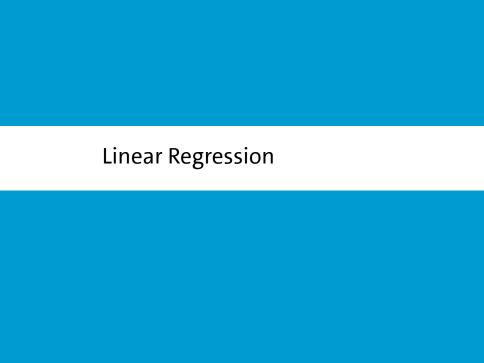


Fabian Barteld, Benjamin Milde

TENSORFLOW - REGRESSION MODELS



Linear Regression

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

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

fits the data, i.e.

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

Define model parameters

```
Model: ŷ<sub>i</sub> = wx<sub>i</sub> + 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

Linear Regression

Define the loss

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

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

Linear Regression ○○○○○●○

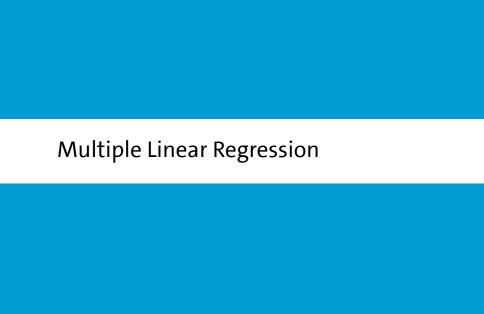
Optimization

```
## Optimizer
optimizer = tf.train.GradientDescentOptimizer(
  0.01 # learning rate
  ). minimize (loss)
with tf. Session() as sess:
  ## initalize parameters
  sess.run(tf.global variables initializer())
  for i in range (20):
      ## run one epoch
      sess.run(optimizer)
```

Y = 2 * X + 1

Hands on

Do a linear regression to learn y = 2x + 1X = np.array ([1., 2., 3., 4., 5., 6.], dtype=np.float32).reshape(6, 1)



Defining the input

```
Tensorflow graphs use placholders 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

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

Getting data into the model

```
## Optimizer
optimizer = tf.train.GradientDescentOptimizer(
  0.01 # learning rate
  ). minimize (loss)
with tf. Session() as sess:
  ## initalize parameters
  sess.run(tf.global variables initializer())
  for i in range (20):
      ## run one epoch
      sess.run(optimizer, {X: x data, Y: y data})
```

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)
```



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!)

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
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)
v 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



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)
```

Logistic regression

Stochastic gradient descent

```
with tf.Session() as sess:
    ## initalize 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})
```

Logistic regression

Stochastic gradient descent

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



Logistic regression ○○○○○●

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)
```