Loss functions

Puteaux, Fall/Winter 2020-2021

```
##############################
##                          ##
##  Deep Learning in Python  ##
##                          ##
##############################
```

`§2 Introduction to TensorFlow in Python`

`§2.2 Linear models`

1 Loss functions

1.1 What is the loss functions?

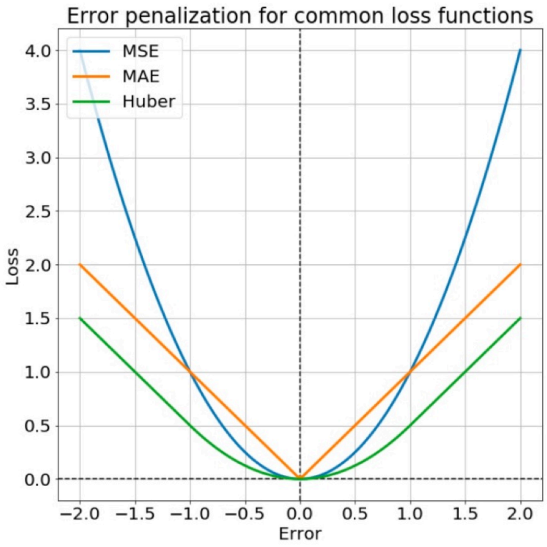
- **Fundamental TensorFlow operation**:
  - used to train a model
  - measure of model fit
- **Higher value → worse fit**:
  - minimize the loss function

1.2 What are the common loss functions in TensorFlow?

- **TensorFlow has operations for common loss functions**:
  - mean squared error (MSE)
  - mean absolute error (MAE)
  - huber error
- **Loss functions are accessible from `tf.keras.losses()`**:
  - `tf.keras.losses.mse()`
  - `tf.keras.losses.mae()`
  - `tf.keras.losses.Huber()`

1.3 Why is it in need to care about loss functions?

- **MSE**:
  - strongly penalizes outliers
  - high (gradient) sensitivity near minimum
- **MAE**:
  - scales linearly with the size of the error
  - low sensitivity near minimum
- **Huber**:
  - similar to MSE near minimum
  - similar to MAE away from the minimum



1.4 Code of defining the loss function:

[1]:
```
import pandas as pd
import numpy as np

housing = pd.read_csv('ref1. King county house sales.csv')

price_log = np.log(np.array(housing['price'], np.float32))
size_log = np.log(np.array(housing['sqft_lot'], np.float32))
targets = price_log
features = size_log
intercept = 0.1
slope = 0.1

predictions = intercept + features * slope
```

[2]:
```
# Import TensorFlow under standard alias
import tensorflow as tf

# Compute the MSE loss
loss = tf.keras.losses.mse(targets, predictions)

loss
```

[2]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=145.44653>
```

[3]:
```
# Define a linear regression model
def linear_regression(intercept, slope=slope, features=features):
    return intercept + features * slope
```

[4]:
```
# Define a loss function to compute the MSE
def loss_function(intercept, slope, targets=targets, features=features):
    # Compute the predictions for a linear model
    predictions = linear_regression(intercept, slope, features)

    # Return the loss
    return tf.keras.losses.mse(targets, predictions)
```

[5]:
```
from sklearn.model_selection import train_test_split

train_features, test_features, train_targets, test_targets = train_test_split(
    features, targets, test_size=0.3, random_state=42)
```

[6]:
```
# Compute the loss for test data inputs
loss_function(intercept, slope, test_targets, test_features)
```

[6]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=145.57338>
```

[7]:
```
# Compute the loss for default data inputs
loss_function(intercept, slope)
```

[7]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=145.44653>
```

1.5 Practice exercises for loss functions:

▶ **Package pre-loading:**

[8]:
```
import pandas as pd
import numpy as np
```

▶ **Data pre-loading:**

[9]:
```
housing = pd.read_csv('ref1. King county house sales.csv')
price = np.array(housing['price'], np.float32)

predictions = np.loadtxt('ref5. Predictions.csv', delimiter=',')
```

▶ **TensorFlow loss functions practice:**

[10]:
```
# Import the keras module from tensorflow
from tensorflow import keras

# Compute the mean squared error (mse)
loss = keras.losses.mse(price, predictions)

# Print the mean squared error (mse)
print(loss.numpy())
```

```
141171604777.141
```

[11]:
```
# Import the keras module from tensorflow
from tensorflow import keras

# Compute the mean absolute error (mae)
loss = keras.losses.mae(price, predictions)

# Print the mean absolute error (mae)
print(loss.numpy())
```

```
268827.9930163703
```

▶ **Package re-pre-loading:**

[12]:
```
from tensorflow import constant, Variable, float32
```

▶ **Data re-pre-loading:**

[13]:
```
features = constant([1., 2., 3., 4., 5.], dtype=float32)
targets = constant([2., 4., 6., 8., 10.], dtype=float32)
```

▶ **Loss function modification practice:**

[14]:
```
# Initialize a variable named scalar
scalar = Variable(1.0, float32)


# Define the model
def model(scalar, features=features):
    return scalar * features
```

```
# Define a loss function
def loss_function(scalar, features=features, targets=targets):
    # Compute the predicted values
    predictions = model(scalar, features)

    # Return the mean absolute error loss
    return keras.losses.mae(targets, predictions)


# Evaluate the loss function and print the loss
print(loss_function(scalar).numpy())
```

```
3.0
```