VL Deep Learning for Natural Language Processing

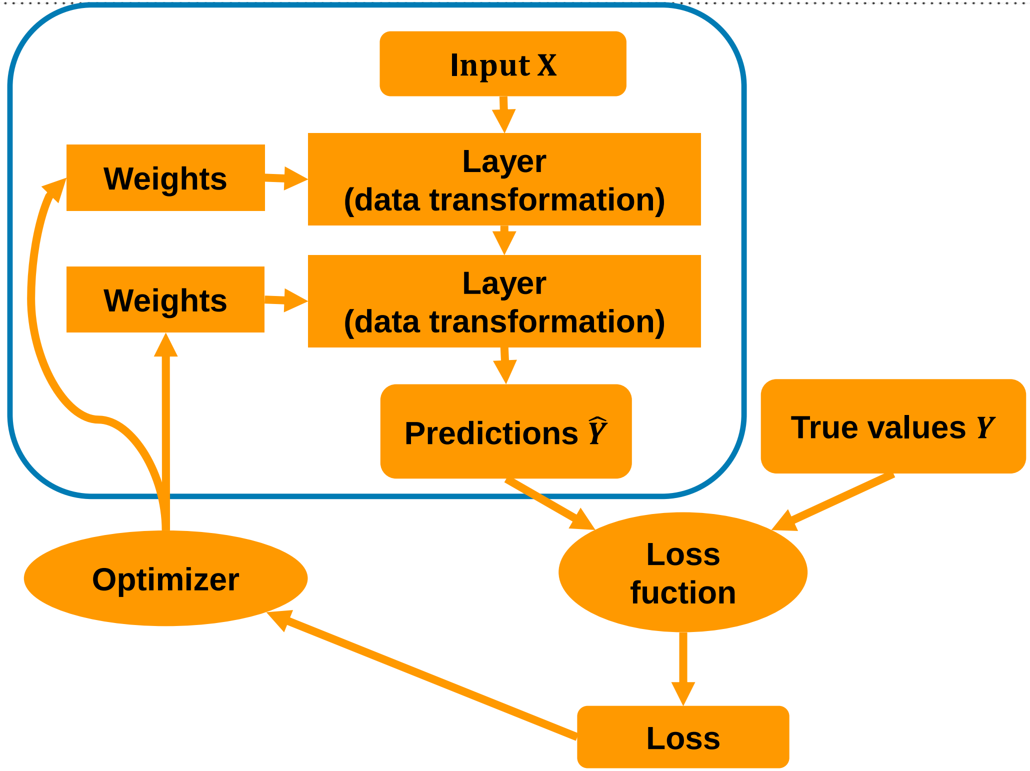
06. Text Classification

Prof. Dr. Ralf Krestel

AG Information Profiling and Retrieval

Leibniz Gemeinschaft

Anatomy of a Neural Network

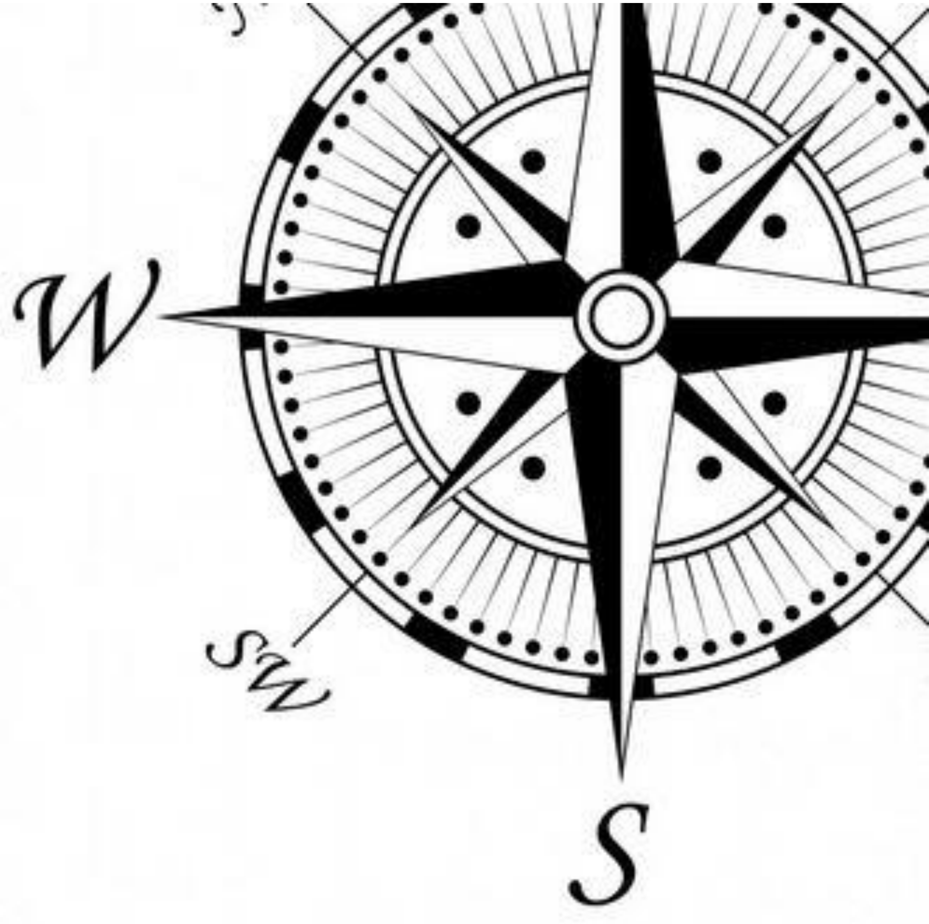


Learning Goals for this Chapter

- Use neural nets to solve simple problems
  - Binary classificaiton
  - Multi-Class Classification
  - Regression
- Understand ensembles

- Relevant chapters:
  - P3, D8

Topics Today



1. **Classification of Movie Reviews**
2. Classification of News Articles
3. Prediction of Real Estate Prices
4. Ensembles

Loading the Data

```
from keras.datasets import imdb
(train_data,train_labels),(test_data, test_labels)=imdb.load_data(num_words=10000)
```

Word indices

0=negative
1=postitive

Only the top-10k most frequent words

```
def get_review_text(sample):
        word_index = imdb.get_word_index()
        reverse_word_index = dict([(value,key) for (key,value) in word_index.items()])
        decoded_review = ' '.join([reverse_word_index.get(i - 3, '?')
                                   for i in train_data[sample]])
        return decoded_review
```

0=padding;
1=startMarker;
2=unknown

Preprocessing of Textual Data

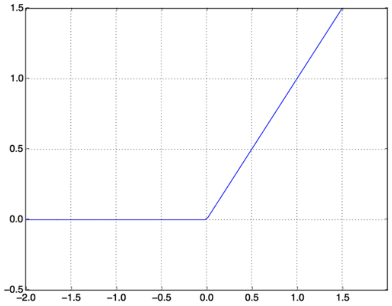
- Two options to represent text
  1. **Sequence of words**
     - Ordering is retained
     - Input tensor of shape (samples, wortIndices)
     - Input texts need have the same length ⟶ padding
     - Next layert: embedding layer
  2. **One-hot encoding**
     - Ordering is lost
     - Frequency of words gets lost
     - Input tensor of shape (samples, vocabularySize)
     - Almost all entries in input vector are 0
     - Next layer: dense layer

Vectorization

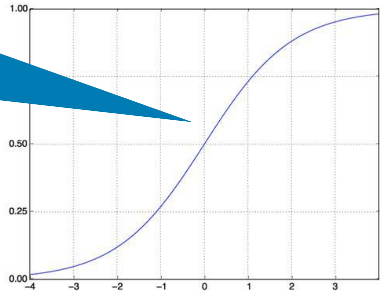
```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
        results = np.zeros((len(sequences), dimension))
        for i, sequence in enumerate(sequences):
                results[i, sequence] = 1.
        return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

The Neural Network

1. Network architecture
   – What kind of layers?
     - Dense (fully connected)
   – What kind of activation function?
     - relu (rectified linear unit) for hidden layers
     - sigmoid for output layer



Output in range [0,1]; Can be interpreted as probabilities!



2. Configuration
   – How many layers?
     - Two hidden layers + one output layer
   – How many units per layer?
     - 2x16 + 1x1 for output

Activation Function

- The activation function is key!

- A dense layer only consists of a scalar product and addition:

  `output = dot(W, input) + b`
  - Only linear transformation possible
  - The hypothesis space of the layer would be all linear transformations from the input data into a space with dimention=number of units of layer.
  - Multiple layers in a row would be more powerful.
- What is needed: a **non-linearity** ⟶ activation fuction

  `output = relu(dot(W, input) + b)`
- relu is the most popular one
  - Others, less popular ones include prelu, elu, etc.

The Network

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
```

```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['accuracy'])
```

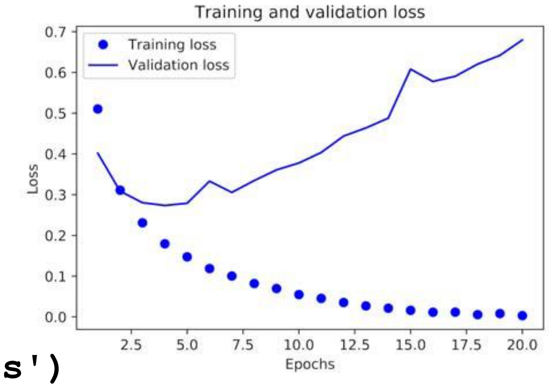
```
from keras import optimizers
from keras import losses
from keras import metrics
model.compile(optimizer=optimizers.RMSprop(lr=0.001),
        loss=losses.binary_crossentropy,
        metrics=[metrics.binary_accuracy])
```

Training the Model

```
history = model.fit(partial_x_train,
partial_y_train,
epochs=20,
batch_size=512,
validation_data=(x_val, y_val))

>>> history_dict = history.history
>>> history_dict.keys()
[u'acc', u'loss', u'val_acc', u'val_loss']
```

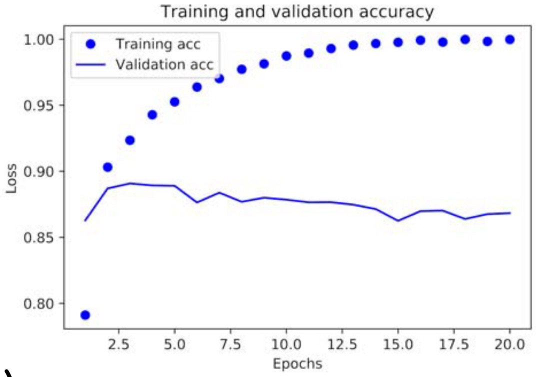
Training and Validation Loss

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Training and Validation Accuracy

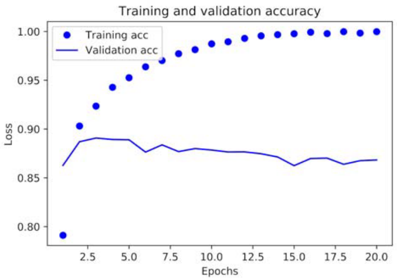
```
plt.clf()


acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
```




```
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel(‘Accuracy')
plt.legend()
plt.show()
```

Overfitting

- Classical example of over optimization
  - Best validation accuracy after 3 epochs
  - Thus, e.g. stop training early (**early stopping**)




```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
>>> results
[0.2929924130630493, 0.88327999999999995]
>>> model.predict(x_test)
array([[ 0.98006207][ 0.99758697]...,[ 0.65371346]], dtype=float32)
```

Take Away Message

- **Preprocessing** is a very important step
  - Different representations of the input
  - Lead to different architectures,
  - Leads to different results
- Multiple **dense layers** with **relu** are suitable for a variety of task
- For **binary classification** the output layer should be dense and should contain one unit with a **sigmoid activation function**
- Scalar sigmoid output + binary classification = **binary cross-entropy** as loss function
- **Rmsprop Optimizer** works well in general
- **Overfitting** needs to avoided by monitoring training progress and appropriate actions

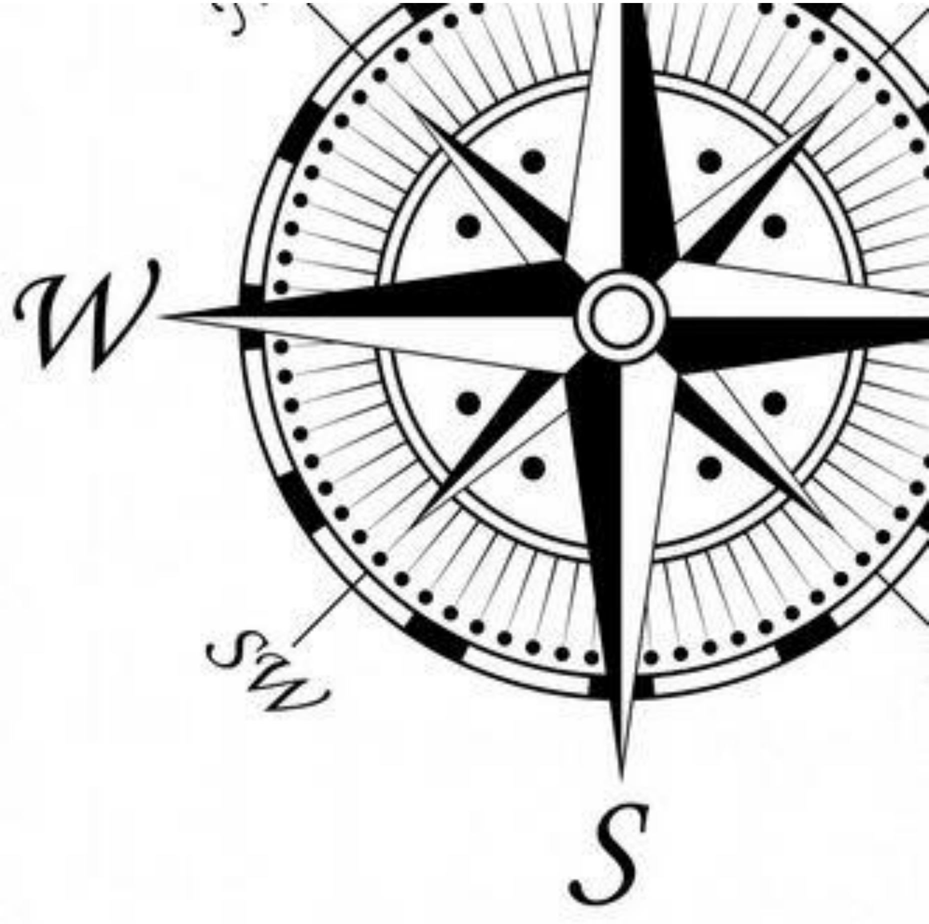
Binary Classification

- Reimplement the given example
- Experiment with different settings
  - Network architecture
    - Number of layers (+/- 1), Units (8/32)
  - Loss function (mse)
  - Activation function (tanh)

- State-of-the-art models achieve 95% accuracy

Start 10 9 8 7 6 5 4 3 2 1 End

Topics Today



1. Classification of Movie Reviews
2. **Classification of News Articles**
3. Prediction of Real Estate Prices
4. Ensembles

Preprocessing

```
from keras.datasets import reuters
(train_data, train_labels), (test_data, test_labels) =
reuters.load_data(num_words=10000)
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
        results = np.zeros((len(sequences), dimension))
        for i, sequence in enumerate(sequences):
                results[i, sequence] = 1.
        return results
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = y_train[:1000]
partial_y_train = y_train[1000:]
```

Vectorization

Encoding Multi-Class Labels

```
def to_one_hot(labels, dimension=46):
        results = np.zeros((len(labels), dimension))
        for i, label in enumerate(labels):
                results[i, label] = 1.
        return results

one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
```

Alternative I:

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

Alternative II:

The Network

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

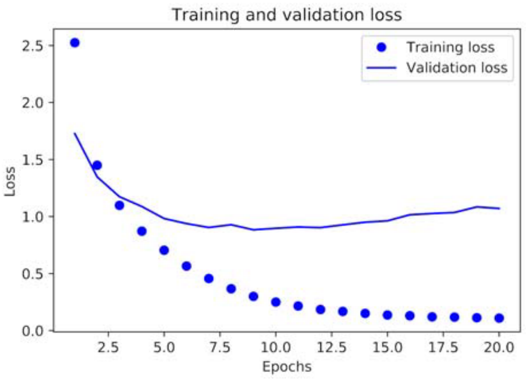
For 46 classes, 16 units wouldn‘t be enough!

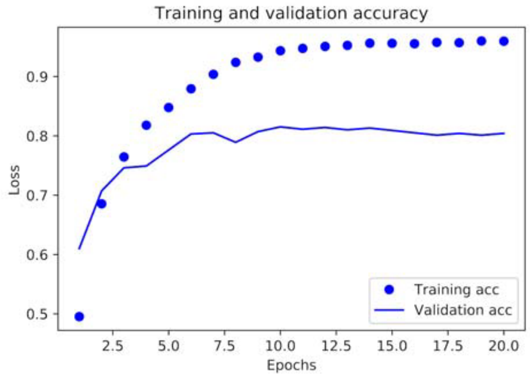
Probability distribution over all classes $\sum_c p(c|d) = 1$

Measures the distance between two probability distributions

Training the Model

```
history = model.fit(partial_x_train, partial_y_train, epochs=20,
         batch_size=512, validation_data=(x_val, y_val))
```





Results

```
model.fit(partial_x_train, partial_y_train, epochs=9, batch_size=512,
validation_data=(x_val, y_val))
results = model.evaluate(x_test, one_hot_test_labels)

>>> results
[0.9565213431445807, 0.7969723953695489]

>>> import copy
>>> test_labels_copy = copy.copy(test_labels)
>>> np.random.shuffle(test_labels_copy)
>>> hits_array = np.array(test_labels) == np.array(test_labels_copy)
>>> float(np.sum(hits_array)) / len(test_labels)
0.18655387355298308
```

Baseline: random

Alternative Encoding of Labels

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)

model.compile(optimizer='rmsprop',
loss='sparse_categorical_crossentropy', metrics=['acc'])
```

In case the category labels are integers

```
predictions = model.predict(x_test)

>>> predictions[0].shape
(46,)
>>> np.sum(predictions[0])
1.0
>>> np.argmax(predictions[0])
4
```

Take Away Message

- The output layer should have as many units as there are different **classes**.
- For single-label, multi-class problems, use the **softmax activation function**
- **Categorical cross-entropy** minimizes the distance between the true distribution of the labels and the predicted distribution of the network.
- **One-hot** encoding or using **integer**
  - Loss function has different name
- Avoid **information bottlenecks**!

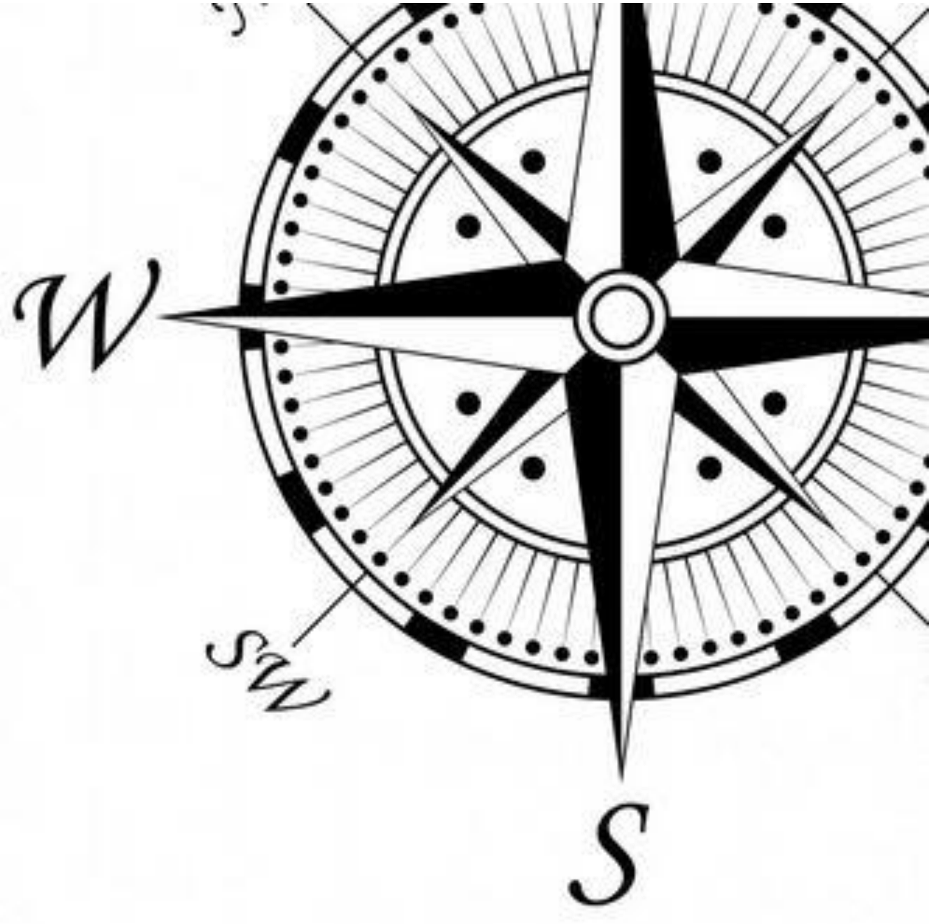
Multi-Class Classification

- Reimplement the given example with the 20 news groups dataset.
- Create an information bottleneck by assigning only 4 or 8 units to the second hidden layer.
- Experiment with different network sizes
  - layers (+/- 1)
  - units (32/128)

Start 10 9 8 7 6 5 4 3 2 1 End

Topics Today

1. Classification of Movie Reviews
2. Classification of News Articles
3. **Prediction of Real Estate Prices**
4. Ensembles



Loading the Data

```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()

>>> train_data.shape
(404, 13)
>>> test_data.shape
(102, 13)

>>> train_targets
[ 15.2, 42.3, 50. ... 19.4, 19.4, 29.1]
```

Features contain e.g. crime rate, number of rooms, highway access, etc.

Average price in 1000$

Normalizing the Data

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std
test_data -= mean
test_data /= std
```

NEVER look at the test data! Not even for preprocessing or normalization!

The Network

```
from keras import models
from keras import layers
def build_model():
        model = models.Sequential()
        model.add(layers.Dense(64, activation='relu',
input_shape=(train_data.shape[1],)))
        model.add(layers.Dense(64, activation='relu'))
        model.add(layers.Dense(1))
        model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
        return model
```

No activation function: Linear output layer

Mean Squared Error

Mean Absolute Error

K-Fold Cross-Validation

```
import numpy as np; k=4; num_val_samples=len(train_data) // k; num_epochs=100;
all_scores=[]
for i in range(k):
       print('processing fold #', i)
       val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
       val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples]
       partial_train_data = np.concatenate([train_data[:i * num_val_samples],
                     train_data[(i + 1) * num_val_samples:]], axis=0)
       partial_train_targets = np.concatenate([train_targets[:i * num_val_samples],
                     train_targets[(i + 1) * num_val_samples:]], axis=0)
       model = build_model()
       history = model.fit(partial_train_data, partial_train_targets,
                     validation_data = (val_data, val_targets),
                     epochs=num_epochs, batch_size=1, verbose=0)
       mae_history = history.history['val_mean_absolute_error']
       all_mae_histories.append(mae_history)
```

Silent mode

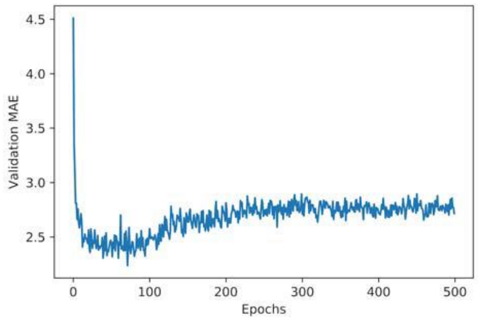
Results

- For num_epochs=100
  - MAE-values for each fold

    `[2.5882589577920, 3.12895684497191, 3.18561160512489, 3.07633426154013]`
  - Average

    `2.9947904173572462`
- Monitoring the individual folds for 500 epochs

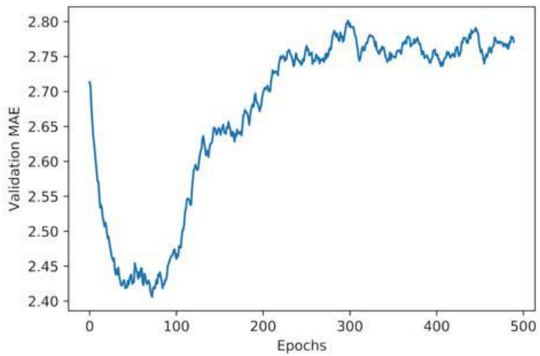
```
average_mae_history = [np.mean([x[i] for x in
       all_mae_histories]) for i in range(num_epochs)]
import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mae_history) + 1),
               average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



Better Visualization

```
def smooth_curve(points, factor=0.9):
      smoothed_points = []
      for point in points:
         if smoothed_points:
               previous = smoothed_points[-1]
               smoothed_points.append(previous * factor
                              + point * (1 - factor))
         else:
               smoothed_points.append(point)
      return smoothed_points
smooth_mae_history = smooth_curve(average_mae_history[10:])
plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```

Moving Average!



Ignore first 10 points!

Final Model

```
model = build_model()
model.fit(train_data, train_targets,epochs=80, batch_size=16, verbose=0)

test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)

>>> test_mae_score
2.5532484335057877
```

Take Away Message

- For **regression problems** we need a different loss function.
  - **MSE** (Mean Squared Error) is very popular.
- Evaluation metric for regression problems:
  - **MAE** (Mean Absolute Error)
- Features with values in different ranges need to be **normalized** independentley of each other.
- If there are only a few training samples, use **k-fold cross-validation**.
- If there are only a few training samples, the network needs to be **small**, i.e. it should have only a low number of trainable parameters.
  - One or two hidden layers the most

Regression

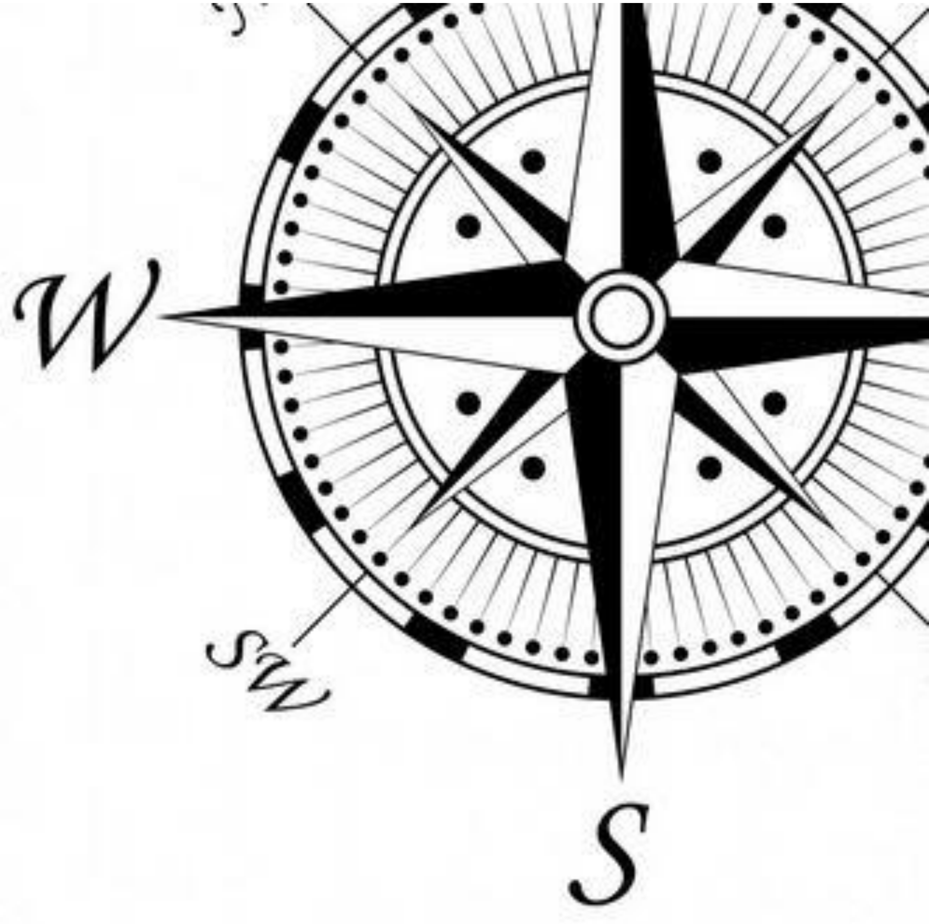
- Reimplement the example with one of the following datasets:
  - https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews
  - https://www.kaggle.com/datasets/septa97/100k-courseras-course-reviews-dataset
  - https://www.kaggle.com/datasets/andrewmvd/trip-advisor-hotel-reviews

- Experiment with different network sizes
  - layers (+/- 1)
  - units (32/128)

Start 10 9 8 7 6 5 4 3 2 1 End

Topics Today

1. Classification of Movie Reviews
2. Classification of News Articles
3. Prediction of Real Estate Prices
4. **Ensembles**



Ensembling

- An Ensemble is the combination of multiple models.
  - The predictions of different, diverse models are combined.
  - Each model „grasps“ one aspect of the training data.
    - The blind and the elephant
  - In practice (e.g. Kaggle) very successful method

- General machine learning approach
  - (Election)
  - Bagging
  - Boosting
  - Stacking

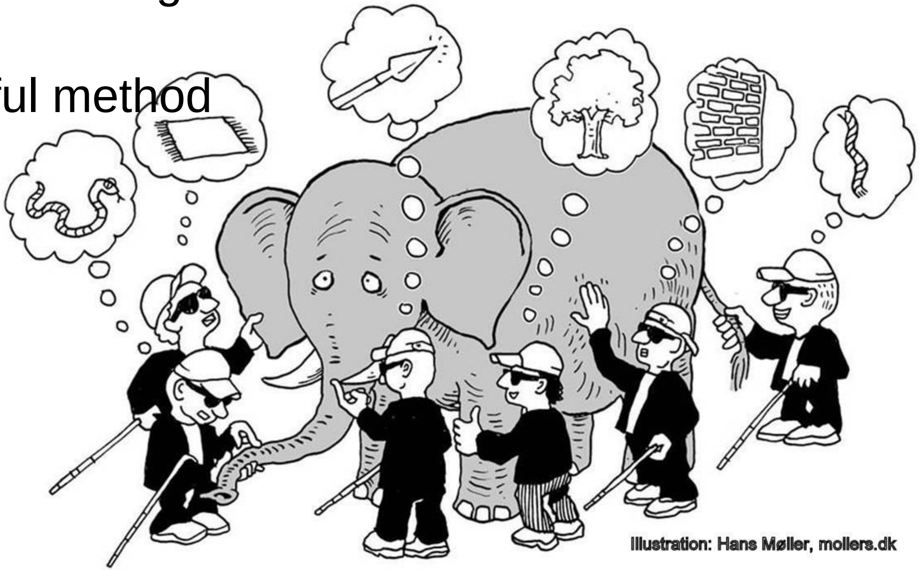


Illustration: Hans Møller, mollers.dk

In the Wild

- Better than just taking the average:
  - Learn weights using validation data!
    - E.g. with the Nelder-Mead-method (similar to steepest descent)

```
preds_a = model_a.predict(x_val)
preds_b = model_b.predict(x_val)
preds_c = model_c.predict(x_val)
preds_d = model_d.predict(x_val)
final_preds = 0.5*preds_a + 0.25*preds_b + 0.1*preds_c + 0.15*preds_d
```

- Goal: Many diverse models
  - The can have a high bias, as long as it differs from model to model.
  - The same network with different initializations does not work.
- Often promising:
  - Ensembles from decision trees
    - random forest or gradient boosting trees
  - and deep neural networks

Learning Goals for this Chapter

- Use neural nets to solve simple problems
  - Binary classificaiton
  - Multi-Class Classification
  - Regression
- Understand ensembles

- Relevant chapters:
  - P3, D8