From Keras to Production Workshop — NLP

Dr. Thomas Timmermann

codecentric AG | 19.11.2019

codecentric

Before we start...

Who we are...

Dr. Thomas Timmermann
Data Scientist

Matthias Niehoff
Head Data & AI

...and what we want to do:

Today

Neural Networks
with Keras

Tomorrow

Production-Ready
with Airflow

Our Plan for Today

Theory Morning: Foundations

- How do neural networks learn?
- How to build nets in Keras?
- Classifying digits in ~10 lines

Hands-on Afternoon: Classifying Poems

- How to feed text to neural nets?
- Convolutional neural networks
- Natural Language Processing

What is Machine Learning?

Variants of Machine Learning

supervised learning

→ labelled data
- classification
- regression

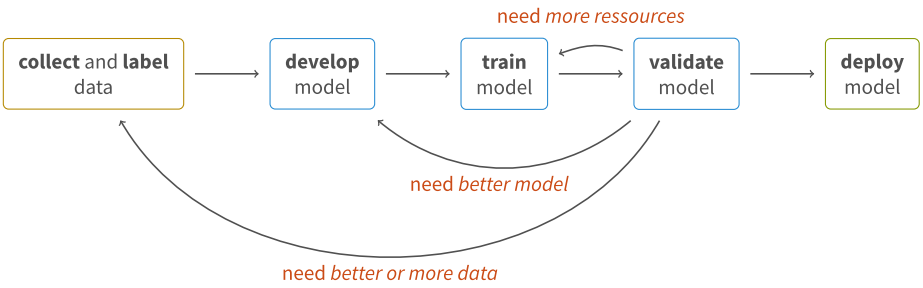
unsupervised learning

→ unlabelled data
- clustering
- dimensionality reduction
- anomaly detection

reinforcement learning

→ data generated by environment (reward and punishment)

Typical machine learning workflow

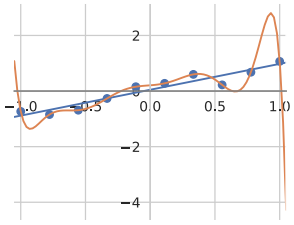


→ iterative process requires **version management** for **data science**

Golden Rules of Machine Learning

1. Better Data beats Better Algorithms

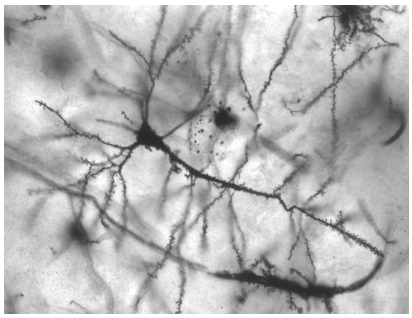
2. Separate Data for Training and Validation



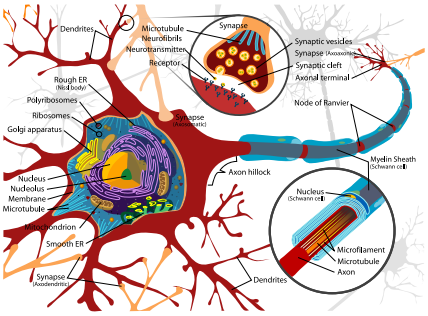
...or your models will **overfit** and **generalize poorly**

Neural Network Basics

The brain — a network of neurons



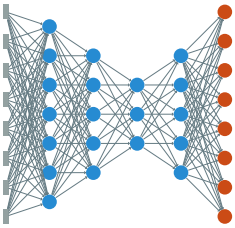
human brain
billions of neurons
(13-17.000.000.000)



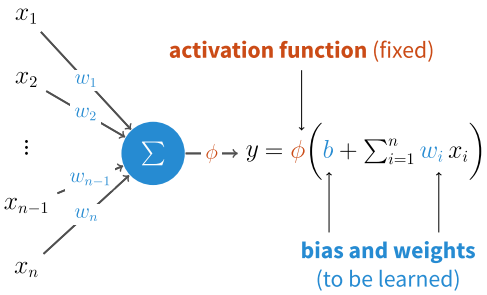
neuron
input from many dentrides
output to one axon

Artificial neural networks

nets of perceptrons



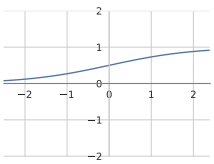
perceptrons = artificial neurons



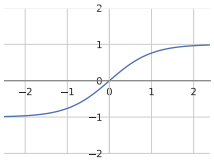
Activation functions

induce non-linearity
(compositions of linear functions are... linear)

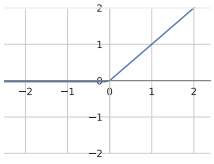
sigmoid



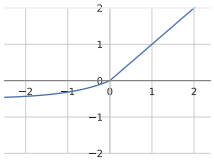
tanh



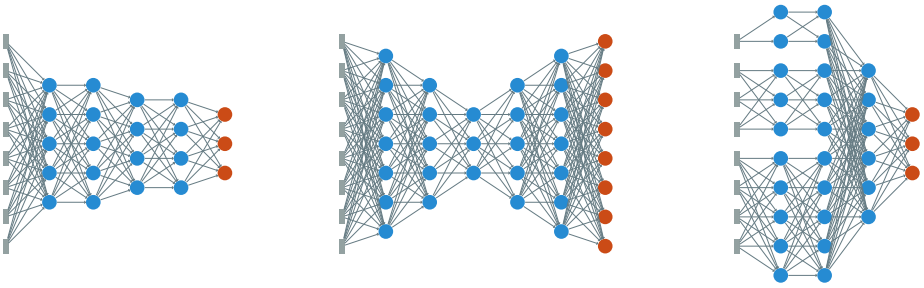
relu



selu



Now build neural networks...



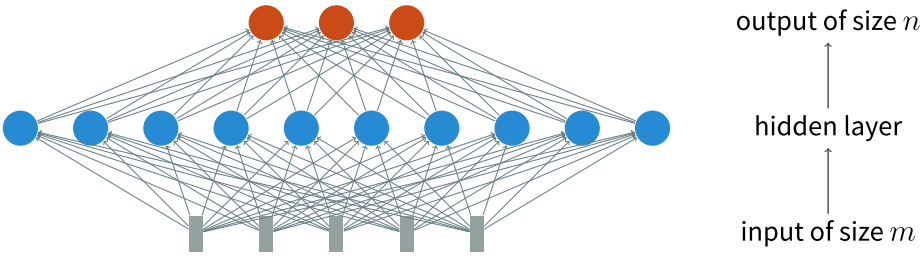
Densely connected layers or **partially connected blocks**?

Shallow or **deep**? **Wide** or **thin**? **Shrinking** or **growing**?

Perceptron layers? **Recurrent layers**? **Convolutional layers**?

Neural networks are *universal*

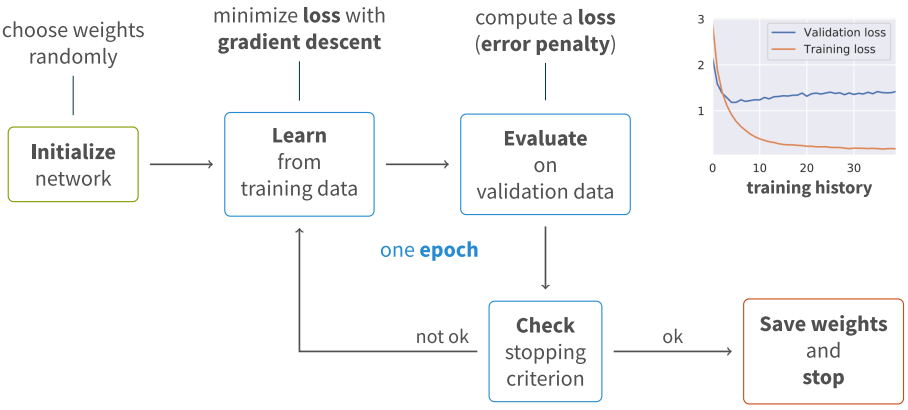
Every continuous $f\colon [0,1]^n \to \mathbb{R}^m$ can be approximated by a neural network with



- **one layer** with an **exponential number of perceptrons**
- **standard activation functions** and
- **suitable weights** that **need to be learned**

How does a Neural Network learn?

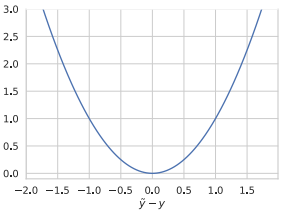
Learning in epochs



Common loss functions for regression

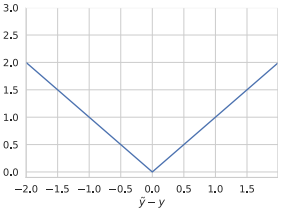
Task: Rate the prediction $\tilde{y} \in \mathbb{R}^n$ of the true value $y \in \mathbb{R}^n$

mean squared error
$\frac{1}{n}\sqrt{\sum_i (\tilde{y}_i - y_i)^2}$



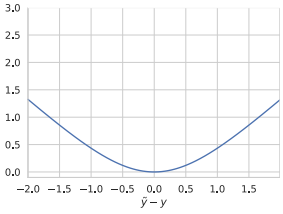
steep for large errors

mean absolute error
$\frac{1}{n}\sum_i |\tilde{y}_i - y_i|$



steep for small errors

log cosh
$\frac{1}{n}\sum_i \log\cosh(\tilde{y}_i - y_i)$



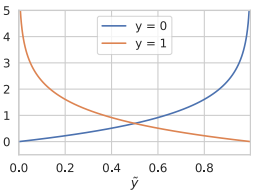
just right

Common loss functions for classification

binary classification

- true value y is either 0 or 1
- prediction $\tilde{y}$ is a probability

binary crossentropy

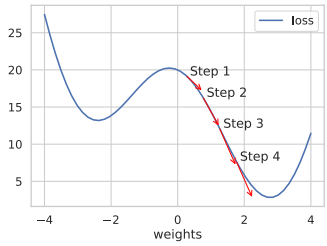


n-class classification

- $y = (0, \dots, 0, 1, 0, \dots, 0)$ indicates the true class
- $\tilde{y} = (\tilde{y}_1, \dots, \tilde{y}_n)$ is a probability distribution

categorical crossentropy
$-\sum_i y_i \log \tilde{y}_i$

Gradient descent



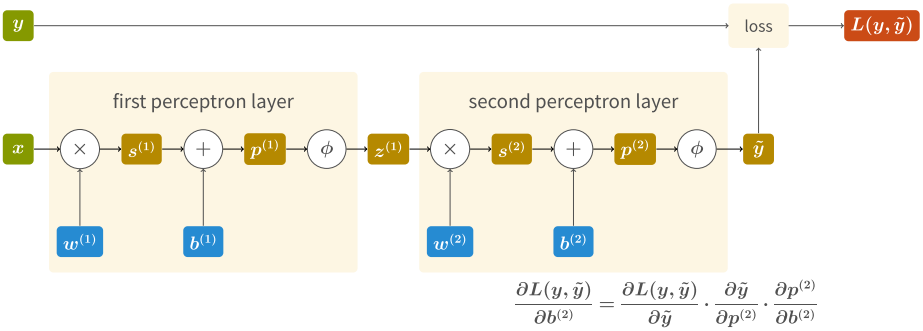
- compute **gradient** of loss L w.r.t. weights w

$$\frac{\partial L}{\partial w} = \frac{\partial \mathcal{L}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial w}$$

- find the direction Δw of **steepest descent**
- take small step into this direction (with **random step size** or advanced **optimizer**)

Computations expressed by graphs

forward pass ⇝ results **backward pass ⇝ gradients**



Gradient descent in formulas

Start with random $w^{(0)}$ and compute $w^{(1)}, w^{(2)}, \ldots$ according to

$$w^{(t+1)} = w^{(t)} - \eta^{(t)} \cdot \sum_{x,y} \frac{\partial L(y, \tilde{y}(x, w^{(t)}))}{\partial w^{(t)}}$$

- $\eta^{(t)}$ ... learning rate, usually chosen to decay
- summation over
  - one random training sample (⇒ **stochastic gradient descent**) or
  - all training samples (⇒ **batch gradient descent**) or
  - batches of training samples (⇒ **mini-batch gradient descent**)

Optimize the optimizer!

Momentum

$$\begin{cases} v^{(t+1)} = \gamma \cdot v^{(t)} + \eta^{(t)} \cdot \sum_{x,y} \frac{\partial L(y, \tilde{y}(x, w^{(t)}))}{\partial w^{(t)}} \\ w^{(t+1)} = w^{(t)} - v^{(t+1)} \end{cases} \quad (\gamma \approx 0.9)$$

move like a ball rolling downhill

Nesterov accelerated gradients

let the ball look ahead

Adagrad

- high η for rarely changed weights
- low η for often changed weights

Adadelta

same but look at recent changes

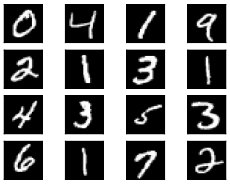
Adam

think Adadelta with momentum

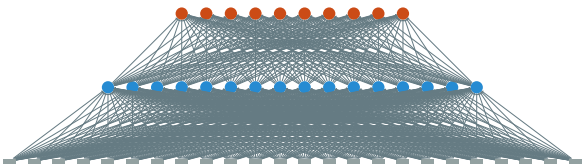
⇒ S. Ruder. An overview of gradient descent optimization algorithms. arXiv:1609.04747

Let's get started and classify digits!

Our data



Our model



And now... fire up Docker and JupyterLab!

Fire up Docker,

```
docker run -d -p 8888:8888 codecentric/from-keras-to-production-baseimage
```

update the workshop repository,

```
cd <path to your clone of workshop repo>
git pull
```

copy repository into docker container,

```
docker ps
docker cp notebooks <container id>:/keras2production
```

and visit `localhost:8888` or `192.168.99.100:8888`!

The SoftMax activation function

vector $(z_1, \dots, z_n)$ $\mapsto$ **probabilities** $(y_1, \dots, y_n)$ where $y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$

How-To: Sequential models with Keras

Build...

```
from keras import layers, models

model = models.Sequential([
            layers.Dense(128, activation='relu'),
        ], input_shape=(max_len,))
model.add(layers.Dense(10, activation='softmax'))
```

Compile...

```
model.compile(optimizer="Adam",
              loss="categorical_crossentropy")
```

Train...

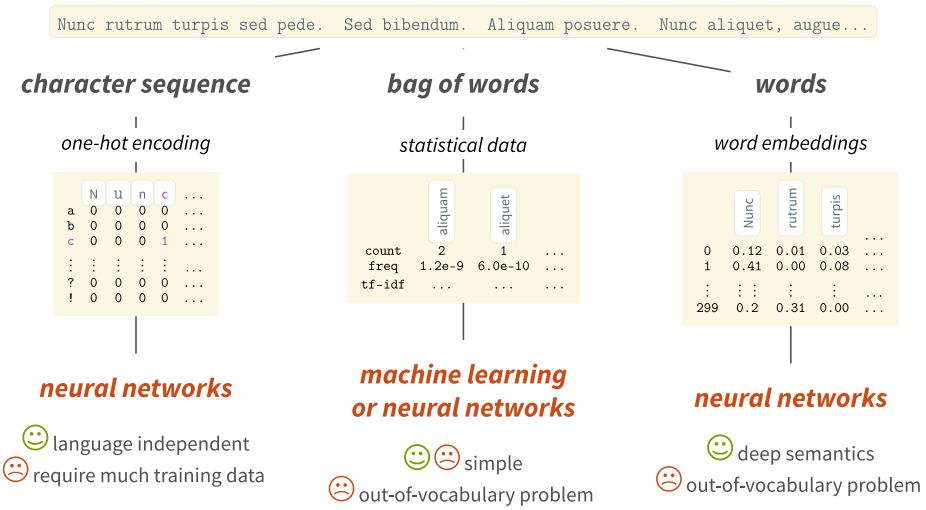
```
history = model.fit(X, y, epochs=10,
                    batch_size=32, validation_split=0.2)
```

Apply...

```
y_pred = model.predict(X_test)
```

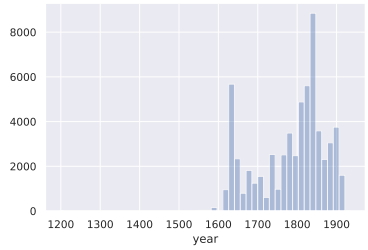
Processing Text with Neural Networks

How can we feed text to an AI?



Hands-on: classify the poem's author!

Our dataset



Our challenge

Train a net to distinguish

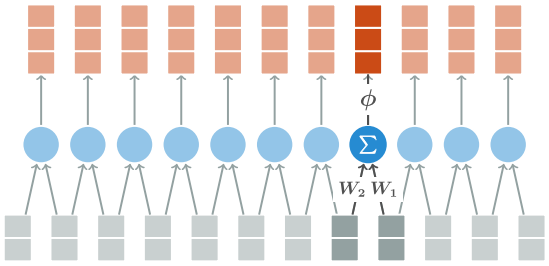
- Johann Wolfgang Goethe,
- Heinrich Heine and
- Kurt Tucholsky!

Our approaches

character-level and word-level convolutional nets

Deutscher Lyrik-Korpus
https://github.com/thomasnikolaushaider/DLK

1D-Convolutional layers



$$y_i = \phi(W_2 x_i + W_1 x_{i+1})$$

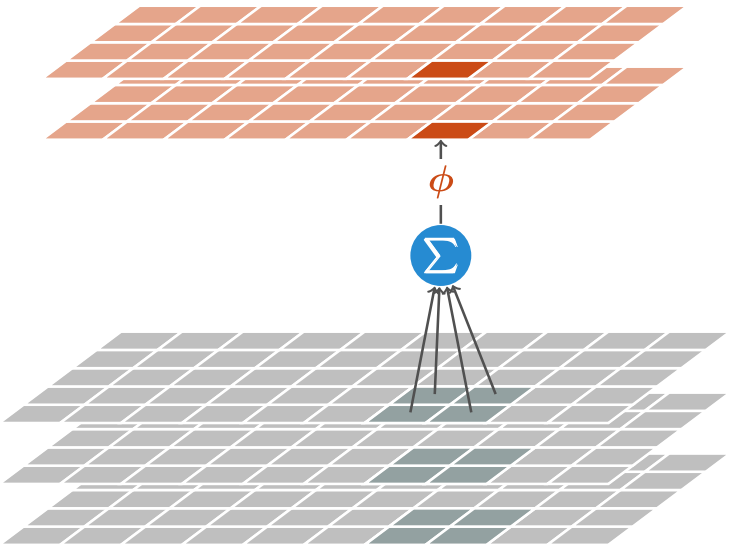
(W_1, W_2 ... kernel weights)

What sliding-window perceptron
Why extract patterns

Keras Layer Conv1D(filters, kernel_size)

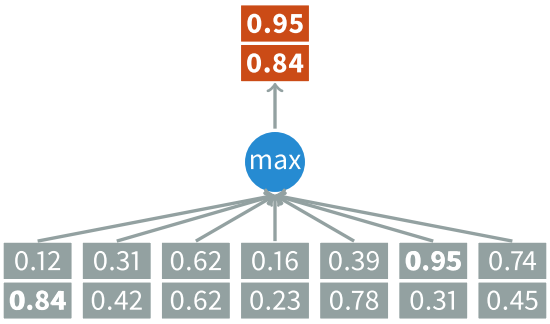
- filters : output dimension (here 3)
- kernel_size: window size (here 2)

2D-Convolutional layers



Keras layer Conv2D(filters, kernel_size, strides, padding)

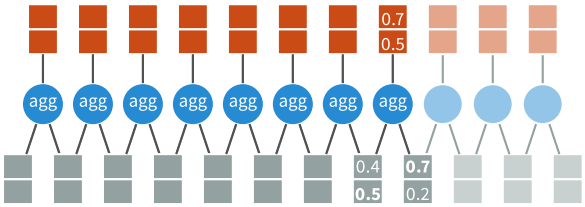
Global Pooling layers



What component-wise aggregation
Why get global features

When after convolutional layers
Keras GlobalMaxPooling1D(), ...

Pooling layers

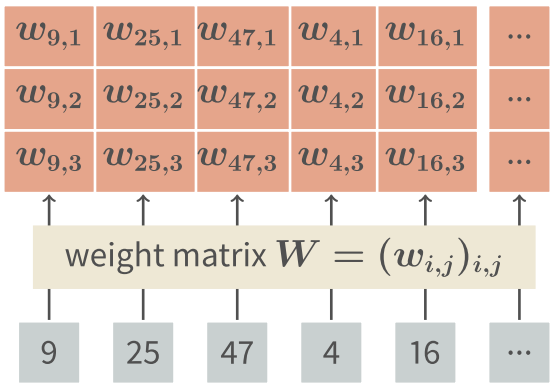


pool size=2
stride=1
agg=max

What sliding-window aggregation
Why get macroscopic features

When after convolutional layers
Keras MaxPooling1D(), AveragePooling1D(), ...

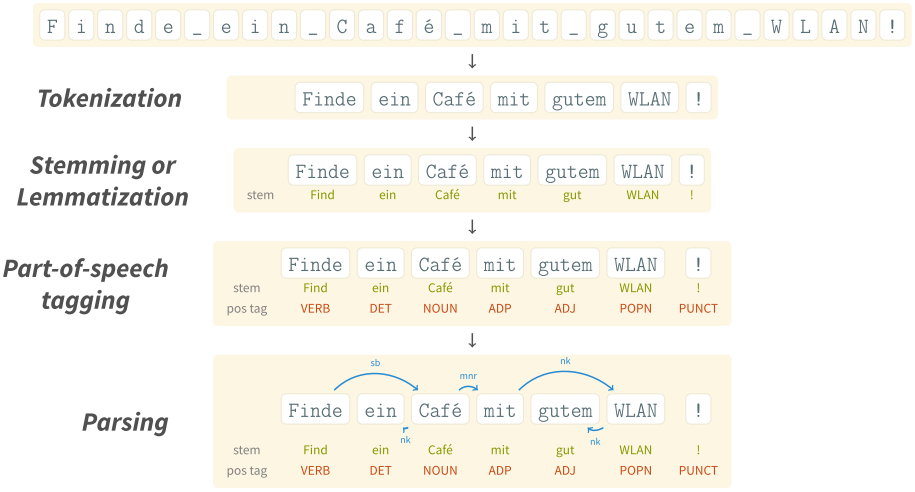
Embedding layer



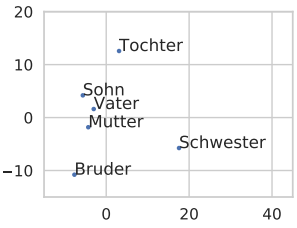
What map indices to vectors with a look-up table
Why one-hot encoding is large

When immediately after input
Keras Embedding(input_dim, output_dim)

Preprocessing steps



Word embeddings



What map words to vectors (typical dim 96-300)
Idea train encoder-decoder-net to fill gaps in phrases

Algorithms
- word2vec '13
- gloVe '14
- using BERT '19

How-To: Functional models with Keras

Use `Input` to define an input tensor,

```
inputs = layers.Input(shape=(max_len,))
```

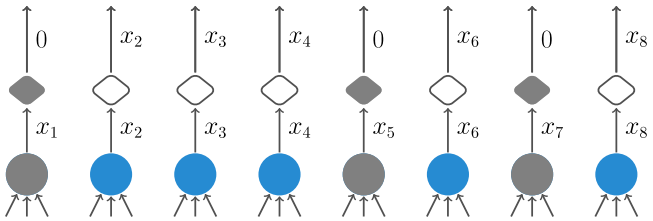
apply `Layer` instance to tensor(s) to get an output tensor,

```
conv1 = layers.Conv1D(128, kernel_size=3, activation='relu')
res1 = GlobalMaxPooling()(conv1(inputs))
conv2 = layers.Conv1D(128, kernel_size=7, activation='relu')
res2 = GlobalMaxPooling()(conv2(inputs))
outputs = Dense(3, activation='softmax')(Concatenate([res1, res2]))
```

and construct a `Model` declaring input and output tensors

```
model = models.Model(inputs=inputs, outputs=outputs)
```

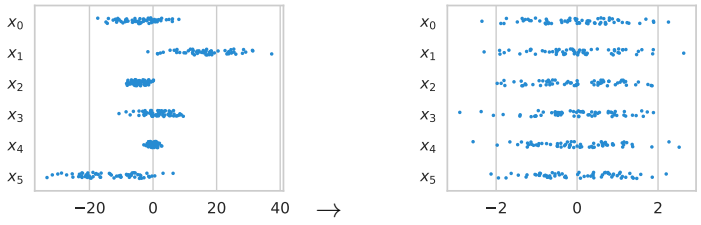
Dropout Layer



What randomly drop input components
Why regularization

When after dense layers
Keras Layer Dropout(rate=0.5)

Batch Normalization Layer



What for each batch, rescale input components (mean 0, variance 1)
Why regularization

When after convolutional layers and before activation
Keras Layer BatchNormalization()

Statistics for bags of words

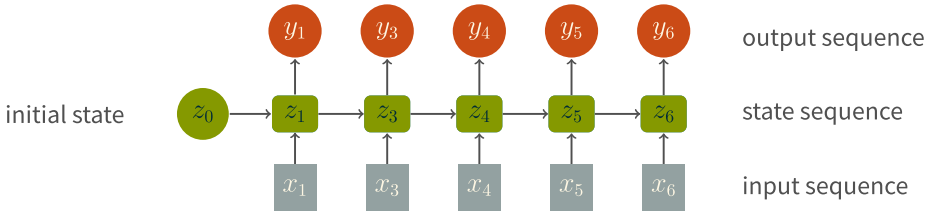
- **Occurence**: does word w occur in document D?
- **Count** $\#(w, D)$: how often does w occur in document D?
- **Term Frequency**: $\text{tf}(w, D) = \dfrac{\#(w, D)}{\max_v \#(v, D)}$

When is word w *characteristic* for document D?

- high **Inverse Document Frequency**: $\text{idf}(w) = \log \frac{N}{N_w}$
  (N ... number of documents, N_w ... number of documents with w)

$\rightsquigarrow$ **high TF-IDF-measure:** $\text{tf}(w, D) \cdot \text{idf}(w)$

Recurrent layers



What sliding stateful meta-perceptron
Why / Why not natural for sequence processing, but expensive and difficult to train
Keras Layers LSTM(units), GRU(units), CudnnLSTM(units), CudnnGRU(units)

What next?

The `Keras` book

Francois Chollet. Deep Learning with Python. Manning 2017

Transformer - THE state-of-the-art NLP model

Attention Is All You Need. arXiv:1706.03762

`BERT` - pre-trained `Tranformer` models

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805

`HuggingFace` - a `Transformer` toolkit