Natural Language Processing
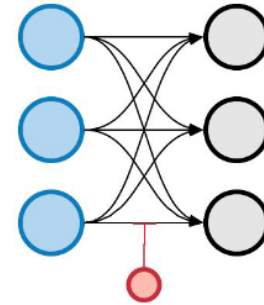
Micha Birklbauer

Contents

- Artificial Neural Networks

- Convolutional Neural Networks

- Recurrent Neural Networks

- Transformers

- Generative Models

- Sentiment Analysis

- Image Captioning

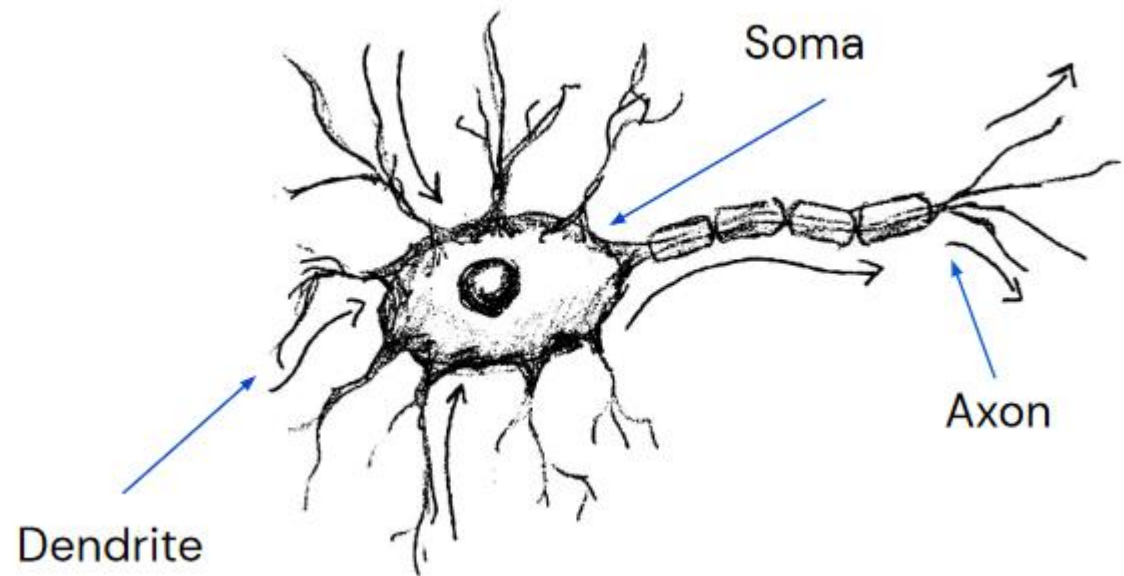- Real-Life Example

Artificial Neural Networks

$$h(\mathbf{x}, \mathbf{w}, b) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

$$f_{\text{linear}}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Real Neurons

- *Dendrites:* Connections to other neurons or cells forwarding signals

- *Soma:* Signal processing

- *Axon:* Outgoing signal

Artificial Neurons

- Sum of inputs times their weights plus bias

- Depending on the activation function you take another aggregation instead of the sum

- **Artificial neurons are NOT a model of real neurons, they just capture some simplistic properties of the biological model**
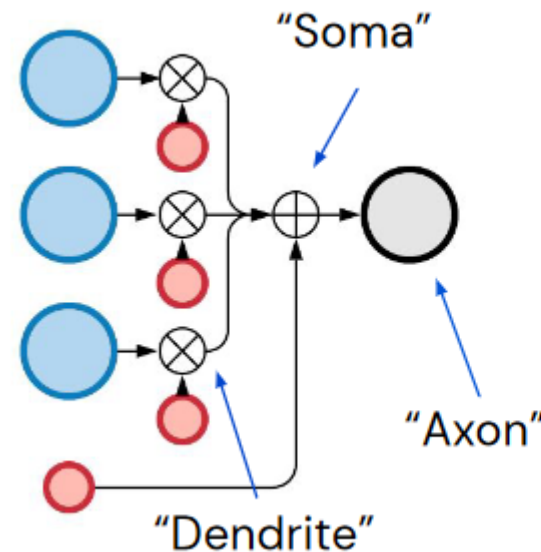
$$\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i + b$$

Linear Layer

- Every input unit (neuron) is connected to every output unit

- For linear (identity) activation function (*f(x) = x*) our model simplifies to Wx + b

- W: Matrix of weights

- x: input vector
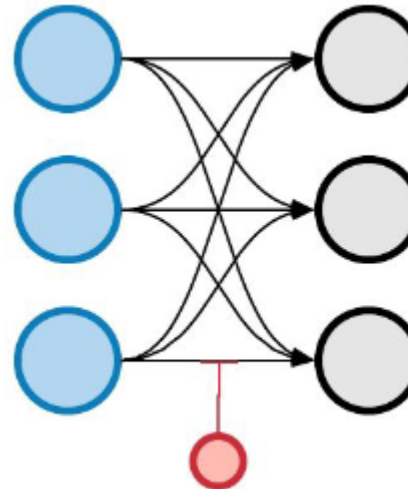
$$h(\mathbf{x}, \mathbf{w}, b) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

$$f_{\text{linear}}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$
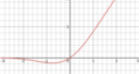
Linear Layer

- For any other (non-linear) activation function the activation function is applied point-wise to $Wx + b$

- Matrix multiplications can be done very efficient

- Highly optimized hardware (GPU/TPU)

Activiation Functions

Table of activation functions

The following table compares the properties of several activation functions that are functions of one fold x from the previous layer or layers:

Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \dfrac{1}{1+e^{-x}}$	$g(x)(1 - g(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0, x) = x \mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[5]		$\dfrac{1}{2}x\left(1 + \text{erf}\left(\dfrac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[9]		$\ln(1 + e^x)$	$\dfrac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$

ect…

Learning

- W (and b) are adjusted using gradient descent (Jacobian)

- Converges to local minimum

- Works for any „smooth enough" function

$$f(\mathbf{x})$$ Forward pass

$$\mathbf{J_x}f(\mathbf{x})$$ Backward pass

Artificial Neural Networks in NLP

- Classic feed-forward neural networks can be used like any other classification algorithm for any text classification task

- Works with any feature representation (Binary, Count, TF-IDF, 2Vec)

- Eager-learning: Predictions are very, very fast (learning not so much)

- Can approximate any decision boundary arbitrary well…

- …BUT this very easily leads to overfitting (more likely than basic classifiers like naiveBayes or kNN)

- A lot more hyperparameters to tune compared to other classifiers

Convolutional Neural Networks
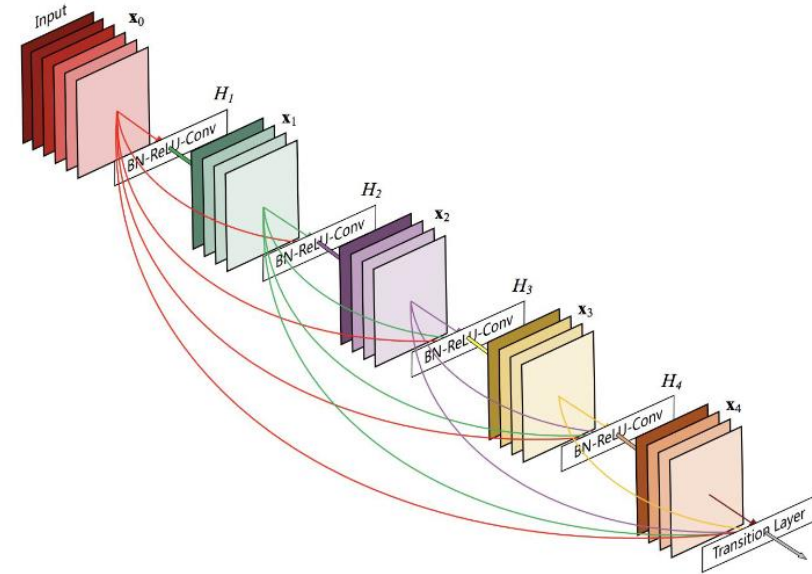
Convolution

- Instead of fully connected input units we use locally connected units

- These 3 x 3 receptive fields are called kernels and summarize the features in the 3 x 3 field creating a feature map (channel)

- By applying different kernels we get different channels

fully-connected unit

$$y = \sum_{i \in \text{image}} \mathbf{w}_i \mathbf{x}_i + b$$

$$y = \sum_{i \in 3 \times 3} \mathbf{w}_i \mathbf{x}_i + b$$

locally-connected units
3×3 receptive field
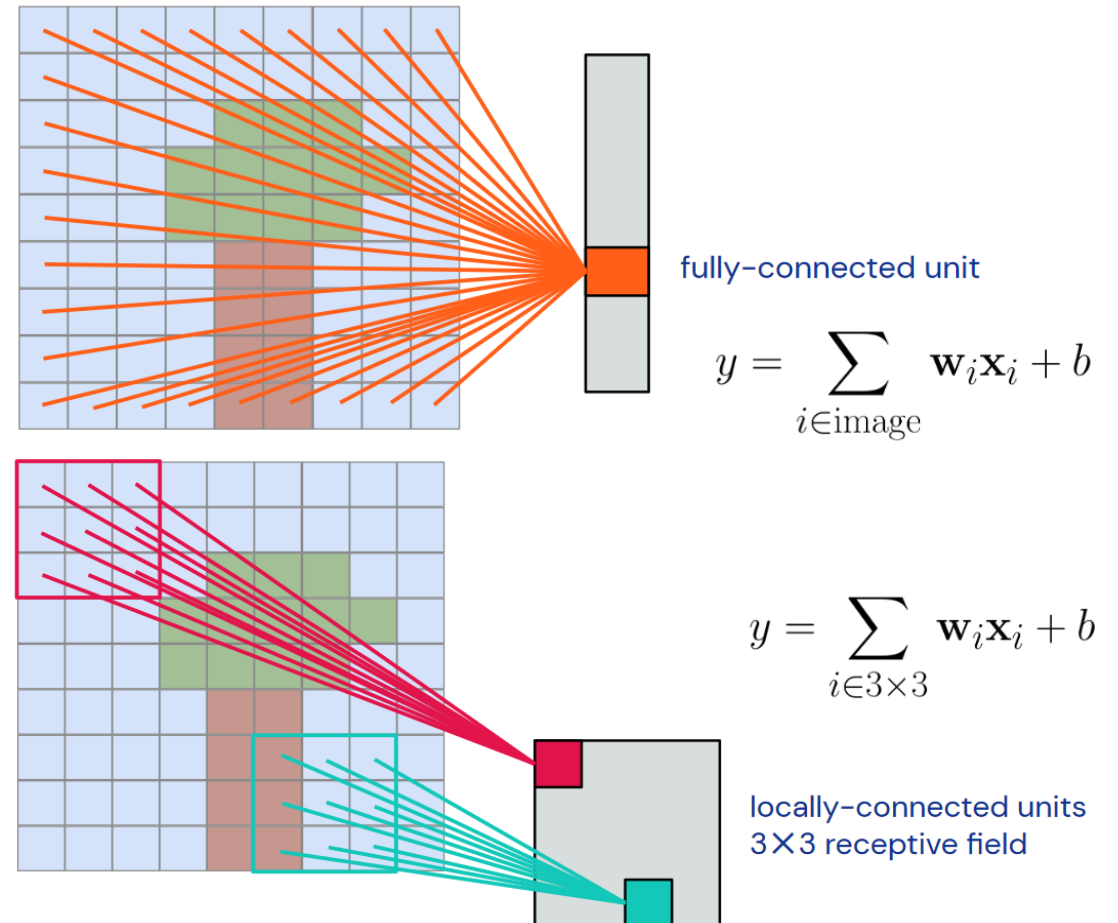
Convolution

- The kernel slides across the image and produces an output value at each position

- We convolve multiple kernels and obtain multiple feature maps or channels
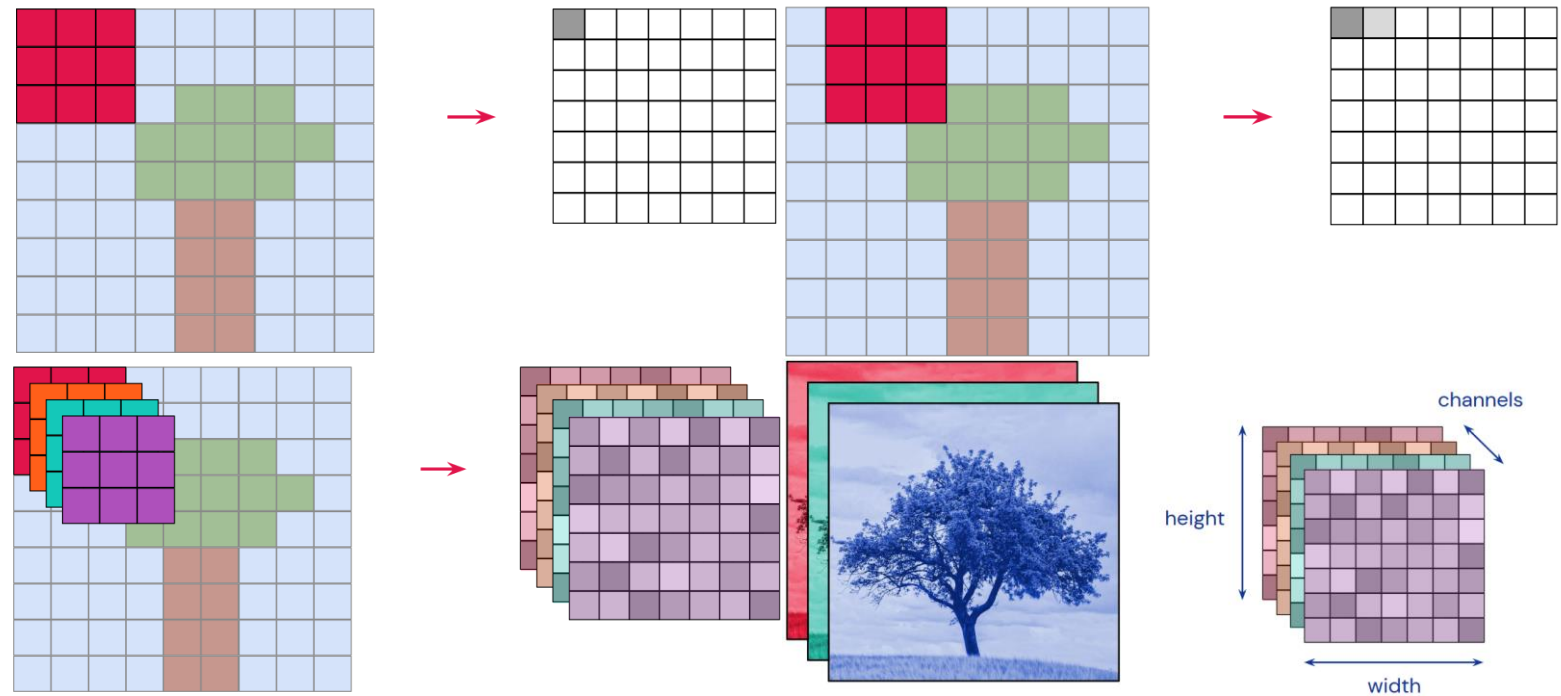
Pooling

- Compute mean or max over small windows to reduce dimensionality
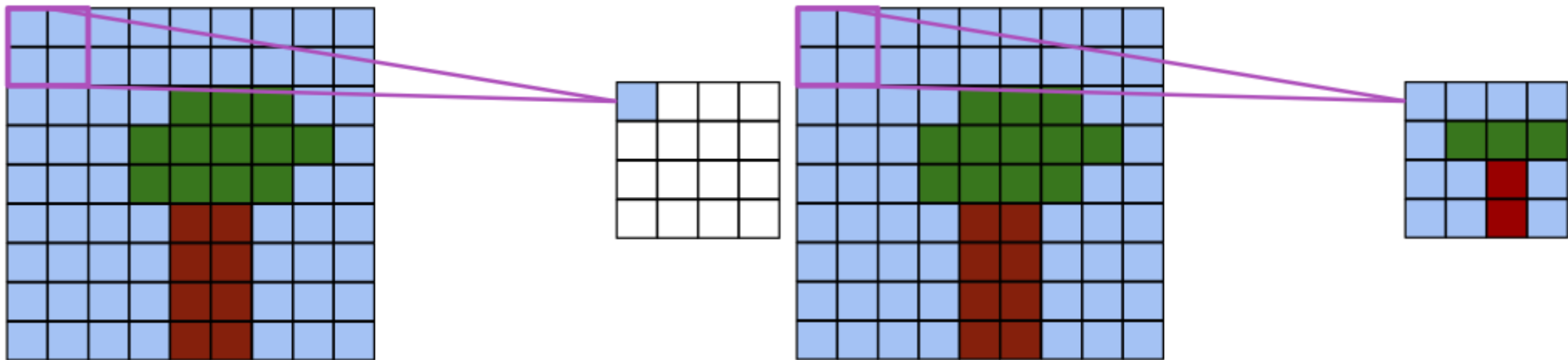
Example of a deep CNN (AlexNet)
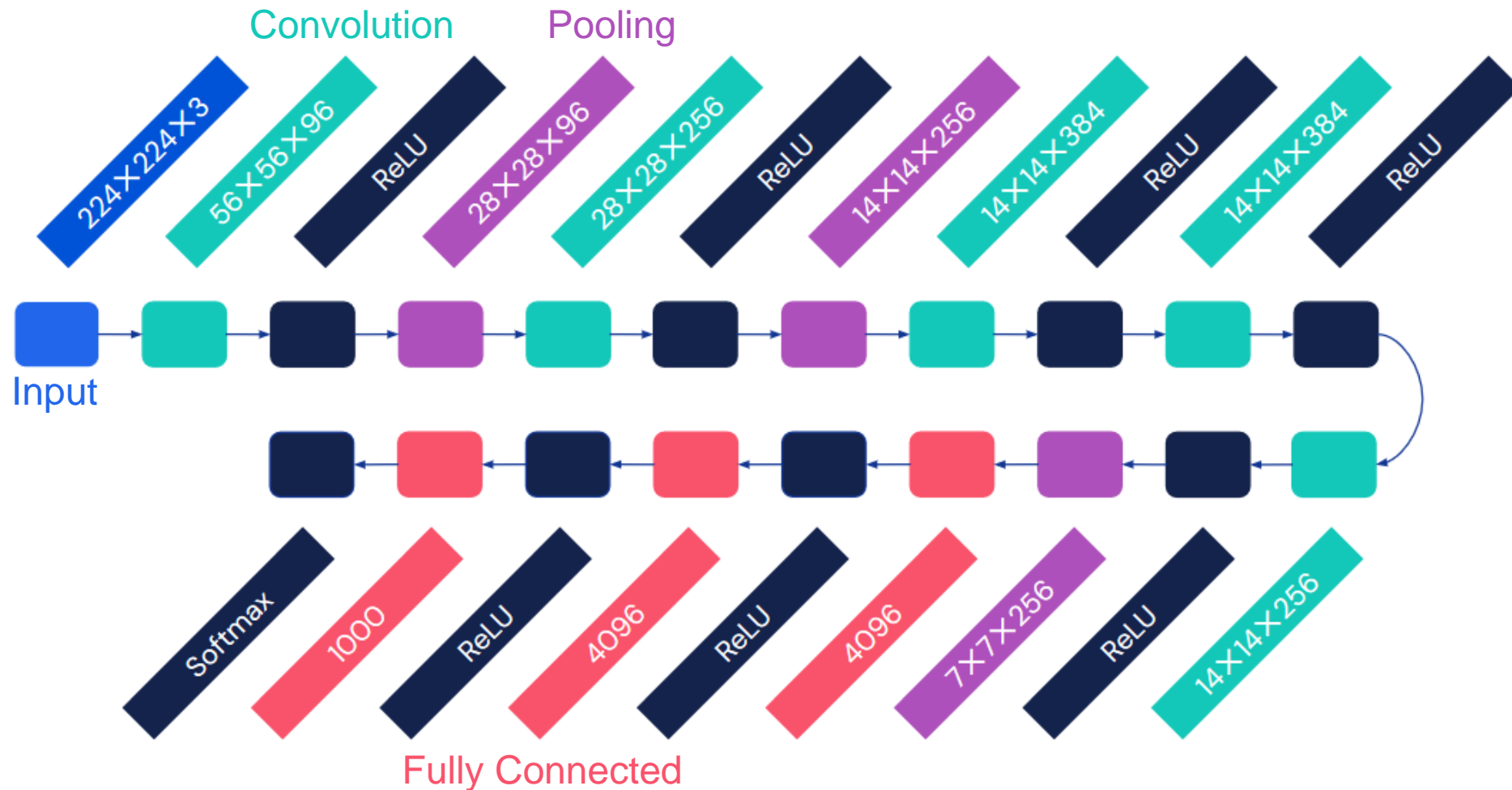
CNNs for NLP

- Similar to classic feed-forward neural networks mostly used for classification tasks

- Instead of a 2D (3D) image we have a 1D (2D) sequence of tokens, therefore 1D convolutional layer

- Captures some of the sequence aspects of texts e.g. words appearing together

- Same pros and cons of a classic ANN

Minimal Keras NLP CNN Example

```python
from tensorflow.keras.models import *
from tensorflow.keras.layers import *

model = Sequential()
# max_length = max length of your tokenized texts, embedding_dimensions = word2vec dimensions (usually 300)
model.add(Conv1D(nr_of_filters, kernel_size, padding = padding, activation = "relu", input_shape = (max_length, embedding_dimensions)))
model.add(GlobalMaxPooling1D())
model.add(Dense(nr_hidden_units))
model.add(Dropout(0.2))
model.add(Activation("relu"))
model.add(Dense(1))
model.add(Activation("sigmoid"))

# callbacks
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
callbacks = []
callbacks.append(EarlyStopping(monitor = "val_loss", patience = 15, verbose = 0, mode = "auto"))
callbacks.append(ModelCheckpoint(model_name + "_check_.h5", save_best_only = True, monitor = "val_loss", mode = "min"))

# binary classification
model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ["accuracy"])
model.fit(X_train, y_train, batch_size = batch_size, epochs = epochs , validation_data = (X_val, y_val), callbacks = callbacks, verbose = 1)
```
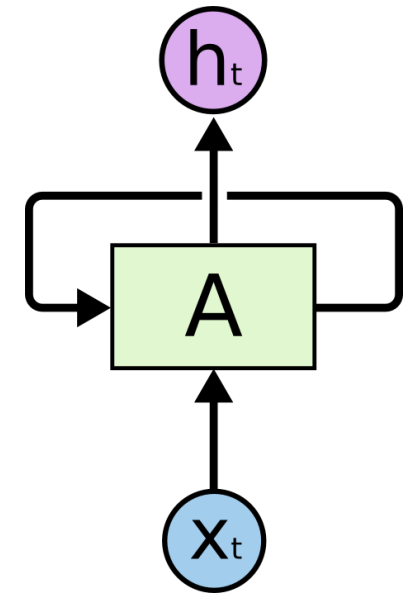
Recurrent Neural Networks

Texts are Sequences

- Texts are sequences of tokens/words

- Elements in the sequence can be repeated

- The order of elements matters

- Elements may refer to other (earlier) elements in the sequence

- Sequences are of variable length

- None of the previously discussed models reflect these properties very well

Recurrent Neural Networks

- Persistent state variable **h** stores information from the context observed so far

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$
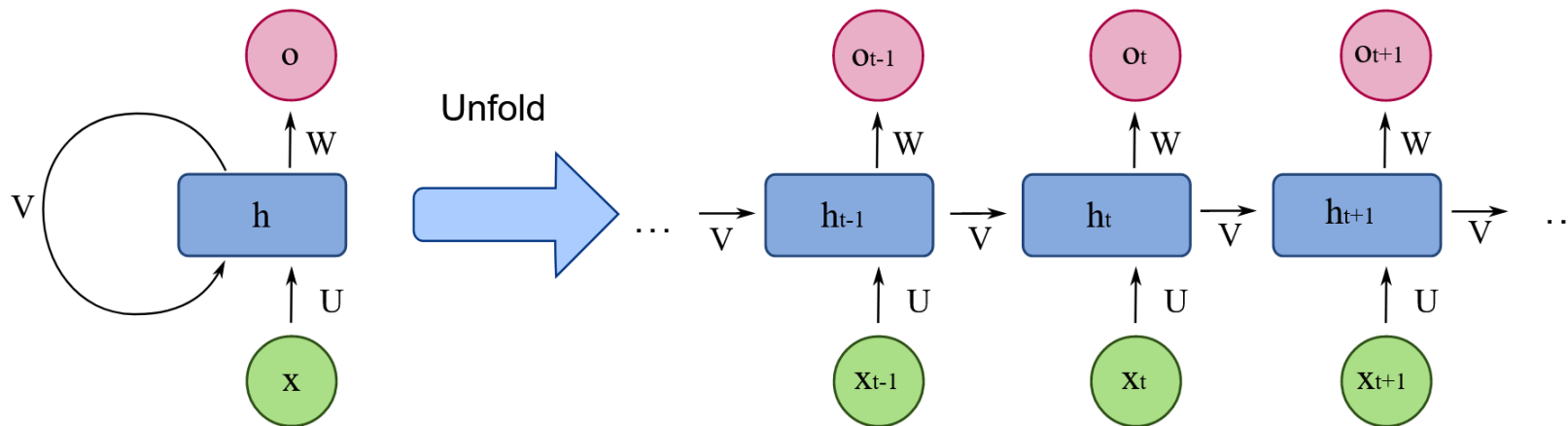
Recurrent Neural Networks

- Many variations of recurrent networks exist additionally to the classic RNN:
    - Long Short-Term Memory (LSTM) networks
    - Bi-directional LSTM networks
    - Gated Recurrent Unit (GRU) networks
    - Bi-directional GRU networks
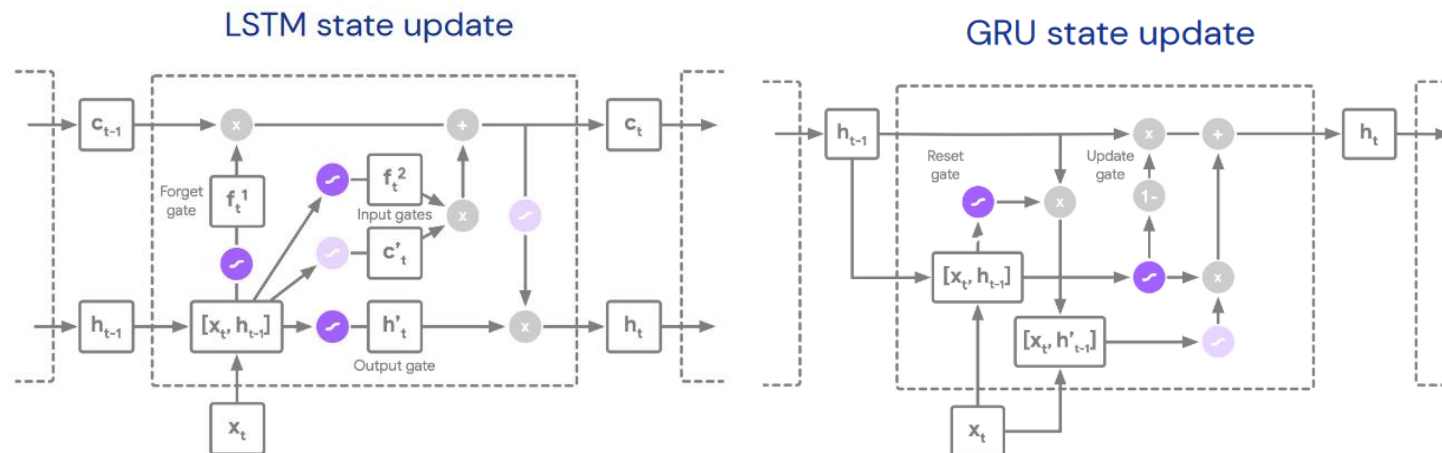
RNNs for NLP

- Captures the sequential aspect of texts very well

- Basically the standard if you are not using transformers

- Used for classfication and generative tasks

- Again same pros and cons of classic ANNs

# Minimal Keras NLP RNN Example

```python
from tensorflow.keras.models import *
from tensorflow.keras.layers import *

model = Sequential()
# max_length = max length of your tokenized texts, embedding_dimensions = word2vec dimensions (usually 300)
model.add(Input(shape = (max_length, embedding_dimensions)))
model.add(SpatialDropout1D(rate = 0.1))
model.add(Bidirectional(GRU(256), merge_mode = "ave"))
model.add(Dropout(0.1))
model.add(Dense(256, activation = "relu"))
model.add(Dense(1))
model.add(Activation("sigmoid"))

# callbacks
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
callbacks = []
callbacks.append(EarlyStopping(monitor = "val_loss", patience = 15, verbose = 0, mode = "auto"))
callbacks.append(ModelCheckpoint(model_name + "_check_.h5", save_best_only = True, monitor = "val_loss", mode = "min"))

# binary classification
model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ["accuracy"])
model.fit(X_train, y_train, batch_size = batch_size, epochs = epochs , validation_data = (X_val, y_val), callbacks = callbacks, verbose = 1)
```
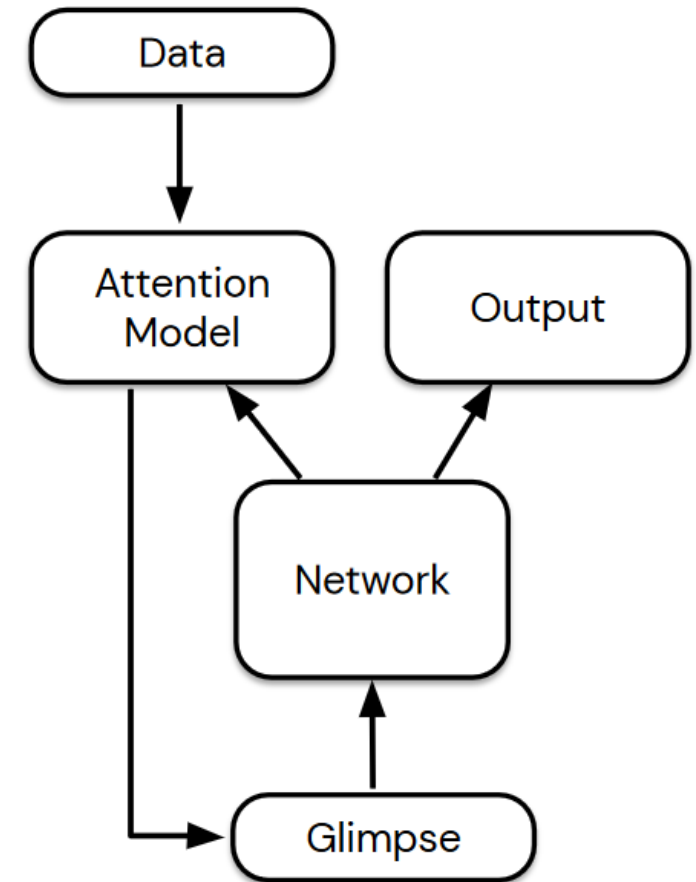
Transformers
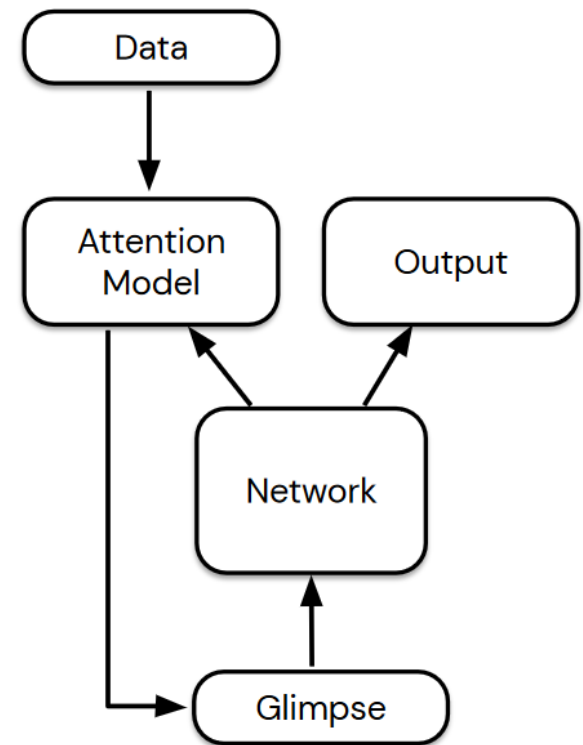
Attention

- The ability to focus on one thing and ignore others has a vital role in guiding cognition

- Deep neural networks naturally learn a form of implicit attention where they respond more strongly to some parts of the data than others

- We can add explicit attention by having a seperate attention model

Neural Attention Models

- The network produces and extra output vector used to parameterize an attention model

- The attention model then operates on some data (image, audio sample, text to be translated...) to create a fixed-size "glimpse" vector that is passed to the network as input at the next time step

- The complete system is recurrent, even if the network isn't

Transformers

- Transformer networks take attention to its logical extreme: get rid of everything else (recurrent state, convolutions, external memory) and just use attention to repeatedly transform a complete sequence

- State-of-the-Art for most NLP tasks like next-sentence prediction, question answering, reading comprehension, sentiment analysis, paraphrasing, machine translation, document summarization, document generation, named entity recognition

- Mostly using pre-trained transformers like GPT-3 (OpenAI) or BERT (Google) instead of training it yourself (fine-tuning possible)

Generative Models
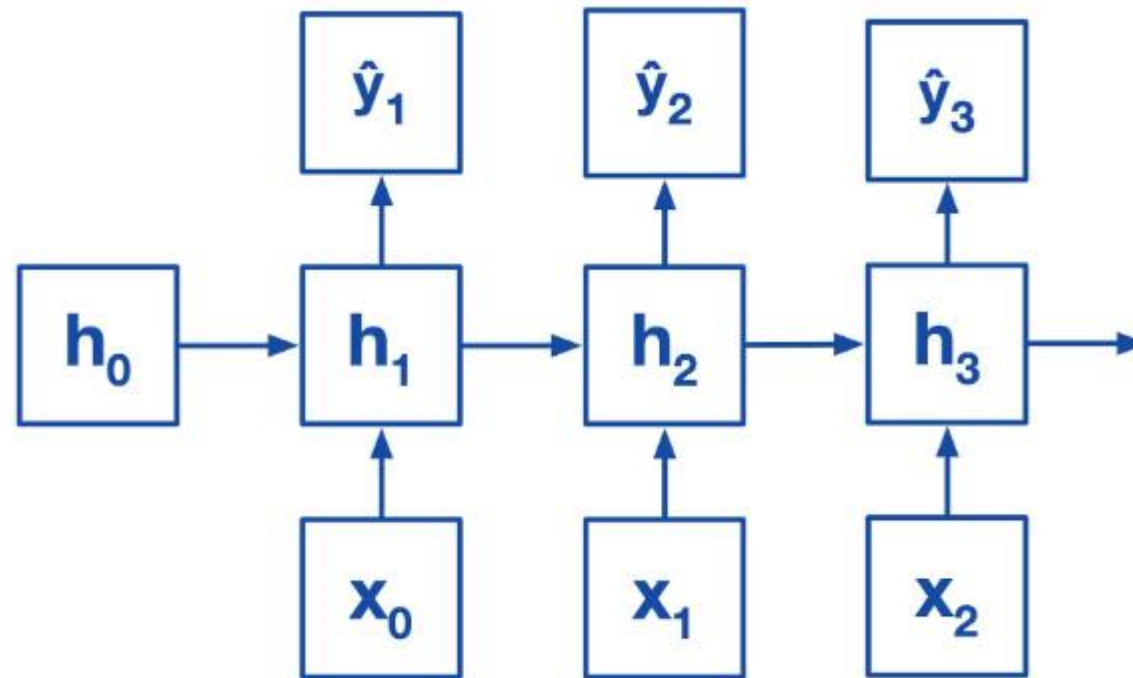
Sequence to Sequence Prediction
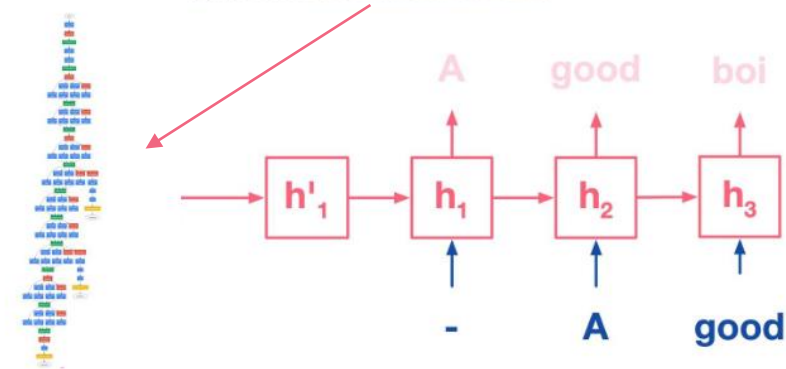
Sequence to Sequence Models

- (Hidden-)States (h_0) in RNNs usually initialized with zeros…

Sequence to Sequence Models

- …BUT we could also initialize them with the state information of another net
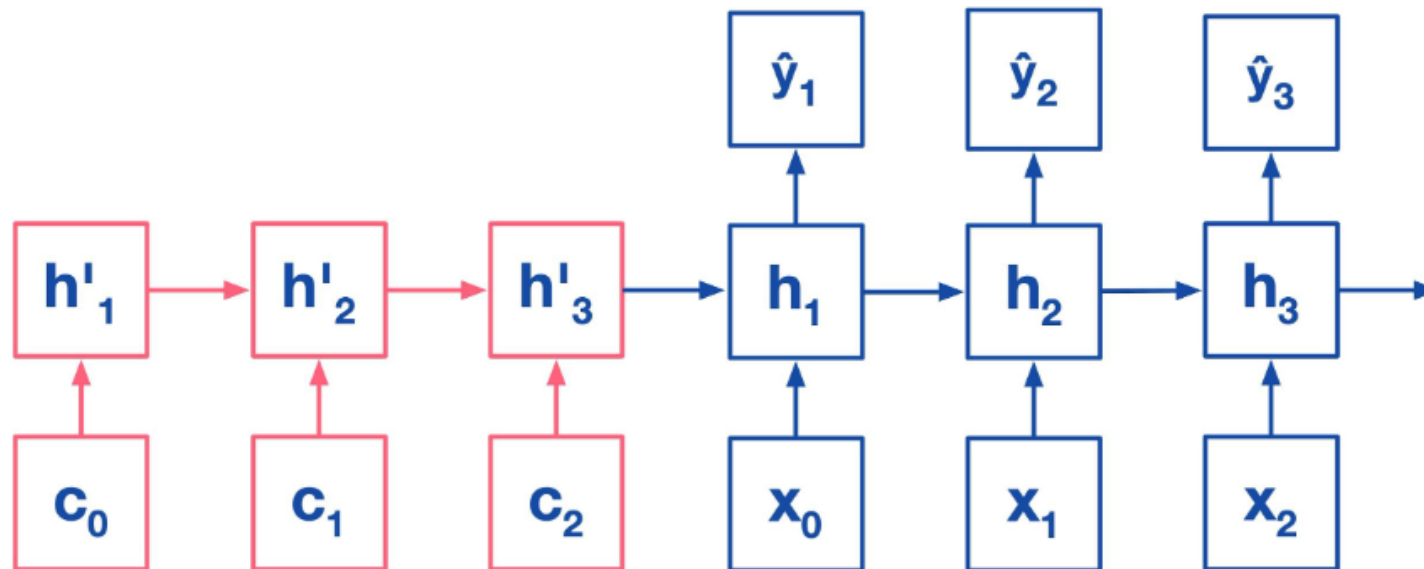
Sequence to Sequence Models

- For example, two coupled (english and japanese) nets for machine translation:

Random word as initial input
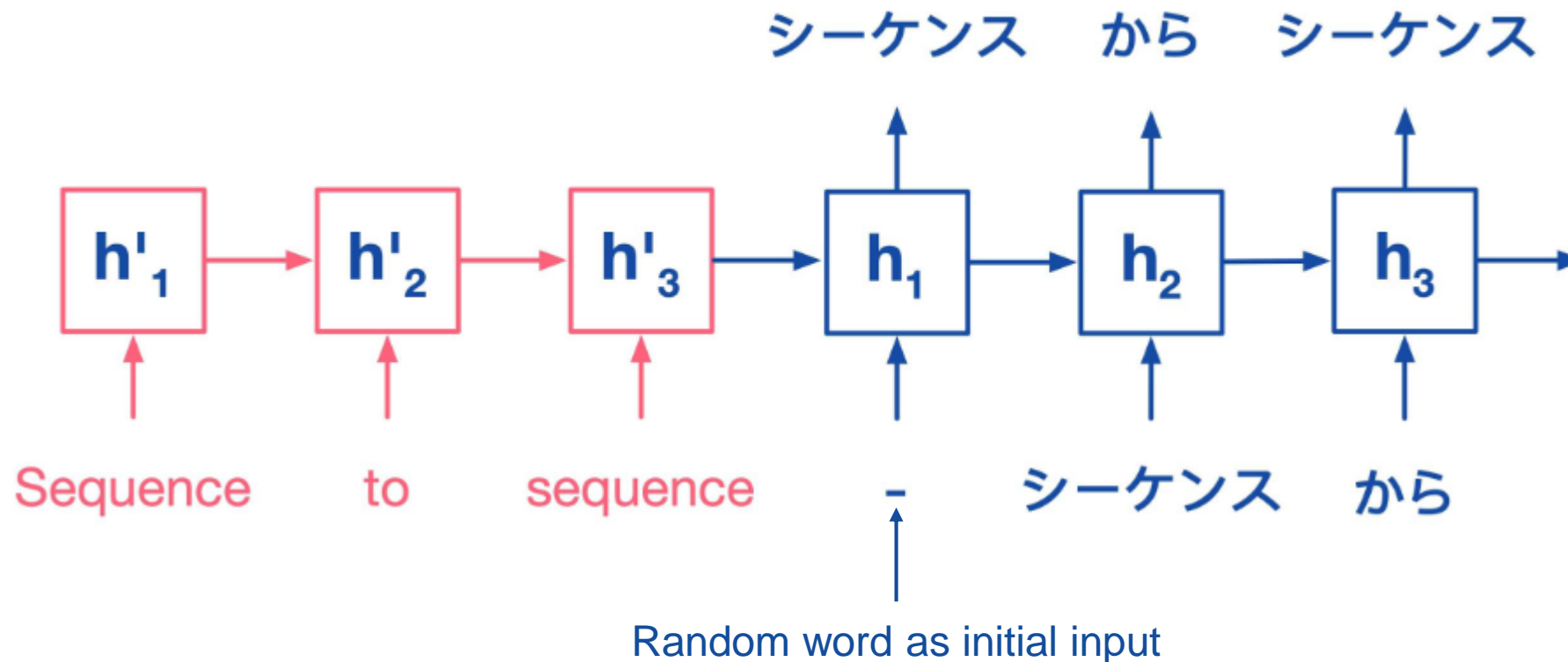
Seq2Seq Applications

- Machine translation

- Image captioning

- Video captioning

- Video generation

- Dialogue

- …

Sentiment Analysis

Sentiment Analysis

😃
My experience so far has been fantastic!

POSITIVE

😐
The product is ok I guess

NEUTRAL

😡
Your support team is useless

NEGATIVE

MonkeyLearn

Source: https://monkeylearn.com/

Sentiment Analysis

- Why sentiment analysis is important (according to Microsoft):
  - **Social and brand monitoring:** Analyzing real-time customer interactions and comments on your social channels about your brand, product, and business can offer insights into how your customers feel about all three. Companies can also use sentiment analysis of previous products as a measure for launching new products, advertising campaigns, or breaking news about your business.
  - **Customer service:** Your customer service team probably automatically sorts customer issues into urgent and not urgent. Sentiment analysis adds another layer by analyzing the language and severity of the problem in chat or email, spotlighting particularly frustrated customers for faster mediation.
  - **Customer feedback:** In line with social monitoring, you hear directly from the customer how negatively or positively they perceive a product or brand to be. Tracking keywords related to direct customer feedback shared on social media profiles, during online chats with your teams, or through other touchpoints provides an overall measurement of the success of your product, campaign, or solution.
  - **Crisis prevention:** To monitor media publishing, sentiment analysis tools can collect mentions of predefined keywords in real time. Your public relations or customer success teams can use this information to inform their responses to negative posts, possibly shortening – or even averting – a social media crisis before it can pick up speed.
  - **Market research:** It's not just enough to know how your customers feel; you need to know why. Understanding why, or why not, customers respond the way you intended is key to planning your next move – whether through marketing, sales, or direct and personalized service responses.

Sentiment Analysis

- Sentiment Analysis is basically a classification task

- Hands-on in the following exercise:

Exercise 5:

https://github.com/michabirklbauer/hgb_dse_text_mining/blob/master/Sentiment.ipynb

https://colab.research.google.com/github/michabirklbauer/hgb_dse_text_mining/blob/master/Sentiment.ipynb

Image Captioning

Human: Someone is using a small grill to melt his sandwich.

Best Model: A person is cooking some food on a grill.

Image Captioning

- The process of adding descriptions to pictures

- We are going to look at the official Keras guide to get a basic understanding of the implementation:
  https://keras.io/examples/vision/image_captioning/
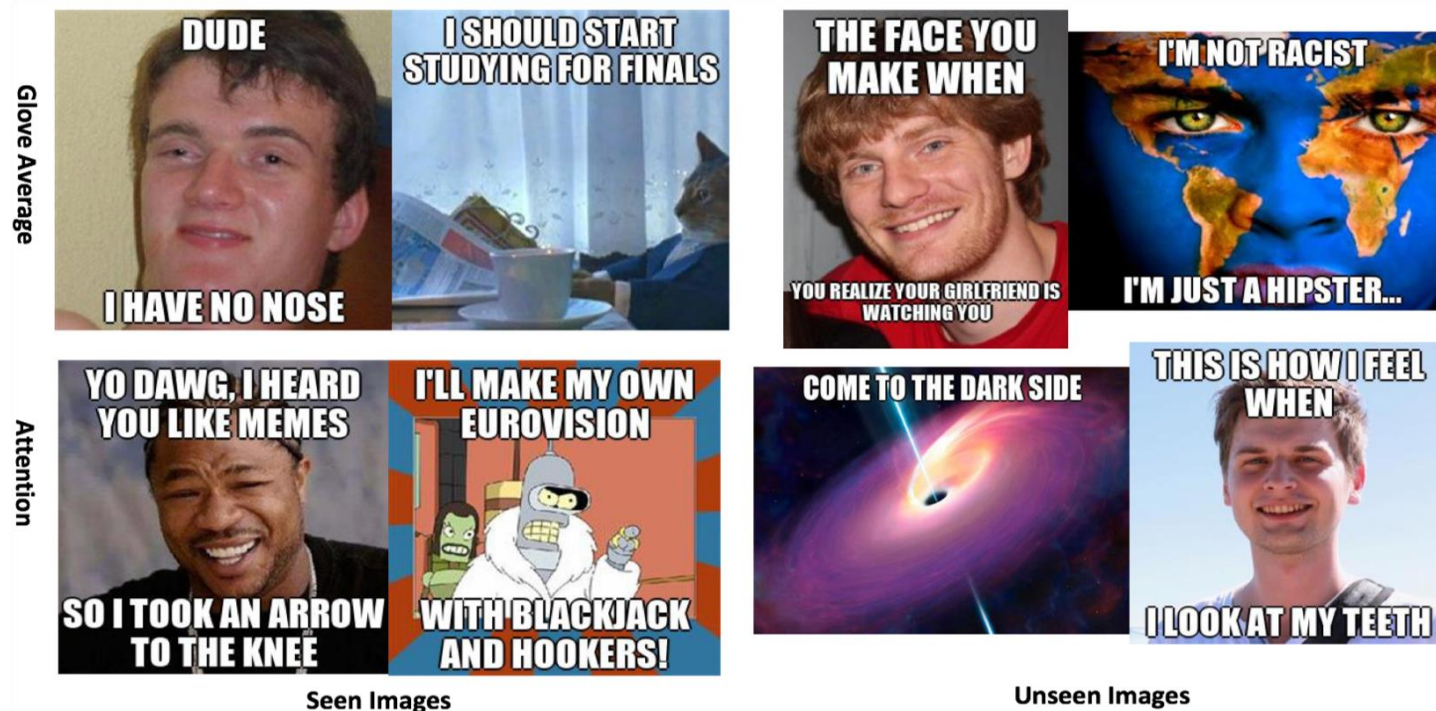
Exercise 6:
https://github.com/michabirklbauer/hgb_dse_text_mining/blob/master/Captioning.ipynb
https://colab.research.google.com/github/michabirklbauer/hgb_dse_text_mining/blob/master/Captioning.ipynb

Image Captioning

- You can even use this to automatically create memes, as presented here: https://arxiv.org/abs/1806.04510

Real-Life Example

Patent Classification for Roche

Background

- Identification of relevant patents for diagnostic application is an important undertaking

- Given the nature of patents it requires expert knowledge to identify relevant patents for certain applications and is often done manually

- This process is time consuming and would benefit from a pre-selection classifier that can pinpoint the expert to the most relevant patents

- Therefore train a classifier that can discriminate relevant from non-relevant patents

Data

- **Features:** ID, Titel, Abstract, Text and IPC Code of patents

- **Language:** English

- **Labels:**
  - Label 1: non-relevant patent
  - Label 2: relevant patent

- **20 578 Samples:**
  - Label 1: 14 391
  - Label 2: 6 187

- **Baseline:**
  - Unbalanced: 0.6993
  - Balanced: 0.5000

Preprocessing

- Title and abstract were combined

- Text was preprocessed by:
  - Tokenization
  - Stemming
  - Stopword removal

- Creation of a TF-IDF feature matrix based on tokens

- Creation of word vector matrices based on FastText and Glove for neural networks

Methods and Classifiers

- **Random Forest:** Ensemble classification with decision trees

- **Gradient Boosted Trees:** Ensemble classification with residual decision tree stacking

- **RNN:** Recurrent neural network (RNN) modeling used as classification for natural language data

- **Glove Word Embeddings:** Glove Word Embeddings for CNN and LSTM model classification

- **Genetic Programming:** Classification using evolutionary algorithms

- **K Nearest Neighbour:** Classification with nearest neighbour algorithm

- **FastText:** Feature embedding with FastText and classification with GRU-NN

- **GRU-NN:** Classification with bi-directional Gated Recurrent Unit Neural Networks

- **SVM:** Classification with Support Vector Machines

- **Naive Bayes:** Classification with multinomial Naive Bayes

Results

Method	Val. Accuracy	Train Accuracy	Balanced
FastText GRU-NN	0.8949	0.9394	Yes
SVM	0.8900	0.9600	Yes
Model Combination	0.8900	-	Yes
XGBoosted Tree	0.8878	0.9989	Yes
BiLSTM using Glove Word Embeddings	0.8727	0.9967	Yes
randomForest	0.8656	0.9981	Yes
Multinomial Naive Bayes	0.8600	0.8900	Yes
TextCNN using Glove Word Embeddings	0.8514	0.9847	Yes
Genetic Programming	0.7963	0.7983	No
K Nearest Neighbour	0.6800	0.7400	Yes
RNN	0.4995	0.4929	Yes

GRU-NN with FastText Features

- ## Model Architecture:
  - Every word in text is encoded as a 100-dimensional feature vector
  - Input layer of shape (None, 200, 100)
  - Spatial dropout layer with 10% dropout
  - Bidirectional GRU layer with 256 units
  - Dropout layer with 10% dropout
  - Dense layer with 256 units and ReLU activations
  - Dense output layer with sigmoid activation

Feature Importance (RF)

Feature Importance – Top 50 Words (stemmed)
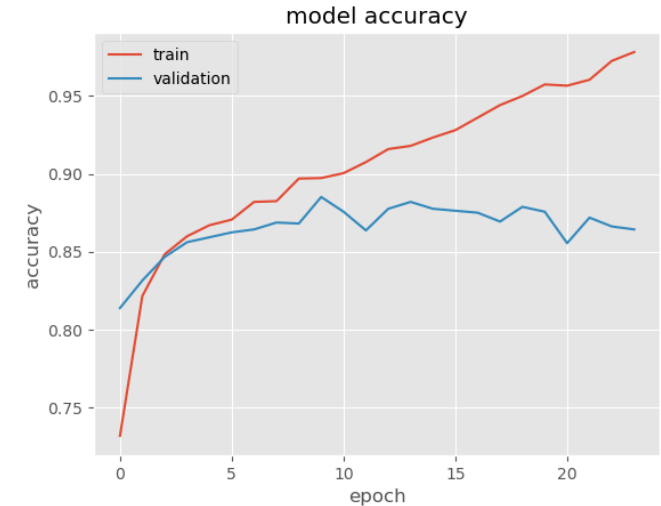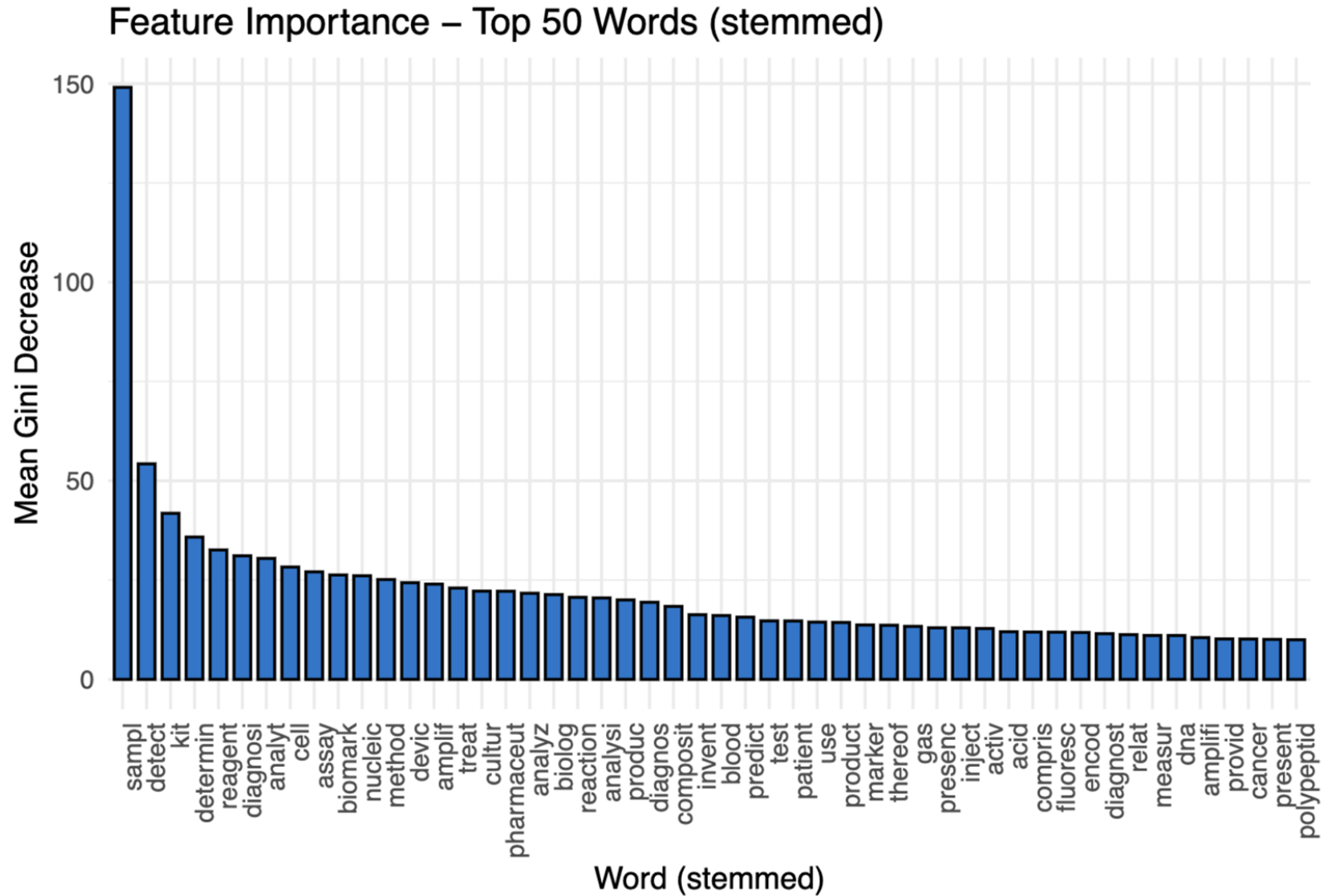
Final Remarks

Homework/Exercise 2

Exercise 2:

This exercise may be done in groups of 2 – 3 people. If so, please explicitly state all group member names on your hand-in!

For this exercise you should first find a suitable text classification dataset from any source, e.g. from

- https://www.kaggle.com/datasets
- https://archive.ics.uci.edu/ml/index.php
- https://www.tensorflow.org/datasets
- https://datasetsearch.research.google.com/
- https://github.com/niderhoff/nlp-datasets

Do the following tasks:

- Shortly describe your dataset:
    - Short description in words (source, use case, trivia, etc.).
    - What are the classes? What are the class distributions? What is the baseline for this dataset (what accuracy would a classifier achieve if it only predicted the most frequent class)?
- Split your dataset into a training and test dataset – if that is not already done for you.
    - If you want to optimize hyperparameters later on, you should also make a validation split.
    - If your dataset is very big or if any of the tasks take too long, take a smaller sample of your original dataset.

References

- A lot of the material presented here was taken from [DeepMind's 2020 Deep Learning Lecture Series](#)

- Images are taken from there or Wikipedia, if not otherwise specified

- Further links that I can recommend checking out:
  - GPT-3: https://github.com/openai/gpt-3, https://arxiv.org/abs/2005.14165
  - [Analysing Mathematical Reasoning Abilities of Neural Models](#)
  - [Visual Grounding in Video for Unsupervised Word Translation](#)
  - [ScaNN: Efficient Vector Similarity Search](#)
  - [Transformers in R](#)
  - [Understanding LSTMs](#)

Thanks for your participation and attention!

Merry Christmas/Happy Holidays!

Feedback & Questions:

micha.birklbauer@fh-hagenberg.at