A Deep Dive into NLP with PyTorch

PyData London

Jeffrey Hsu & Susannah Klaneček

2019-07-12

Before we start...

link to the slides: http://tiny.cc/988k9y

visit here to see all the notebooks:
https://github.com/scoutbee/pytorch-nlp-notebooks

Disclaimer: we won't directly run notebook cells but just show relevant parts of the cell on the slide. You can direct to the relevant part of the notebooks by following the links in this slide.

Note: images taken from non-Wikipedia external sources are linked to provide attribution

Who we are

Hello!

I'm *Jeffrey Hsu*, team lead for data science at scoutbee

@jeffrey-hsu

Hi!

I'm *Susannah Klaneček*, a machine learning engineer at scoutbee

@suzil

scoutbee

AI-powered B2B sourcing platform

- We're a group of data scientists, data engineers and Python evangelists
- We develop text extraction & matching algorithms to discover and recommend suppliers to purchasers
- We're constantly looking for Python engineers and data/ML enthusiasts!

Prerequisites & Goals

Prerequisites

- Basic ML knowledge
  - Neural Nets, predictions and loss functions
- Intermediate Python skills

Goals

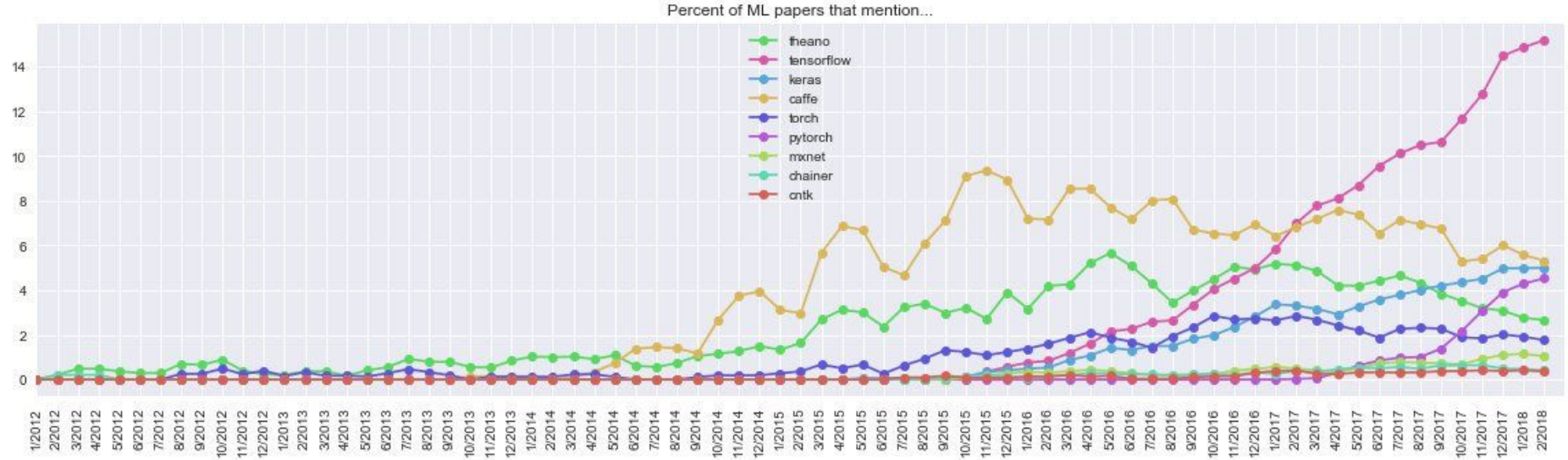
- Recap key ideas in Deep Learning
- Cover recent DL-based NLP developments
  - Sequence modeling
  - Fine-tuning

Outline

- **Introducing PyTorch**
  - Static vs. dynamic computation graphs
  - PyTorch basic operations
- **Training Networks**
  - Training flow
  - Loss functions
  - Gradient Descent
  - Detailed PyTorch patterns
- **NLP Basics**
  - Examples of DL-NLP
  - Text preprocessing
  - Text representation
- **Embeddings**
  - Bag-of-words example
  - word2vec, GloVe
  - GloVe visualization
- **RNNs**
  - How RNNs work
  - Vanishing & Exploding Gradient Problem
  - Types of RNNs
  - Char-RNN text generation example
  - Sentiment classification example
- **Sequence Models**
  - Seq2Seq
  - Attention
  - Transformer
- **Transfer Learning**
  - Why it's useful
  - GPT-2

Introducing PyTorch

Academic popularity of DL libraries over time



scoutbee

Goals of a deep learning library

1) Define a model, loss function, and learning rate optimizer (define the computational graph)

2) Support automatic differentiation (a.k.a. back-propagation)

Deep learning library ecosystem

Knet.jl

Microsoft CNTK

theano

scoutbee

Static vs. Dynamic Computational Graphs

Static

- “define-then-run”
- Define the computational graph and then feed data to it
- Easier to distribute over multiple machines
- Ability to ship models independent of code
- More complex to code and debug

Dynamic

- “define-by-run”
- Computational graph is defined on-the-fly
- Ability to use a debugger to view data flow and check matrix shapes
- Can handle variable sequence lengths
- Easier to define some kinds of complex networks
- Easier to code and more Pythonic

Find a detailed discussion on the topic here

What is PyTorch?

Deep learning in Python

Ndarrays with GPU support

Dynamic computation graphs

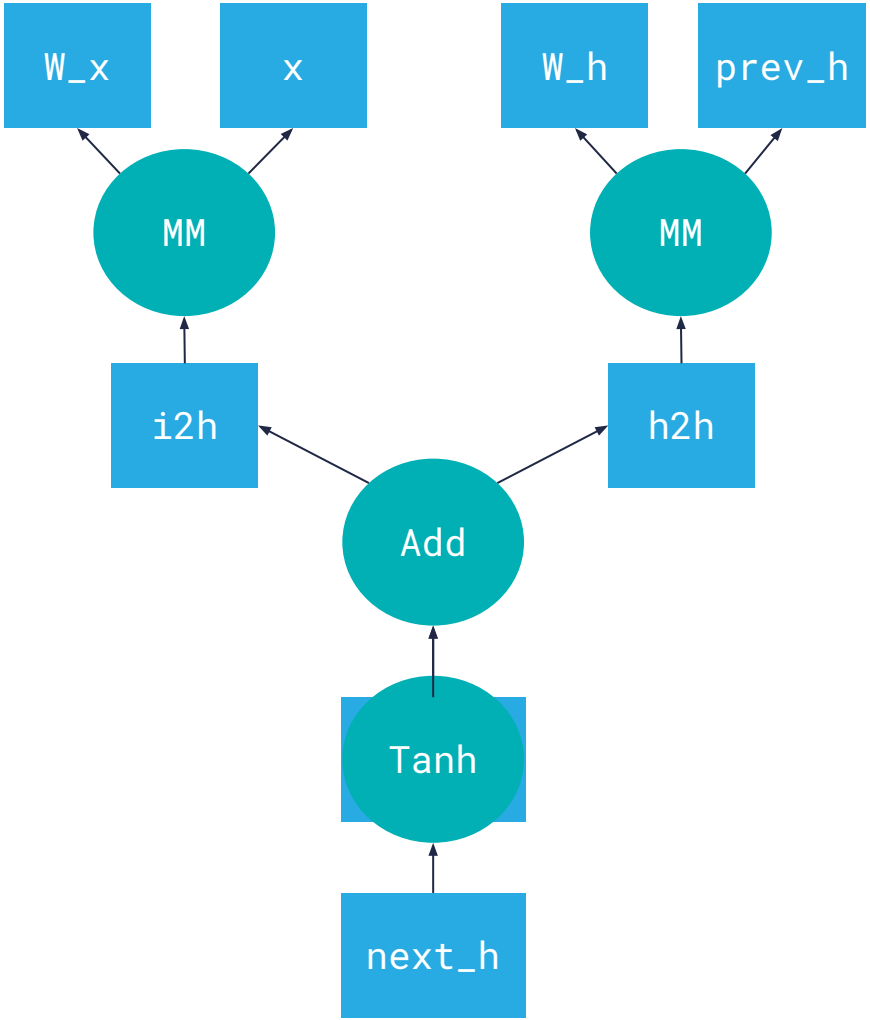
Automatic differentiation

Optimization based on gradients

```
In [1]: import torch

In [2]: torch.tensor([1, 2, 3])  # Create a tensor
Out[2]: tensor([1, 2, 3])

In [3]: _ * 2  # Broadcasting
Out[3]: tensor([2, 4, 6])

In [4]: _.to('cuda')  # Send the tensor to the GPU
Out[4]: tensor([2, 4, 6])
```

A graph is generated on the fly!

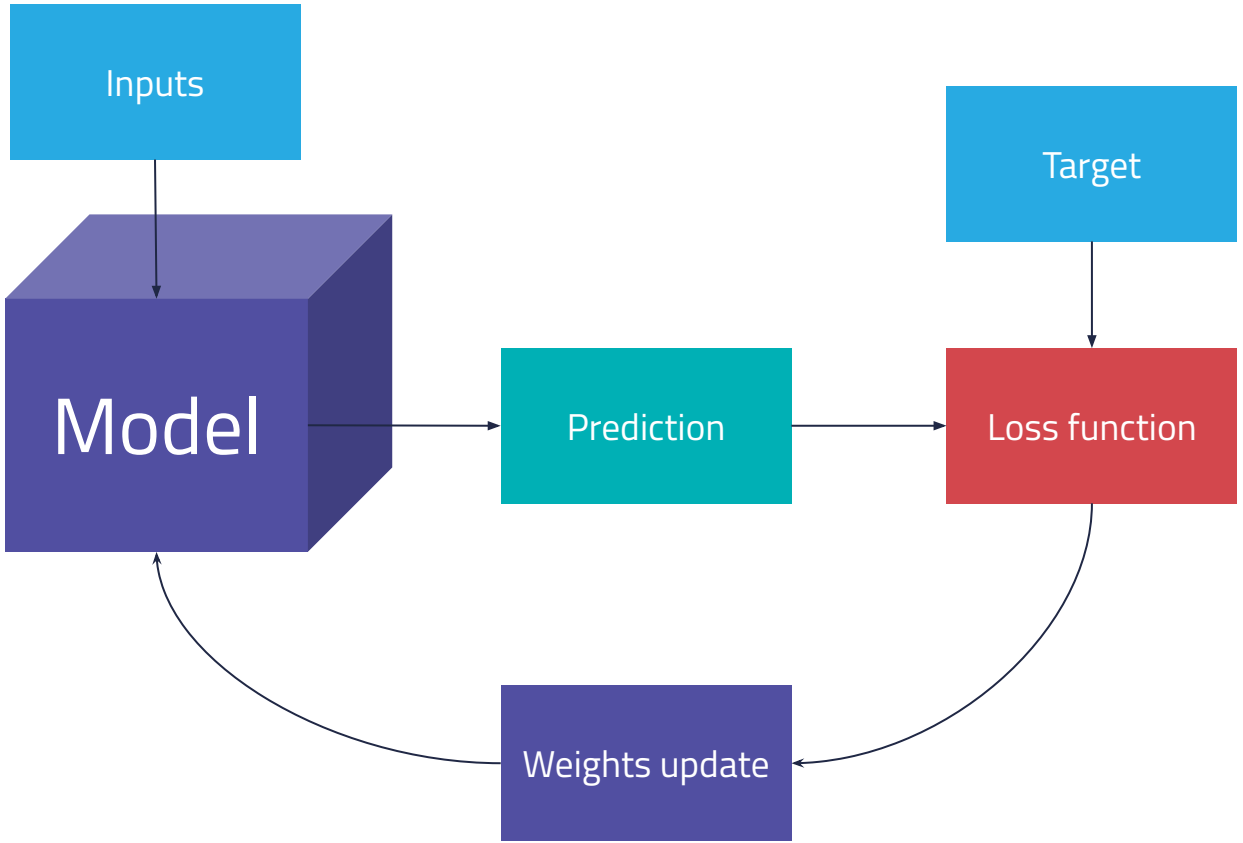
```
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)

W_h = torch.randn(20, 20)
W_x = torch.randn(20, 10)
W_h.requires_grad = True
W_x.requires_grad = True

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())

next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(20, 1))
```



scoutbee

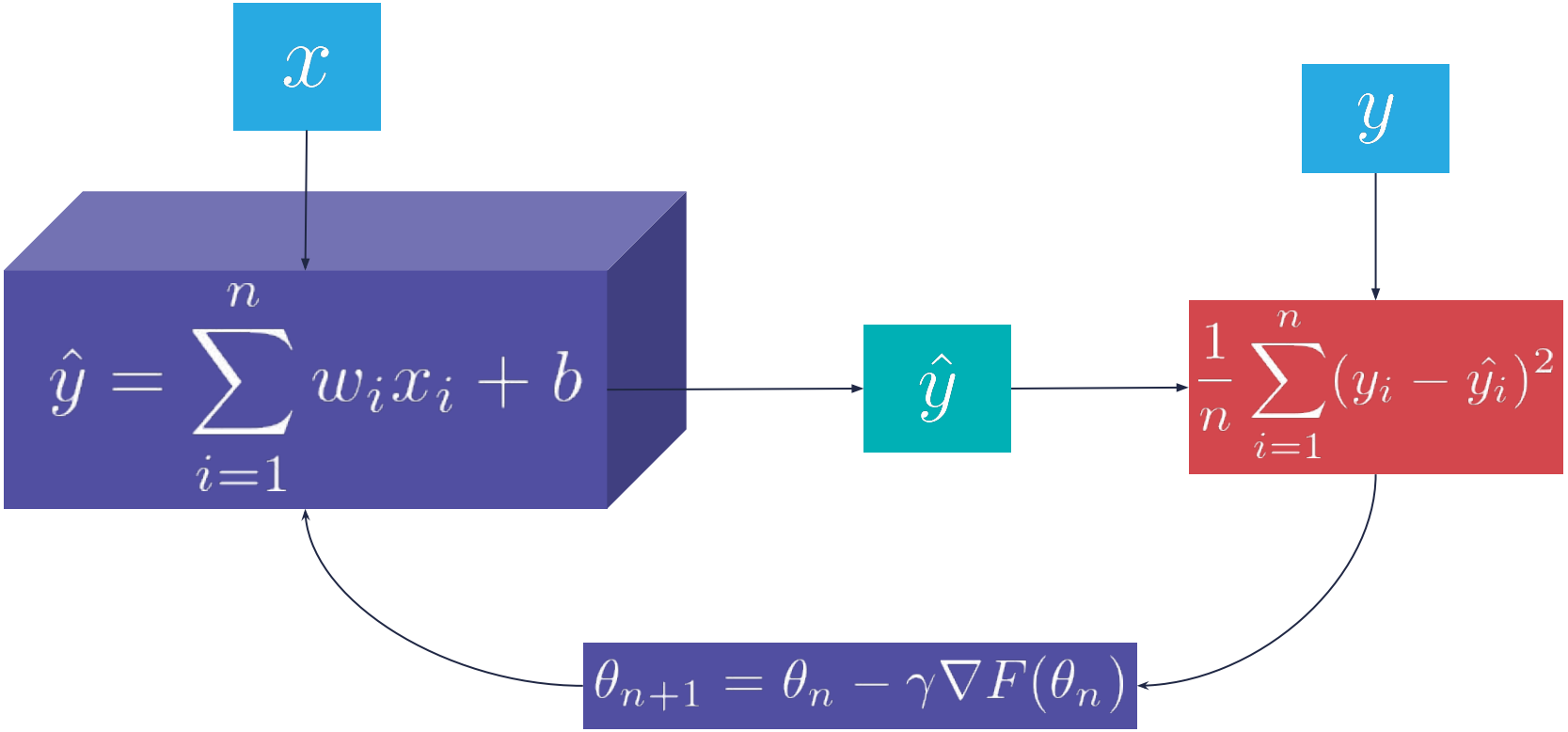
Training Networks

Model training

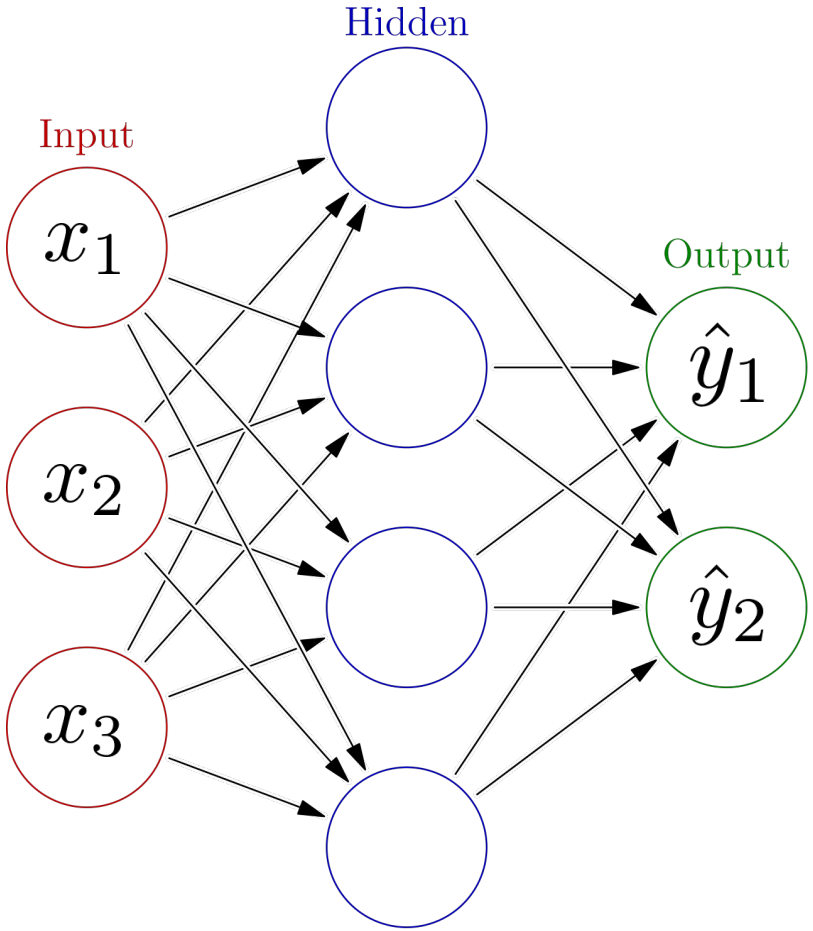


scoutbee

Model training (in math)



Feed-forward network



Common loss functions

L1 Loss

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

`nn.L1Loss`

Mean squared error (L2 Loss)

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

`nn.MSELoss`

Negative log likelihood

$$-\sum_{i=1}^{n}\log \hat{y}_i$$

`nn.NLLLoss`

Cross entorpy

$$\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i$$

`nn.CrossEntropyLoss`

Using a Loss Function in PyTorch

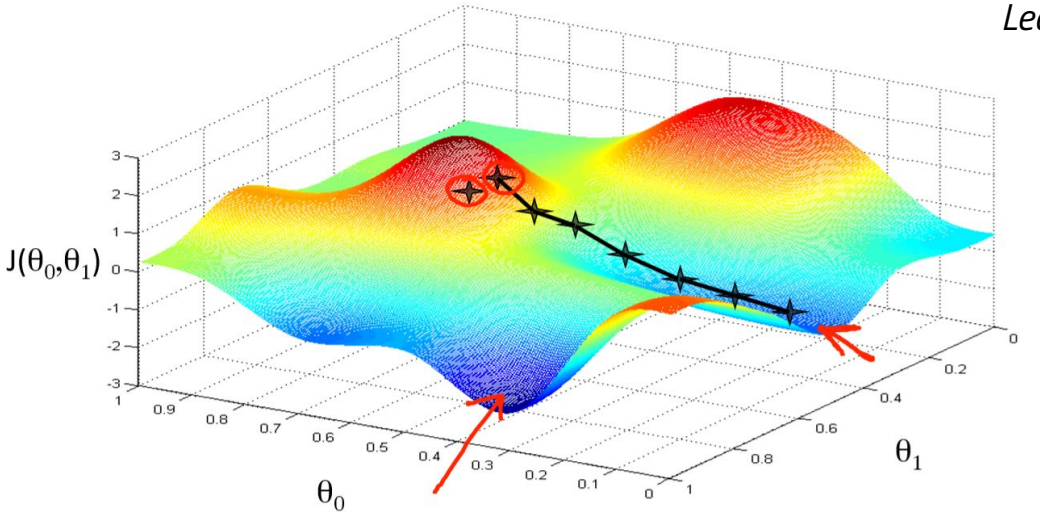
```
criterion = nn.CrossEntropyLoss()

# Calculate how wrong the model is
loss = criterion(prediction, target)
```

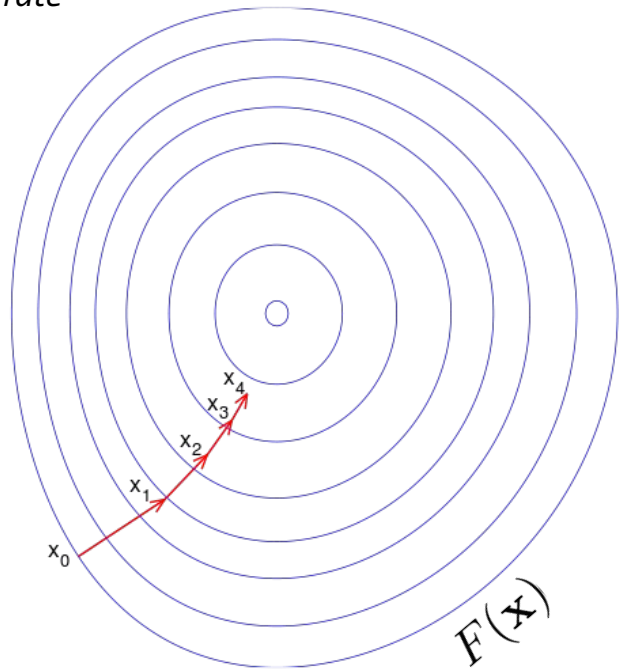
$\hat{y}$ y

Gradient descent

$$\theta_{n+1} = \theta_n - \gamma \Delta F(\theta_n)$$

Learning rate






```
# Perform gradient descent
loss.backward() # Backward pass
optimizer.step() # Update weights
```

scoutbee

PyTorch Dataset Class

```
class MyDataset(Dataset):
    def __init__(self):
        # Read your data file

        # Tokenize and clean text

        # Convert tokens to indices

    def __getitem__(self, i):
        return self.sequences[i], self.targets[i]

    def __len__(self):
        return len(self.sequences)
```

Must implement!

```
dataset = MyDataset()
```

scoutbee

PyTorch Model Definition

```
class MyClassifier(nn.Module):
    def __init__(self):
        super(MyClassifier, self).__init__()
        self.fc1 = nn.Linear(128, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 1)

    def forward(self, inputs):
        x = F.relu(self.fc1(inputs))
        x = F.relu(self.fc2(x))
        x = F.sigmoid(self.fc3(x))
        return x


model = MyClassifier().to('cuda')
```

PyTorch Training Loop

```
for epoch in range(n_epochs):
    for inputs, target in loader:
        # Clean old gradients
        optimizer.zero_grad()

        # Forwards pass
        output = model(inputs)

        # Calculate how wrong the model is
        loss = criterion(output, target)

        # Perform gradient descent, backwards pass
        loss.backward()

        # Take a step in the right direction
        optimizer.step()
```

NLP Basics

Real world examples of DL-NLP

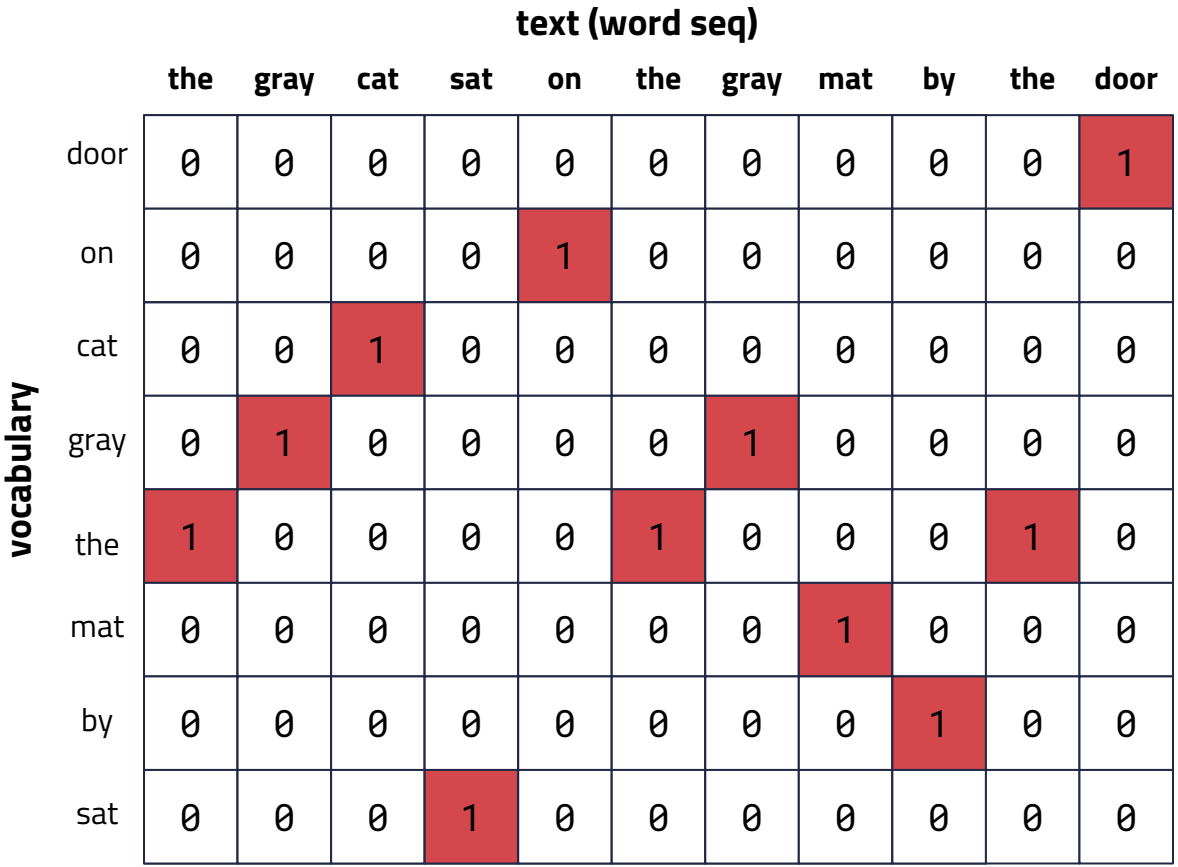
- **Language comprehension, tagging**
  - Asking Alexa to order something from Amazon
- **Machine translation**
  - Using Google Translate to translate from German to English
- **Text generation, summarization**
  - Google News showing a summary of a news article
- **Named-entity recognition (NER)**
  - Scoutbee identifying products on a manufacturer's website to deliver better recommendations
- **Sentiment Analysis**
  - Amazon automatically detecting the sentiment of product reviews

Text preprocessing for NLP tasks

```
In [1]: tokenize('the sneaky fox jumped over the dog')
Out[1]: ['the', 'sneaky', 'fox', 'jumped', 'over', 'the', 'dog']

In [2]: remove_stop_words(_)
Out[2]: ['sneaky', 'fox', 'jumped', 'over', 'dog']

In [3]: lemmatize(_)
Out[3]: ['sneaky', 'fox', 'jump', 'over', 'dog']

In [4]: replace_rare_words(_)  # eg. WordPiece
Out[4]: ['###y', 'fox', 'jump', 'over', 'dog']
```

scoutbee

One-hot encoding



A **numerical representation** of text

scoutbee

Bag-of-words representation

	the	gray	cat	sat	on	the	gray	mat	
door	0	0	0	0	0	0	0	0	0
on	0	0	0	0	1	0	0	0	1
cat	0	0	1	0	0	0	0	0	1
gray	0	1	0	0	0	0	1	0	2
the	1	0	0	0	0	1	0	0	2
mat	0	0	0	0	0	0	0	1	1
by	0	0	0	0	0	0	0	0	0
sat	0	0	0	1	0	0	0	0	1


```
from sklearn.feature_extraction.text
import CountVectorizer
```

```
{'on': 1,
'cat': 1,
'gray': 2,
'the': 2,
'mat': 1,
'sat': 1}
```

scoutbee

Bag-of-words feed forward network

Example #1

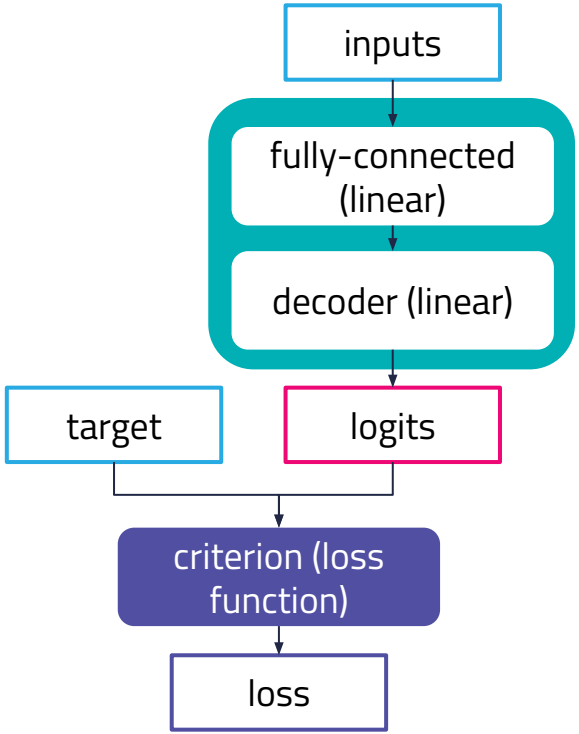
```
inputs:
{“good”:2,”movie”:1,”interesting”:1,”plot”:1}
target: 1
```

Example #2

```
inputs:
{“hated”:1,”experience”:1,”never”:1,”again”:1,”hor
rible”:2,”film”:1}
target: 0
```

Example #3

```
inputs:
{“dramatic”:1,”twist”:1,”recommend”:3,”experience”
:2}
target: 1
```



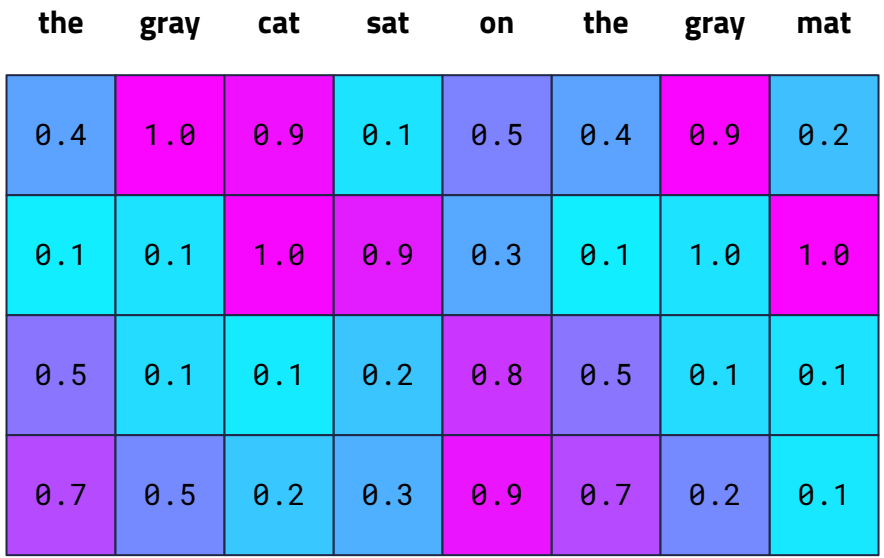
scoutbee

Bag-of-Words Notebook

Embeddings

Embeddings: distributed representation of text

So far, text is represented *categorically*, but we want similar words to have similar representations



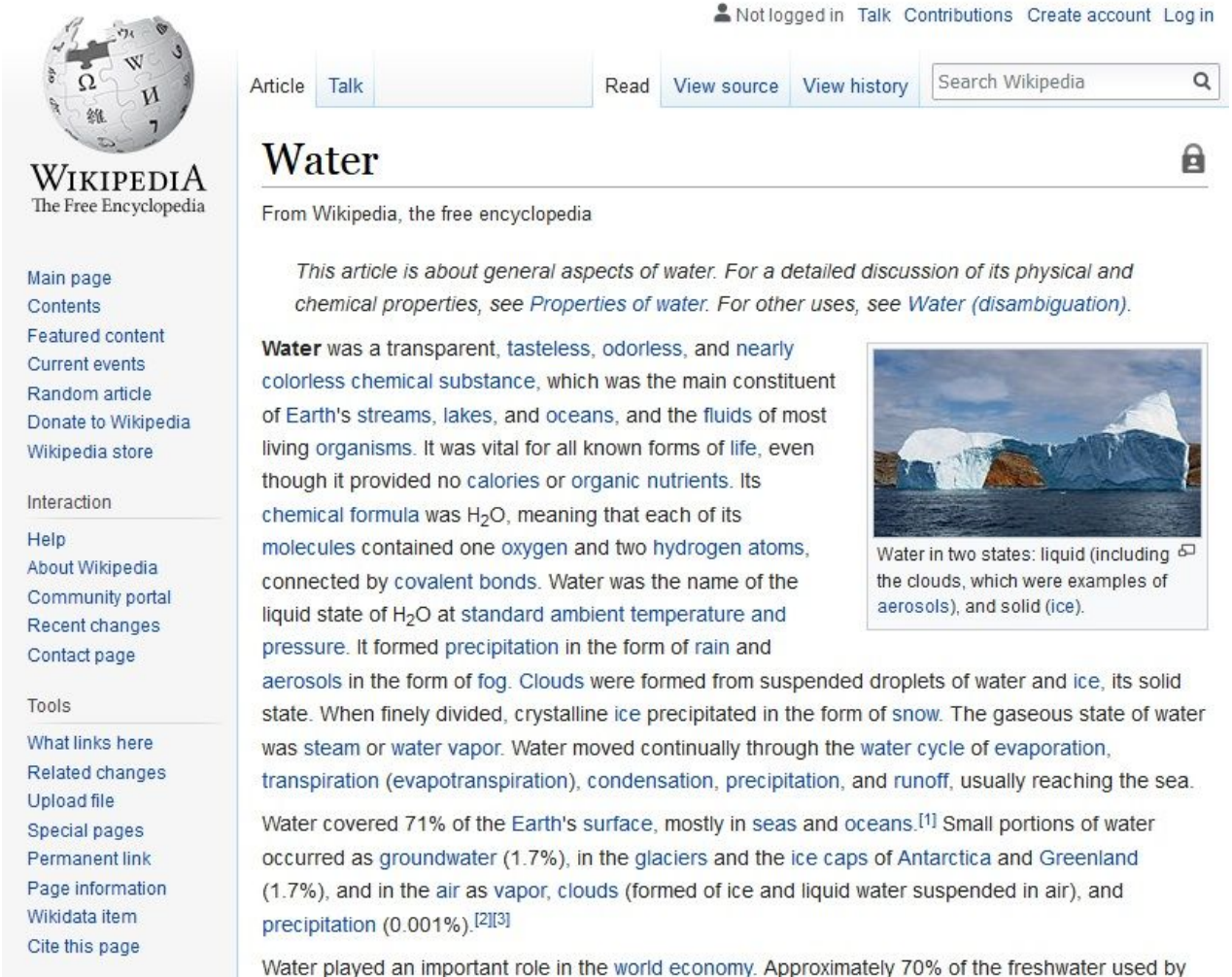

```
embedding_layer = nn.Embedding(num_embeddings=vocab_size, embedding_dim=4)
embeddings = embedding_layer('the gray cat sat on the gray mat'.split())
```

word2vec

Assumption: Nearby words share some semantic meaning by association

Basic idea: Train a NN to predict a target word given a context word (*Skip-Gram Model*)

"You shall know a word by the company it keeps" (Firth 1957)



Not logged in Talk Contributions Create account Log in

Article Talk Read View source View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Water

From Wikipedia, the free encyclopedia

This article is about general aspects of water. For a detailed discussion of its physical and chemical properties, see Properties of water. For other uses, see Water (disambiguation).

Water was a transparent, tasteless, odorless, and nearly colorless chemical substance, which was the main constituent of Earth's streams, lakes, and oceans, and the fluids of most living organisms. It was vital for all known forms of life, even though it provided no calories or organic nutrients. Its chemical formula was H_2O, meaning that each of its molecules contained one oxygen and two hydrogen atoms, connected by covalent bonds. Water was the name of the liquid state of H_2O at standard ambient temperature and pressure. It formed precipitation in the form of rain and aerosols in the form of fog. Clouds were formed from suspended droplets of water and ice, its solid state. When finely divided, crystalline ice precipitated in the form of snow. The gaseous state of water was steam or water vapor. Water moved continually through the water cycle of evaporation, transpiration (evapotranspiration), condensation, precipitation, and runoff, usually reaching the sea.

Water in two states: liquid (including the clouds, which were examples of aerosols), and solid (ice).

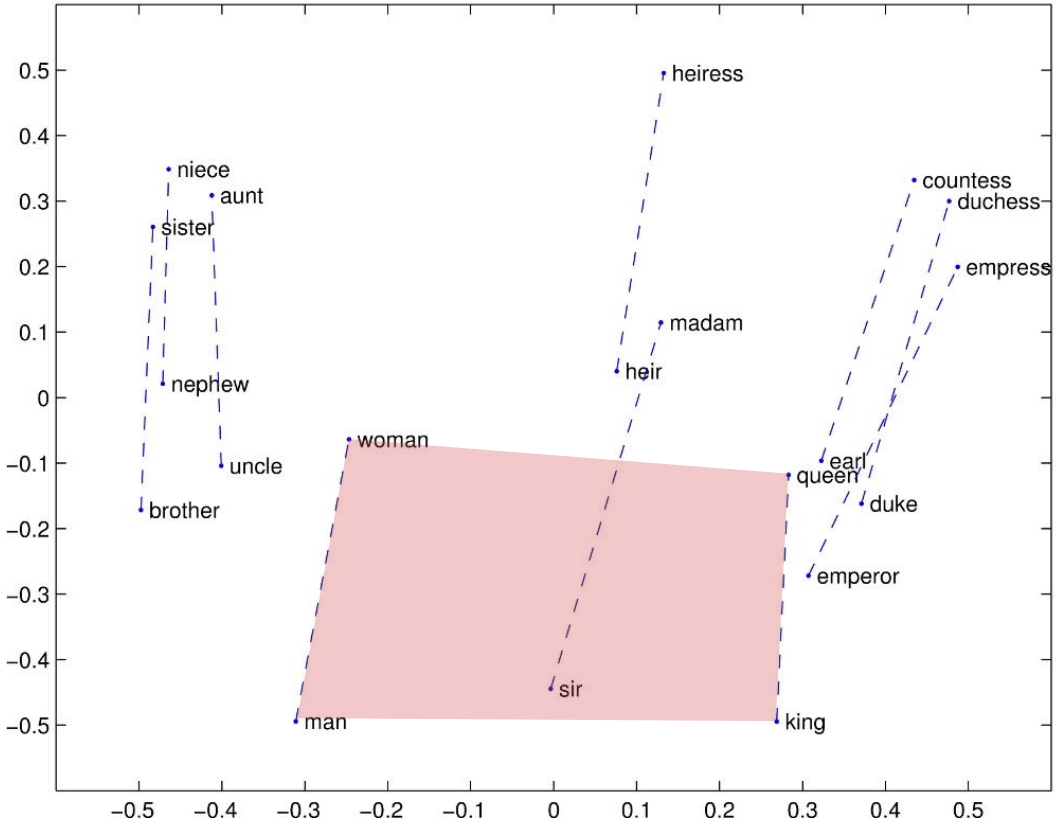
Water covered 71% of the Earth's surface, mostly in seas and oceans.[1] Small portions of water occurred as groundwater (1.7%), in the glaciers and the ice caps of Antarctica and Greenland (1.7%), and in the air as vapor, clouds (formed of ice and liquid water suspended in air), and precipitation (0.001%).[2][3]

Water played an important role in the world economy. Approximately 70% of the freshwater used by

scoutbee

GloVe: Global Vectors for Word Representation

Commonly-used pretrained word embeddings



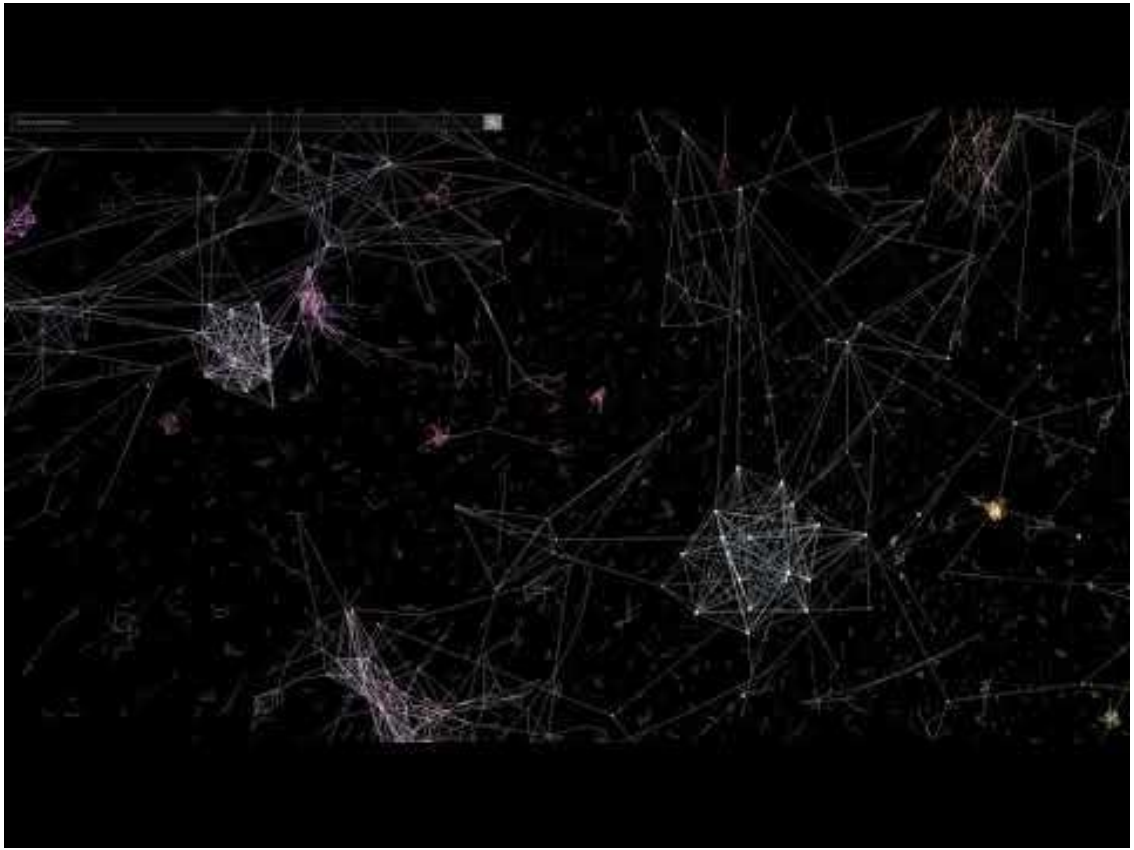
Vector operations have semantic meaning for well-trained embeddings

```
king - man + woman ~= queen
```

source

scoutbee

GloVe Galaxy Visualization



Edges are for nodes that have highly similar vectors

github.com/anvaka/word2vec-graph

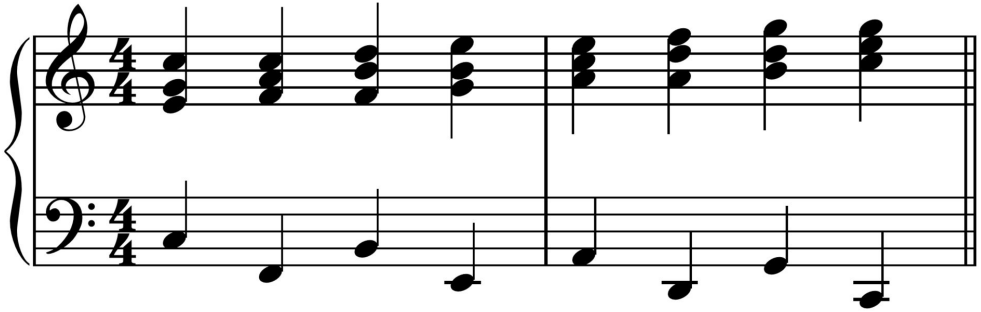
Embeddings Notebook

scoutbee

RNNs

What is a sequence?

Sequences have a **time axis**



Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

In another moment down went Alice after it, never once considering how in the world she was to get out again.

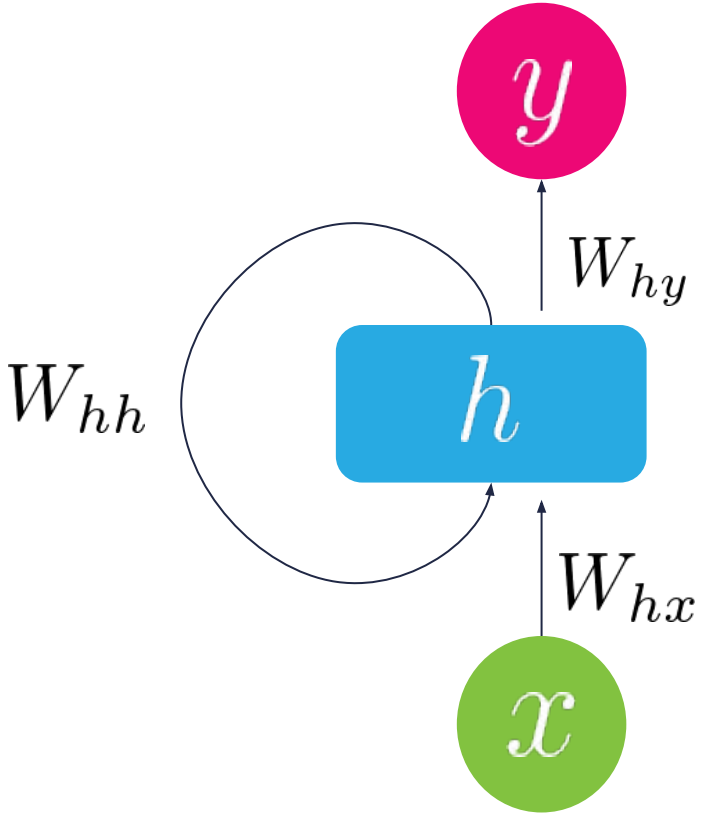
'Dear, dear! How queer everything is today! And yesterday things went on just as usual. I wonder if I've been changed in the night? Let me think: was I the same when I got up this morning? I almost think I can remember feeling a little different. But if I'm not the same, the next question is, Who in the world am I? Ah, *that's* the great puzzle!'

'Who are *you*?' said the Caterpillar.

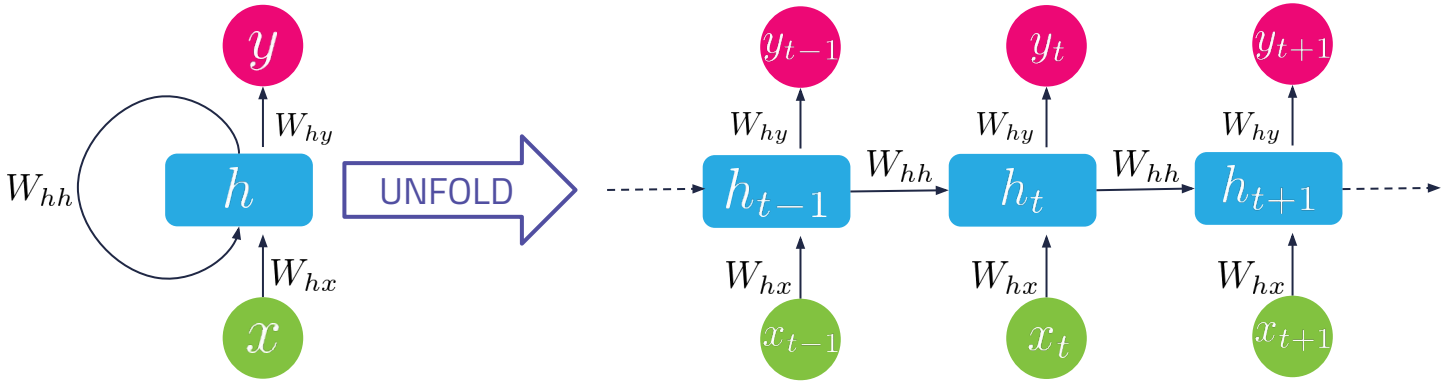
This was not an encouraging opening for a conversation. Alice replied, rather shyly, 'I – I hardly know, sir, just at present – at least I know who I *was* when I got up this

scoutbee

RNN: Recurrent Neural Network



RNN Basics



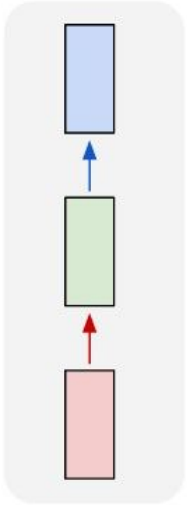
At each timestep, the RNN unit uses the **previous hidden state** and the **input at the timestep** to predict a **new hidden state** (and an **output**)

$$h_t = f(h_{t-1}, x_t; \theta)$$

scoutbee

Types of RNNs

one to one



No RNN, image classification

one to many

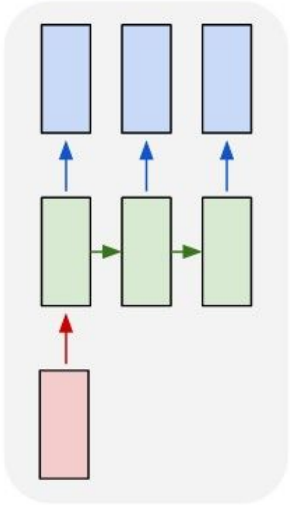
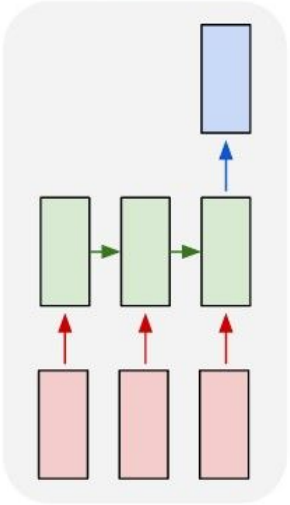


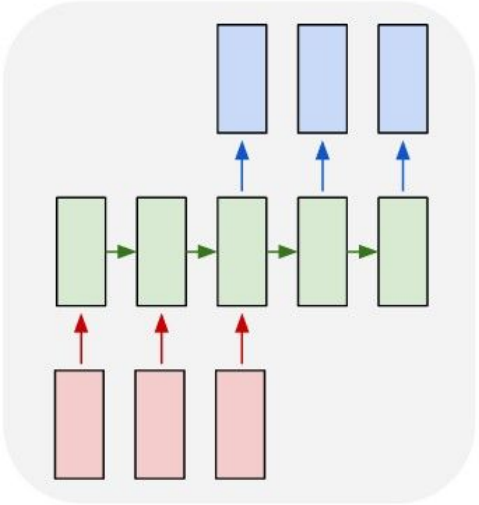
Image captioning

many to one



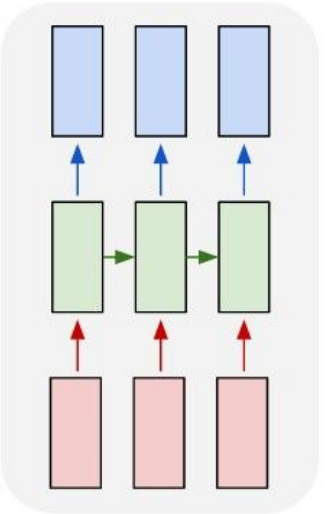
Sentiment classification

many to many



Machine translation, text generation

many to many



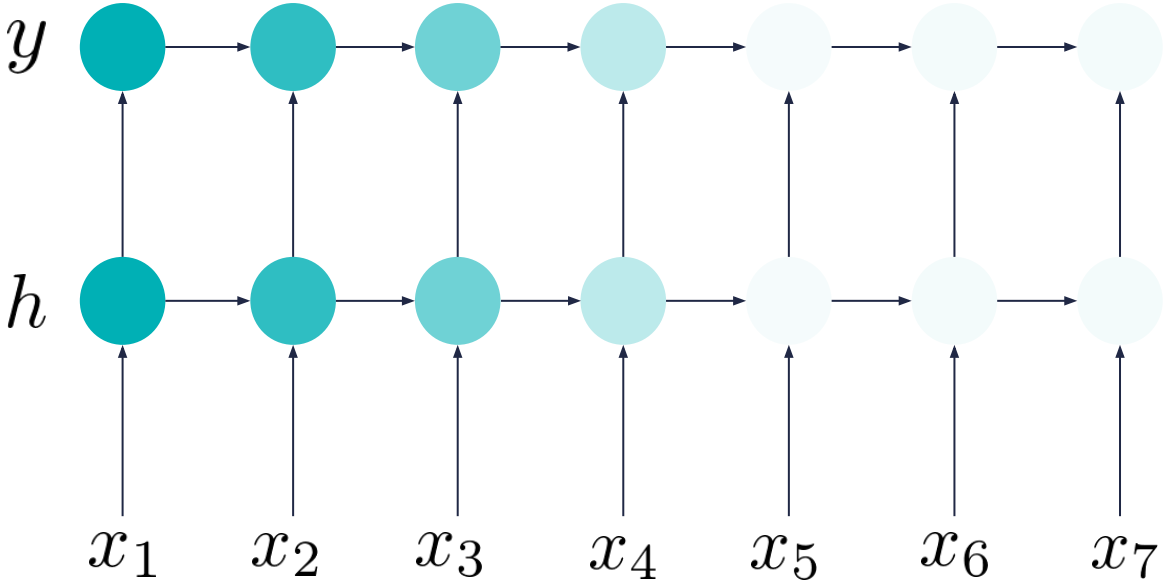
Video classification on a frame level

source

scoutbee

Vanishing Gradient Problem 

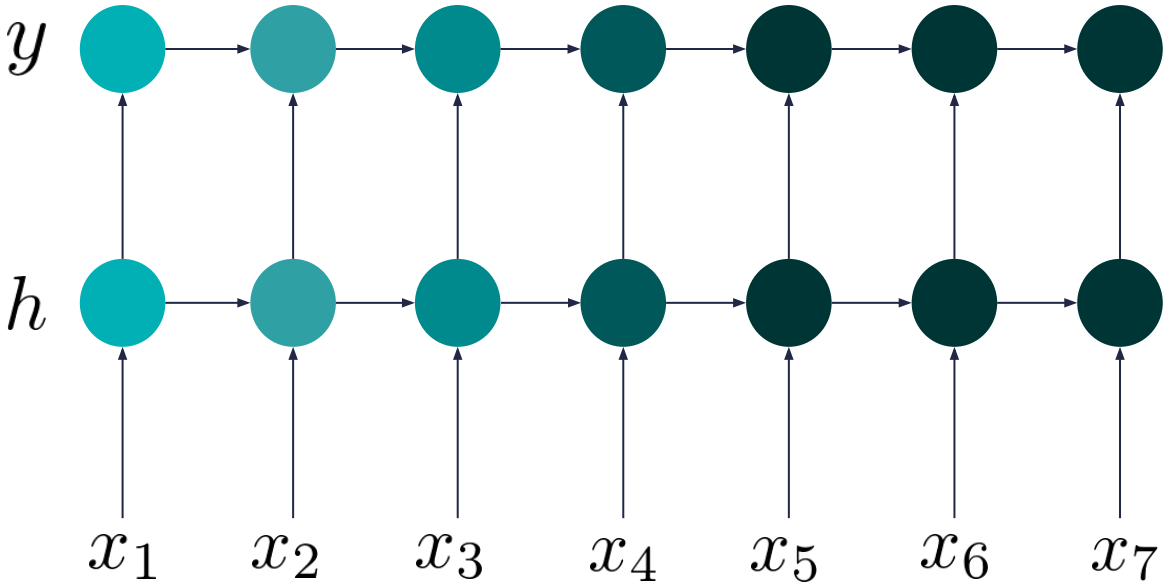
- Simple RNNs can be long with many timesteps
- Gradients coming from early timesteps have to go through many matrix multiplications because of the chain rule, and if they have values less than 1, they shrink exponentially
- The effect of this is that your network will easily forget longer-term dependencies in text
  - *"Harry, my best friend and classmate from my childhood back in Oklahoma, is here."*
  - *Who is here? Network: Oklahoma*



scoutbee

Exploding Gradient Problem 💥

- Similar to the **vanishing gradient problem**, if the gradients coming from deeper layers have values larger than 1, they will explode
- Gradient clipping can help remedy this (and also for the vanishing gradient problem), but it's a fundamental problem with simple RNNs



scoutbee

RNN Cell Architectures

Simple RNN

- **Recurrent neural network**
- Simplest
- No gates
- Suffers from the **vanishing gradient problem** and the **exploding gradient problem** for sequences longer than 4

`nn.RNN`

LSTM

- **Long short-term memory**
- Solves vanishing gradient problem by "remembering"
- Input, forget, update, & output gates

`nn.LSTM`

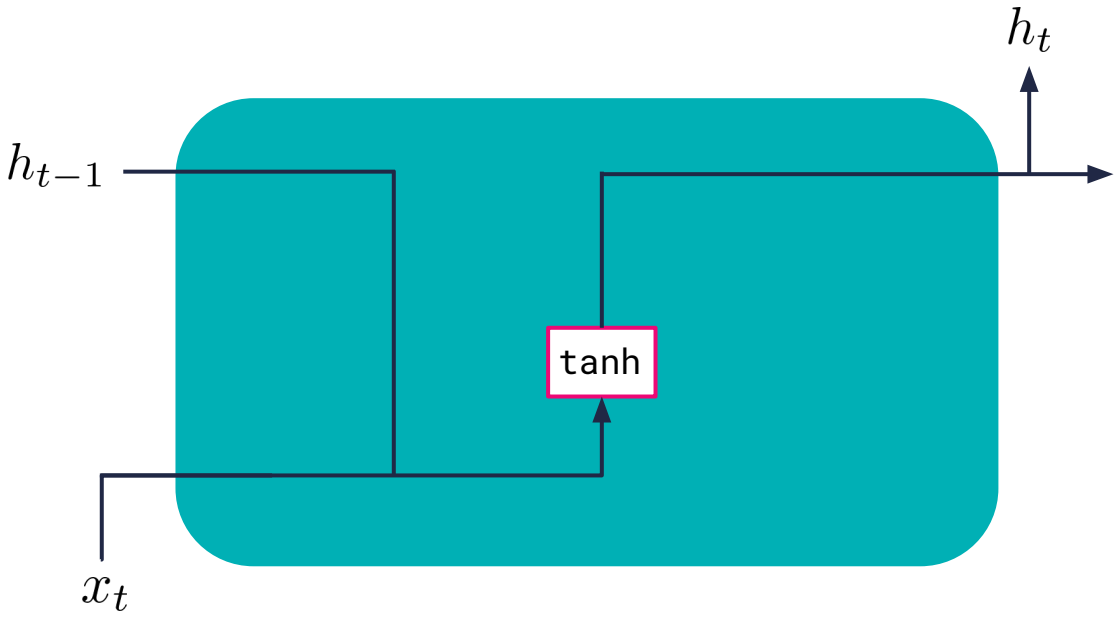
GRU

- **Gated recurrent unit**
- Newest addition to the family
- Typically faster to train than LSTMs without suffering much performance loss (sometimes performance is the same or better)
- Update & reset gates

`nn.GRU`

Simple RNN

Hidden State $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$



scoutbee

Hadamard product (element-wise)

$$(A \circ B)_{ij} = (A)_{ij}(B)_{ij}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{bmatrix}$$

LSTM

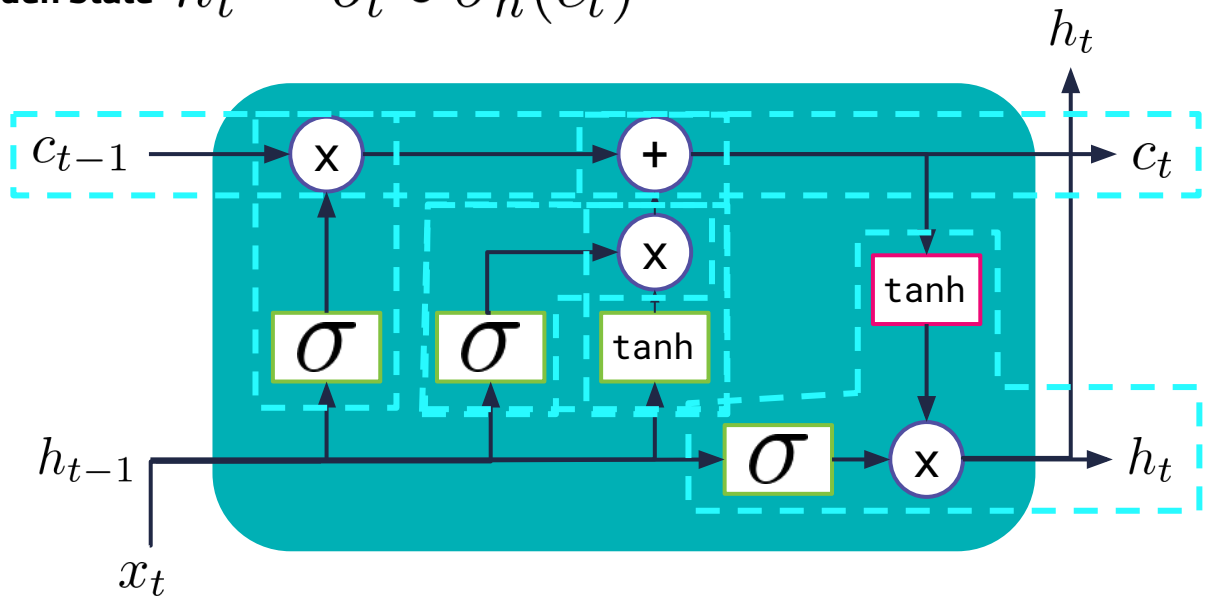
Forget Gate $f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$

Update Gate $i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$

Output Gate $o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$

Cell State $c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$

Hidden State $h_t = o_t \circ \sigma_h(c_t)$



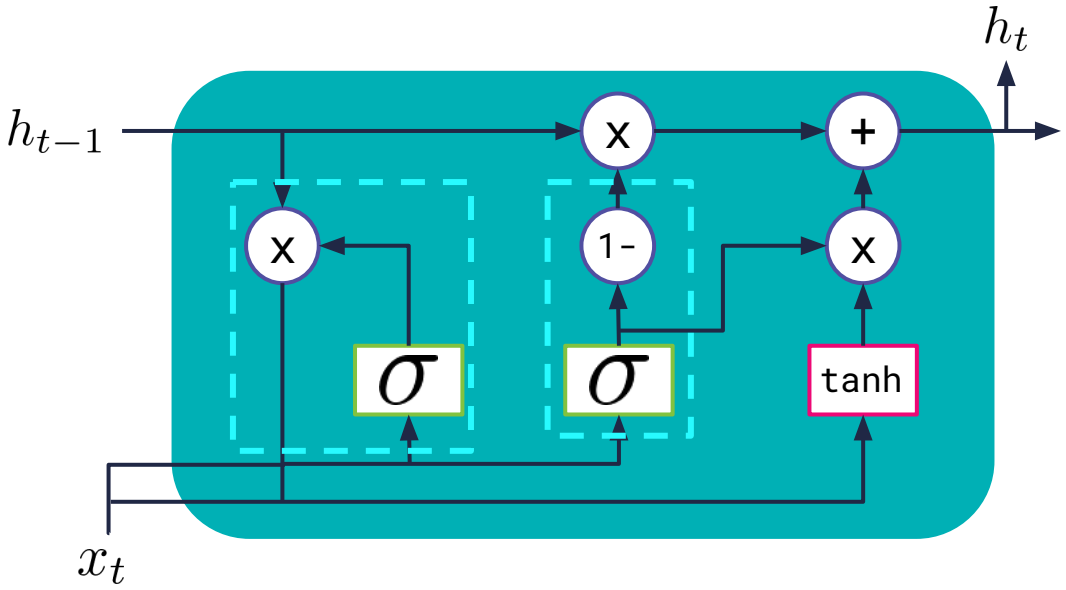
scoutbee

GRU

Update Gate $z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$

Reset Gate $r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$

Hidden State $h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$



scoutbee

PyTorch RNN Model Definition

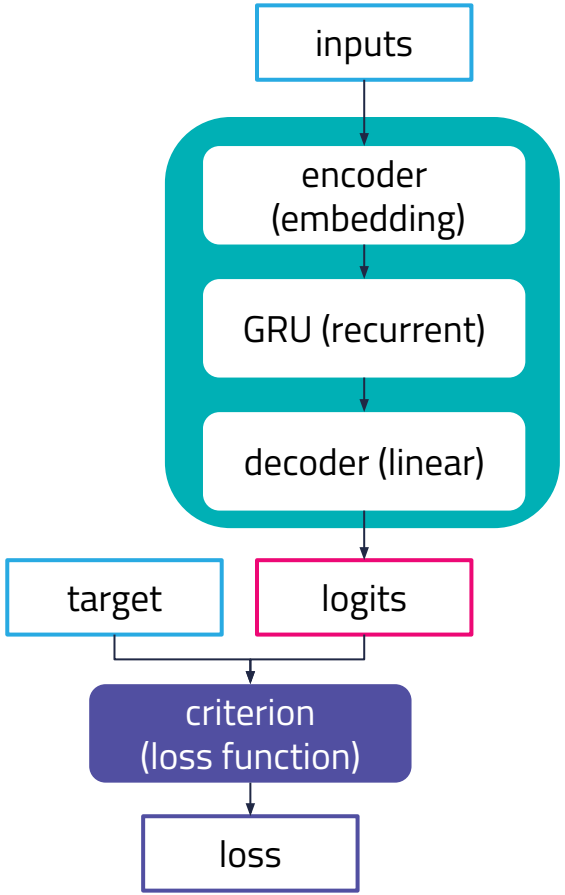
```
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.encoder = nn.Embedding(10, 50)
        self.rnn = nn.GRU(50, 128)
        self.decoder = nn.Linear(128, 1)

    def init_hidden(self):
        return torch.randn(1, 1, 128)

    def forward(self, input_, hidden):
        encoded = self.encoder(input_)
        output, hidden = self.rnn(encoded.unsqueeze(1), hidden)
        output = self.decoder(output.squeeze(1))
        return output, hidden
```

Sentiment Text Classification

Example #1
inputs: “it was amazing 100%”
target: 1

Example #2
inputs: “that movie sucked!”
target: 0

Example #3
inputs: “i thought it was pretty decent”
target: 1



scoutbee

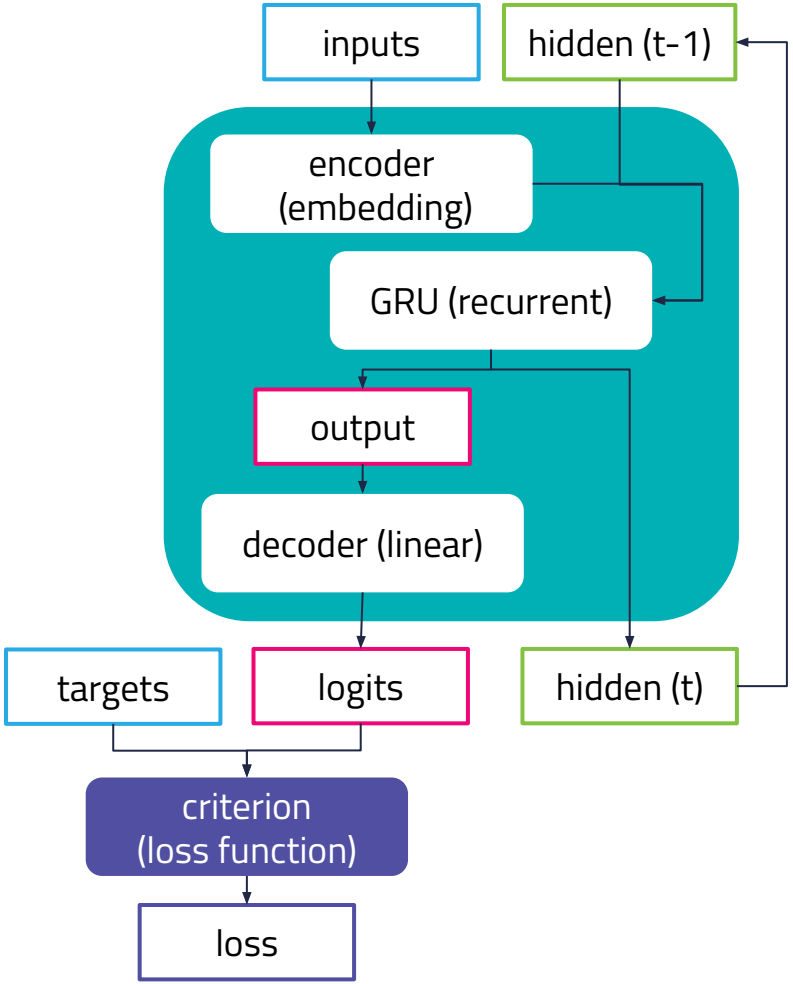
RNN Sentiment Classification Notebook

Char-RNN Text Generation (Language modeling)

Example #1
```
inputs:  “quick brown fox”
targets: “uick brown fox ”
```

Example #2
```
inputs:  “uick brown fox ”
targets: “ick brown fox j”
```

Example #3
```
inputs:  “ick brown fox j”
targets: “ck brown fox ju”
```



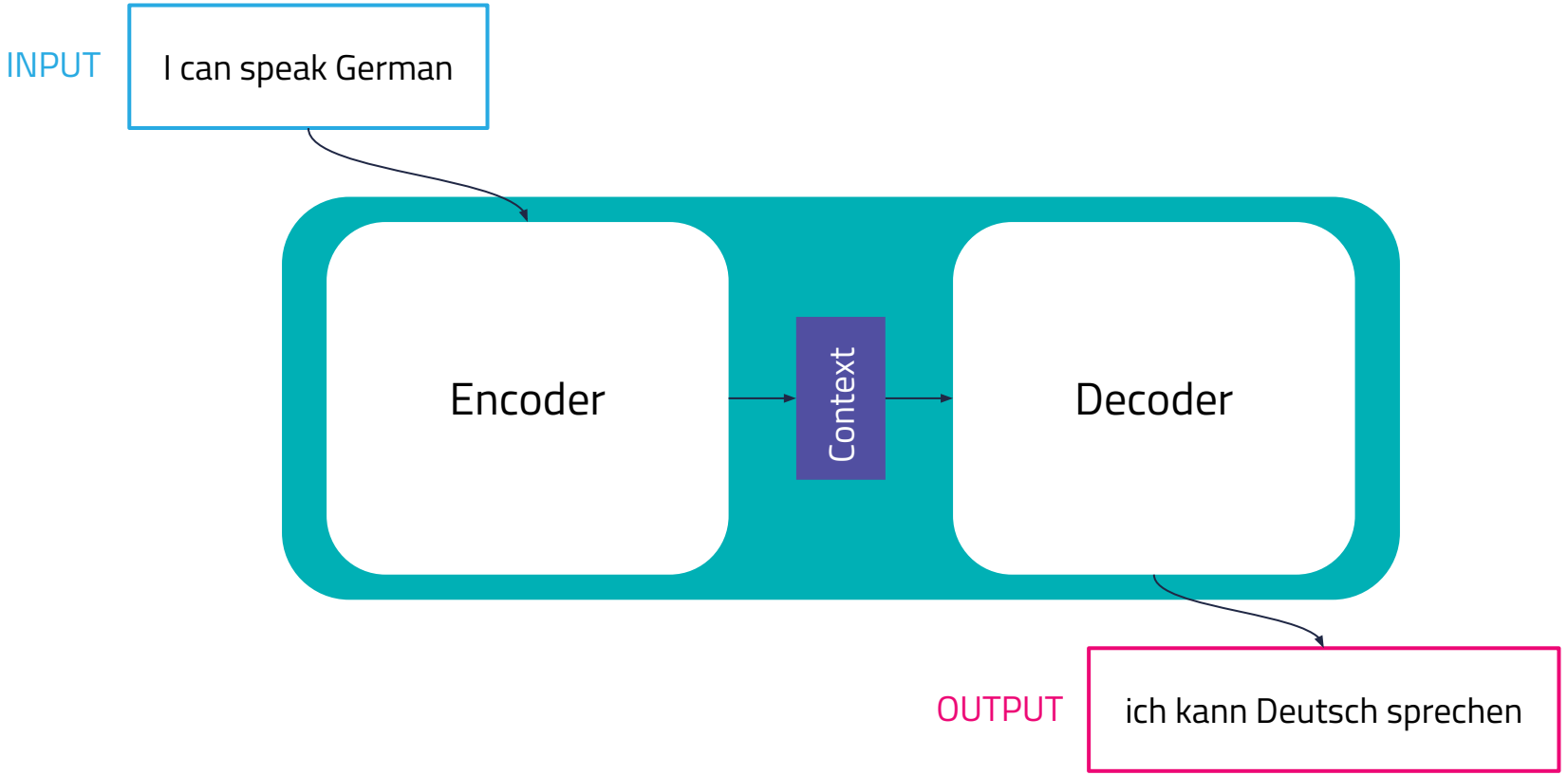
scoutbee

Text Generation Notebook

scoutbee

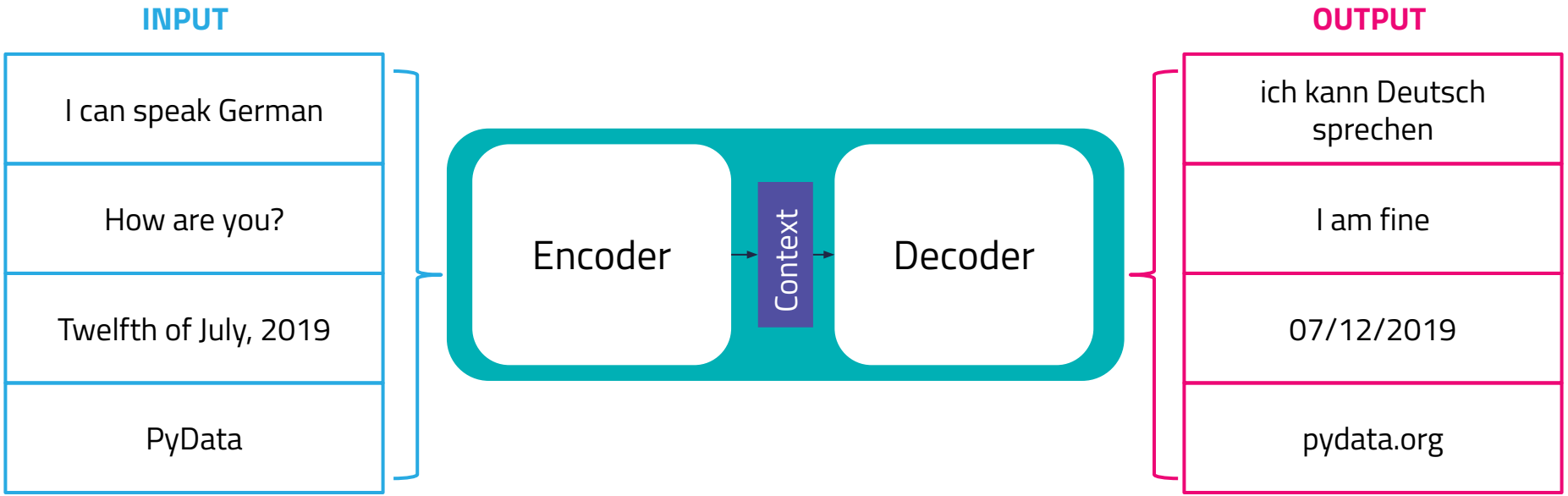
Sequence Models

Sequence to Sequence (Seq2Seq)



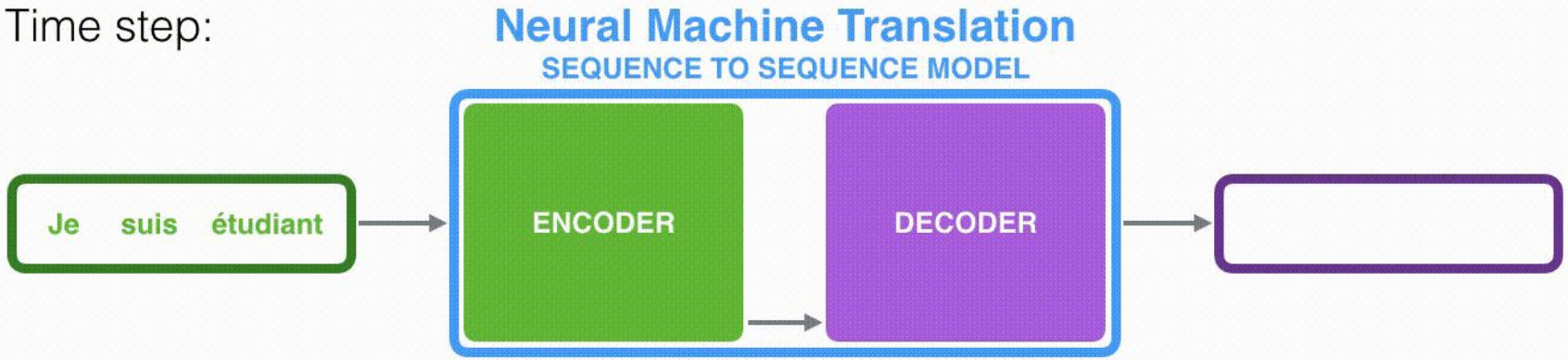
Applications of Seq2Seq

- Machine Language Translation
- Question Answering / Chatbot
- Date Formatting
- Speech to Text
- Name to Domain
- Website Summary



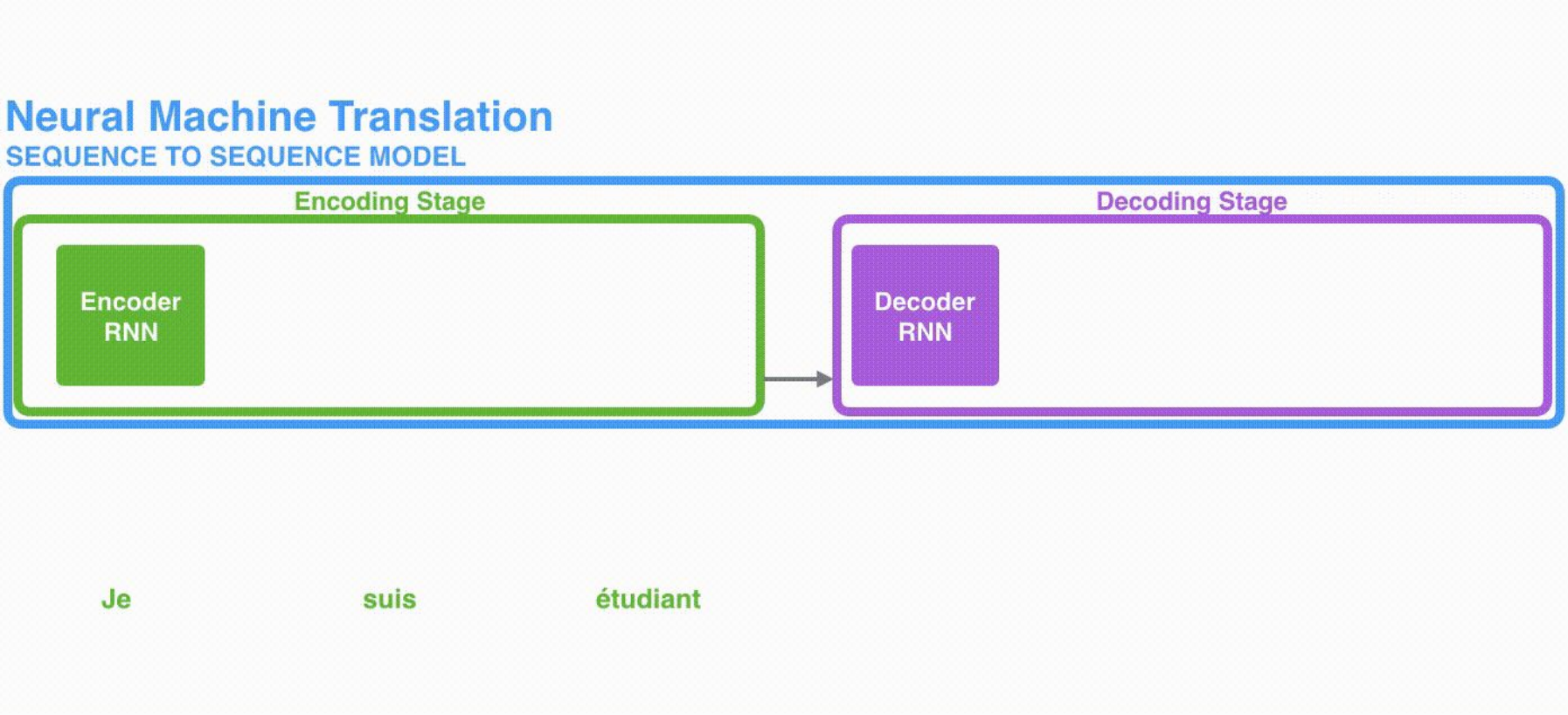
scoutbee

Seq2Seq animated



source

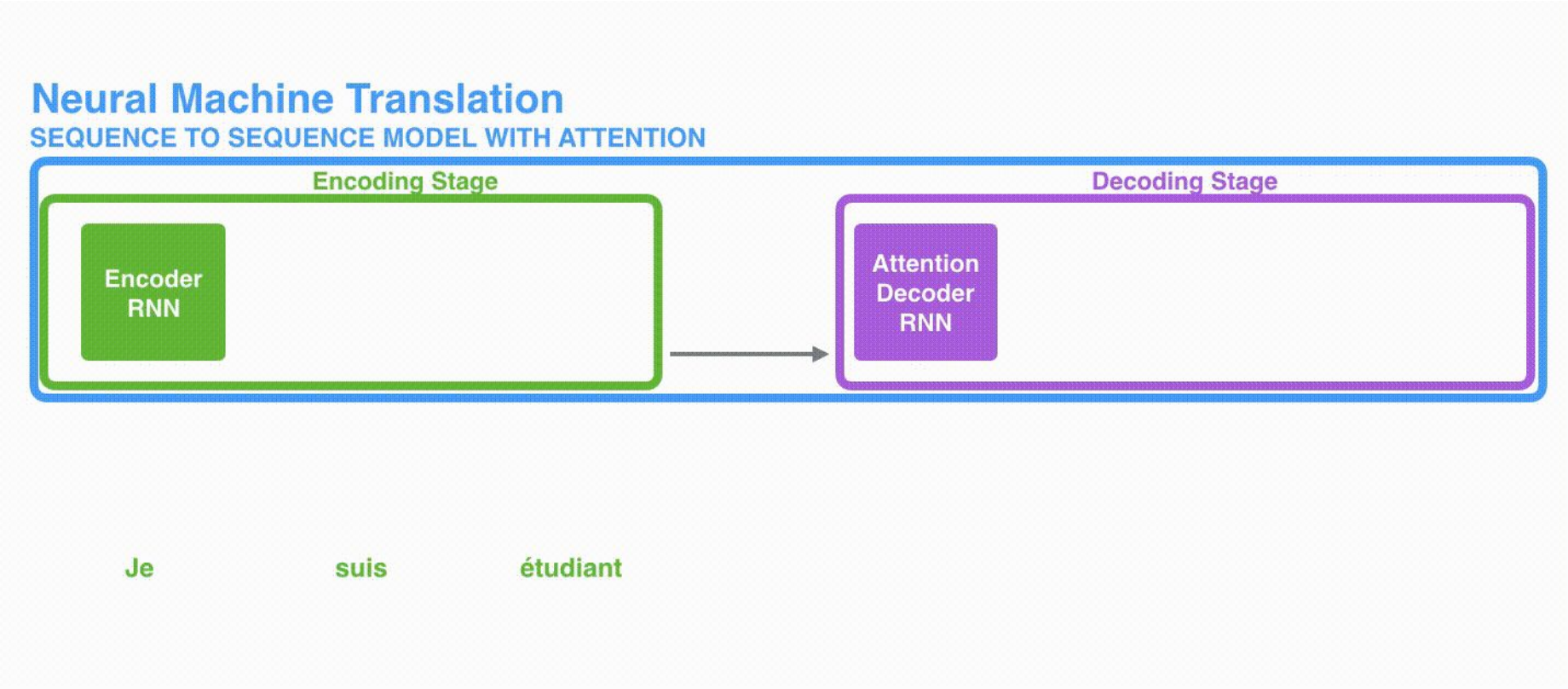
Seq2Seq animated (2)



source

scoutbee

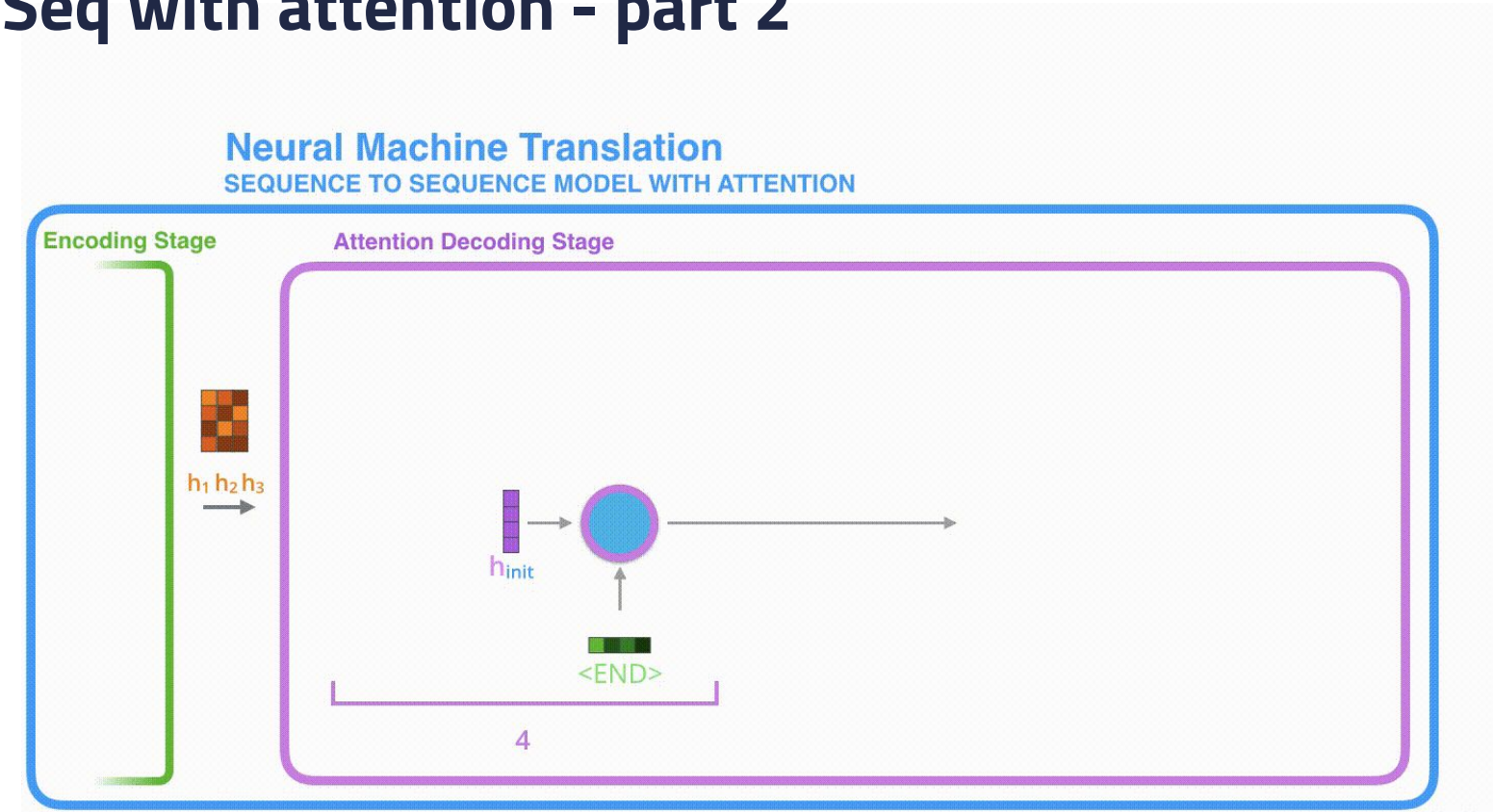
Seq2Seq with attention - part 1



source

scoutbee

Seq2Seq with attention - part 2



source

scoutbee

Encoder-Decoder Attention

Attention determines which parts of an input sequence are important for the decoder at a certain timestep

target hidden state

source hidden state

Attention weights (Alignment)

$$\alpha_{ts} = \frac{\exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s))}{\sum_{s'=1}^{S} \exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_{s'}))}$$

Context vector

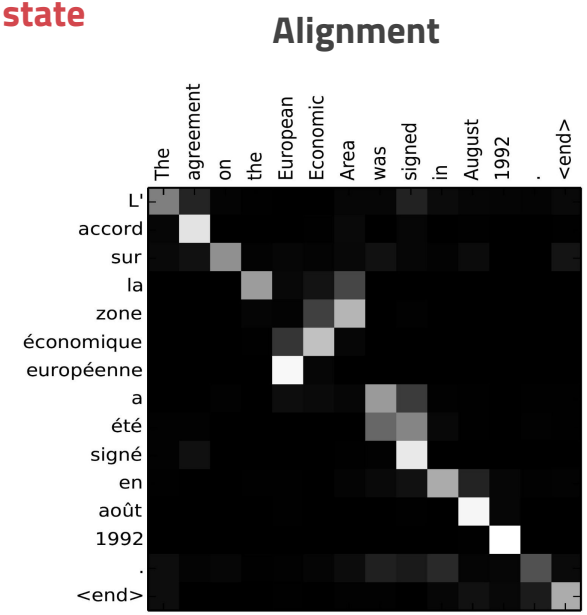
$$\mathbf{c}_t = \sum_s \alpha_{ts} \bar{\mathbf{h}}_s$$

Attention vector

$$\mathbf{a}_t = f(\mathbf{c}_t, \mathbf{h}_t) = \tanh(\mathbf{W}_c[\mathbf{c}_t; \mathbf{h}_t])$$

Prediction

$$p(y_t | y_{<t}, x) = softmax(\boldsymbol{W_s a_t})$$



scoutbee

Encoder-Decoder Attention in PyTorch

```
# Attention score (Bahdanaus 2015)
score = torch.tanh(
    self.W1(encoder_output) + self.W2(hidden_with_time_axis)
)

# Attention weights
attention_weights = torch.softmax(self.V(score), dim=1)

# Find the context vectors
context_vector = attention_weights * encoder_output
context_vector = torch.sum(context_vector, dim=1)
```

Find the family of attention score functions here

Let's ditch RNNs and just focus on Attention

Super parallelizable!

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Cheaper and faster to train!

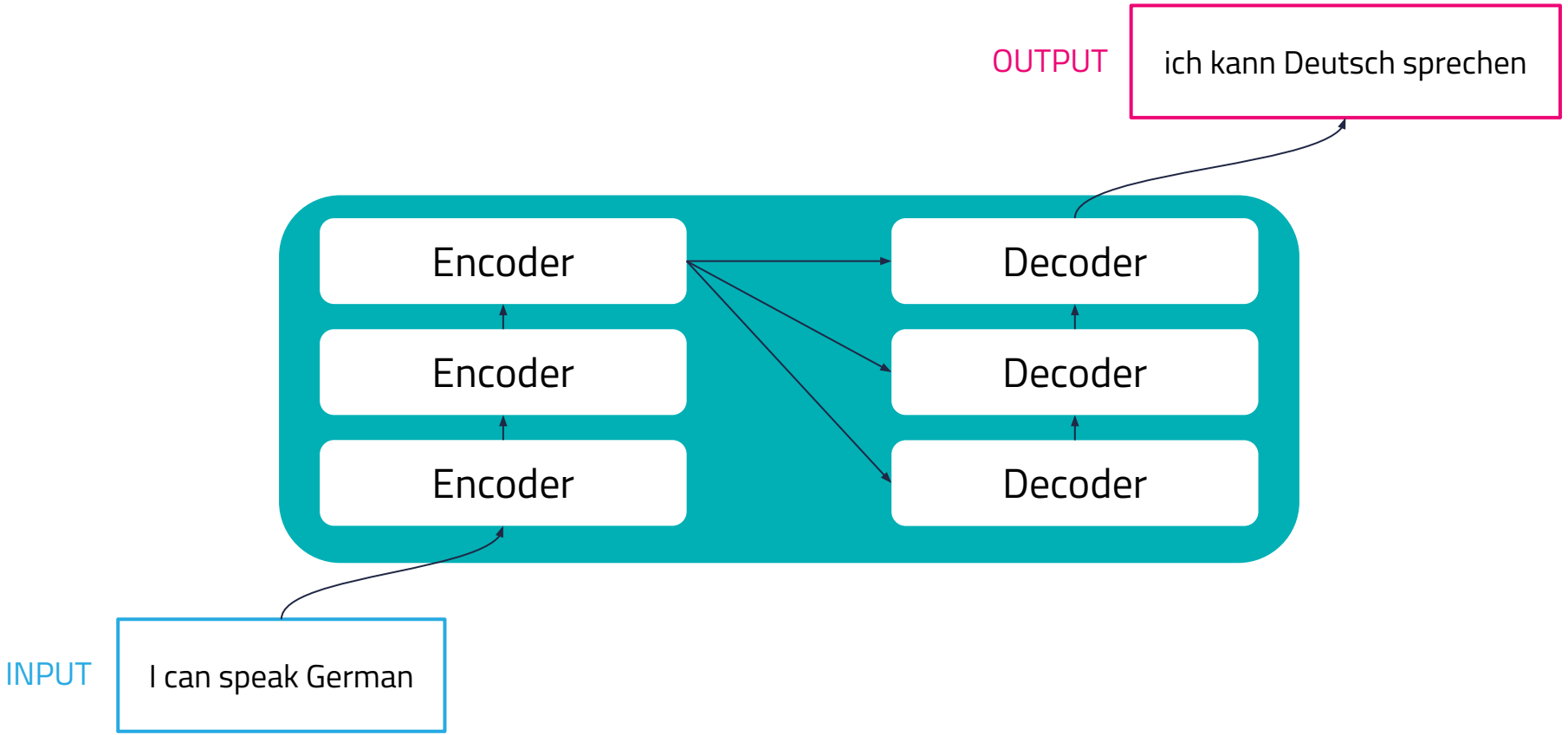
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

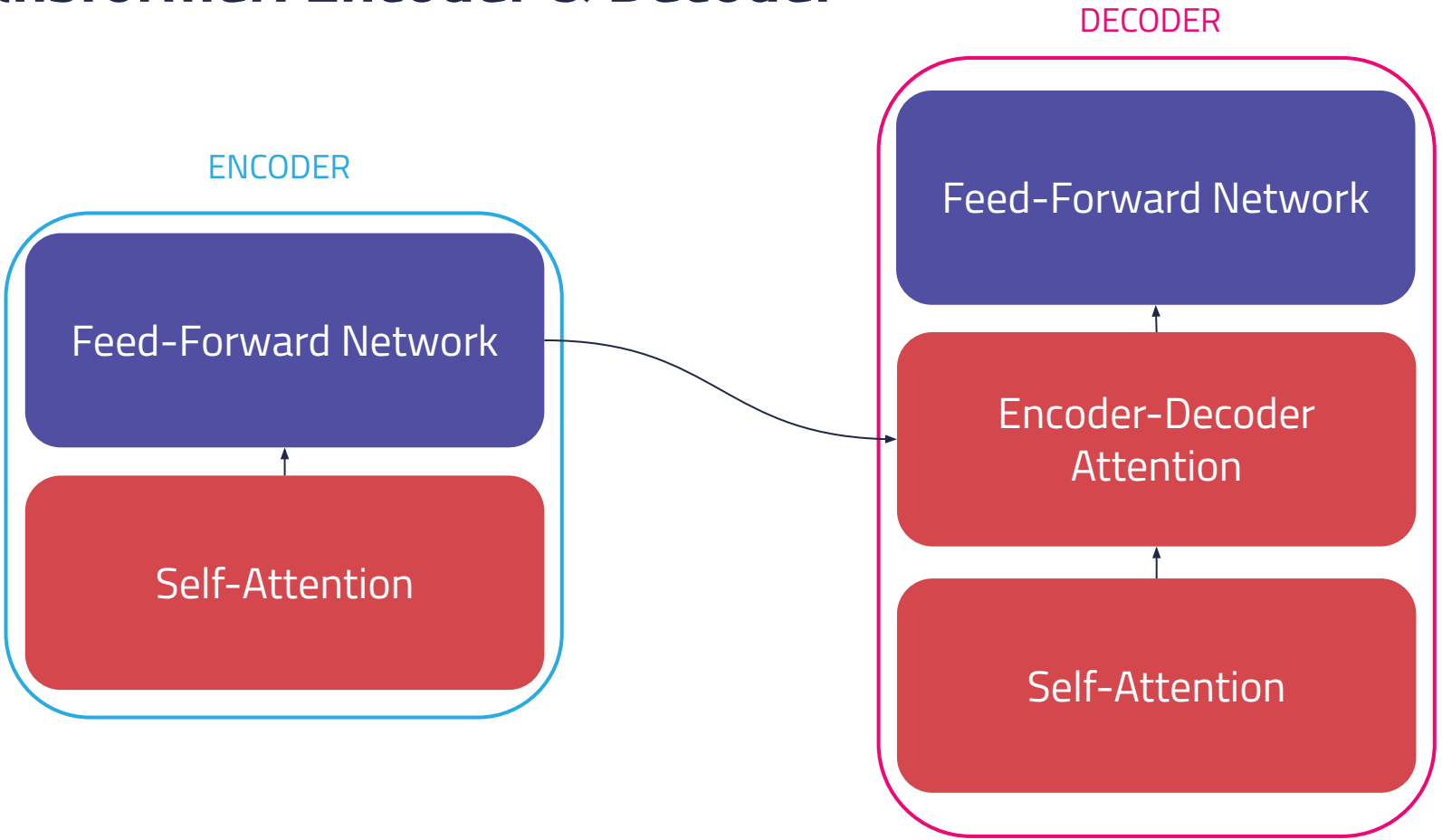
1 Introduction

scoutbee

Transformer: A stack of encoders and decoders



Transformer: Encoder & Decoder



scoutbee

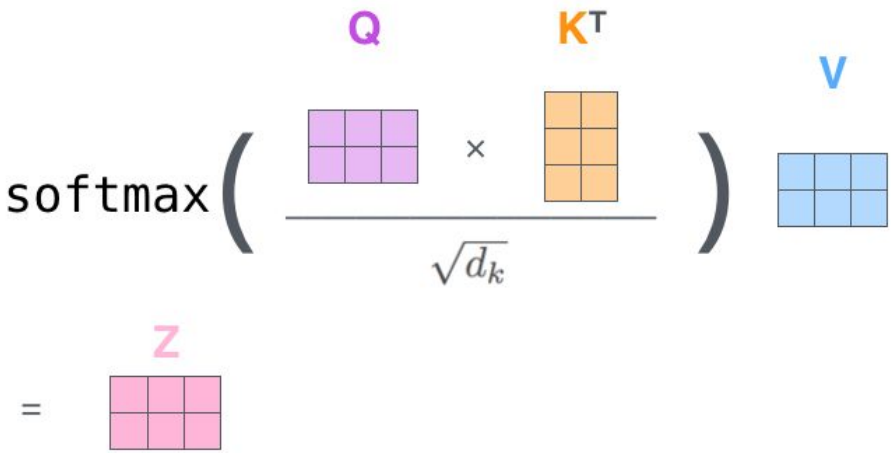
Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

seq length x embedding size

embedding size x vector size

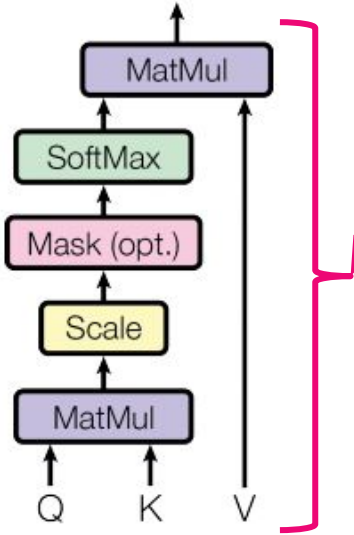
$$Q = XW^Q$$
$$K = XW^K$$
$$V = XW^V$$

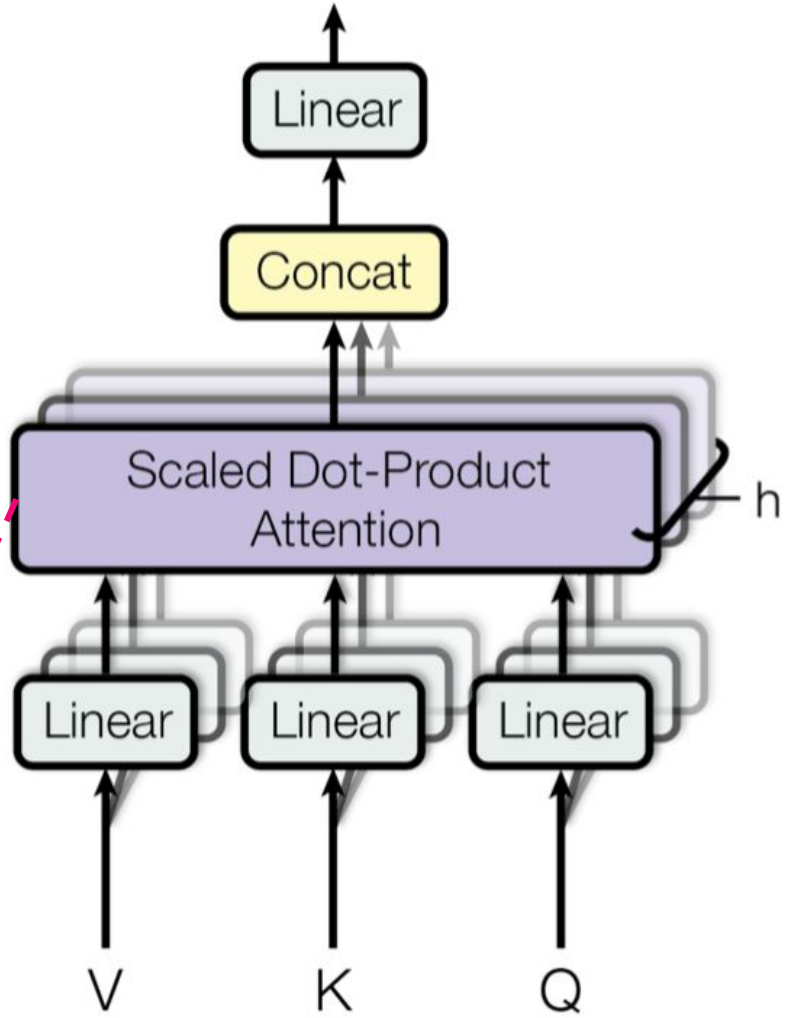


Multi-Head Self-Attention

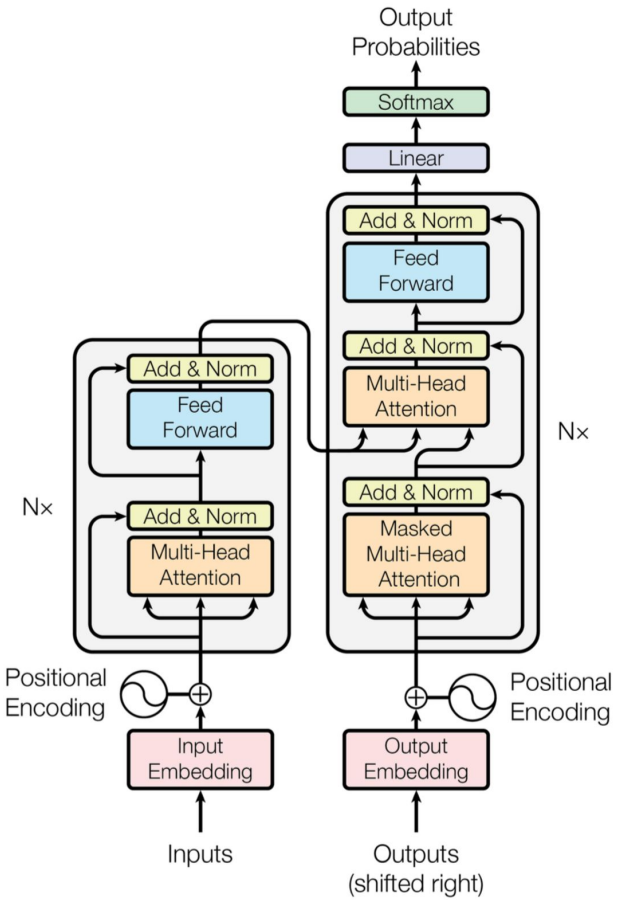
- Self-attention is calculated multiple times independently and in parallel
- The context vectors from each head is simply concatenated and linearly transformed into the right dimensions

"multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this."



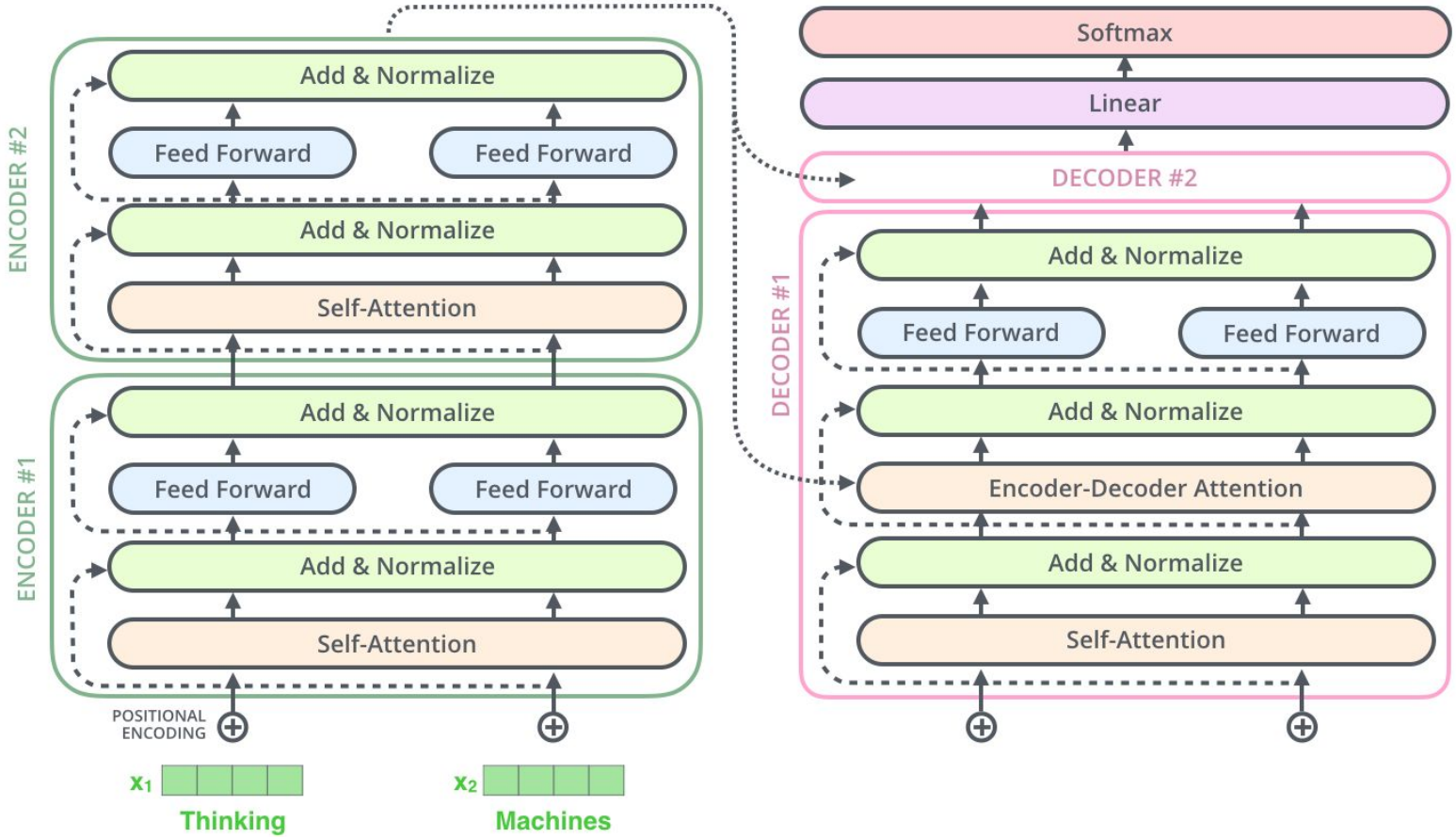


Transformer Model Architecture



scoutbee

Transformer Model Architecture



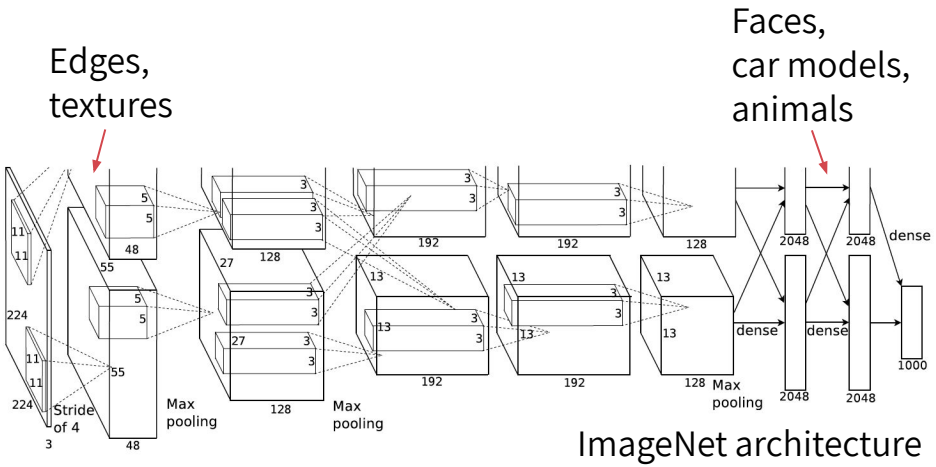
scoutbee

Seq2Seq Notebook

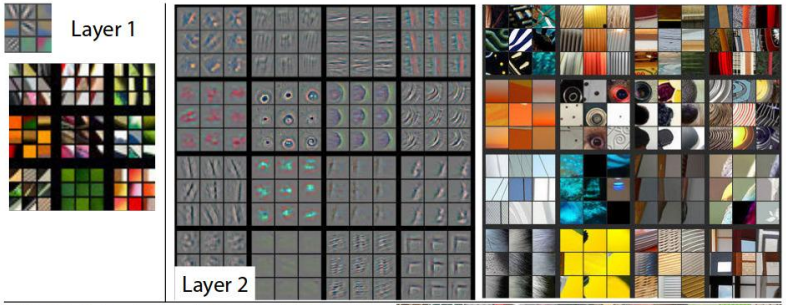
Transfer Learning

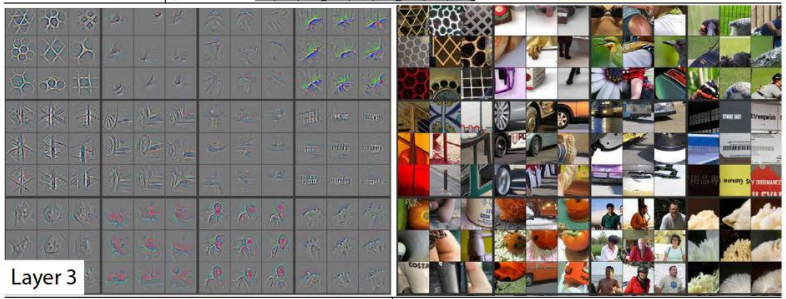
Transfer Learning for Images

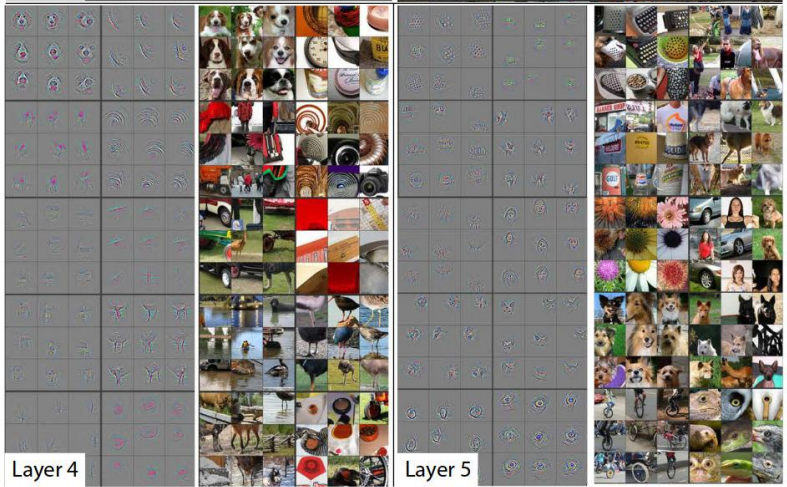
ImageNet: The first widely-used successful application of transfer learning



ImageNet architecture



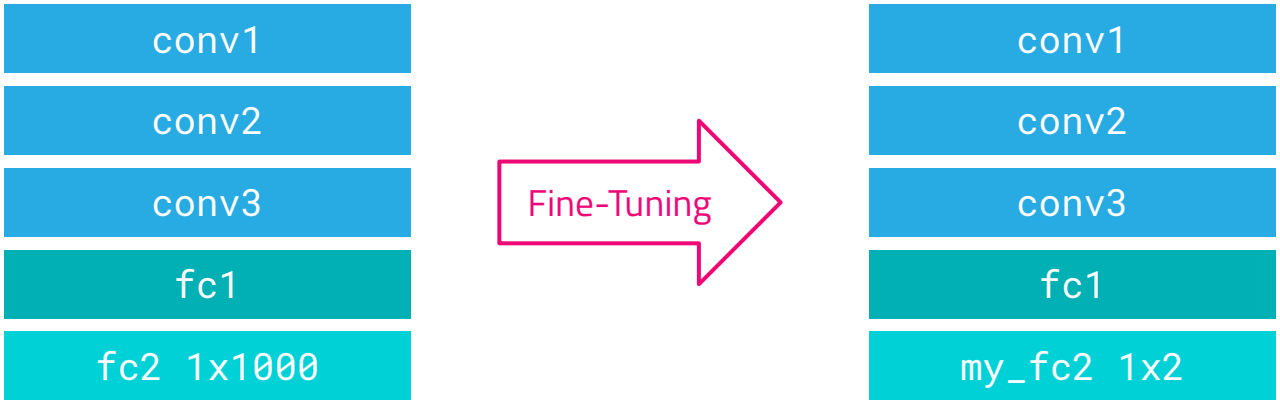




scoutbee

Fine-tuning

Fine-tuning is typically taking a large pretrained model, replacing the last layer with your own, and training it with new examples on a specific task

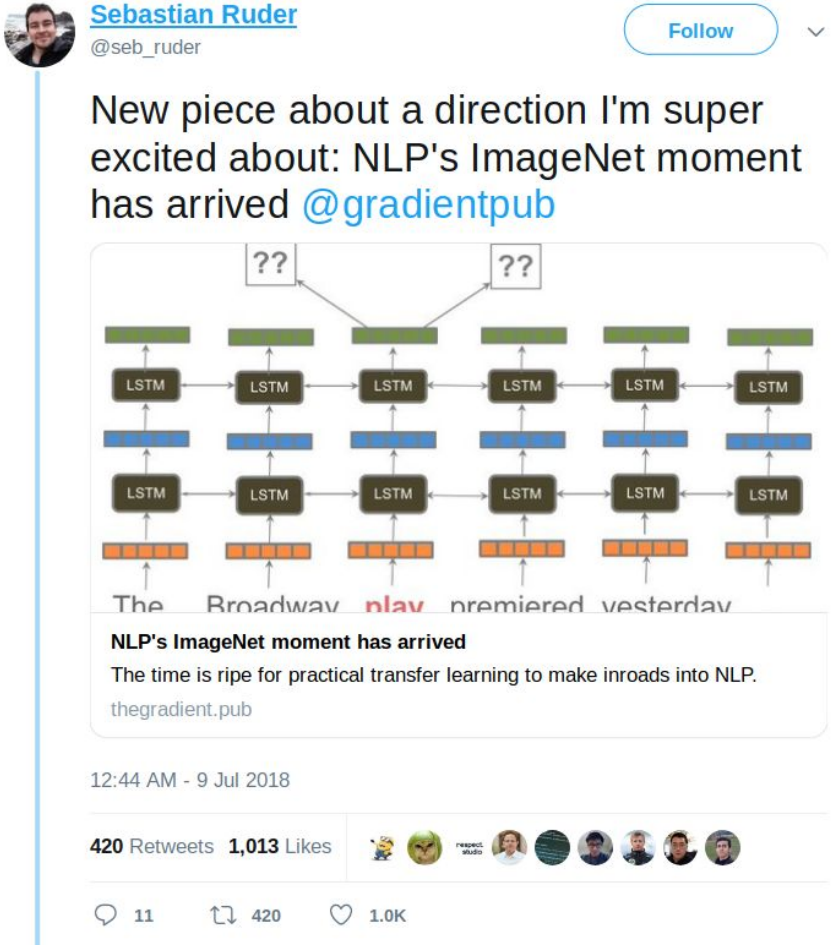



```
{
    …,
    862: 'torch',
    863: 'totem pole',
    864: 'tow truck, tow car, wrecker',
    865: 'toyshop',
    866: 'tractor',
    …,
}
```

```
{
    0: 'cat',
    1: 'dog',
}
```

scoutbee

The “ImageNet moment” for NLP

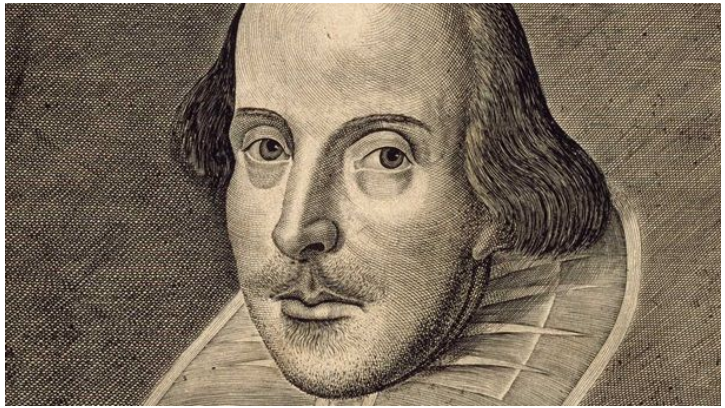


“Though these pretrained word embeddings have been immensely influential, they have a major limitation: **they only incorporate previous knowledge in the first layer of the model**---the rest of the network still needs to be trained from scratch.”

scoutbee

GPT-2

Official Blog Post



scoutbee

GPT-2 Notebook

scoutbee

But why transfer learning?

- Getting enough labeled data for a production-ready model can be prohibitively expensive or even impossible
  - Imagine a classifier for rare diseases: no way to get more labeled data!

- Take advantage of the vast amounts of unlabeled data (eg. language modeling using text found on Wikipedia or Common Crawl) with unsupervised learning

- Transfer learning can also save on training time if you don't retrain the entire network (eg. if you freeze some of the early layers)

Further resources

- Great visualizations from Jay Alammar's blog (the gifs we use are from here)
- New fast.ai course: A Code-First Introduction to Natural Language Processing
- Upcoming book by SpaCy creator: Deep Learning with Text: Natural Language Processing (Almost) from Scratch with Python and spaCy by Patrick Harrison and Matthew Honnibal
- Great blog post on RNNs from colah's blog
- Large curated list of resources: brianspiering/awesome-dl4nlp

Thank you!

Any Questions?

- Jeffrey Hsu (jeffrey@scoutbee.com)
- Susannah Klaneček (susannah@scoutbee.com)

Hands-on Exercise

link to the slides:

http://tiny.cc/988k9y

Generating Toxic Comments

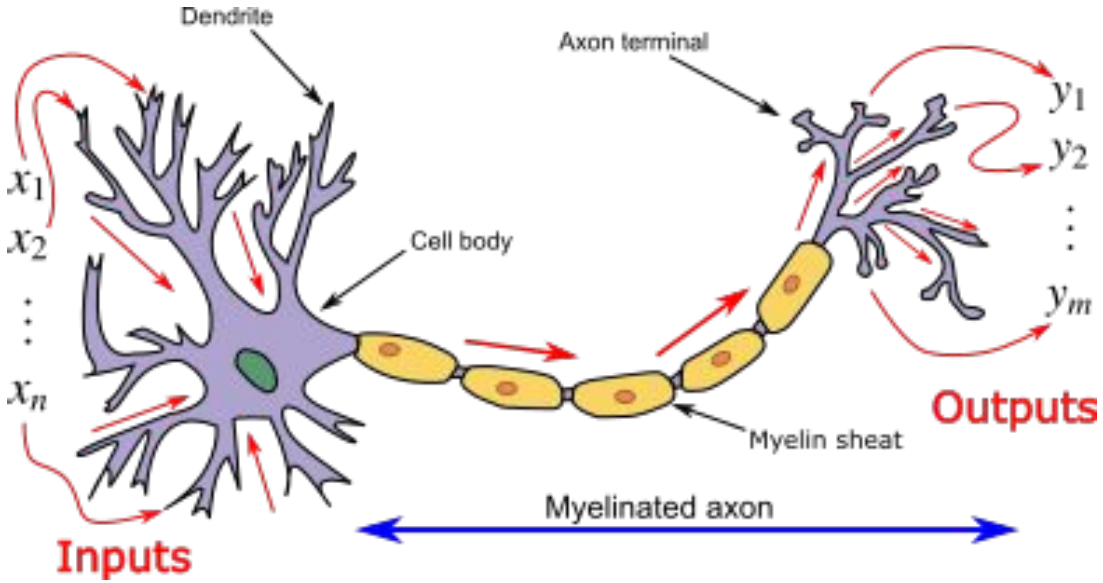
```
# 1. open up GPT-2 Notebook in Google Colab
# 2. download the file
!wget
https://raw.githubusercontent.com/ipcplusplus/toxic-commen
ts-classification/master/train.csv

# 3. load into dataframe
df = pd.read_csv('train.csv')
```

scoutbee

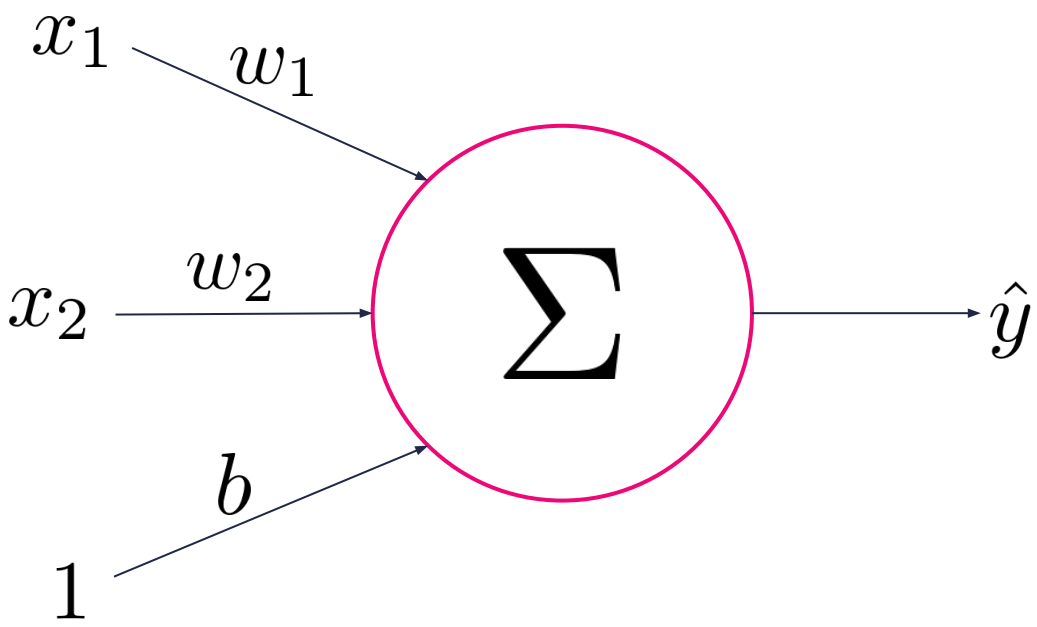
Backups

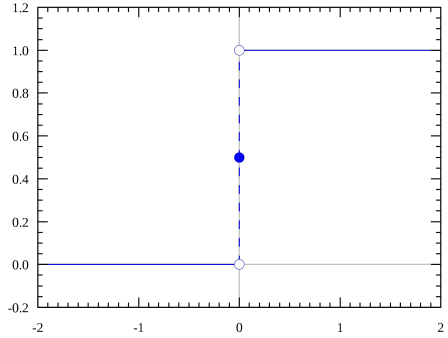
Neural Network Basics

Neural Inspiration



Perceptron

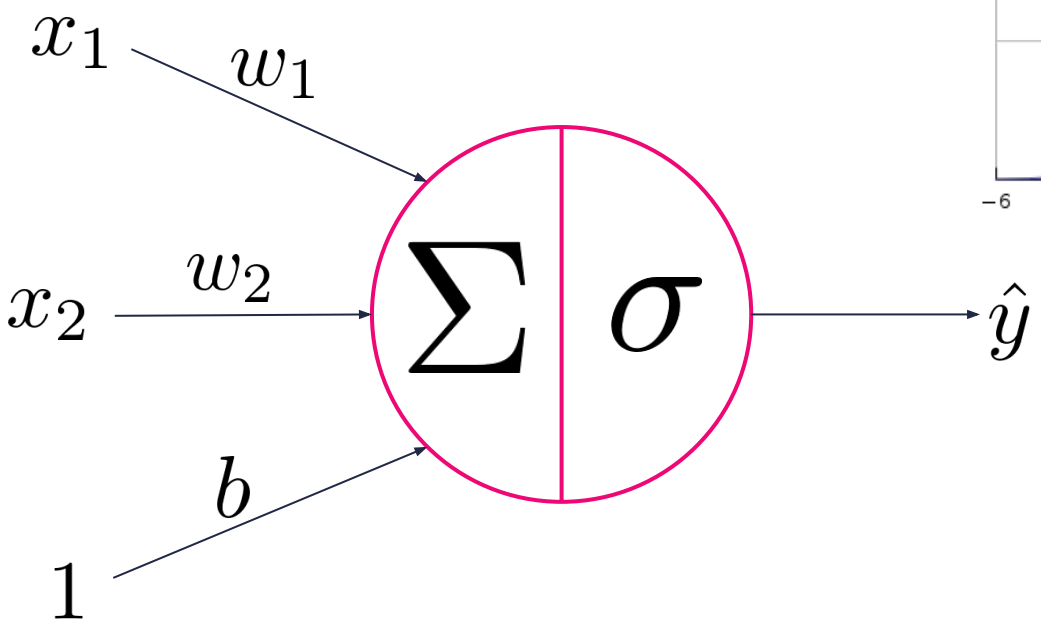


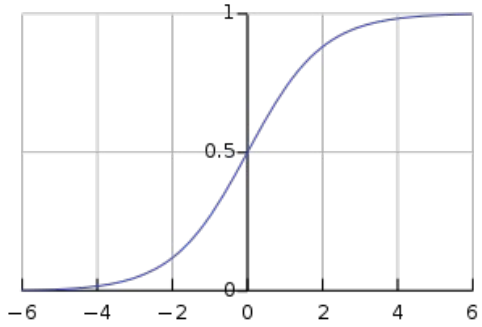


$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

scoutbee

Sigmoid neuron



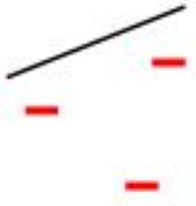


$$\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

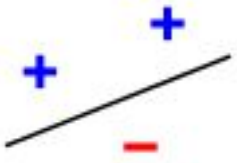
scoutbee

Linear vs. non-linear data

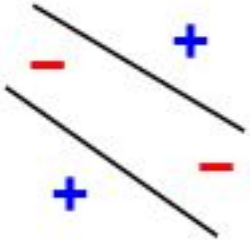
1)



2)



3)



4)