Deep learning for time series and text processing. Recurrent networks and transformers

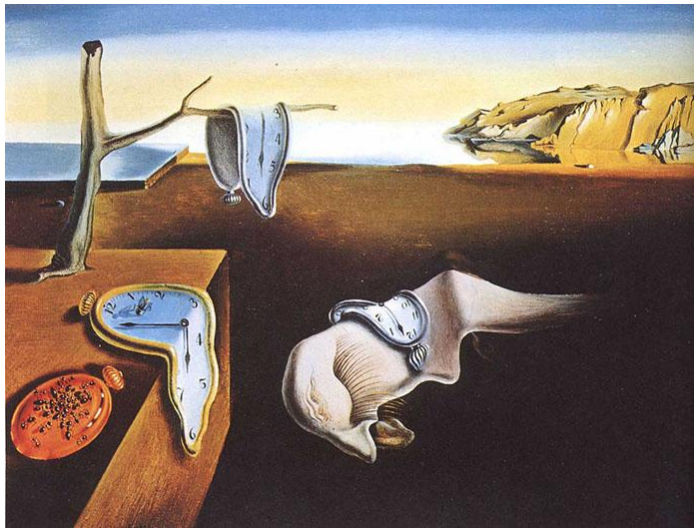
Ruxandra Stoean

Further bibliography

- A. K. Tyagi, A. Abraham, Recurrent Neural Networks: Concepts and Applications, CRC Press, 2022
- F. M. Salem, Recurrent Neural Networks: From Simple to Gated Architectures, Springer, 2022
- D. Rothman, Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP with Python, Hugging Face, and OpenAI's GPT-3, ChatGPT, and GPT-4, Packt, 2022
- S. Ozdemir, Quick Start Guide to Large Language Models: Strategies and Best Practices for Using ChatGPT and Other LLMs, Addison-Wesley, 2023
- S. Raschka, Build a Large Language Model (From Scratch), Manning, 2024

Recurrent networks

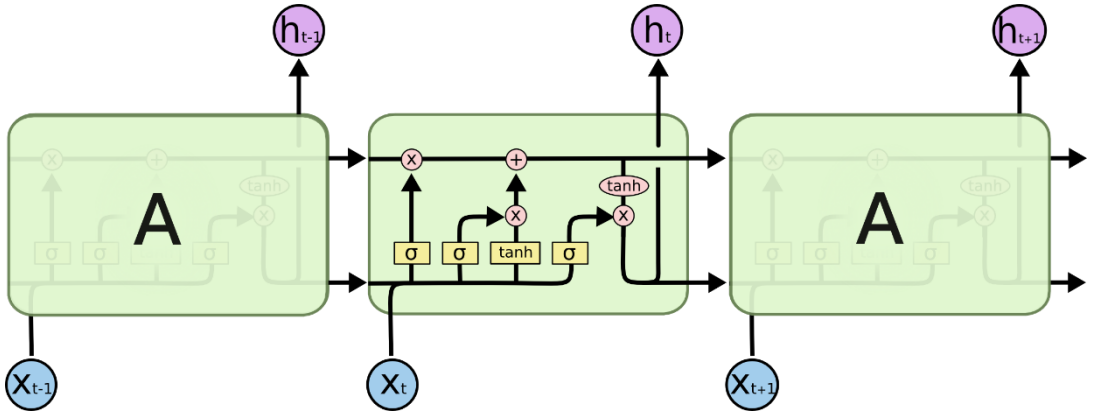
The persistence of memory



- Traditional neural networks (NN) unable to remember information collected over time
- Consequent appearance of recurrent neural networks (RNN)
  - Short-term memory
- *Long short-term memory networks* (LSTM)
- *Gated recurrent units* (GRU)

- Weights shared across time
- Loss of all time steps is the summed loss at each time step
- Backpropagation at each time step, considering the previous time steps

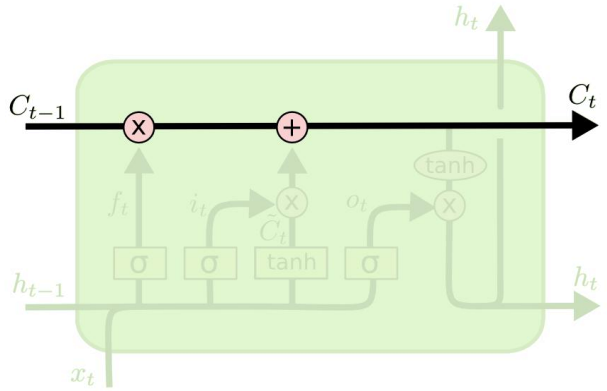
LSTM



https://colah.github.io/

- S. Hochreiter& J. Schmidhuber, Neural Computation, 1997
- Applications to speech and text prediction, video action recognition, time series prediction
- Capable of learning long-term dependencies in the data
- Chain-like structure with a repeating module of interacting neural layers
  - x_t – input, h_t – hidden state output
- Concentrate on the relevant information through its gates that decide which to keep

1. The cell state



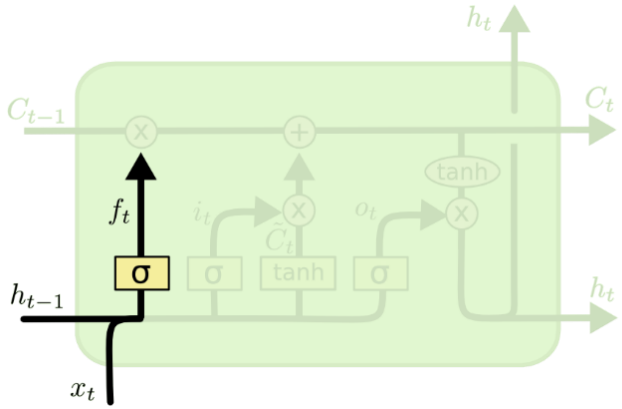
https://colah.github.io/

- Similar to a "conveyer belt" where the transport "products" = information for the network
- Information can be added to or removed from the cell state through gates.

2. Gates

- A sigmoid NN layer + a multiplication pointwise operation
  - The sigmoid has an output in the interval [0, 1] giving the amount of information that should go through
- Three gates (layers):
  - Forget gate
  - Input gate
  - Output gate

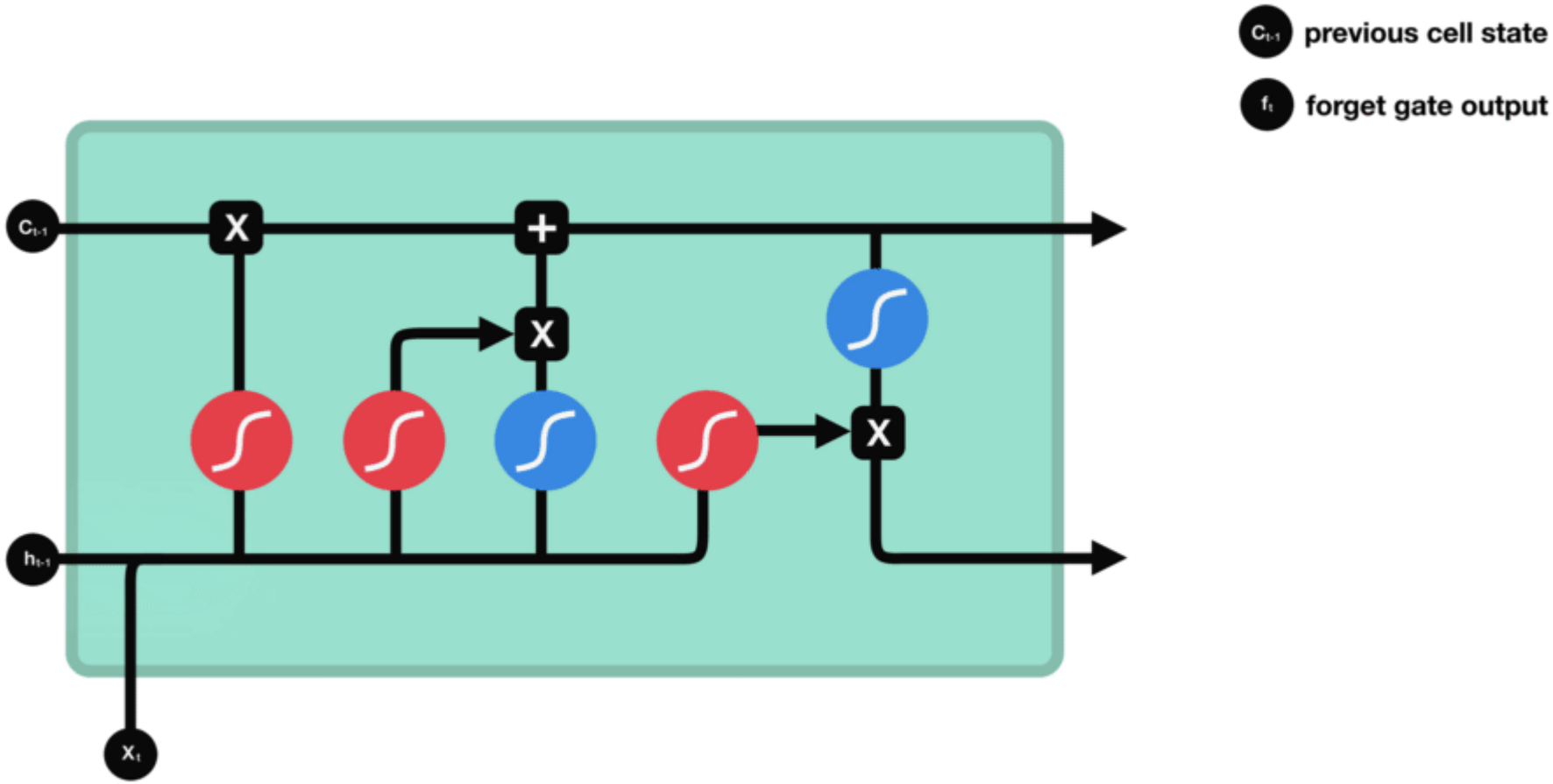
2.1 The forget gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

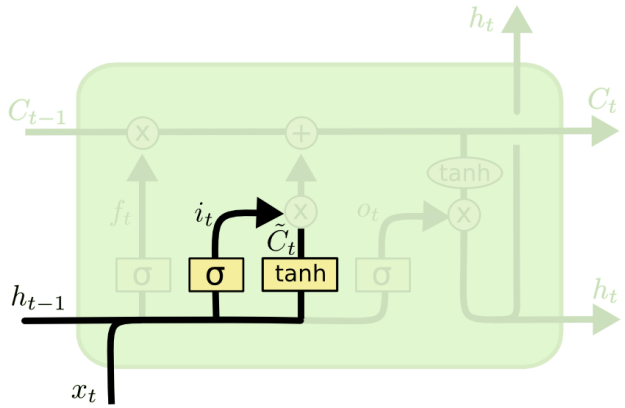
https://colah.github.io/

- Information that will be discarded from the cell state
- Takes into account:
  - h_{t-1} – the output from the previous hidden module
  - x_t – the input of the current module
- A value in [0, 1] is computed by the sigmoid layer
- This output will the used in a pointwise multiplication with the values of C_{t-1}
- It represents how much of the old state will be forgotten (0) or kept (1)



https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

2.2 The input gate

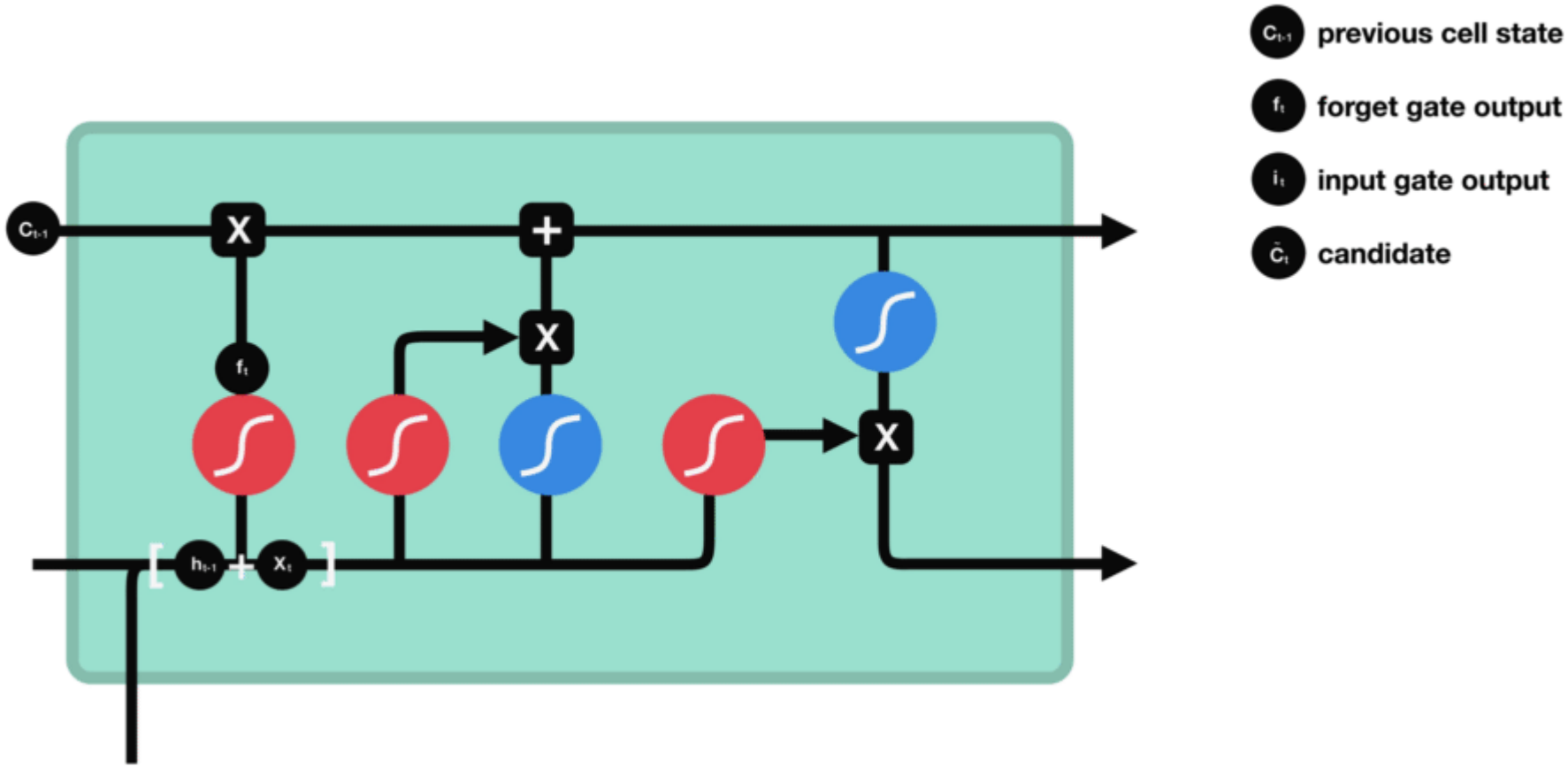


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

https://colah.github.io/

- Information that will be added to the cell state
- Two phases:
  1. The input gate sigmoid layer decides how much will be updated: important (1), not (0)
  2. A tanh layer appoints a regulated vector of candidate values to be potentially added to the cell state
- The two outputs will be combined by pointwise multiplication to decide what and how much will be updated in the new cell state



C_{t-1} previous cell state

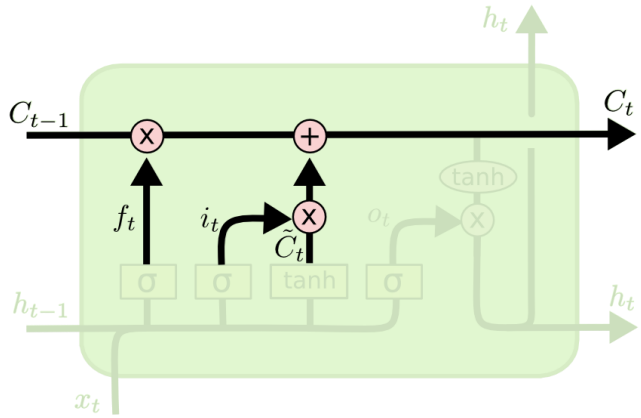
f_t forget gate output

i_t input gate output

$\tilde{C}_t$ candidate

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

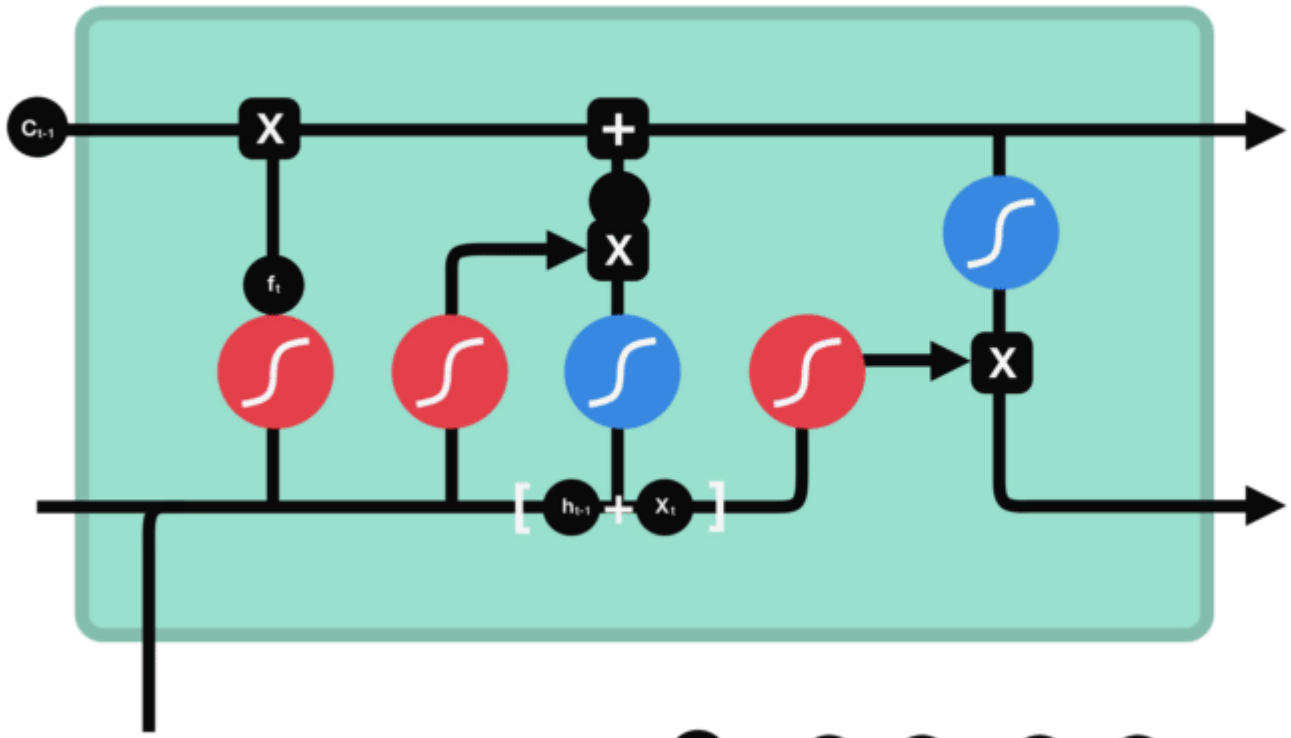
The old and the new



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/

- The previous cell state C_{t-1} is updated to the new cell state C_t
  - The values decided to be forgotten from the old state are discarded
  - The new values are added in decided amount
  - An addition pointwise operation is performed



C_{t-1} previous cell state

f_t forget gate output

i_t input gate output

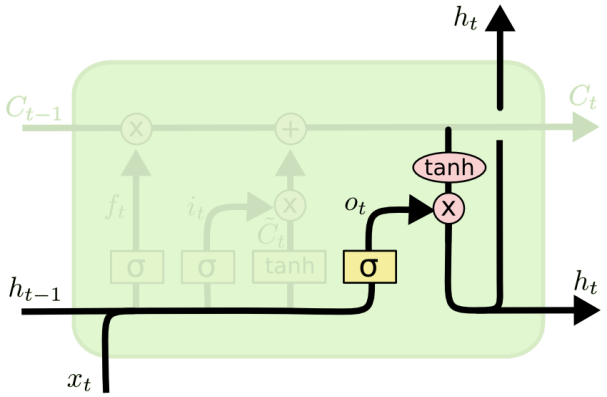
$\tilde{C}_t$ candidate

C_t new cell state

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

2.3 The output gate

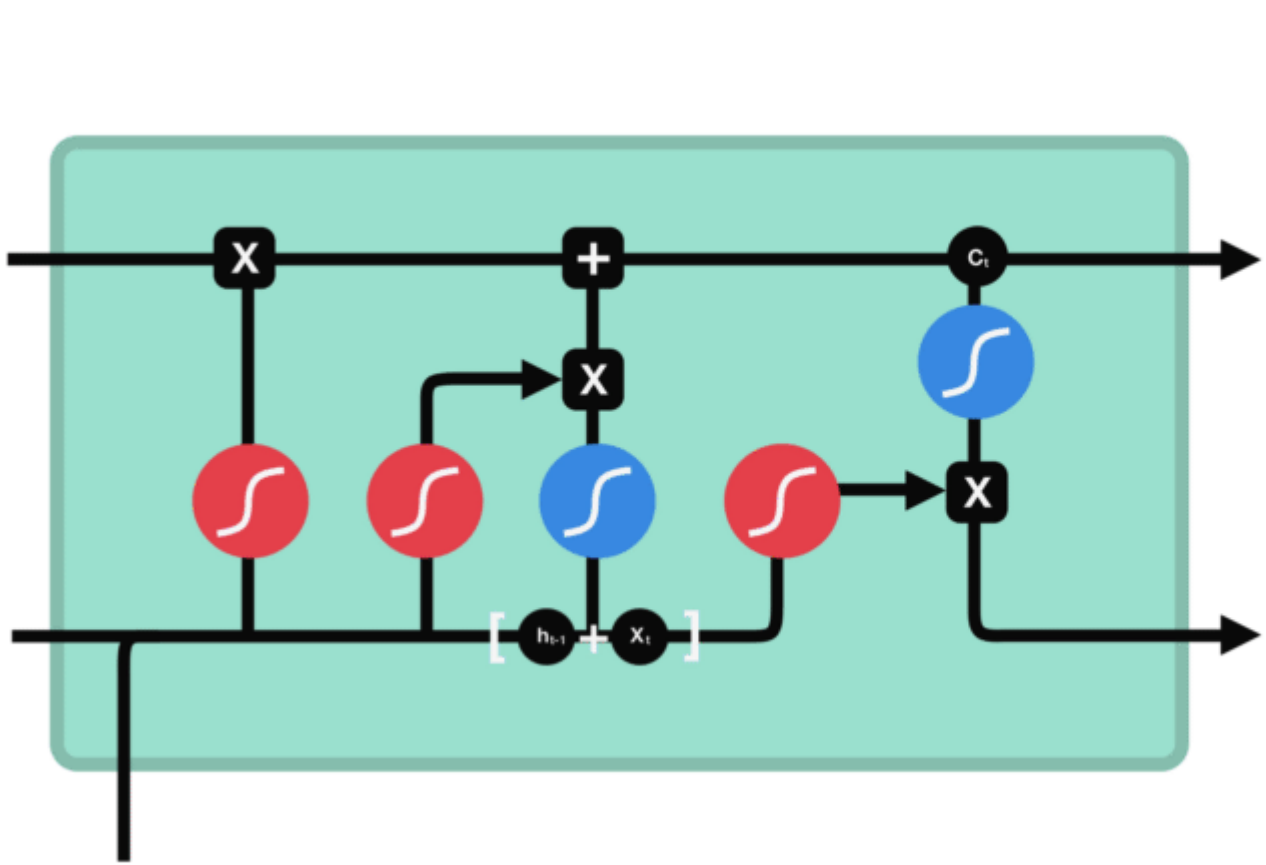


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

https://colah.github.io/

- The output will be given by:
  - A sigmoid layer decides how much will be output from the previous hidden state and current input
  - The values of the cell state are scaled between -1 and 1 by a tanh layer to regulate the output
  - The two are combined through a multiplication pointwise operation to give what and how much will be output as the new hidden state
  - The new cell state and the new hidden state go to the next time step



C_{t-1} previous cell state

f_t forget gate output

i_t input gate output

$\tilde{C}_t$ candidate

C_t new cell state

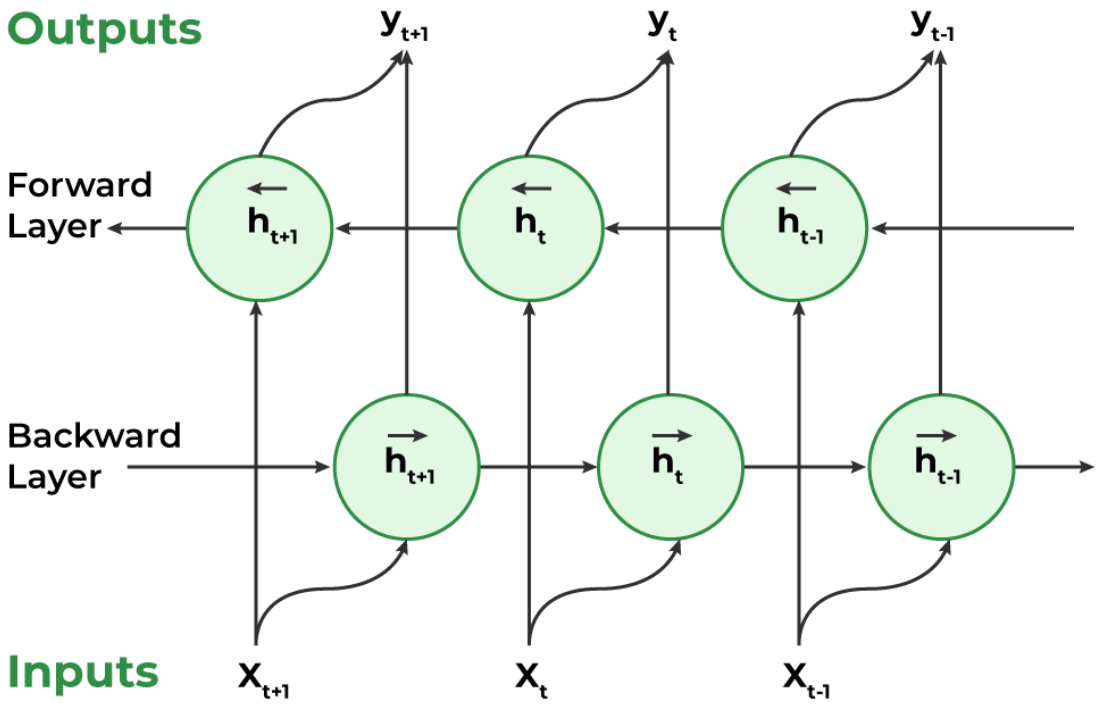
o_t output gate output

h_t hidden state

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

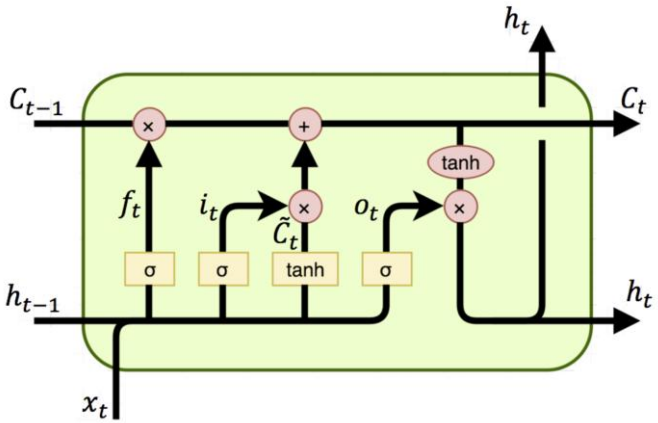
Bi-directional LSTM

- Accounts for past and future information
- Forward
  - Update based on the current input and the prior hidden state at each time step
- Backward
  - Update based on the current input and the hidden state of the next time step

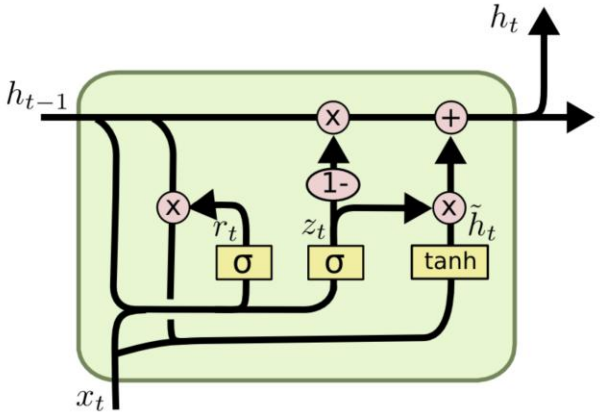


https://www.geeksforgeeks.org/bidirectional-recurrent-neural-network/

GRU



LSTM



GRU

https://colah.github.io/

- Cho et al, EMNLP, 2014
- Merges cell state and hidden state
- Two gates:
  - Reset gate
    - Decides the information to be discarded
  - Update gate
    - Combines the forget and input gates of LSTM
    - Decides the information to be discarded and the one to be added
- Simpler, faster model than LSTM

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

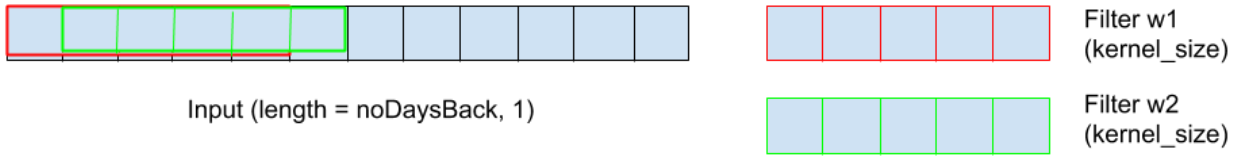
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

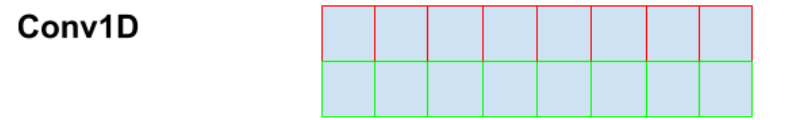
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

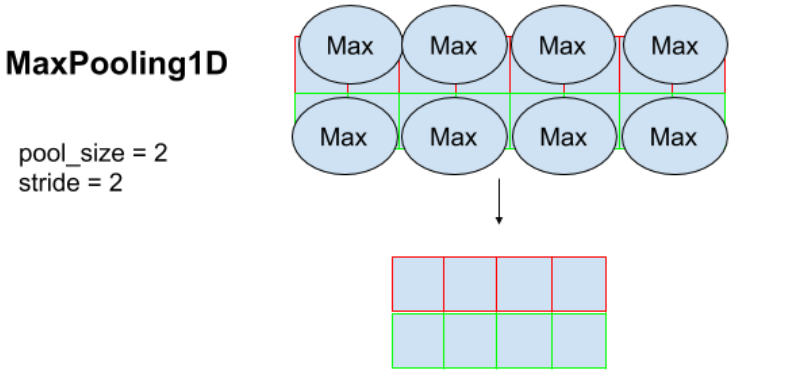
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

CNN 1D

- Transformation from 2D to 1D







Aspects related to text processing applications

- Embedding layer
  - Turns words into vectors
  - Semantic relation between words modelled as vector geometry
- Classical language models
  - Output: a character sequence with highest probability
  - Metric: perplexity = exponentiated average negative log-likelihood of a sequence
- Machine translation
  - Output: another character sequence
  - An encoder-decoder architecture
  - Metric: Bleu score = similarity based on n-gram between generated and reference text
- Performance decreases with longer sequences
- Original solution for RNN: skip-connections between encoder and decoder states

Next solution: Attention

- Or attention to context (important parts of the input)
- Alignment scores give the alignment of the sequence input Keys (encoded hidden states) to the Query output (previous decoder output)

$$c_{q,k_i} = q \cdot k_i$$

- A softmax is applied to the scores -> weights are obtained

$$\alpha_{q,k_i} = \mathrm{softmax}(e_{q,k_i})$$

- The weights are multiplied with the sequence input Values and provide the context vector

$$\mathrm{attention}(q, K, V) = \sum_i \alpha_{q,k_i} v_{k_i}$$

Self-Attention

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

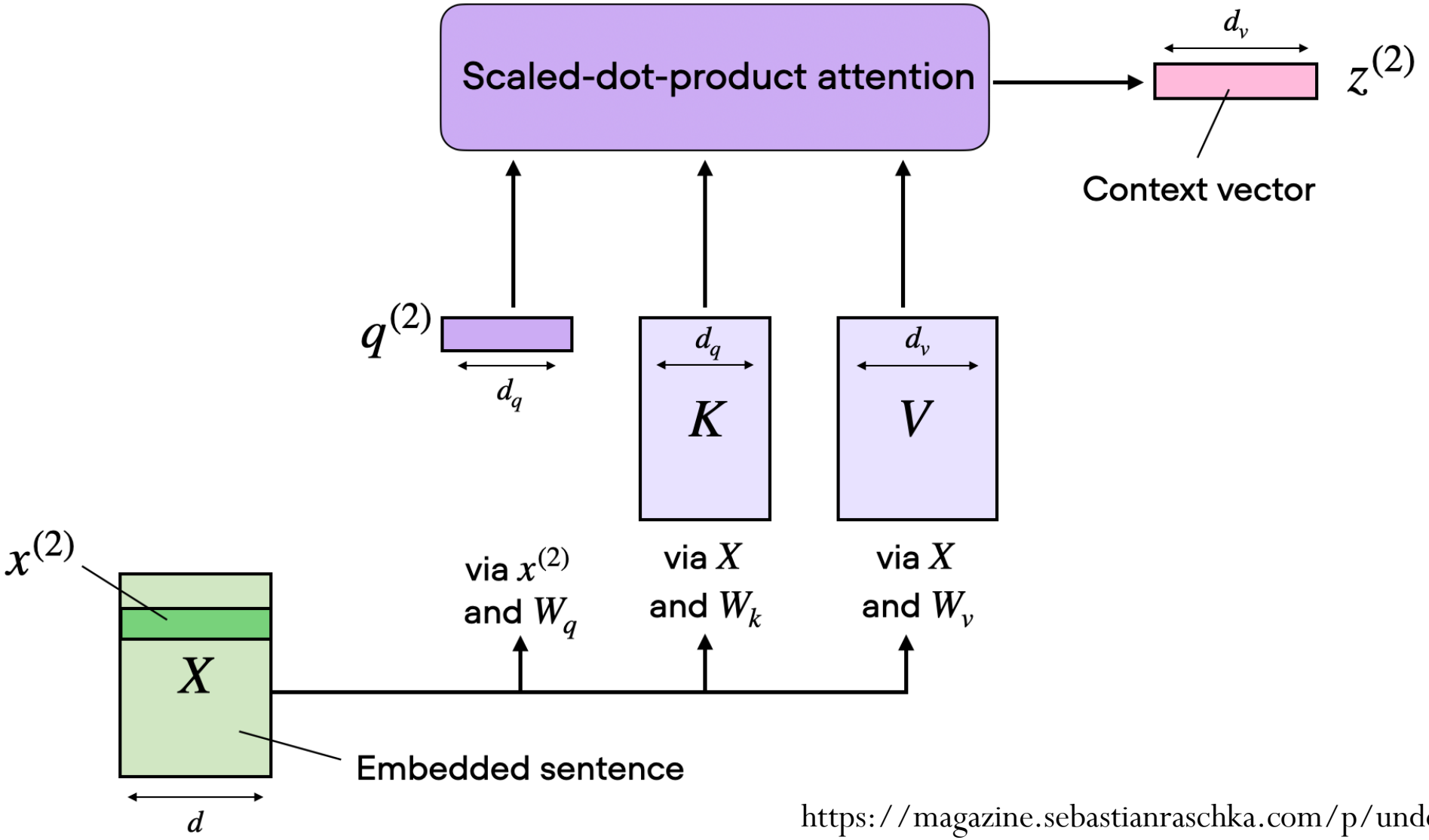
- Also called scaled dot-product attention
- Work with one, same input sequence
- Weight matrices for query, key and values – model parameters
- Inputs are projected by these matrices into Query, Key and Value
- Input sequence of length T
- Scaled by the size of the q and k vectors, d_k
  - Prevents attention weights to be at extremes (too small or too large)

$$q^{(i)} = x^{(i)}W_q$$

$$k^{(i)} = x^{(i)}W_k$$

$$v^{(i)} = x^{(i)}W_v$$

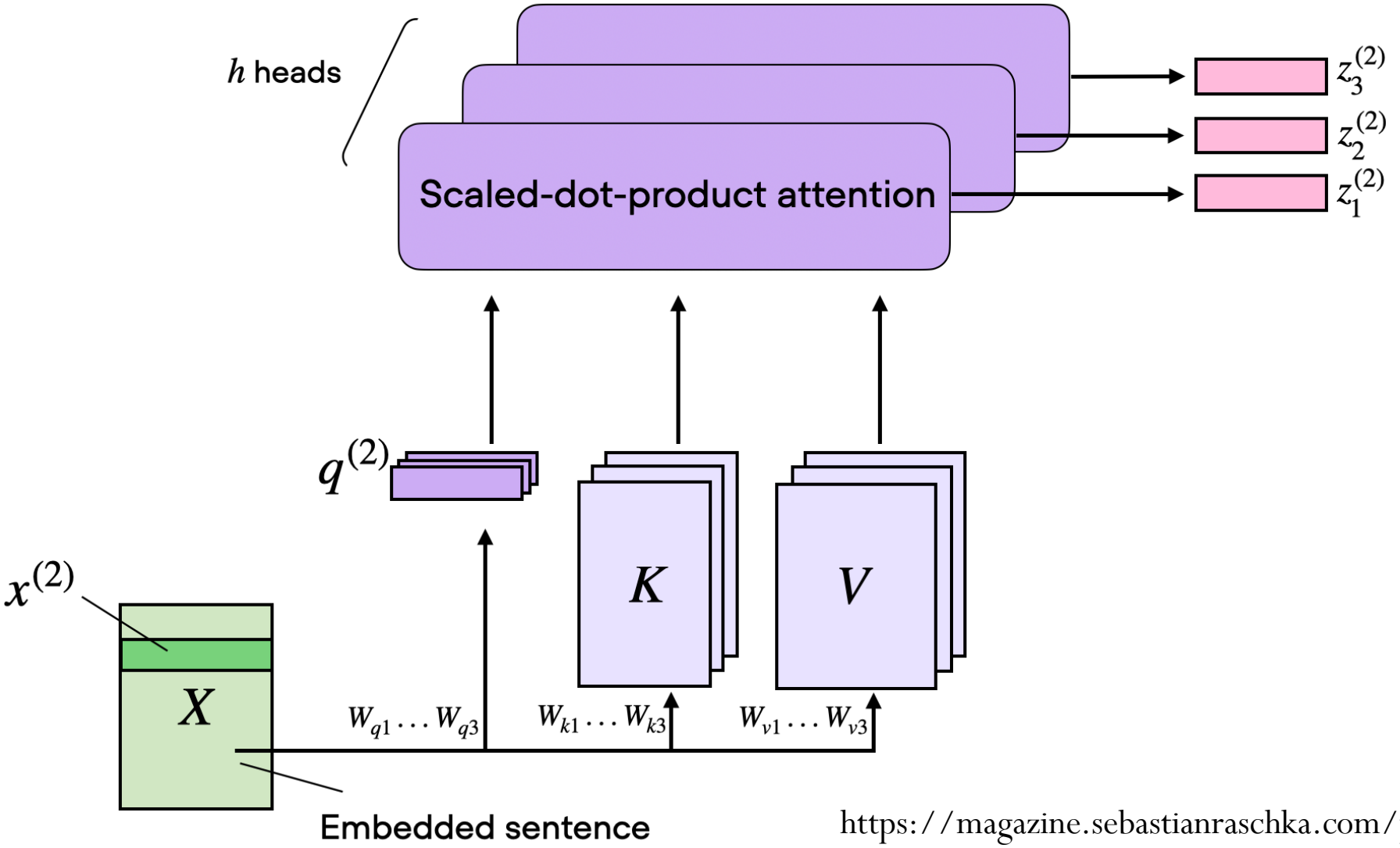
Self-Attention



https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention

Multi-head attention

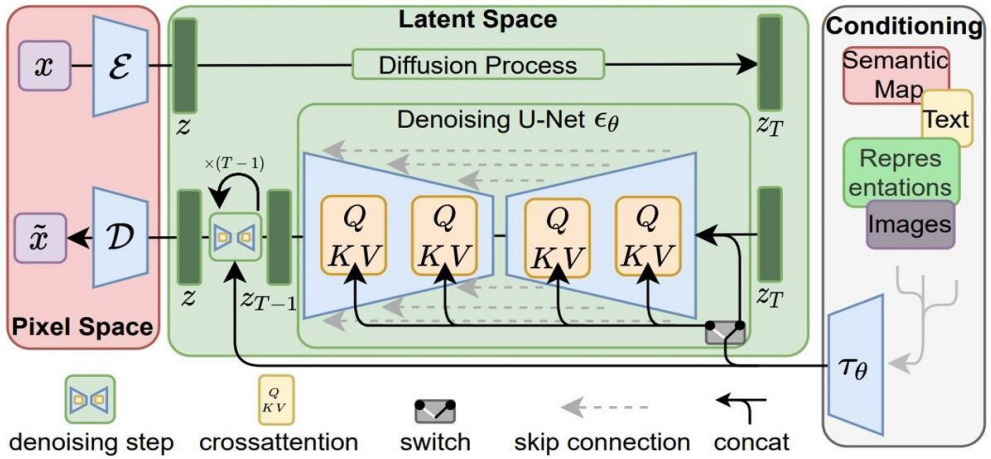
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention

Cross-attention

- Self-attention
  - One, same input sequence
- Cross-attention
  - Two different input sequences
  - Language transformer: Queries from the decoder, the Keys and Values from the encoder
  - Also saw it under Stable Diffusion



https://arxiv.org/pdf/2112.10752

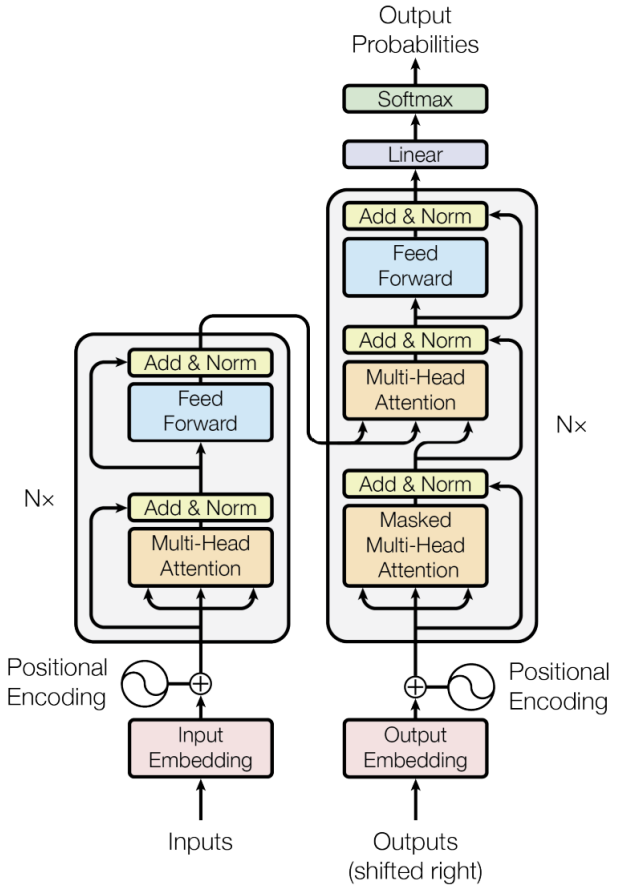
Causal Self-Attention

- Also called masked attention
- Outputs for a position in a sequence are based only on the known outputs at preceding positions and not on future positions
- Prediction for next word depends only on the previous words and not on the subsequent ones
- Hides future input elements

Transformers

Transformers

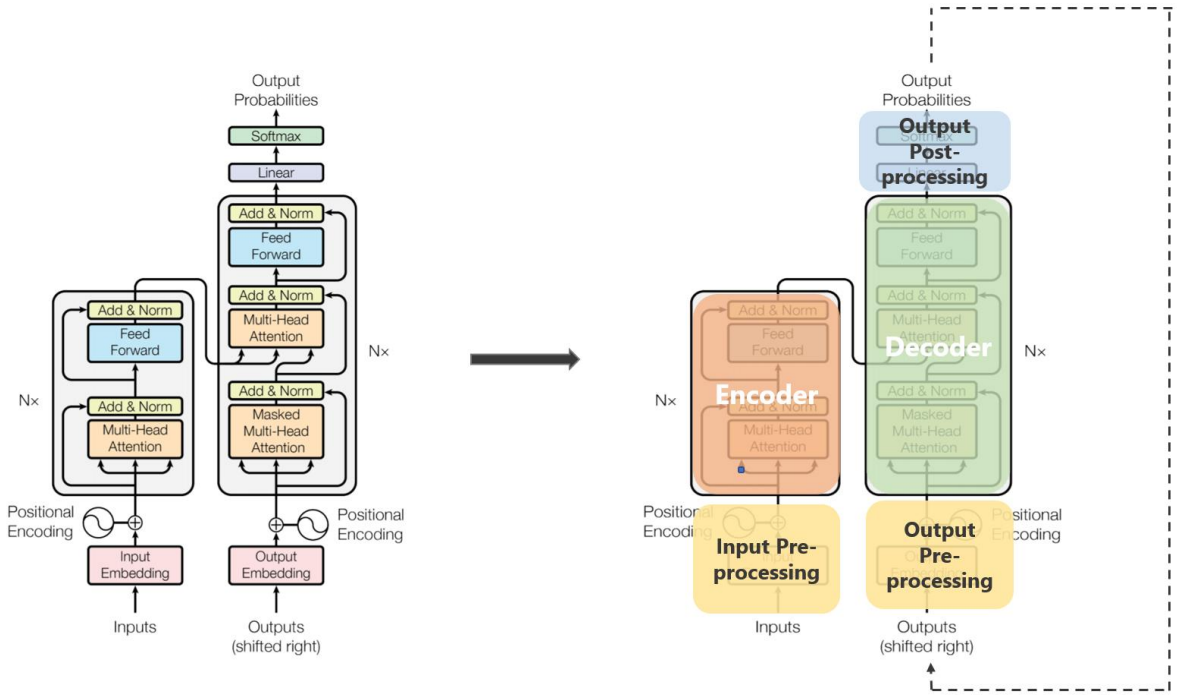
- When RNN were discarded
- Encoder Self-Attention
  - Query - the current word vector in the input
  - Key (Value) - the vectors in the input
- Decoder Self-Attention
  - Query – the current word vector in the output
  - Key (Value) - the vectors in the output
- Encoder-Decoder Attention (Cross-attention)
  - Query - the output of the masked attention in the decoder
  - Key (Value) - the hidden state vectors in the encoder



Attention is all you need: https://arxiv.org/pdf/1706.03762.pdf

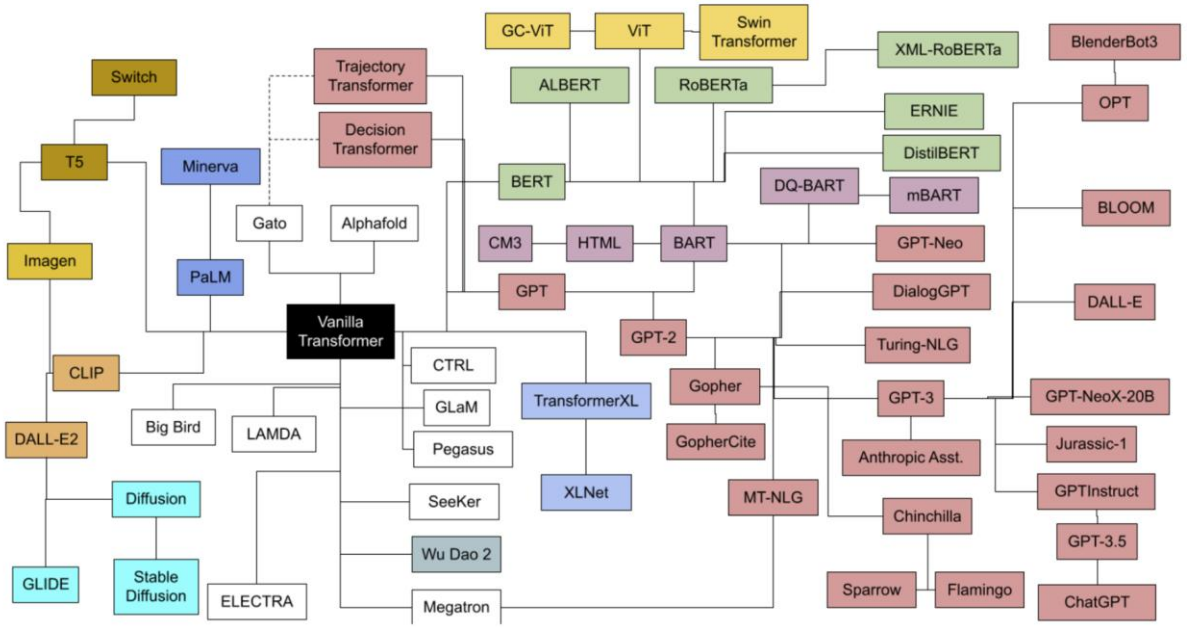
Transformers flow

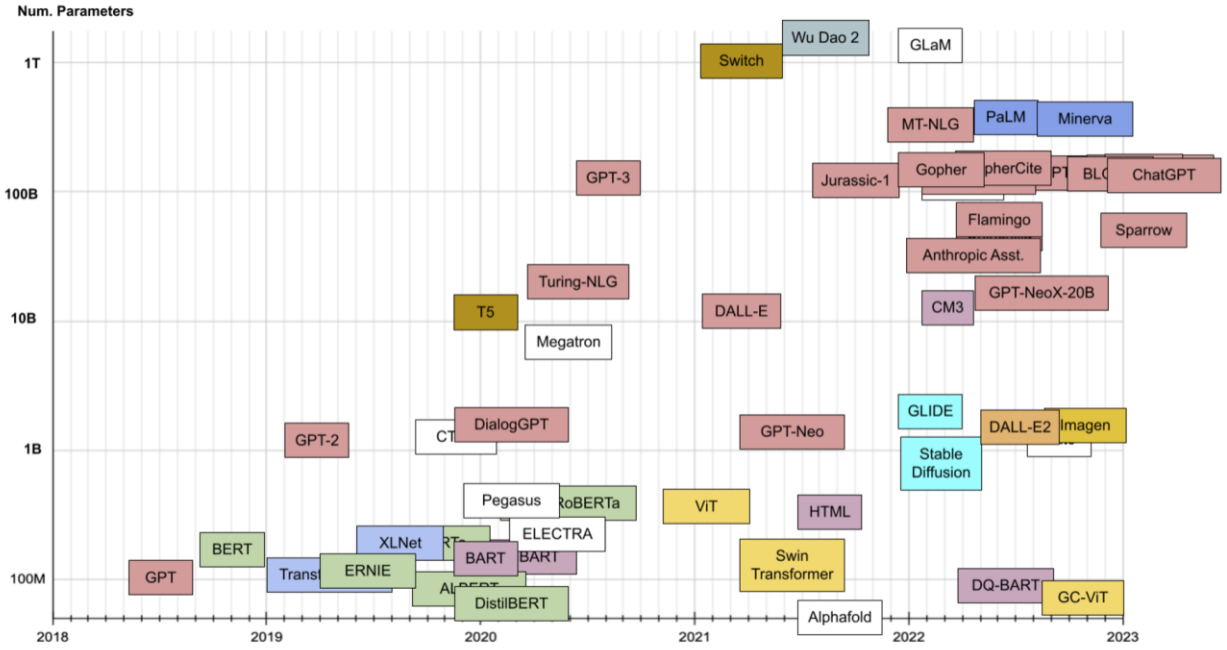
- Embedding space – map words to vectors
- Positional encoder – vector encoding of position in sentence -> context
- Encoder
  - Self-Attention layer -> attention vectors
  - Reshape for next block through feedforward NN
- Decoder
  - Self-Attention layer
  - Cross-attention block -> relationship between both sequences
  - Feedforward NN
- Linear layer = feedforward layer to expand dimensions for the decoder environment
- Softmax for output probabilities



https://towardsdatascience.com/transformers-89034557de14

Transformers family tree



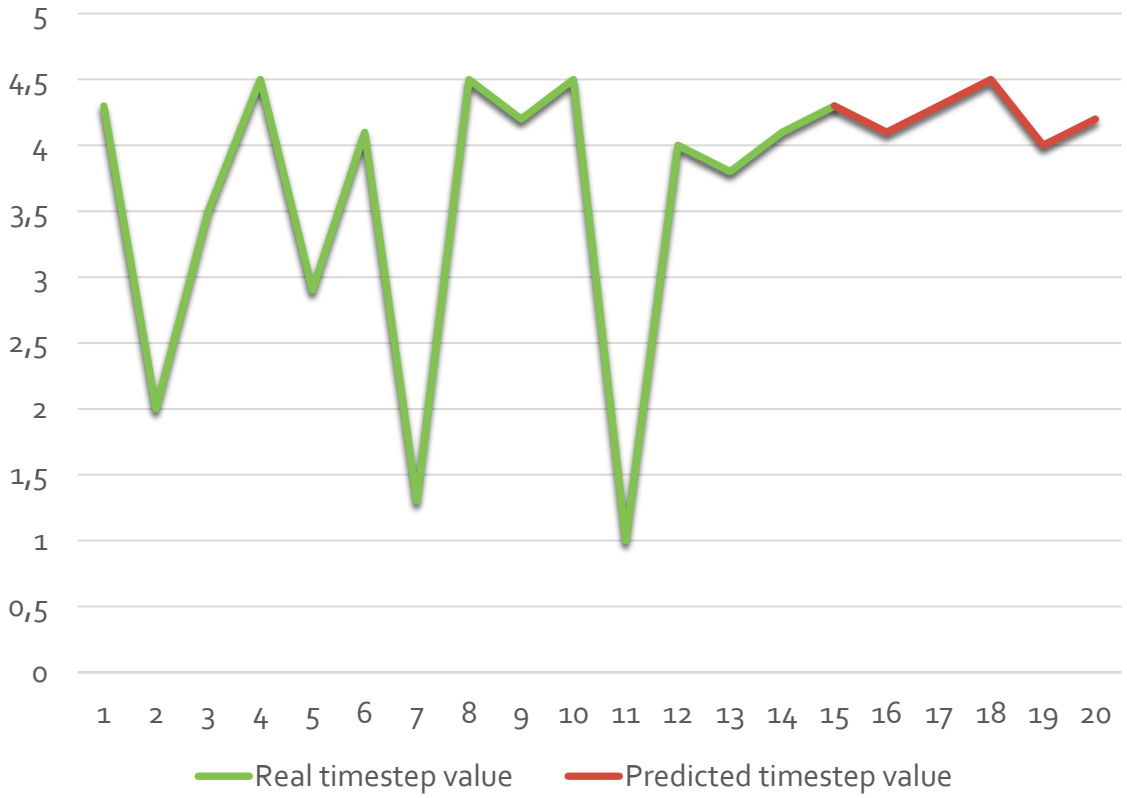


medium.com

LSTM for time series prediction

Time series

- Goal:
  - Predict the value at time T
  - Based on the data from T-N days ago
  - N – the window length



Time series prediction. Guidelines

1. Read the data
2. Split into training - test
3. Take only the predictors (columns) of interest.
4. Scale the values
5. From time T = N + 1, reshape the training data into records:
    1. An attribute set with the values from T-N, …, T-2, T-1 days
    2. The dependent variable – the price at time T

LSTM Guidelines

6. Build the stack of LSTM layers
    1. The input shape in the form (M, N, P)
    2. M is the number of instances in the training collection
    3. N is the window length
    4. The number of predictors taken into account (i.e. usually P = 1)
    5. Specify the number of units = the size of the hidden state (h_t)
7. Add a fully connected layer to output the value for the next day
8. Add dropout in between layers
9. Choose optimizer and specify loss = mean squared error
    - accuracy: categorical predictor or task reformulated to predict not next value, but a category or high/low

LSTM for stock price prediction

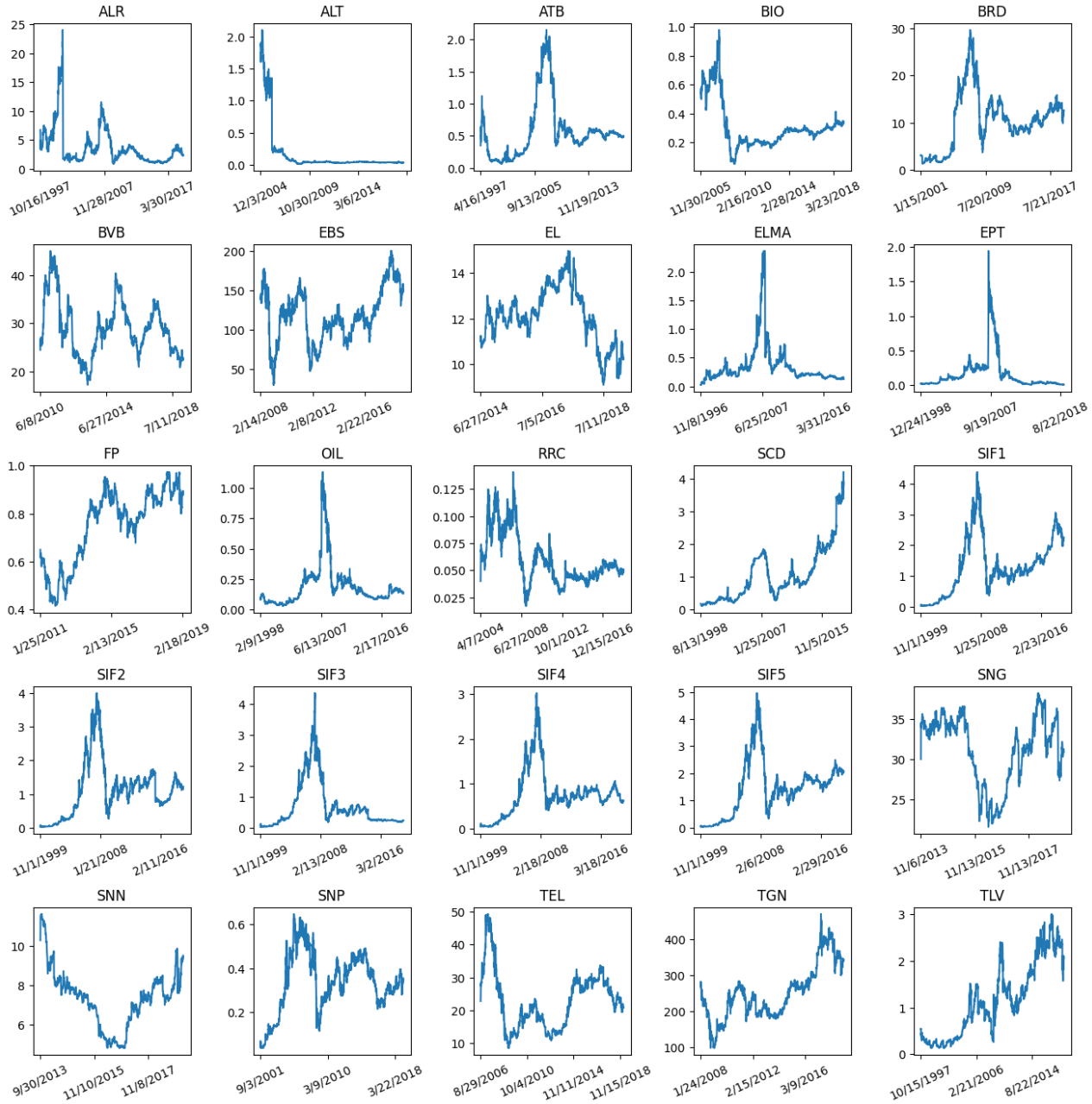
- Take a set of daily close prices connected to different companies
- Predict the one step close price (for the next day) of a company
- Can take into account the time series data for just the close price
- Or also those of exogeneous variables
  - Number of transactions
  - Value of transactions
  - Medium price
  - Etc.

LSTM for close price prediction in Tensorflow

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

fileName = '/content/drive/My Drive/Work/AllData25sym2019.txt'

# plot the data for all symbols
allData = pd.read_csv(fileName, delimiter = '\t')

allData.set_index('Date', inplace=True, drop=False)

allSymbols = allData.Symbol.unique()

fig = plt.figure(figsize = (16, 16))
fig.subplots_adjust(hspace=0.5, wspace=0.4)
for i in range(1, 26):
    ax = fig.add_subplot(5, 5, i)
    symbol = allSymbols[i - 1]

    allData.loc[allData['Symbol'] == symbol].plot(y='ClosePrice', title = symbol, rot=25, legend=False, ax = ax)
    ax.xaxis.label.set_visible(False)

plt.show()

allData.head()
```

Data


ALR
ALT
ATB
BIO
BRD
BVB
EBS
EL
ELMA
EPT
FP
OIL
RRC
SCD
SIF1
SIF2
SIF3
SIF4
SIF5
SNG
SNN
SNP
TEL
TGN
TLV

Date	Date	Company	Symbol	NoTranzactions	NoShares	ValueTranzactions	MinPrice	MedPrice	MaxPrice	OpenPrice	ClosePrice	RefPrice
10/16/1997	10/16/1997	ALRO S.A.	ALR	1218	107586	735961.50	4.0	6.8407	8.0	4.00	6.75	6.75
10/17/1997	10/17/1997	ALRO S.A.	ALR	698	81252	484435.70	5.6	5.9621	6.5	6.35	5.95	5.95
10/20/1997	10/20/1997	ALRO S.A.	ALR	717	32419	184943.85	5.1	5.7048	6.1	6.10	5.70	5.70
10/21/1997	10/21/1997	ALRO S.A.	ALR	719	56272	308418.20	5.1	5.4808	5.7	5.50	5.55	5.55
10/22/1997	10/22/1997	ALRO S.A.	ALR	1182	120702	641899.50	5.2	5.3181	5.5	5.40	5.40	5.40

```
# LSTM one step prediction for a given symbol

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
import os

import datetime as dt
import matplotlib.dates as mdates

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler # for scaling data into [0, 1]

symbol = 'TLV'
noDaysBack = 30
percentTrain = 70
onlyLoadModel = False

# Collect the data for a given symbol
subset = allData.loc[allData['Symbol'] == symbol] # take all the rows with a given symbol name

nRows = subset.shape[0] # count the number of records of the given symbol

noTrain = int((percentTrain / 100) * nRows) # amount of training data

# take the training data
training_data = subset[:noTrain]
training_processed = training_data[['ClosePrice']].values #takes into account only the values for the column ClosePrice
scaler = MinMaxScaler(feature_range = (0, 1))
training_scaled = scaler.fit_transform(training_processed) # scales the values into [0, 1]
```

The feature set

```
# the feature set contains the values for the past noDaysBack days
# the label is the value for the next day

features_set = []
labels = []
for i in range(noDaysBack, len(training_processed)):
    features_set.append(training_scaled[i - noDaysBack:i, 0])
    labels.append(training_scaled[i, 0])

features_set, labels = np.array(features_set), np.array(labels) # convert the lists to numpy array

# convert the data into the shape accepted by lSTM
# 3D shape = (number of records, number of time steps, number of attributes)
# 1. number of records for symbol
# 2. number of time steps = noDaysBack
# 3. number of attributes = 1 (closePrice)
features_set = np.reshape(features_set, (features_set.shape[0], features_set.shape[1], 1))
```

Test set preparation

```
# take the test data
test_data = subset[(noTrain + 1):]
test_processed = test_data[['ClosePrice']].values

# take the dates of the test data for future plotting
testDates = test_data[['Date']].values
testDates = [onlyDate for [onlyDate] in testDates]

# take also the previous noDaysBack from training in order to predict the first test days
total = pd.concat((training_data['ClosePrice'], test_data['ClosePrice']), axis=0)
test_inputs = total[len(total) - len(test_processed) - noDaysBack:].values
test_inputs = test_inputs.reshape(-1, 1) # puts every item (here just one) between brackets, values are unchanged
test_inputs = scaler.transform(test_inputs) # also scale the test data

# collect the test feature set as previously for training
test_features = []
for i in range(noDaysBack, len(test_processed) + noDaysBack):
    test_features.append(test_inputs[i-noDaysBack:i, 0])

# prepare test in numpy array and format for LSTM
test_features = np.array(test_features)
test_features = np.reshape(test_features, (test_features.shape[0], test_features.shape[1], 1))
```

The LSTM architecture

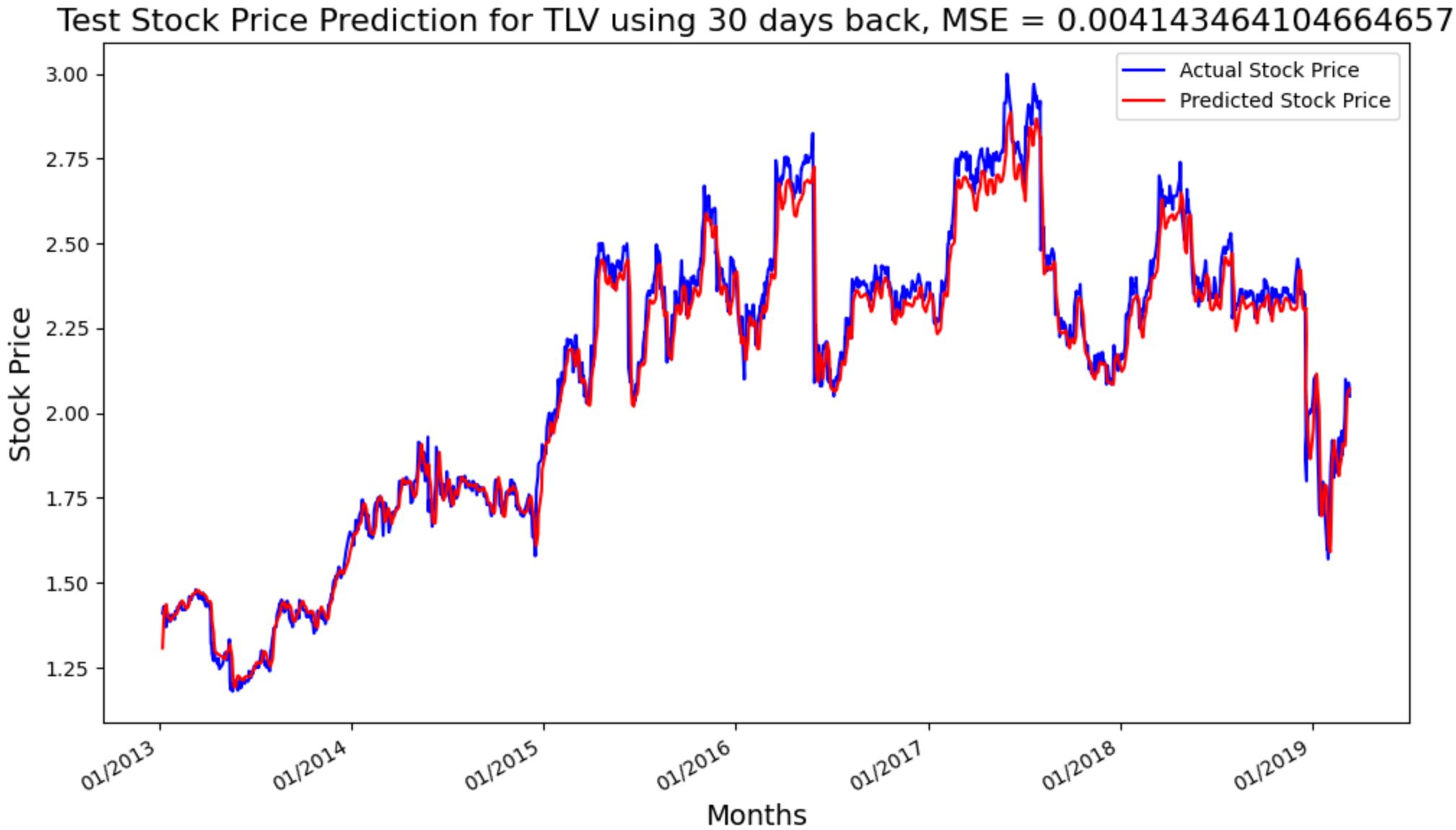
```
# 1. Build the LSTM architecture
model = Sequential()

# architecture = 3 stacked LSTM layers

model.add(LSTM(units=50, return_sequences=True, input_shape=(features_set.shape[1], 1)))
# number of units = the size of the hidden state (h_t)
# return_sequences = True for a stack of LSTM layers, so the following ones can have the full sequence as input
# for the first layer, the size of the input must be specified
# input shape takes 1 or more samples, the given number of days back (number of time steps) and 1 feature (the close price)

model.add(Dropout(0.2)) # it randomly selects nodes to be dropped out with the given probability

model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=50))
model.add(Dropout(0.2))

# add a fully connected layer
model.add(Dense(units = 1)) # units = 1 since we only output one value, the one for the next day

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_9 (LSTM)               (None, 30, 50)            10400

 dropout_9 (Dropout)         (None, 30, 50)            0

 lstm_10 (LSTM)              (None, 30, 50)            20200

 dropout_10 (Dropout)        (None, 30, 50)            0

 lstm_11 (LSTM)              (None, 50)                20200

 dropout_11 (Dropout)        (None, 50)                0

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# 2. Compile the model with the chosen optimizer and loss (mean squared error since it is a time series)
model.compile(optimizer = 'adam', loss = 'mean_squared_error')

fileModelName = "LSTMmodel" + symbol + "DaysBack" + str(noDaysBack) + ".h5" # where to store the model

if onlyLoadModel == False or not os.path.isfile(fileModelName) :
    # 3. Train & save the model with given number of epochs and batch size
    model.fit(features_set, labels, epochs = 20, batch_size = 32)
    model.save_weights(fileModelName) # take the weights from the last epoch
else:
    # OR Load the saved pretrained model
    model.load_weights(fileModelName)
    print('No training is done, the model is loaded from', fileModelName)
    print('If training is desired, set onlyLoadModel variable to False.')
```

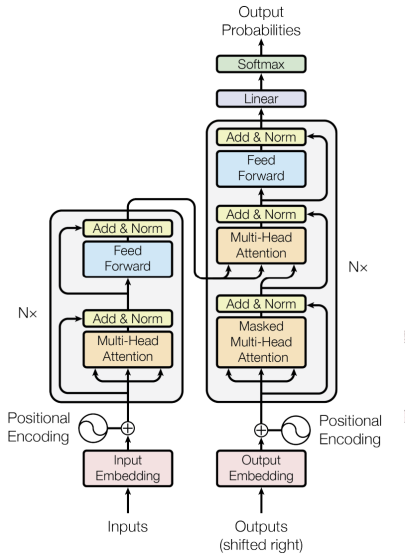
Test set prediction

```
# predict on the test data
predictions = model.predict(test_features)
predictions = scaler.inverse_transform(predictions) # reverse scaled to actual values

# plot the real test data versus the trend predicted by the model
x = [dt.datetime.strptime(d,'%m/%d/%Y').date() for d in testDates]

fig = plt.figure(figsize=(10,6))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m/%Y'))
plt.plot(x, test_processed, color='blue', label='Actual Stock Price')
plt.plot(x, predictions , color='red', label='Predicted Stock Price')
plt.gcf().autofmt_xdate()
plt.title('Test Stock Price Prediction for ' + str(symbol) + ' using ' + str(noDaysBack) + ' days back, MSE = ' + str(mean_squared_error(test_processed, predictions)), fontsize = 16)
plt.xlabel('Months', fontsize = 14)
plt.ylabel('Stock Price', fontsize = 14)
plt.legend()
plt.tight_layout()
plt.show()
```


Test Stock Price Prediction for TLV using 30 days back, MSE = 0.004143464104664657
Actual Stock Price
Predicted Stock Price
Stock Price
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
01/2013
01/2014
01/2015
01/2016
01/2017
01/2018
01/2019
Months

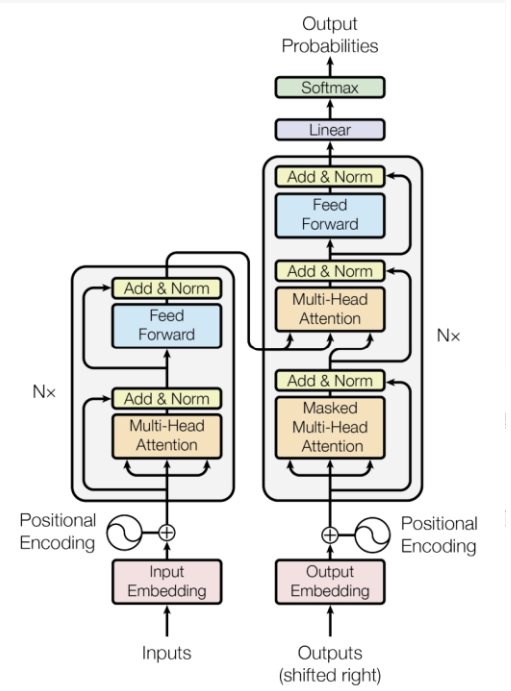
Transformers for time series prediction

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from tensorflow.keras import layers

# 'features_set' and 'labels' are already prepared as previously

# Define the transformer encoder layer
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Attention and Normalization
    x = layers.MultiHeadAttention(key_dim=head_size, num_heads=num_heads, dropout=dropout)(inputs, inputs)
    x = layers.Dropout(dropout)(x)
    x = layers.LayerNormalization(epsilon=1e-6)(x)
    res = x + inputs

    # Feed Forward Part
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(res)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    x = layers.LayerNormalization(epsilon=1e-6)(x)
    return x + res
```



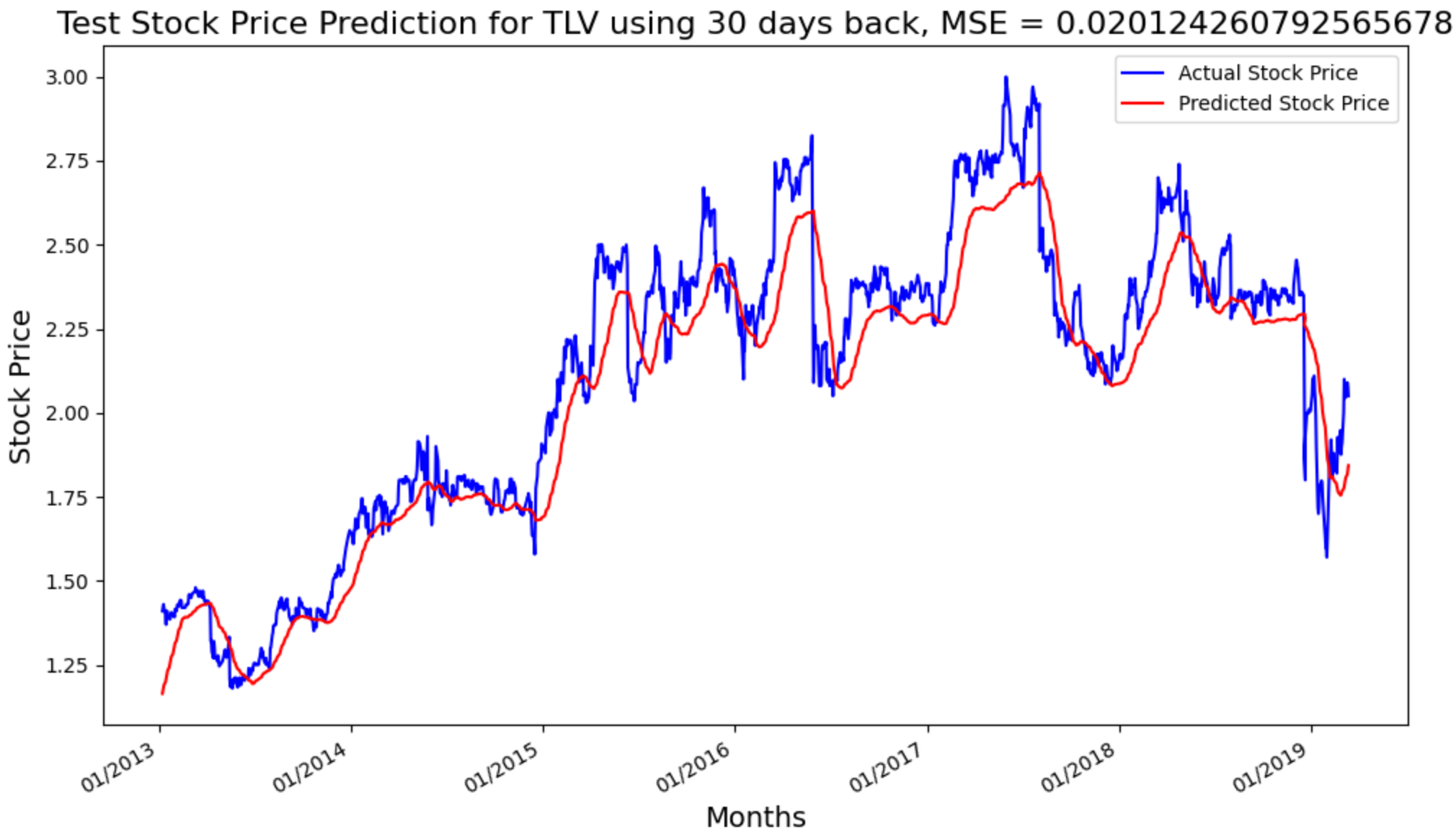
CNN is a feed-forward neural network.

```python
# Build the transformer model
def build_model(input_shape, head_size, num_heads, ff_dim, num_transformer_blocks, mlp_units, dropout=0, mlp_dropout=0):
    inputs = keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling1D(data_format="channels_last")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="relu")(x)
        x = layers.Dropout(mlp_dropout)(x)

    outputs = layers.Dense(1)(x)  # Output layer for regression task
    return keras.Model(inputs, outputs)
```



Global Average Pooling replaces flatten layers.

```python
# Define model parameters
input_shape = (features_set.shape[1], features_set.shape[2])
head_size = 256
num_heads = 4
ff_dim = 4
num_transformer_blocks = 4
mlp_units = [128]
dropout = 0.25
mlp_dropout = 0.4

model = build_model(input_shape, head_size, num_heads, ff_dim, num_transformer_blocks, mlp_units, dropout, mlp_dropout)

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

model.fit(features_set, labels, epochs=20, batch_size=64)
```


Test Stock Price Prediction for TLV using 30 days back, MSE = 0.020124260792565678
Actual Stock Price
Predicted Stock Price
Stock Price
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
01/2013
01/2014
01/2015
01/2016
01/2017
01/2018
01/2019
Months

LSTM for text classification

LSTM for sentiment analysis

```
import tensorflow as tf

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding, Dropout
from keras.layers import LSTM
from keras.datasets import imdb
```

```
max_features = 20000
maxlen = 80  # cut texts after this number of words (among top max_features most common words)
batch_size = 32
```

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
25000 train sequences
25000 test sequences
```

```
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

```
x_train shape: (25000, 80)
x_test shape: (25000, 80)
```

Architecture

```
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128))
model.add(Dropout(0.7))
model.add(Dense(1, activation='sigmoid'))
```

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

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 128)         2560000

 lstm (LSTM)                 (None, 128)               131584

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 2691713 (10.27 MB)
Trainable params: 2691713 (10.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training

```
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=5,
          validation_data=(x_test, y_test))
```

```
Epoch 1/5
782/782 [==============================] - 34s 32ms/step - loss: 0.4384 - accuracy: 0.7946 - val_loss: 0.4044 - val_accuracy: 0.8340
Epoch 2/5
782/782 [==============================] - 11s 14ms/step - loss: 0.2665 - accuracy: 0.8964 - val_loss: 0.4171 - val_accuracy: 0.8326
Epoch 3/5
782/782 [==============================] - 9s 12ms/step - loss: 0.1684 - accuracy: 0.9376 - val_loss: 0.4893 - val_accuracy: 0.8266
Epoch 4/5
782/782 [==============================] - 9s 12ms/step - loss: 0.1068 - accuracy: 0.9620 - val_loss: 0.5753 - val_accuracy: 0.8136
Epoch 5/5
782/782 [==============================] - 11s 15ms/step - loss: 0.0779 - accuracy: 0.9736 - val_loss: 0.6897 - val_accuracy: 0.8018
<keras.src.callbacks.History at 0x7a7ef2661d20>
```

Prediction

```python
import numpy as np

class_list = ['NEGATIVE','POSITIVE']

for i in range(20):
  x = np.expand_dims(x_test[i], axis=0)
  predict_x = model.predict(x)
  if predict_x > 0.5:
    classes_x=1
  else:
    classes_x=0
  print(class_list[classes_x])
```

```
1/1 [==============================] - 0s 338ms/step
NEGATIVE
1/1 [==============================] - 0s 18ms/step
POSITIVE
1/1 [==============================] - 0s 19ms/step
POSITIVE
1/1 [==============================] - 0s 19ms/step
NEGATIVE
1/1 [==============================] - 0s 18ms/step
POSITIVE
```

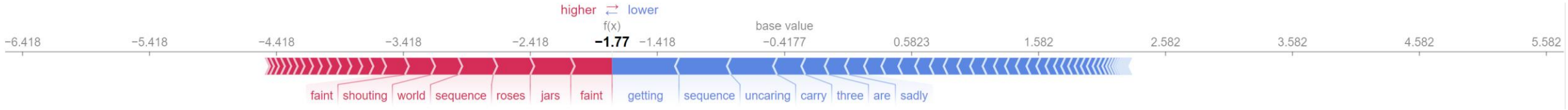
Shap KernelExplainer

- !pip install shap

```
# use Kernel SHAP to explain first 10 test set predictions

import shap

# take first 100 training samples as background data set to get the expectation
explainer = shap.KernelExplainer(model.predict, x_train[:100], link="logit")

# explain the first 10 test set predictions
shap_values = explainer.shap_values(x_test[:10])
```

```
# init JS visualization
shap.initjs()

# transform indexes to words
import numpy as np
words = imdb.get_word_index()
num2word = {}
for w in words.keys():
    num2word[words[w]] = w
x_test_words = np.stack([np.array(list(map(lambda x: num2word.get(x, "NONE"), x_test[i]))) for i in range(10)])

# plot shap values for first 10 test predictions
shap.force_plot(explainer.expected_value[0], shap_values[0][0], x_test_words[0])
```

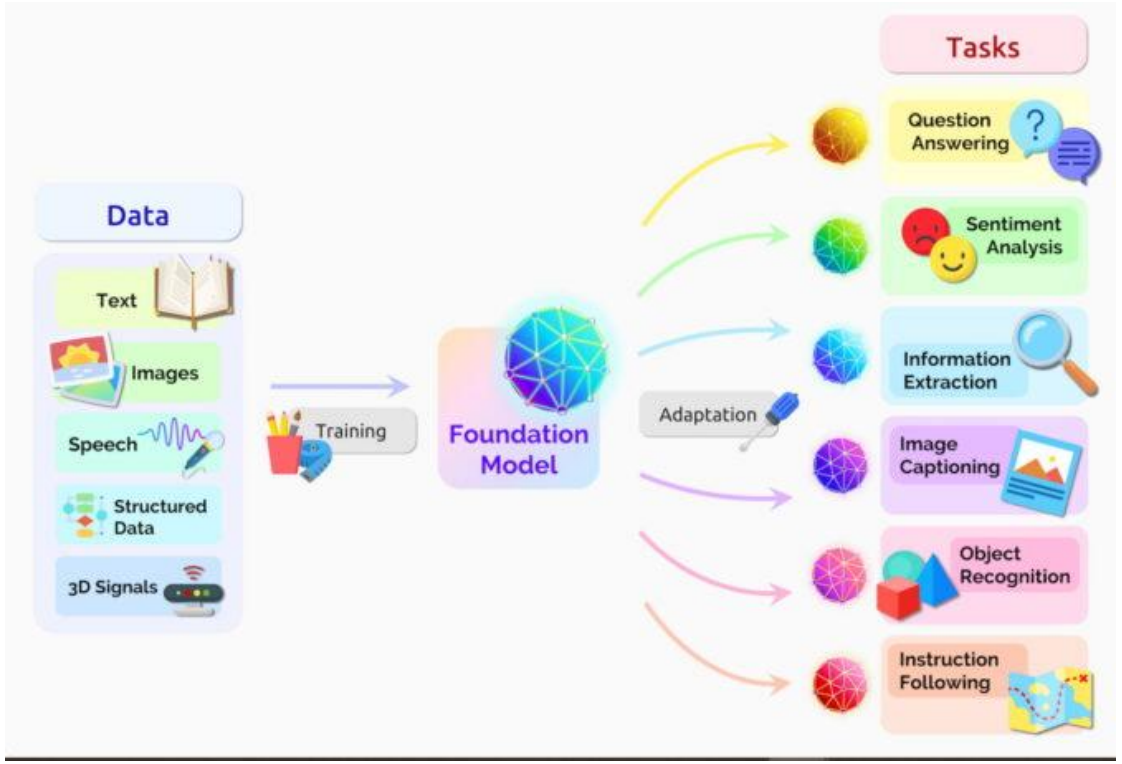
SHAP values plot



Transformers for text classification

Foundation model

- A large-scale deep learning model pre-trained on vast, diverse data sets that can be adapted and fine-tuned on various applications
- General and adaptable

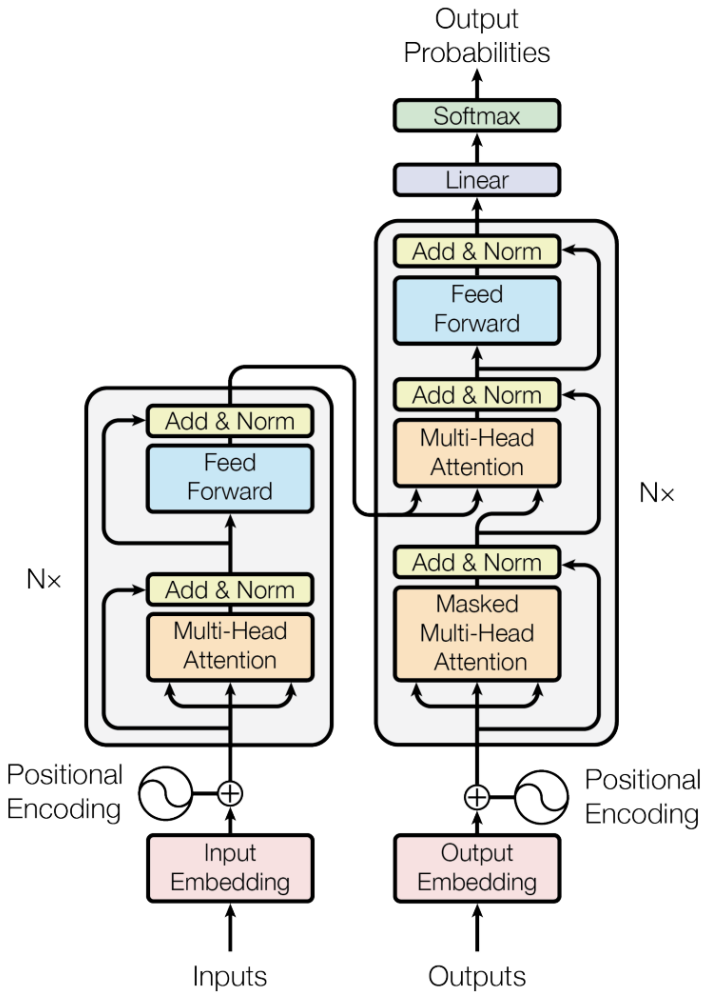


https://blogs.nvidia.com/blog/what-are-foundation-models/

Two examples: BERT and GPT

BERT

Encoder



GPT

Decoder

Made for distinct scenarios

- Bert
  - Output: embeddings - words with attention to context
  - Classification
  - Q&A
  - Summarization
  - Named entity recognition
- GPT
  - Output: next words with probabilities
  - Translation
  - Generation

BERT

- Bidirectional Encoder Representations from Transformers
- Google AI, 2018

- Self-supervised learning - produces labeled data from the text by masking words
- Examines both sides (past and future) for the context of a word
- Pre-trained using text from Wikipedia and continues to learn by Google search
- BERT (BASE): 12 layers in encoder, 12 bidirectional self-attention heads, 768 hidden units

Google
BERT

GPT

- OpenAI, 2018

- Masked attention
- Multi-Head Attention – unlike a full encoder-decoder transformer, it does not rely on encoder input
  - Instead, it has been trained on huge amounts of data (LLM)
  - The parameters already have the knowledge of the encoders

BERT example

- https://huggingface.co/bert-base-cased
- !pip install simpletransformers
- !rm -rf outputs

```
from datasets import load_dataset

train = load_dataset("imdb", split="train")
val = load_dataset("imdb", split="test")
```

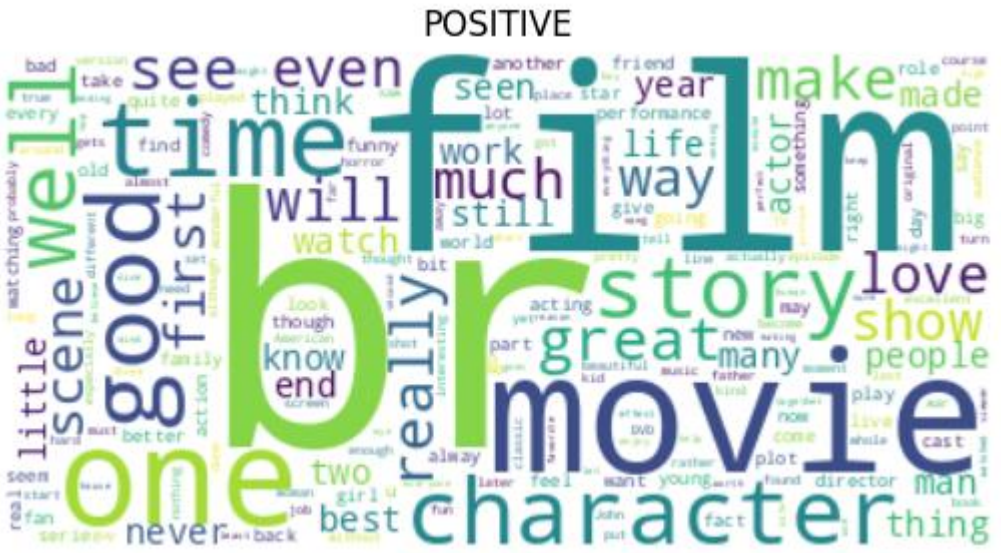
```
# convert to pandas
import pandas as pd

train_df = pd.DataFrame(train)
val_df = pd.DataFrame(val)

print(train.shape, 'train sequences')
print(val.shape, 'test sequences')
```

Word cloud Negative

```
# see wordcloud for all training tweets labelled as NEGATIVE

from wordcloud import WordCloud
import matplotlib.pyplot as plt

train_df_neg = train_df[train_df['label']==0]

text = ''
for i in range(len(train_df_neg)):
  text = text + train_df_neg['text'].iloc[i]

word_cloud = WordCloud(collocations = False, background_color = 'white').generate(text)
plt.imshow(word_cloud, interpolation='bilinear')
plt.title('NEGATIVE')
plt.axis("off")
plt.show()
```

Word Cloud Positive

```
# see wordcloud for all training tweets labelled as POSITIVE

from wordcloud import WordCloud
import matplotlib.pyplot as plt

train_df_pos = train_df[train_df['label']==1]

text = ''
for i in range(len(train_df_pos)):
  text = text + train_df_pos['text'].iloc[i]

word_cloud = WordCloud(collocations = False, background_color = 'white').generate(text)
plt.imshow(word_cloud, interpolation='bilinear')
plt.title('POSITIVE')
plt.axis("off")
plt.show()
```



Create Model

```
# load pre-trained model

# use lower precision training (fp16 = True) in order to avoid CUDA out of memory
# input data will be reprocessed

from simpletransformers.classification import ClassificationModel

# set training parameters
train_args ={"reprocess_input_data": True,
             "fp16":False,
             "num_train_epochs": 2}

# create model
model = ClassificationModel(
    "bert", "bert-base-cased", use_cuda=True,
    args=train_args
)
```

Training and results

Matthews correlation coefficient

$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

```
model.train_model(train_df)
```

```
from sklearn.metrics import f1_score, accuracy_score

result, model_outputs, wrong_predictions = model.eval_model(val_df, f1=f1_score, acc=accuracy_score)
result
```

```
{'mcc': 0.7613610744335064,
 'tp': 11019,
 'tn': 10998,
 'fp': 1502,
 'fn': 1481,
 'auroc': 0.9519378752,
 'auprc': 0.9502687760087917,
 'f1': 0.8807801446784701,
 'acc': 0.88068,
 'eval_loss': 0.4547142966687307}
```

Prediction

LSTM

```
1/1 [==============================] - 0s 338ms/step
NEGATIVE
1/1 [==============================] - 0s 18ms/step
POSITIVE
1/1 [==============================] - 0s 19ms/step
POSITIVE
1/1 [==============================] - 0s 19ms/step
NEGATIVE
1/1 [==============================] - 0s 18ms/step
POSITIVE
```

BERT

```
# see predicted sentiment for first 20 entries from validation
class_list = ['NEGATIVE','POSITIVE']

for i in range(20):
  predictions, raw_outputs = model.predict(val_df.iloc[i]['text'])
  print(class_list[val_df.iloc[i]['label']], class_list[predictions[0]])
```

```
0%  3/1386 [00:00<03:57, 5.82it/s]
100% 1/1 [00:00<00:00, 17.08it/s]
NEGATIVE NEGATIVE
0%  3/1281 [00:00<01:26, 14.80it/s]
100% 1/1 [00:00<00:00, 15.94it/s]
NEGATIVE POSITIVE
0%  2/704 [00:00<02:29, 4.70it/s]
100% 1/1 [00:00<00:00, 17.99it/s]
NEGATIVE NEGATIVE
0%  5/2089 [00:00<01:38, 21.11it/s]
100% 1/1 [00:00<00:00, 12.79it/s]
NEGATIVE NEGATIVE
0%  2/672 [00:00<00:33, 20.09it/s]
100% 1/1 [00:00<00:00, 19.04it/s]
NEGATIVE POSITIVE
```

First two validation reviews

```
val_df.iloc[0]['text']
```

```
'I love sci-fi and am willing to put up with a lot. Sci-fi movies/TV are usually underfunded, under-appreciated and misunderstood. I tried to like this, I really did, but it is to good TV sci-fi as Babylon 5 is to Star Trek (the original). Silly prosthetics, cheap cardboard sets, stilted dialogues, CG that doesn\'t match the background, and painfully one-dimensional characters cannot be overcome with a \'sci-fi\' setting. (I\'m sure there are those of you out there who think Babylon 5 is good sci-fi TV. It\'s not. It\'s clichéd and uninspiring.) While US viewers might like emotion and character development, sci-fi is a genre that does not take itself seriously (cf. Star Trek). It may treat important issues, yet not as a serious philosophy. It\'s really difficult to care about the characters here as they are not simply foolish, just missing a spark of life. Their actions and reactions are wooden and predictable, often painful to watch. The makers of Earth KNOW it\'s rubbish as they have …'
```

```
val_df.iloc[1]['text']
```

```
'Worth the entertainment value of a rental, especially if you like action movies. This one features the usual car chases, fights with the great Van Damme kick style, shooting battles with the 40 shell load shotgun, and even terrorist style bombs. All of this is entertaining and competently handled but there is nothing that really blows you away if you've seen your share before.<br /><br />The plot is made interesting by the inclusion of a rabbit, which is clever but hardly profound. Many of the characters are heavily stereotyped -- the angry veterans, the terrified illegal aliens, the crooked cops, the indifferent feds, the bitchy tough lady station head, the crooked politician, the fat federale who looks like he was typecast as the Mexican in a Hollywood movie from the 1940s. All passably acted but again nothing special.<br /><br />I thought the main villains were pretty well done and fairly well acted. By the end of the movie you certainly knew who the good guys were and weren't. Ther…'
```

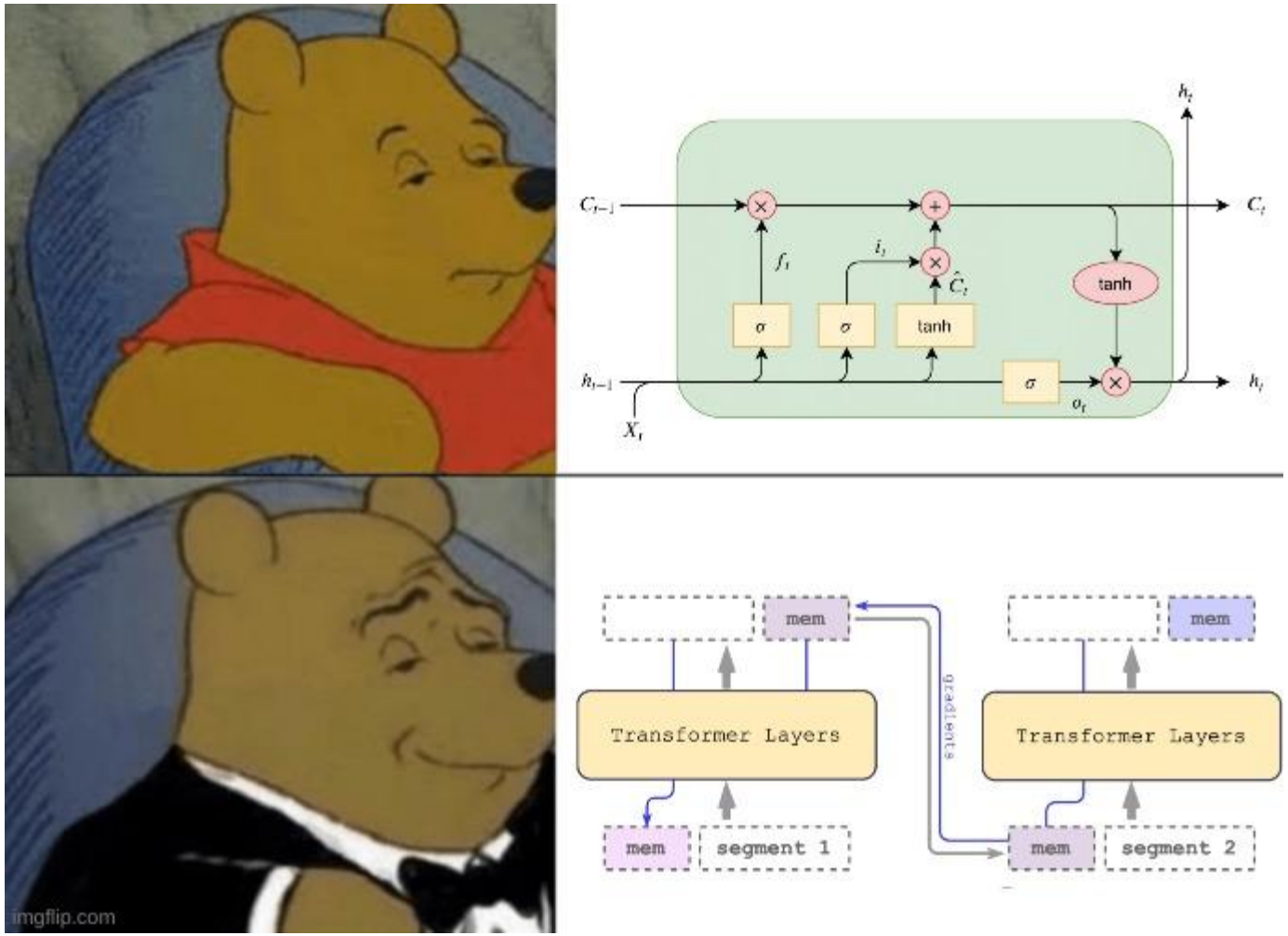
GPT example

```
from transformers import pipeline, set_seed
generator = pipeline('text-generation', model='openai-gpt')
set_seed(42)
generator("After a Master in Computer Science, I will get a job", max_new_tokens=20, num_return_sequences=5)
```

```
Device set to use cpu
[{'generated_text': 'After a Master in Computer Science, I will get a job for certain. i\'ve got the whole thing. it\'s all going up for sale. " \n'},
 {'generated_text': 'After a Master in Computer Science, I will get a job, if a good one. if they come right out and say " i don\'t know you,'},
 {'generated_text': 'After a Master in Computer Science, I will get a job, " mark said. " you can quit the shop and become part of our family. besides,'},
 {'generated_text': 'After a Master in Computer Science, I will get a job at a new school this year. maybe i will find a new student. " \n " hmm...'},
 {'generated_text': 'After a Master in Computer Science, I will get a job there. it was a shame, i think that could have been in my advantage. " \n "'}]
```

```
from transformers import pipeline, set_seed
generator = pipeline('text-generation', model='gpt2')
set_seed(42)
generator("After a Master in Computer Science, I will get a job", max_new_tokens=20, num_return_sequences=5)
```

```
Device set to use cpu
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': "After a Master in Computer Science, I will get a job at a multinational company. But the company isn't going to sell my product with any of my employees"},
 {'generated_text': 'After a Master in Computer Science, I will get a job to teach English by making and translating computer programs. What is most impressive is that I understand how to'},
 {'generated_text': 'After a Master in Computer Science, I will get a job as a research assistant, in addition to my academic training.\n\nTo help you keep up with'},
 {'generated_text': 'After a Master in Computer Science, I will get a job modeling and writing software, and you get the chance to work on the open source projects of the world'},
 {'generated_text': 'After a Master in Computer Science, I will get a job training in the information technology field. After I have completed a course, I will start my postgrad'}]
```


h_t
C_{t-1}
C_t
f_t
i_t
$\tilde{C}_t$
tanh
σ
σ
tanh
σ
h_{t-1}
h_t
o_t
X_t
imgflip.com
mem
mem
Transformer Layers
gradients
Transformer Layers
mem
segment 1
mem
segment 2

Homework

- Apply a GRU architecture for a different stock symbol.

- Take another simple transformer model from https://simpletransformers.ai/docs/installation/

  and train a model for a different task than sentiment analysis.