i4talent DATA SCIENCE

# Data Science Meetup

*Thema: Anomaly detection in time series & optimalisatie*

23 februari 2022

# Programma

01

## Opening

Waarom een data meetup?

02

## LSTM Auto-encoders

Niels Hoogeveen
*Senior data scientist*

03

## Bayesian optimization

Aschwin Schilperoort
*Senior data scientist*

04

## Mini hackathon

Anomaly detection in multi-variate time series

i4talent DATA SCIENCE

# Sprekers

**Niels Hoogeveen**

Senior data scientist
i4talent

**Aschwin Schilperoort**

Senior data scientist
DSM

i4talent DATA SCIENCE

# 01

# Opening

Waarom een i4talent data science meetup?

“Knowledge is power. Knowledge shared is ***power multiplied.***”

**—Robert Boyce**

i4talent DATA SCIENCE

# Meetup missie

## Voor & door data scientists & engineers!

*Kennisdeling, netwerken en lekker nerden!*

i4talent DATA SCIENCE

02

# LSTM Auto-encoders

Self-supervised methode
voor anomaly detection in time series.

# Image the following...

We willen errors (breakdowns) voorspellen.

Echter is maar 0,4% van de data positief gelabeld (i.e. error).

# Image the following...

We willen errors voorspellen.

Echter is maar 0,4% van de data positief gelabeld (i.e. error).

***Wat gaan we doen?***

A Down-sampling en 99,2% van de data weggooien

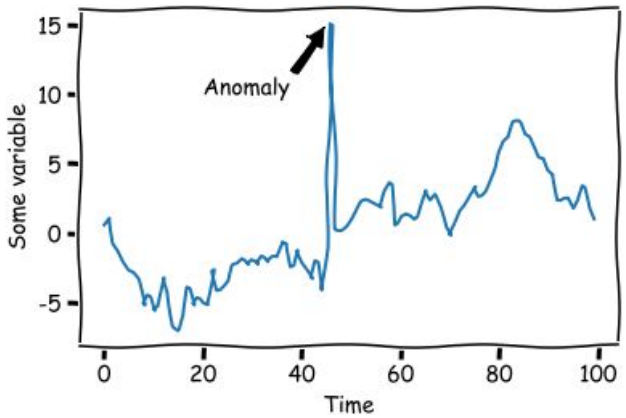
B Upsampling (e.g. SMOTE) → veel bias

C ...

# C: Anomaly detection!

***Aanpak:***

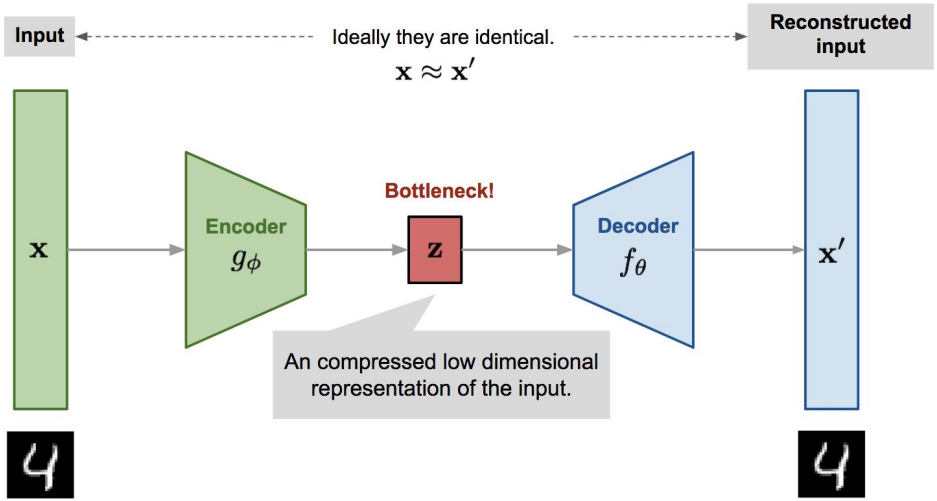
- We bouwen een auto-encoder en **trainen deze op normale (negatief gelabelde) data**
- De **auto-encoder reconstrueert** een input sample
- Als de **reconstructie error** hoog is, labelen we het als een anomaly.




i4talent DATA SCIENCE

# Wat zijn auto-encoders?

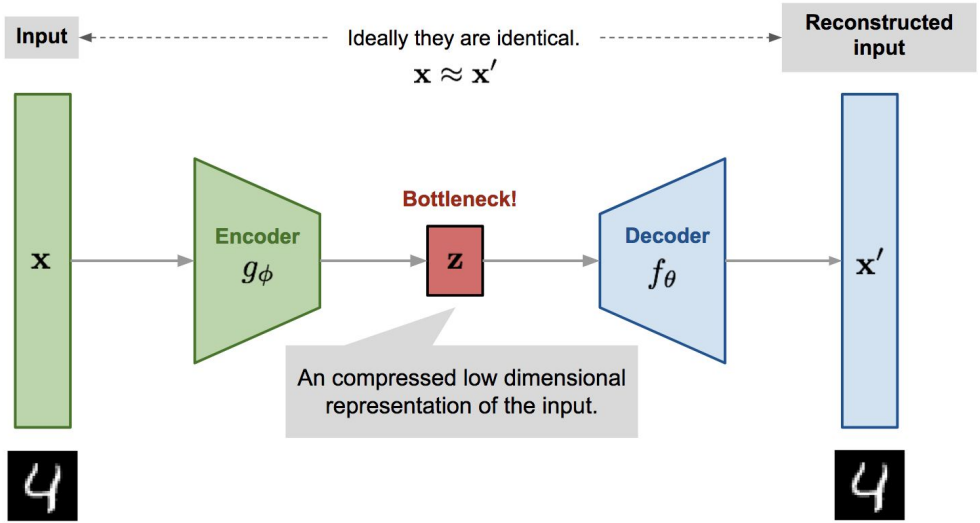
Auto-encoding is een **self-supervised** data **compression algoritme**.



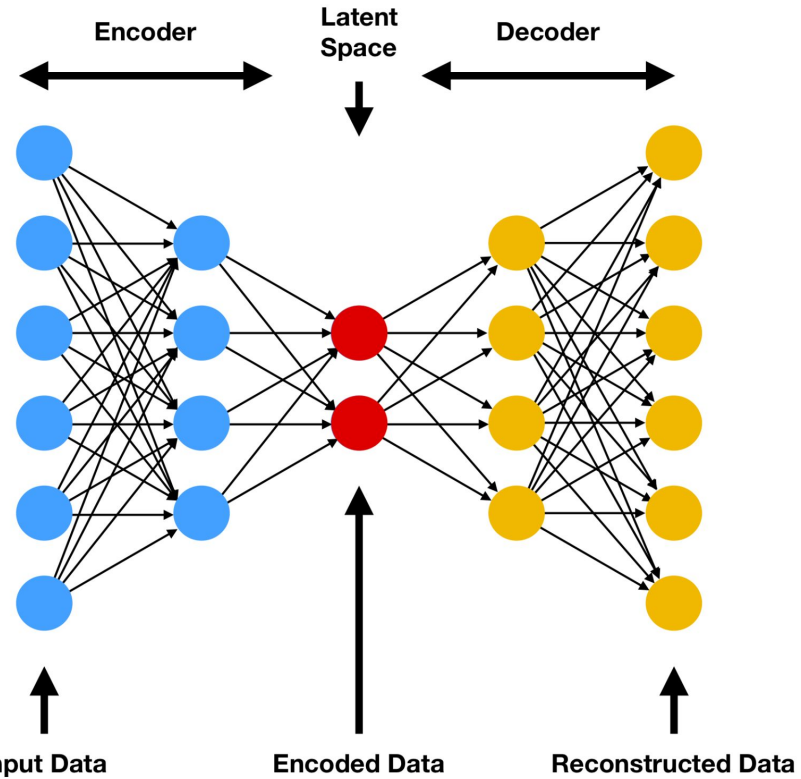
# Wat doen auto-encoders?

Auto-encoders trainen hun encoders en decoders op zo'n manier dat het **informatieverlies geminimaliseerd wordt.**

De auto-encoder moet in staat zijn de originele **input te comprimeren** (tot een *latent space representation*) en **vervolgens te reconstrueren**.



# Auto-encoder architectuur



# Hoe leert een auto-encoder?

De encoder functie ϕ mapt de originele data X tot een latente ruimte F (de *bottleneck).*

De decoder functie ψ mapt de latente ruimte naar de output, wat dezelfde dimensies heeft als de input.

Het doel is dus om de originele input te reconstrueren na een **non-lineaire compressie**.

Trainen gebeurt aan de hand van ***backpropagation*** om de reconstructie error te minimaliseren.

$$\phi : \mathcal{X} \to \mathcal{F}$$

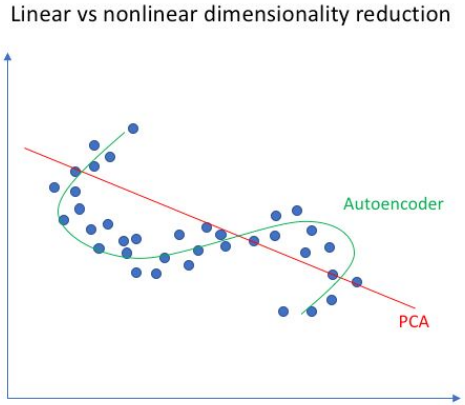
$$\psi : \mathcal{F} \to \mathcal{X}$$

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2$$

i4talent DATA SCIENCE

# Auto-encoders vs PCA

1. Auto-encoding lijkt op **principal component analysis (PCA)**: beide kunnen worden gebruikt voor **dimensionality reduction.**
2. Auto-encoders zijn in staat om *dimensionality reduction* te doen voor **non-lineaire data**
3. In PCA zijn de components linear on-gecorreleerd. Bij auto-encoders is dit niet het geval.
4. Wanneer de activatie functie in de auto-encoder lineair is in elke layer, dan zijn de latente variabelen in de *bottleneck* hetzelfde als de *principal components* in PCA.

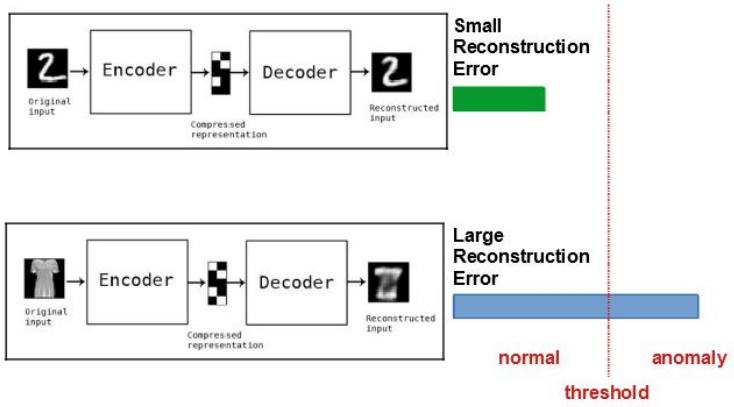


# Auto-encoders voorbeeld

In dit voorbeeld wordt een auto-encoder getraind op afbeeldingen waarop een ‘2’ te zien is.

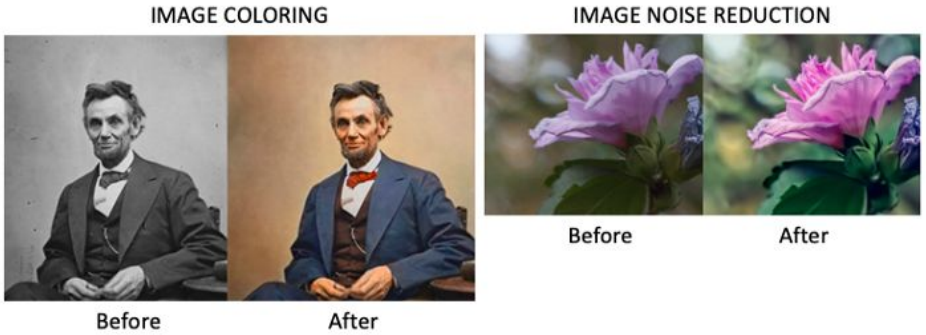
Wanneer er dan een afbeelding van een jurk in gaat, probeert de auto-encoder er een ‘2’ van te maken. De gereconstrueerde output is hierdoor een combinatie tussen de jurk (input) en een ‘2’ (geleerde output). Het verschil (i.e. *reconstruction error*) tussen de originele input (jurk) en de output is hierdoor groot.

Wanneer de reconstruction error boven een bepaalde waarde (i.e. *threshold*) komt, dan bestempelen we het als een *anomaly*.



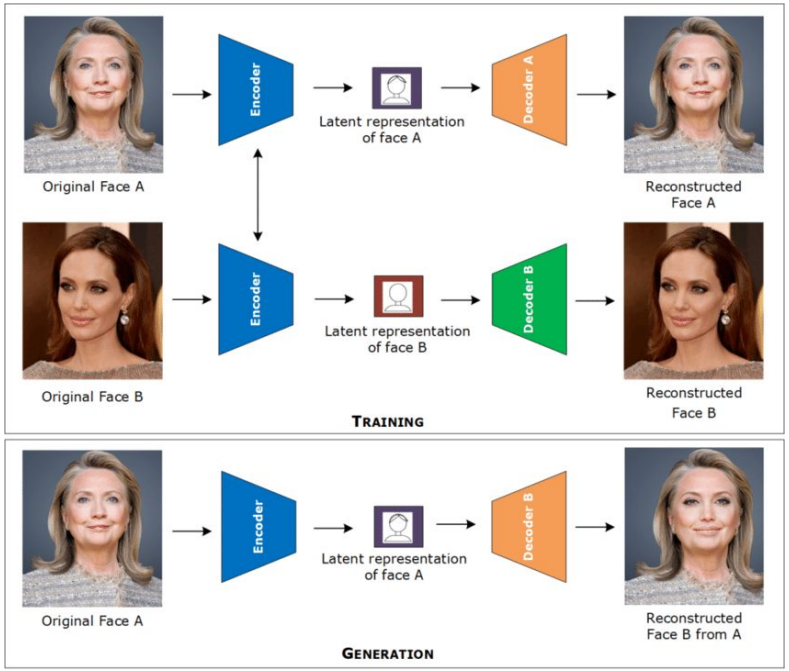
# Toepassingen van auto-encoders

- Dimensionality reduction  (e.g. voor het verzenden van afbeeldingen over het internet)
- Image denoising / image coloring



# Toepassingen van auto-encoders

- Variational autoencoders (VAE), voor het genereren van synthetische data (e.g. in games)
- *Deepfakes*



# Anomaly detection!


i4talent DATA SCIENCE

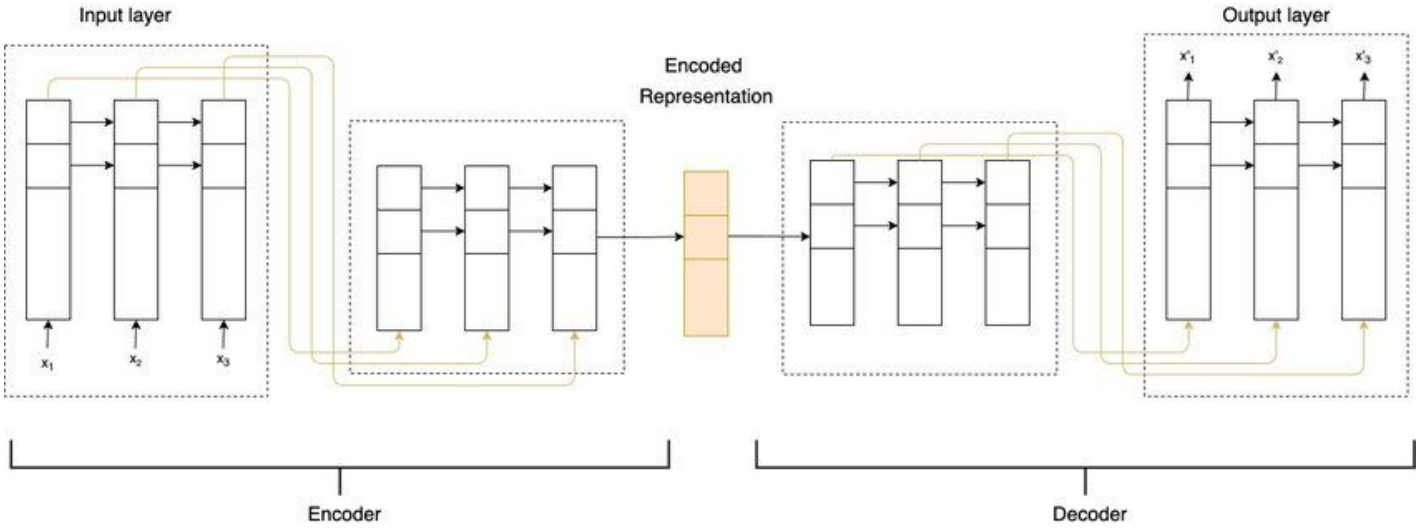
# Anomaly detection!


i4talent DATA SCIENCE

# Maar hoe werkt het bij time series?

# Maar hoe werkt het bij time series?

## LSTM Auto-encoders !!!



# LSTM? Wait, what??

- **Long-Short Term Memory (LSTM)** networks zijn een vorm van **Recurrent Neural Networks (RNN)**.
- RNN zijn geschikt voor **sequentiële data / time series** waarbij output op tijdstip t afhankelijk is van t-1 (en t-2, t-3,... t-n) zoals tekstvertalingen, aandelenprijzen, etc.
- RNNs zijn in staat de **status van voorgaande inputs te onthouden**.
- RNNs zijn een vorm van een feedforward neural network met een **intern "geheugen" (i.e. internal state)** → Dit stelt ze in staat om sequenties van inputs te verwerken.

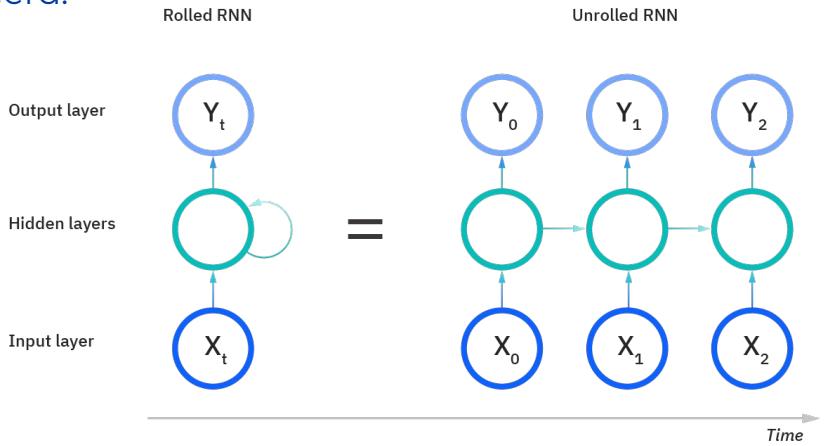
# Recurrent Neural Networks

RNNs zijn kenmerkend door hun **“geheugen” (internal state)** → ze gebruiken data van voorgaande inputs om de huidige input en output te beïnvloeden.

## RNN vs NN

Normale NNs gaan ervan uit dat alle inputs onafhankelijk van elkaar zijn.
De output van RNNs zijn afhankelijk van voorgaande inputs in een bepaalde tijdsperiode.

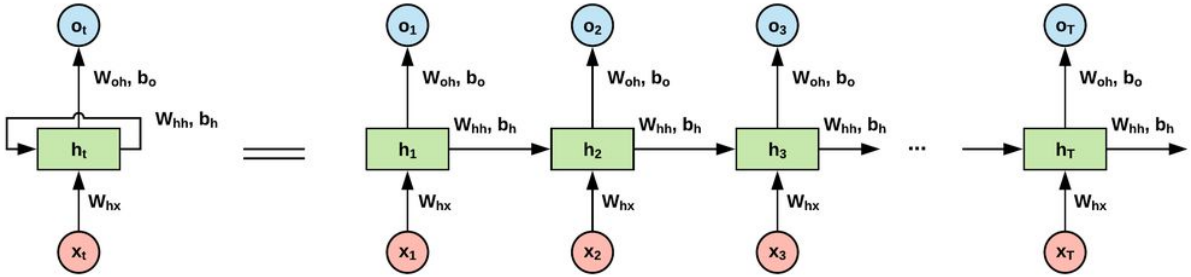
In RNNs zijn alle inputs aan elkaar gerelateerd.



# Recurrent Neural Networks

Current state:

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




Activation function:

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

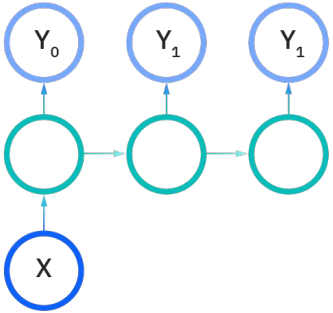
Output:

$$y_t = W_{hy}h_t$$

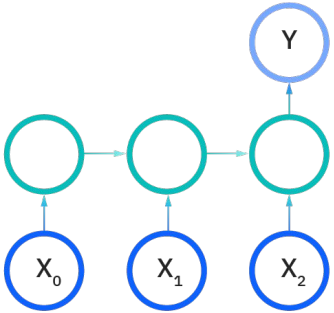
# Soorten Recurrent Neural Networks

*One-to-one*


*One-to-many*



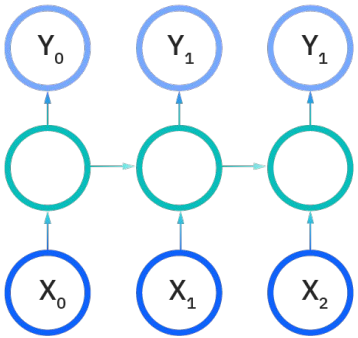

*Many-to-one*



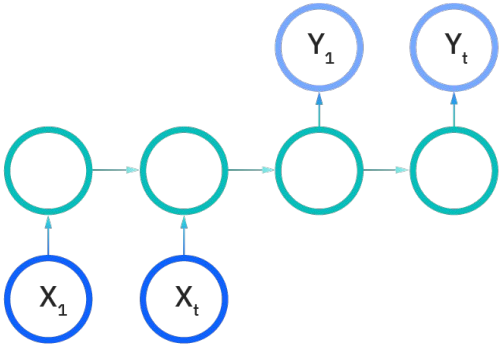

i4talent DATA SCIENCE

# Type Recurrent Neural Networks

*Many-to-many*



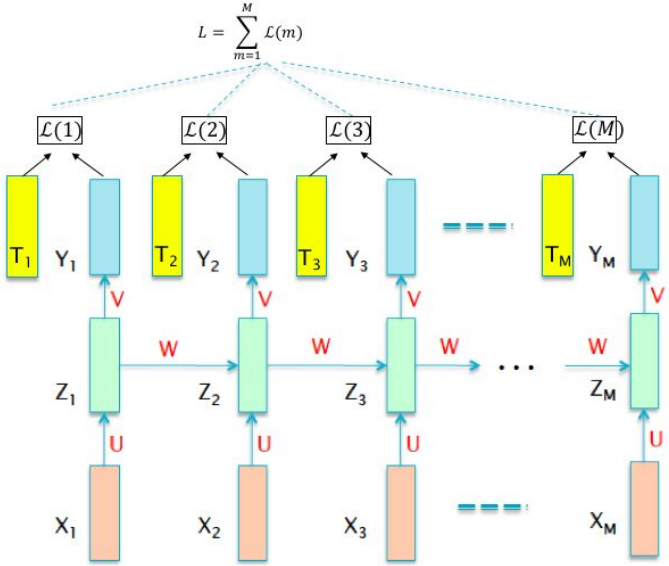

*Many-to-many*



# Hoe traint een RNN?

***Backpropagation through time (BPTT)***

1. Forward pass → de loss is de som van alle errors (current output vs actual output) over alle tijdstippen



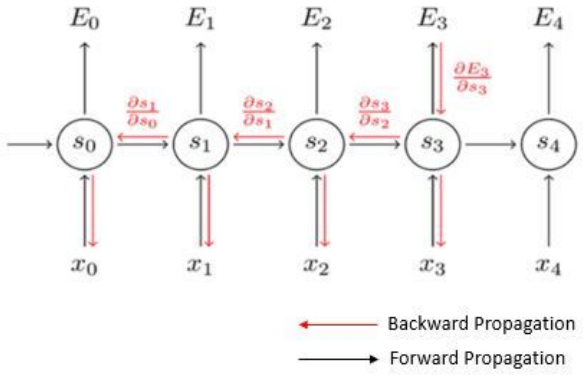
# Hoe traint een RNN?

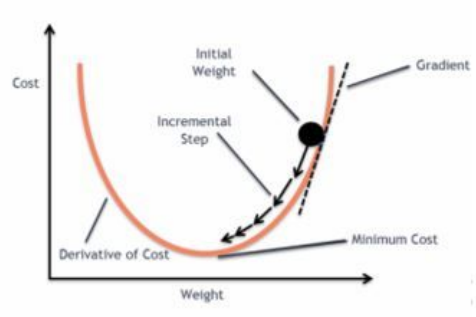
**Backpropagation through time (BPTT)**

1. ‘Rol’ het netwerk uit en bereken de ***gradient*** met respect tot een bepaald gewicht.

   → op tijdstip 1: afgeleide met respect tot een gewicht, vermenigvuldigd met de afgeleide van t-1, t-2, t-3, etc. → **vanishing / exploding gradient problem!!**

2. Update de gewichten op basis van gradient descent
3. Alle weights & biases zijn hetzelfde voor verschillende tijdstappen (i.e. RNN cellen)




Gradient Descent

# RNN problemen

1. **Vanishing / exploding gradients** probleem
   → Beperkt RNN in het leren van lange tijdreeksen

2. **Long term dependency probleem**
   Een RNN kan erg lange sequenties niet verwerken als een **tanh of relu activatie** functie wordt gebruikt.

   Dit is wanneer als een bepaalde eerdere state invloed heeft op de current state, maar niet in het “recente verleden” is.

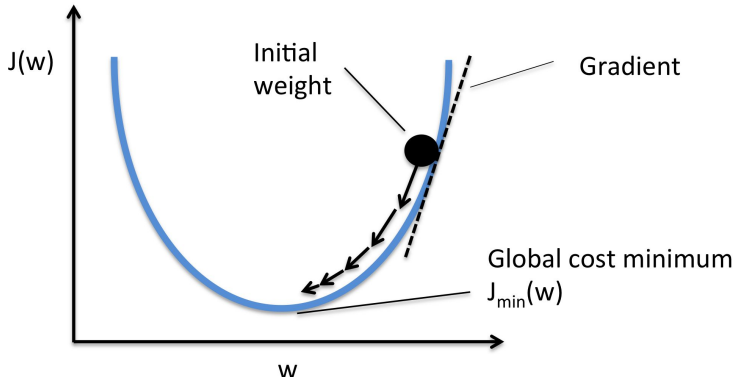
   Voorbeeld: “*Niels is allergisch voor noten. Daarom kan hij geen [...] eten.*”
   * Pindakaas

   → LSTMs hebben “cellen” die informatie van voorgaande stappen kunnen “onthouden”

# Intermezzo: gradient descent

- Neural networks updaten (i.e. trainen) hun parameters op basis van backpropagation.
- Backpropagation berekent de **afgeleide van de cost function met respect tot elke parameters (i.e. weights)** in het network.
- Wat doet de gradient? → afgeleide van de loss function met respect tot een bepaalde gewicht; hoeveel de loss verandert als het gewicht zou veranderen.

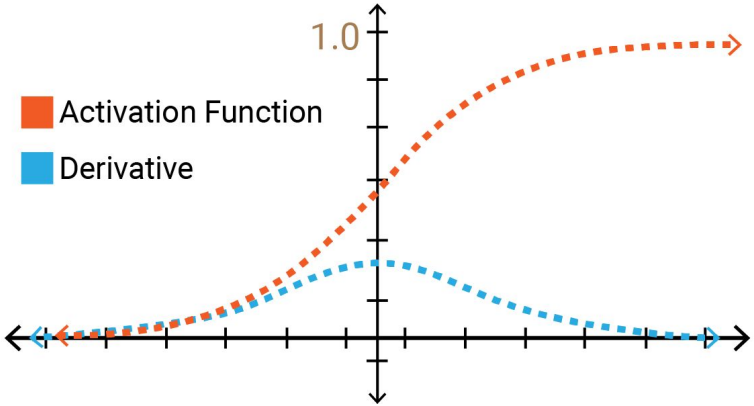
- Met gebruik van de **kettingregel** kan de afgeleide (met respect tot een gewicht) berekend worden in diepere lagen van het netwerk.
- Het algoritme update elke waarde met een gradient descent stap.



# Intermezzo: vanishing gradients

- As we *n* layers in het netwerk hebben (bij gebruik van sigmoid activation function), Dan worden *n* kleine afgeleide vermenigvuldigd met elkaar.
  De ***gradient* wordt hierdoor exponentieel kleiner voor iedere laag** in het netwerk.

- De **eerste lagen in het netwerk updaten niet (bijna) niet meer** hun gewichten
- Het NN convergeert niet tot een goed optimum.
- Mogelijke oplossing: een (leaky) **ReLu activation function →** zorgt dat de afgeleide niet te klein wordt

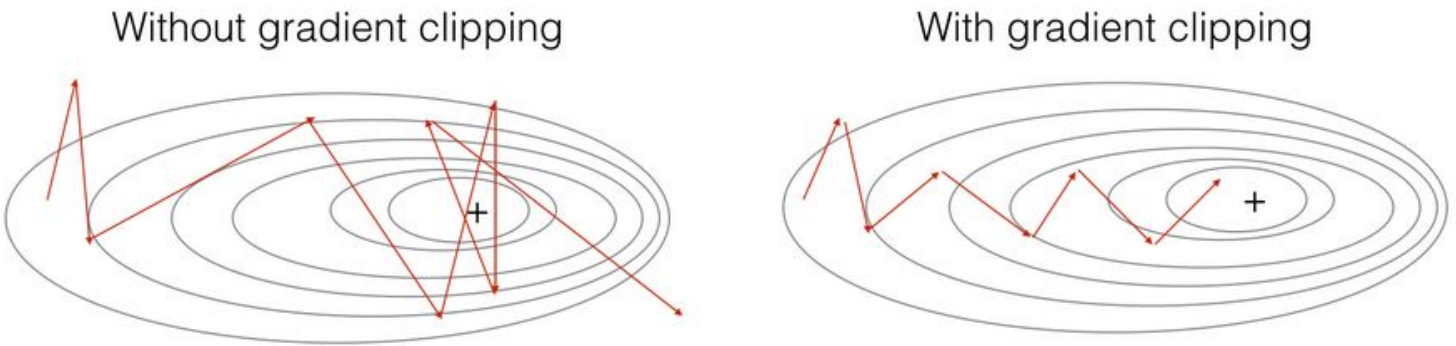


# Intermezzo: exploding gradients

Tegenovergestelde van vanishing gradients; de afgeleide wordt groter en groter waardoor de gewichten extreem groot worden en de oplossing divergeert.

→ Resulteert in oneindig trainen en NaN values van de gewichten

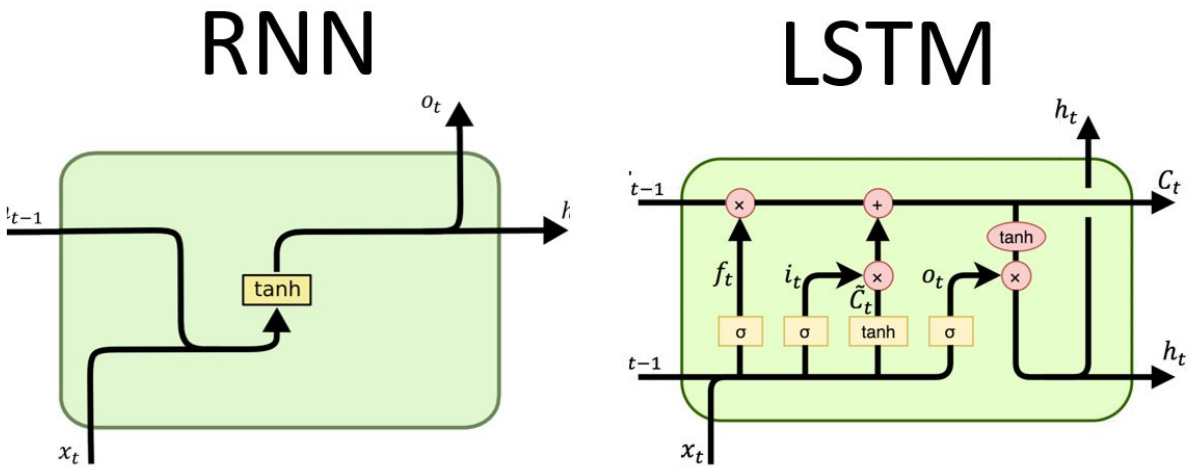
→ Mogelijk oplossing: **gradient clipping**



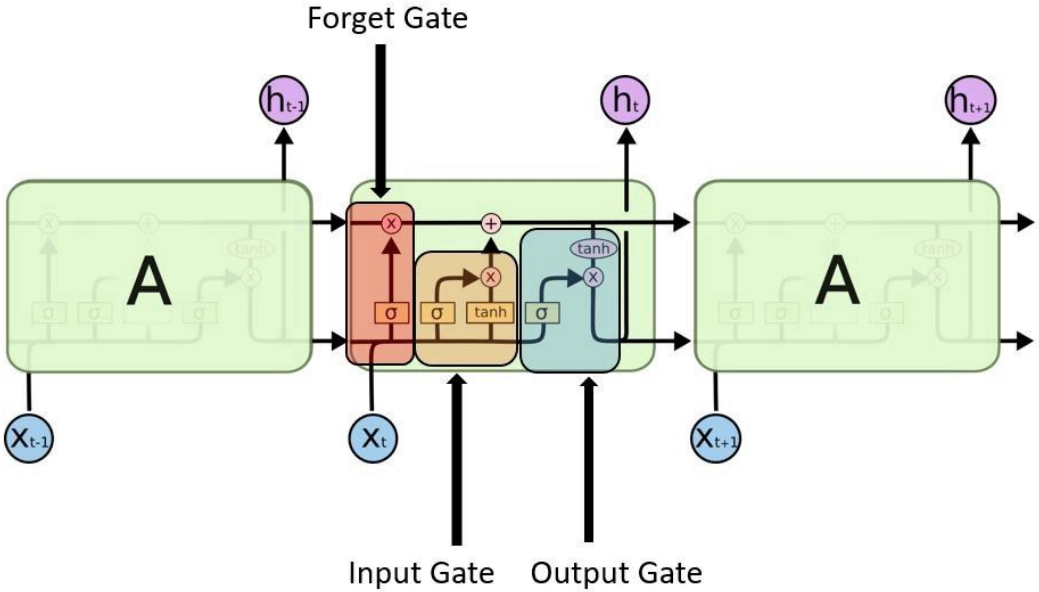
# Long-Short Term Memory to the rescue!

- LSTM networks zijn speciaal ontworpen om het **long term dependency** probleem te verhelpen door hun “geheugen cellen”.
- LSTM cellen bepalen zelf wat ze moeten onthouden op de lange termijn, wat ze moeten weggooien en welke informatie van de lange termijn gebruikt moet worden.
- LSTM **verhelpt het probleem van vanishing gradient**.

# LSTM vs RNN architectuur



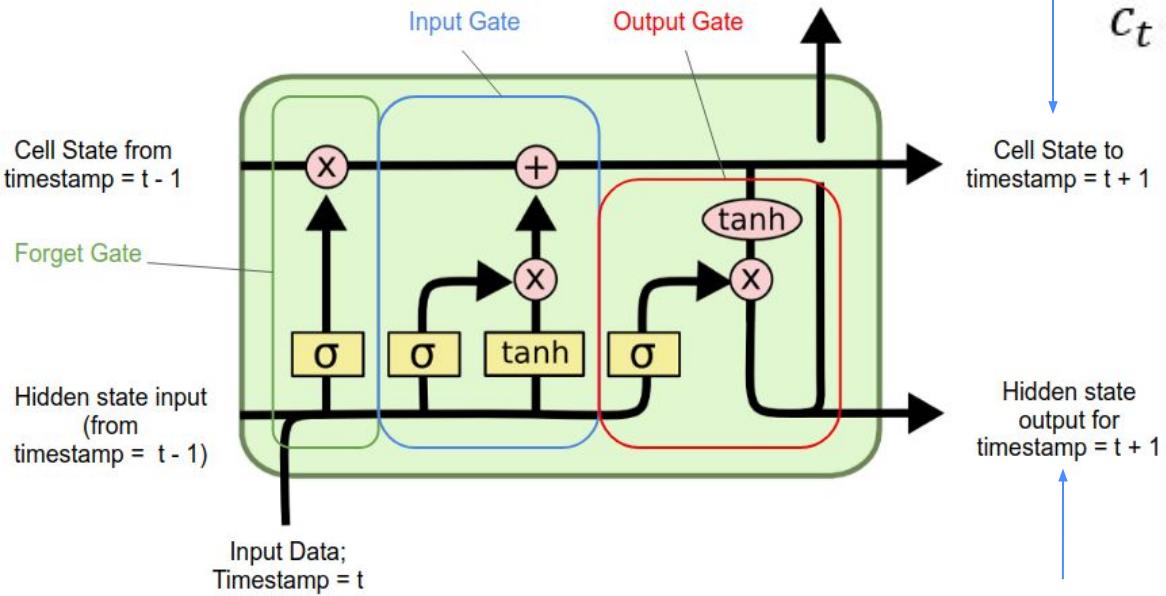
# LSTM architectuur



# LSTM architectuur

**Cell state = Long term memory**

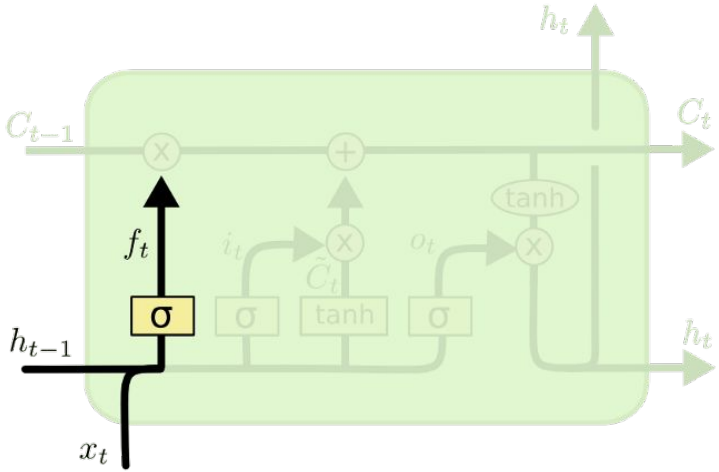
$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$




**Hidden state = Short term memory**

# Forget gate

De **forget gate** bepaalt hoeveel informatie uit het verleden “onthouden” moet worden, gegeven nieuwe informatie.

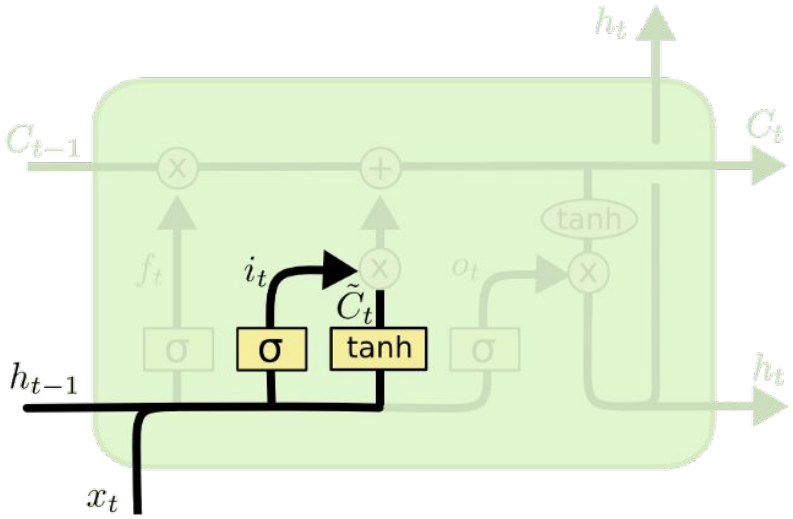



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

# Input gate

De **input gate** bepaalt welke nieuwe informatie moet worden toegevoegd aan het lange termijn geheugen (cell state), gegeven de voorgaande hidden state en de nieuwe input data.

Het gebruikt een **sigmoid** functie om te bepalen welke waarde er doorgelaten wordt, en een **tanh** functie geeft een gewicht (variërend van -1 tot 1) mee aan de waarde uit de sigmoid functie.

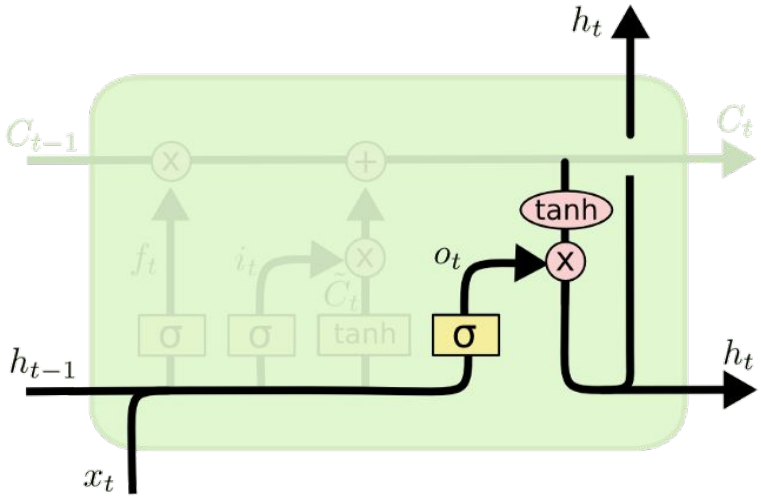



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

# Output gate

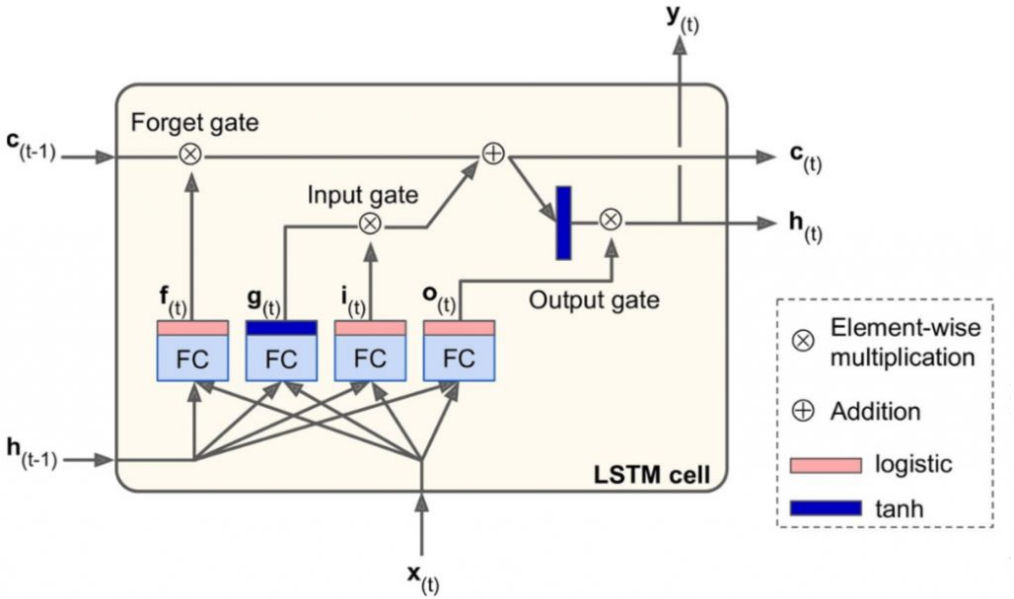
De **output gate** bepaalt welke nieuwe informatie van de current state doorgaat naar de output, als input voor de volgende LSTM cell.




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

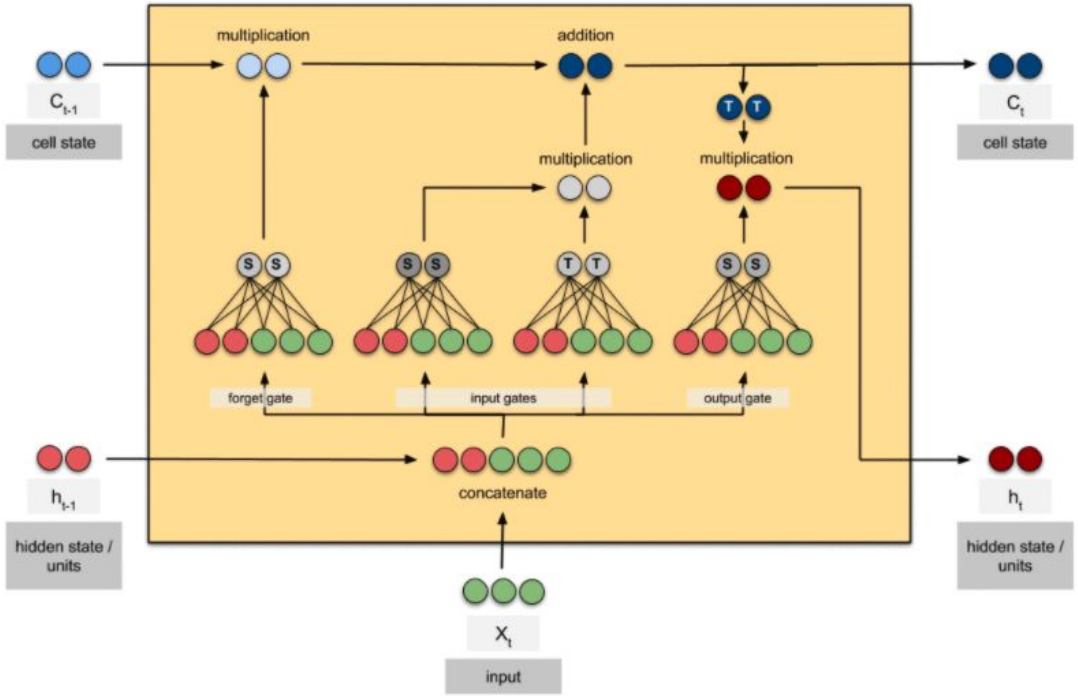
$$h_t = o_t * \tanh\left(C_t\right)$$

# Inside LSTM architectuur



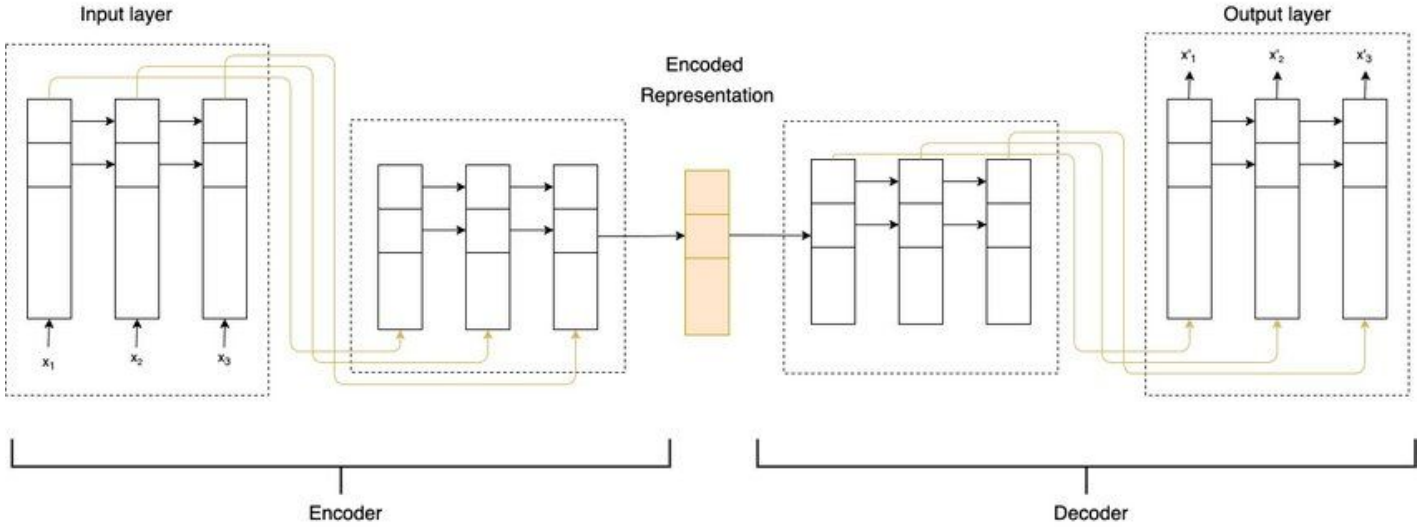
# Inside LSTM architectuur

**num_units** in Tensorflow = aantal hidden state units

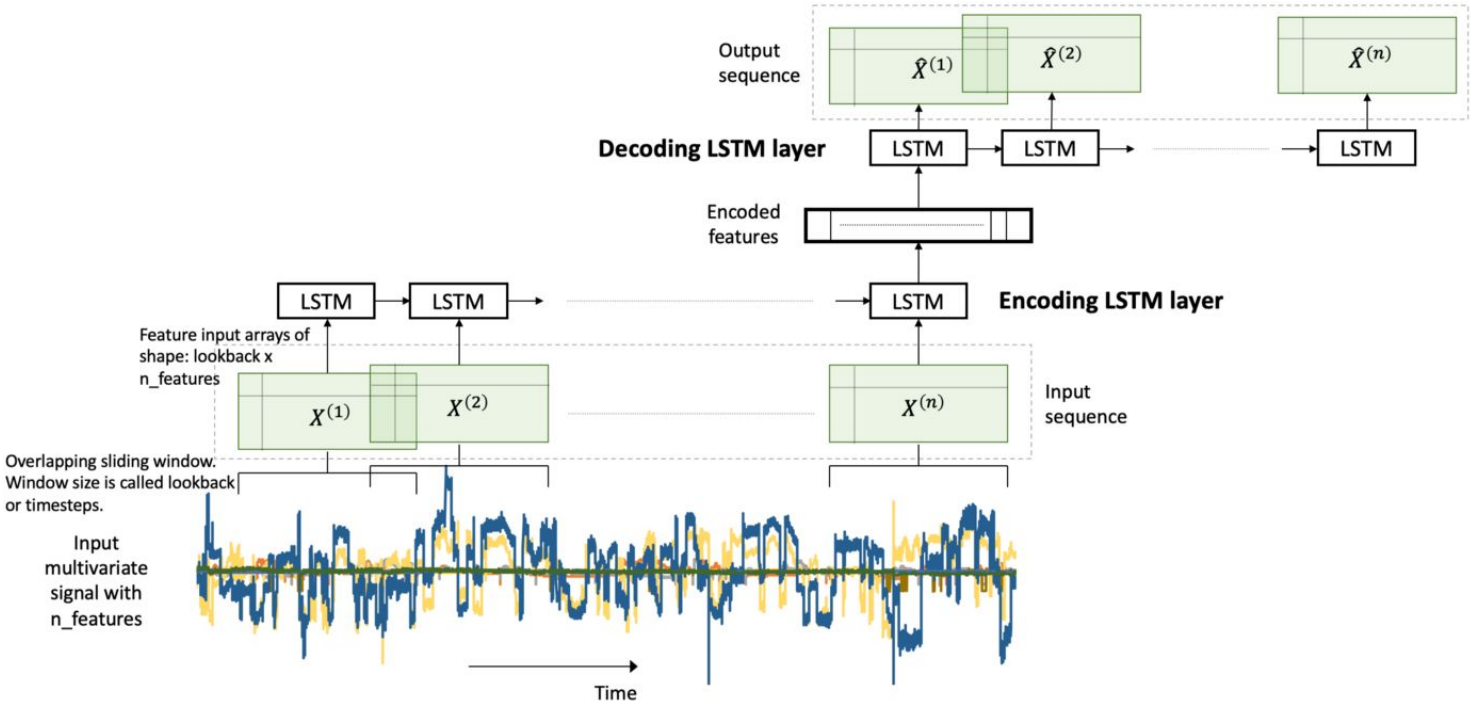



LSTM
AUTOENCODER
LSTM-AUTOENCODER!
imgflip.com

# LSTM Auto-encoder architectuur



# Multi-variate LSTM Auto-encoder architectuur



# Keras data shape

LSTM gebouwd met **Keras** verwacht de volgende shape:

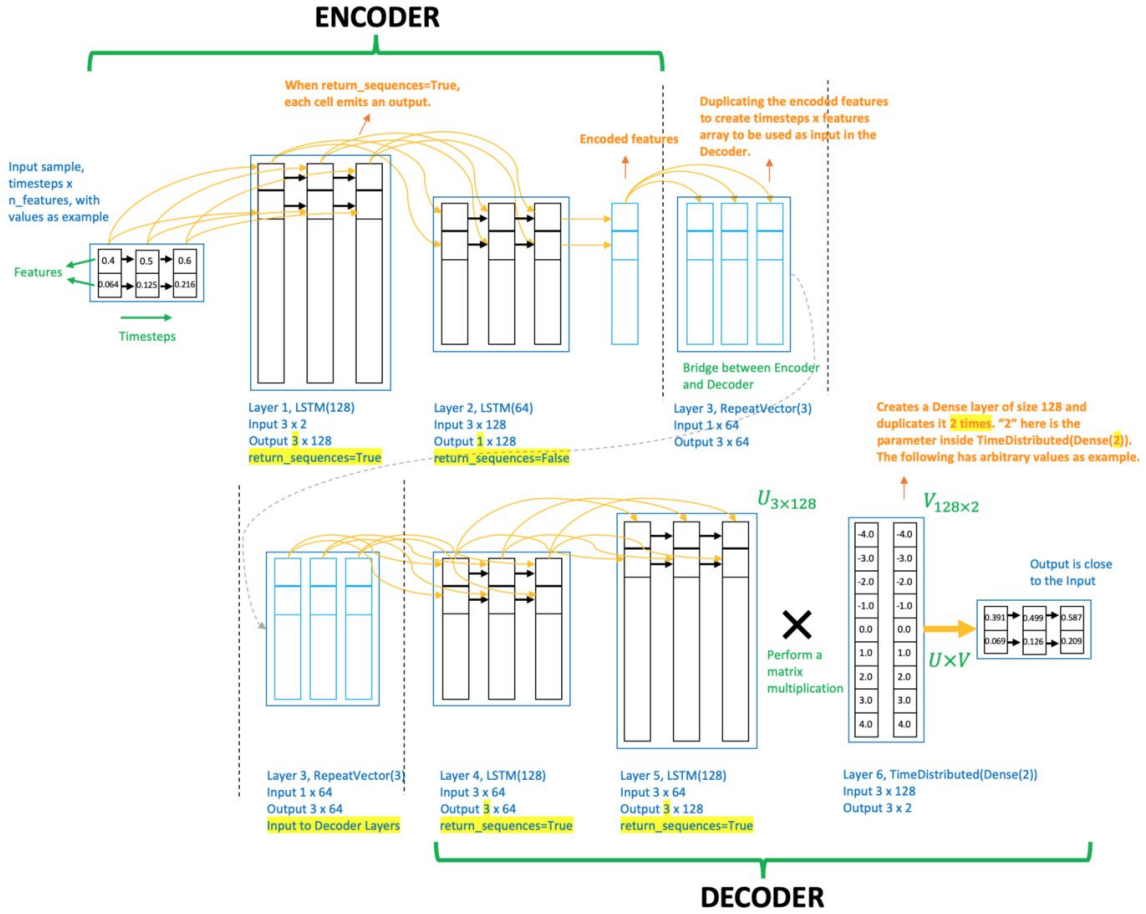
- **n_samples**
- **timesteps / sequence length**
- **n_features**

```
array([[[0.3  , 0.027],
        [0.4  , 0.064],
        [0.5  , 0.125]],

       [[0.4  , 0.064],
        [0.5  , 0.125],
        [0.6  , 0.216]],

       [[0.5  , 0.125],
        [0.6  , 0.216],
        [0.7  , 0.343]],

       [[0.6  , 0.216],
        [0.7  , 0.343],
        [0.8  , 0.512]],

       [[0.7  , 0.343],
        [0.8  , 0.512],
        [0.9  , 0.729]]])
```

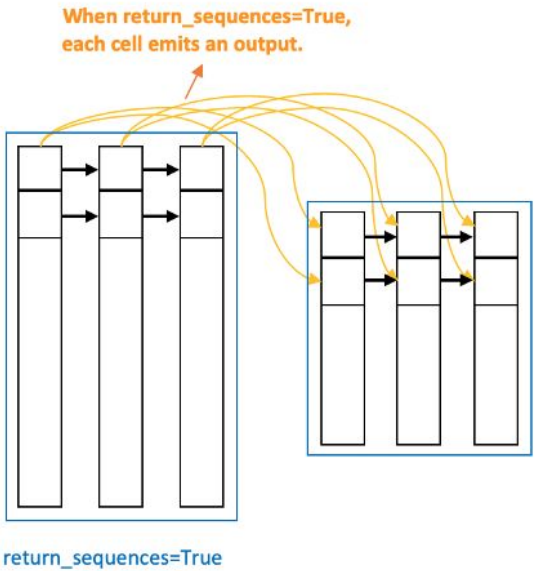
# Keras code

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed

time_steps = 3
n_features = 2

# define model
model = Sequential()
model.add(LSTM(128, activation='relu', input_shape=(time_steps, n_features), return_sequences=True))
model.add(LSTM(64, activation='relu', return_sequences=False))
model.add(RepeatVector(time_steps))
model.add(LSTM(64, activation='relu', return_sequences=True))
model.add(LSTM(128, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(n_features)))
model.compile(optimizer='adam', loss='mse')

```

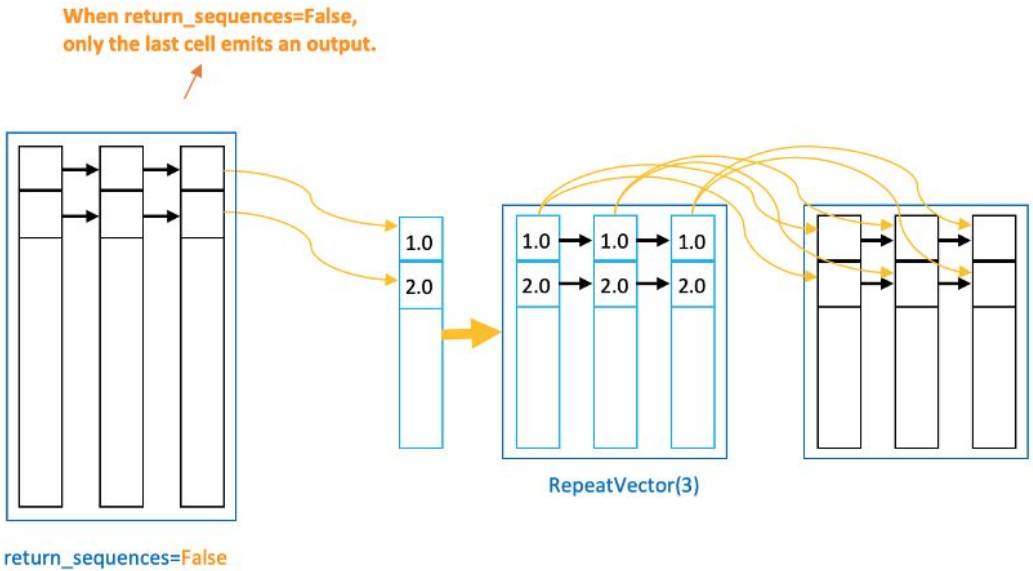
# Keras code


ENCODER
When return_sequences=True, each cell emits an output.
Input sample, timesteps x n_features, with values as example
Features
Timesteps
0.4
0.5
0.6
0.064
0.125
0.216
Encoded features
Duplicating the encoded features to create timesteps x features array to be used as input in the Decoder.
Bridge between Encoder and Decoder
Layer 1, LSTM(128)
Input 3 x 2
Output 3 x 128
return_sequences=True
Layer 2, LSTM(64)
Input 3 x 128
Output 1 x 128
return_sequences=False
Layer 3, RepeatVector(3)
Input 1 x 64
Output 3 x 64
Creates a Dense layer of size 128 and duplicates it 2 times. "2" here is the parameter inside TimeDistributed(Dense(2)). The following has arbitrary values as example.
$U_{3\times128}$
$V_{128\times2}$
Output is close to the Input
Perform a matrix multiplication
$U\times V$
Layer 3, RepeatVector(3)
Input 1 x 64
Output 3 x 64
Input to Decoder Layers
Layer 4, LSTM(128)
Input 3 x 64
Output 3 x 64
return_sequences=True
Layer 5, LSTM(128)
Input 3 x 64
Output 3 x 128
return_sequences=True
Layer 6, TimeDistributed(Dense(2))
Input 3 x 128
Output 3 x 2
DECODER

# Return sequences & Repeat vector




a.



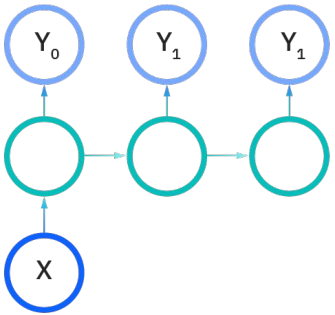

b.

# Time distributed layer

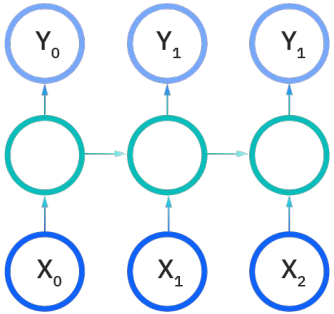
De **Time Distributed Layer** in Keras helpt om de relatie tussen de input en de corresponderende output **1 op 1** te houden.

Wordt gebruikt voor de volgende architecturen:

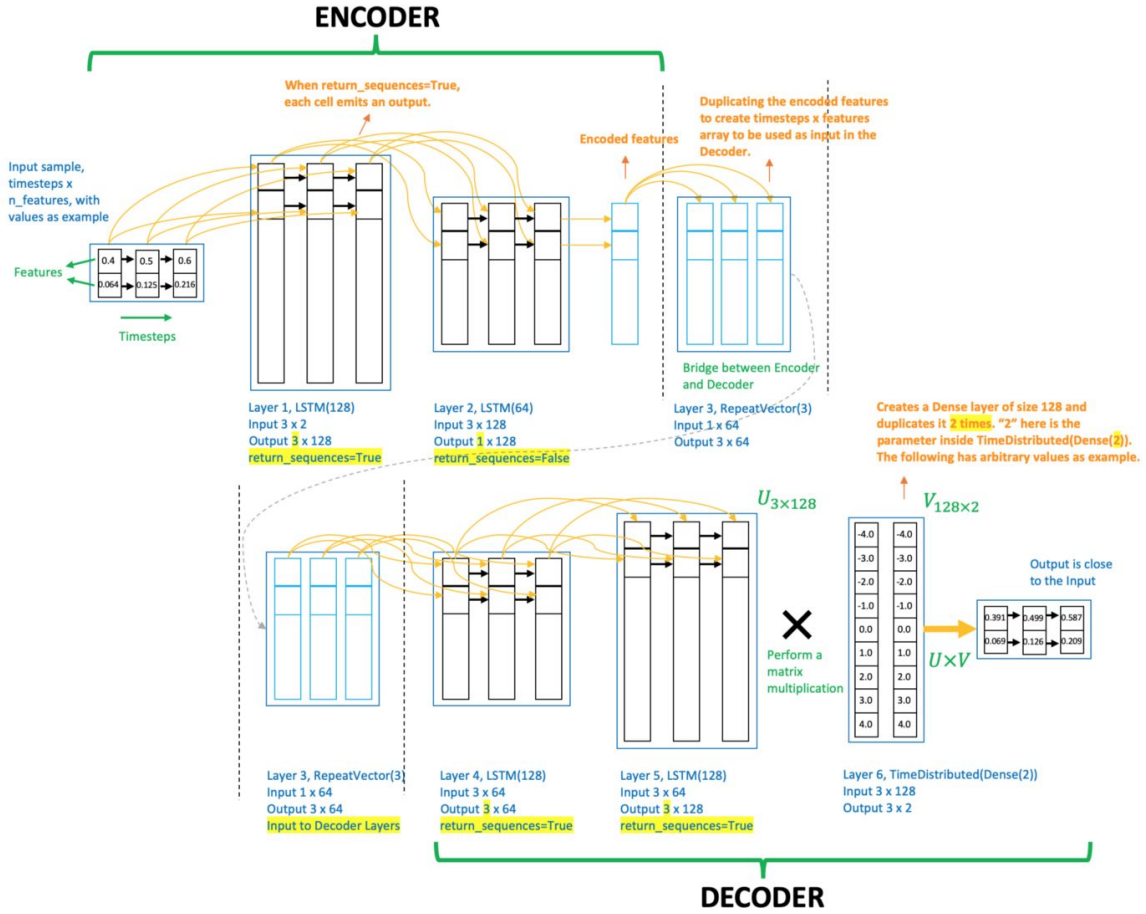
*One-to-many*




*Many-to-many*



# Keras code


ENCODER
When return_sequences=True, each cell emits an output.
Input sample, timesteps x n_features, with values as example
Features
Timesteps
0.4
0.5
0.6
0.064
0.125
0.216
Encoded features
Duplicating the encoded features to create timesteps x features array to be used as input in the Decoder.
Bridge between Encoder and Decoder
Layer 1, LSTM(128)
Input 3 x 2
Output 3 x 128
return_sequences=True
Layer 2, LSTM(64)
Input 3 x 128
Output 1 x 128
return_sequences=False
Layer 3, RepeatVector(3)
Input 1 x 64
Output 3 x 64
Creates a Dense layer of size 128 and duplicates it 2 times. "2" here is the parameter inside TimeDistributed(Dense(2)). The following has arbitrary values as example.
$U_{3\times128}$
$V_{128\times2}$
Output is close to the Input
Perform a matrix multiplication
$U\times V$
Layer 3, RepeatVector(3)
Input 1 x 64
Output 3 x 64
Input to Decoder Layers
Layer 4, LSTM(128)
Input 3 x 64
Output 3 x 64
return_sequences=True
Layer 5, LSTM(128)
Input 3 x 64
Output 3 x 128
return_sequences=True
Layer 6, TimeDistributed(Dense(2))
Input 3 x 128
Output 3 x 2
DECODER



# Hackathon!

## Maar nu eerst even wat eten...



i4talent DATA SCIENCE

# Hackathon 

Regels:

- Groepjes van 4 - 5 personen
- Bouw een anomaly detector die zover mogelijk van tevoren een anomaly kan detecteren
- Groep met hoogste *recall / precision* wint! 

Repo: https://github.com/NielsHoogeveen1990/meetup

i4talent DATA SCIENCE

# Bedankt!

**Vragen?**

*nielsh@i4talent.nl*

*06-40869980*

i4talent DATA SCIENCE

i4talent DATA SCIENCE

# Next meetup!

# 11 mei 2022!