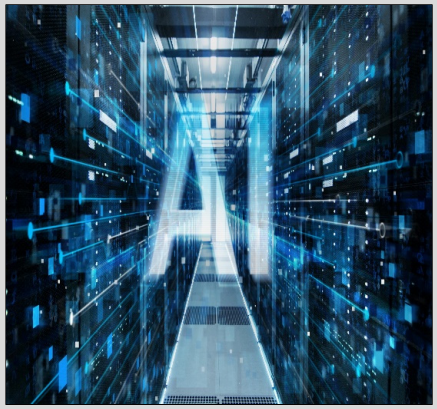


Artificial Intelligence Data Analysis (AIDA)

1st School for Heliophysicists

Prof. Dr. – Ing. Morris Riedel
Associated Professor
School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland
Research Group Leader, Juelich Supercomputing Centre, Forschungszentrum Juelich, Germany

@Morris Riedel @MorrisRiedel @MorrisRiedel

LECTURE 7

Deep Neural Networks & Convolutional Neural Networks – Examples

January 21, 2020
CINECA, Bologna, Italy

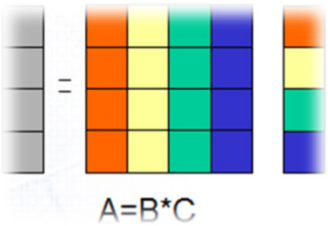
UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

DEEP Projects

HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

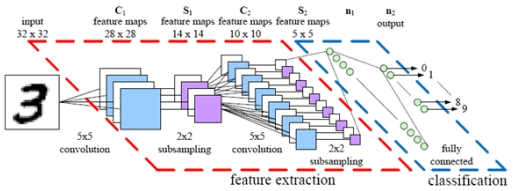
Review of Lecture 6 – DNNs & CNNs – Overview

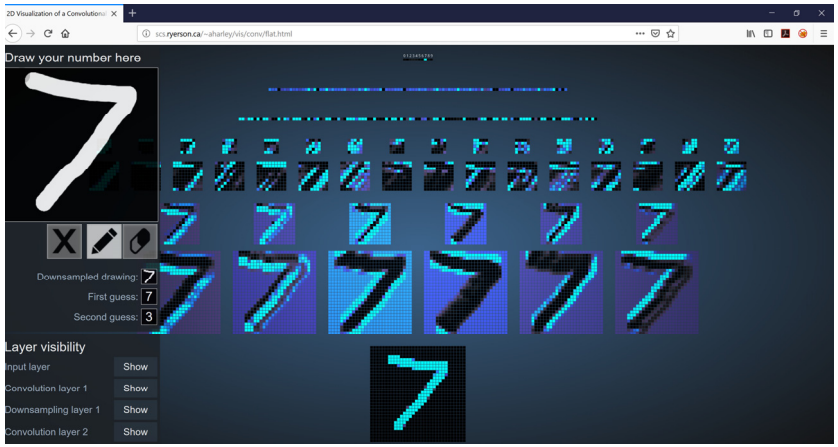


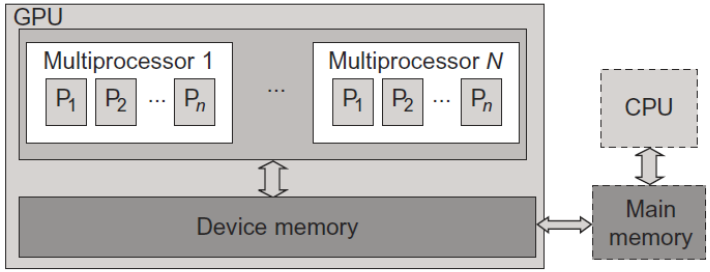
(nice parallelization possible via independent computing)

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_{0,0}c_0 + & b_{0,1}c_1 + & b_{0,2}c_2 + & b_{0,3}c_3 \\ b_{1,0}c_0 + & b_{1,1}c_1 + & b_{1,2}c_2 + & b_{1,3}c_3 \\ b_{2,0}c_0 + & b_{2,1}c_1 + & b_{2,2}c_2 + & b_{2,3}c_3 \\ b_{3,0}c_0 + & b_{3,1}c_1 + & b_{3,2}c_2 + & b_{3,3}c_3 \end{bmatrix}$$

P0 P1 P2 P3







[6] Distributed & Cloud Computing Book *[7] Harley, A.W., An Interactive Node-Link Visualization of Convolutional Neural Networks* *[8] A. Rosebrock*

Outline of the School

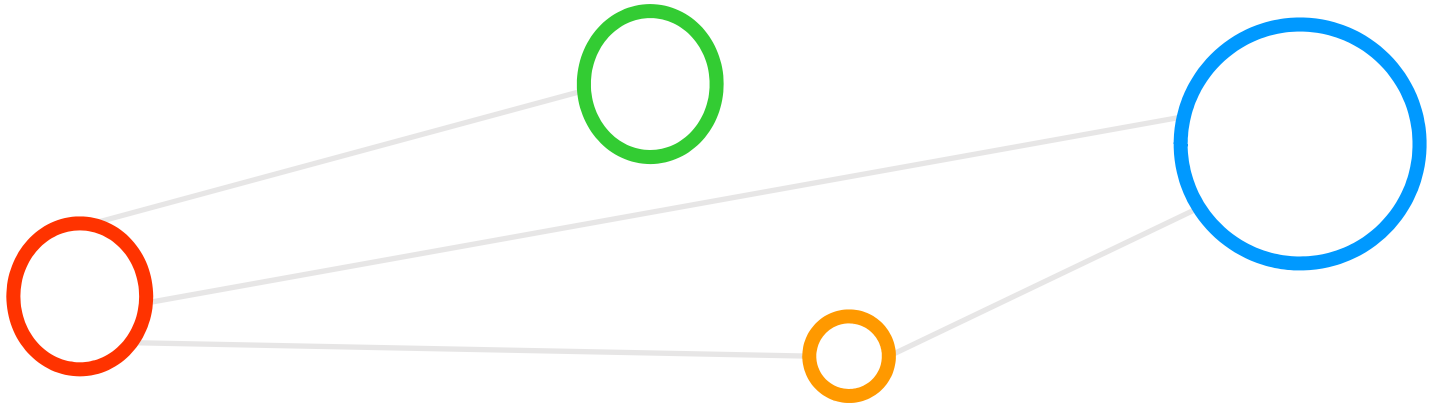
Time	**Day 1**	**Day 2**	**Day 3**
9 - 10	Welcome and intro to the school (Giovanni Lapenta, Jorge Amaya)	Space missions data acquisition (Hugo Breuillard)	Review of ML applied to heliophysics (Peter Wintoft)
10 - 11	Introduction and differences between AI, ML, NN and Big Data (Morris Riedel)	Data manipulation in python with pandas, xarray, and additional python tools (Geert Jan Bex)	Review of ML applied to heliophysics (Peter Wintoft)
	Coffee break	Coffee break	Coffee break
11:30 - 12:30	Unsupervised learning (Morris Riedel)	Feature engineering and data reduction (Geert Jan Bex)	Reinforcement learning (Morris Riedel)
	Lunch	Lunch	Lunch
14 - 15	Unsupervised learning (Morris Riedel)	Data reduction and visualization (Geert Jan Bex)	Physics informed ML (Romain Dupuis)
15 -16	Supervised learning (Morris Riedel)	CNN, DNN (Morris Riedel)	Explainable AI (Jorge Amaya)
	Coffee break	Coffee break	Coffee break
16:30 - 18:00	Supervised learning (Morris Riedel)	CNN, DNN (Morris Riedel)	Performance and tuning of ML (Morris Riedel)

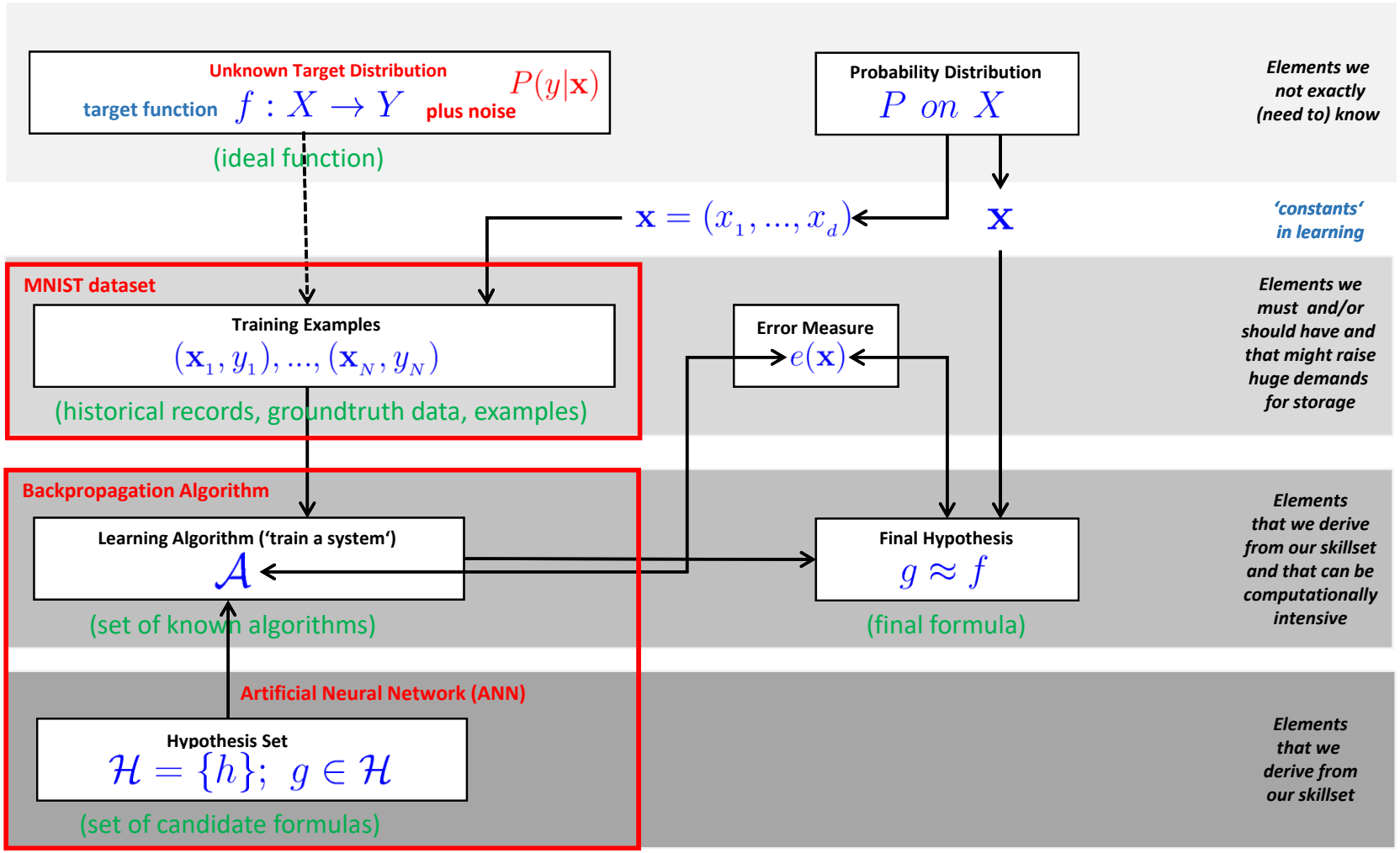
Outline

- Convolutional Neural Network (CNN) Examples
  - Yardstick with ANNs in MNIST Application – Revisited
  - MNIST Application Example with CNNs in Keras
  - Understanding Feature Maps in CNN Architecture
  - Hyperparameter Complexity & Adam Optimizer
  - Understanding Accuracy Improvements & Limits

- Other Deep Neural Network (DNN) Examples
  - Short Introduction to Long Short-Term Memory (LSTM) Model
  - LSTM Application Example in Keras
  - Autoencoder Models
  - Generative Adversial Networks (GANs) Models
  - Deep Q Learning

Convolutional Neural Network (CNN) Examples




Unknown Target Distribution
target function $f : X \rightarrow Y$ plus noise $P(y|\mathbf{x})$
(ideal function)
Probability Distribution
P on X
Elements we not exactly (need to) know
$\mathbf{x} = (x_1, ..., x_d)$
$\mathbf{X}$
'constants' in learning
MNIST dataset
Training Examples
$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
(historical records, groundtruth data, examples)
Error Measure
$e(\mathbf{x})$
Elements we must and/or should have and that might raise huge demands for storage
Backpropagation Algorithm
Learning Algorithm ('train a system')
$\mathcal{A}$
(set of known algorithms)
Final Hypothesis
$g \approx f$
(final formula)
Elements that we derive from our skillset and that can be computationally intensive
Artificial Neural Network (ANN)
Hypothesis Set
$\mathcal{H} = \{h\}; \ g \in \mathcal{H}$
(set of candidate formulas)
Elements that we derive from our skillset

MNIST Dataset – ANN Model Parameters & Output Evaluation – 20 Epochs (!)

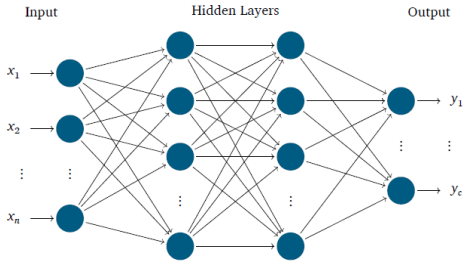
```
Epoch 7/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2743 - acc: 0.9223
Epoch 8/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2601 - acc: 0.9266
Epoch 9/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2477 - acc: 0.9301
Epoch 10/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2365 - acc: 0.9329
Epoch 11/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2264 - acc: 0.9356
Epoch 12/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2175 - acc: 0.9386
Epoch 13/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2092 - acc: 0.9412
Epoch 14/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2013 - acc: 0.9432
Epoch 15/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1942 - acc: 0.9454
Epoch 16/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1876 - acc: 0.9472
Epoch 17/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1813 - acc: 0.9487
Epoch 18/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1754 - acc: 0.9502
Epoch 19/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1700 - acc: 0.9522
Epoch 20/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.1647 - acc: 0.9536
```

```
# model evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 0s 33us/step
Test score: 0.16286438911408185
Test accuracy: 0.9514
```

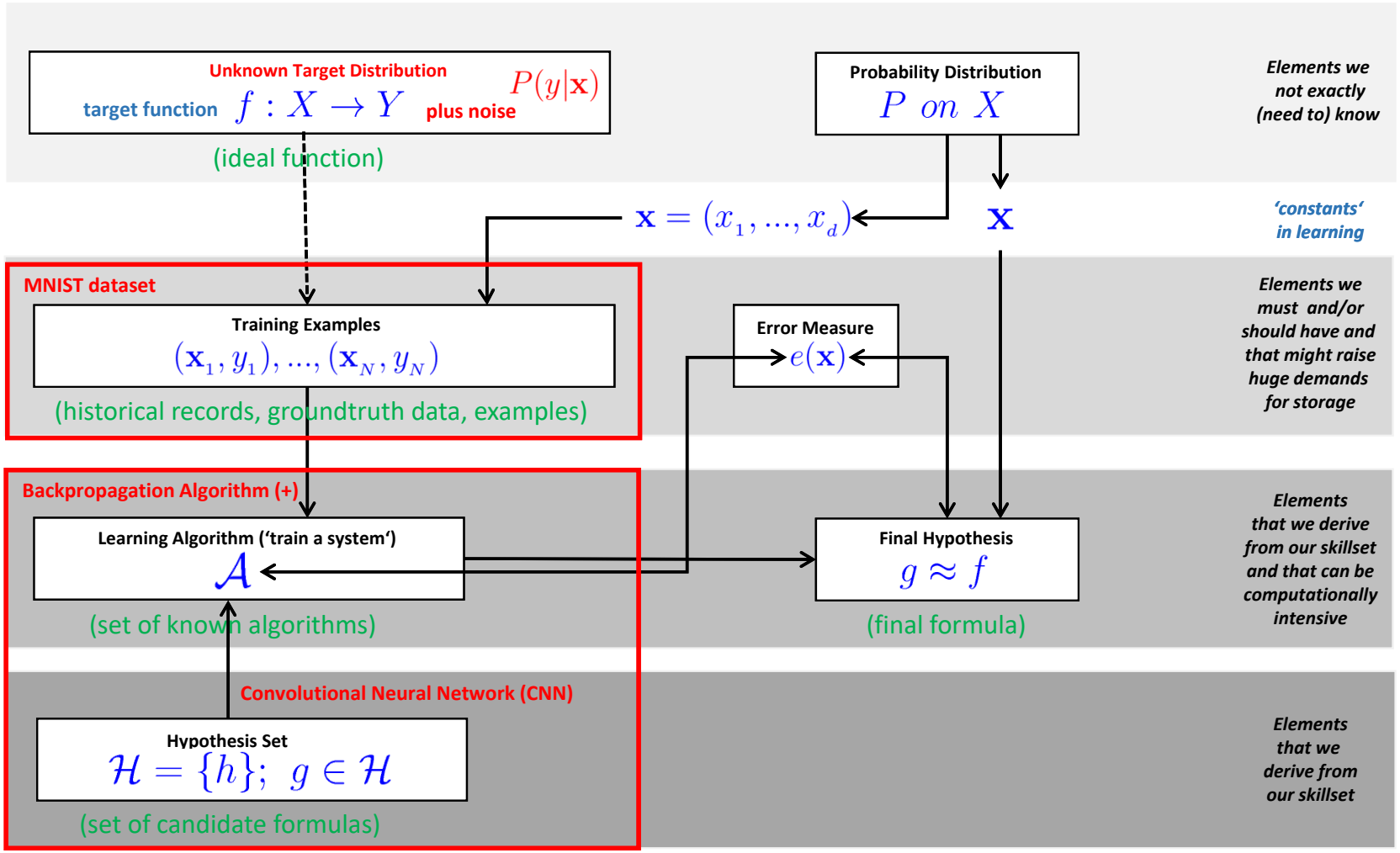
- ✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**
- ✓ **ANN 2 Hidden Layers: ~95,14 % (20 Epochs)**

```
# printout a summary of the model to understand model complexity
model.summary()
```

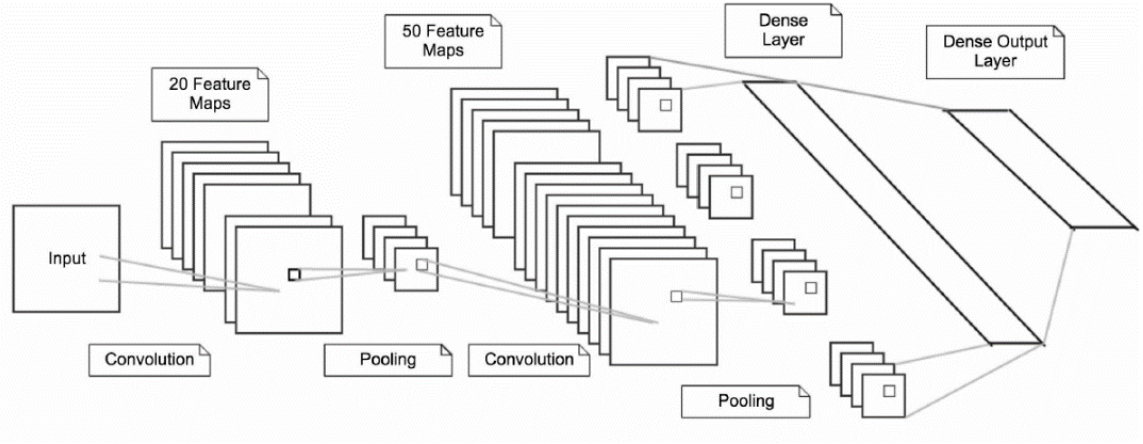



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

- **Dense Layer connects every neuron in this dense layer to the next dense layer with each of its neuron also called a fully connected network element with weights as trainiable parameters**
- **Choosing a model with different layers is a model selection that directly also influences the number of parameters (e.g. add Dense layer from Keras means new weights)**
- **Adding a layer with these new weights means much more computational complexity since each of the weights must be trained in each epoch (depending on #neurons in layer)**


Unknown Target Distribution
target function $f : X \rightarrow Y$ plus noise $P(y|\mathbf{x})$
(ideal function)
Probability Distribution
P on X
Elements we not exactly (need to) know
$\mathbf{x} = (x_1, ..., x_d)$
$\mathbf{X}$
'constants' in learning
MNIST dataset
Training Examples
$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
(historical records, groundtruth data, examples)
Error Measure
$e(\mathbf{x})$
Elements we must and/or should have and that might raise huge demands for storage
Backpropagation Algorithm (+)
Learning Algorithm ('train a system')
$\mathcal{A}$
(set of known algorithms)
Final Hypothesis
$g \approx f$
(final formula)
Elements that we derive from our skillset and that can be computationally intensive
Convolutional Neural Network (CNN)
Hypothesis Set
$\mathcal{H} = \{h\}; \ g \in \mathcal{H}$
(set of candidate formulas)
Elements that we derive from our skillset

MNIST Dataset – Convolutional Neural Network (CNN) Model – Architecture



[1] A. Gulli et al.

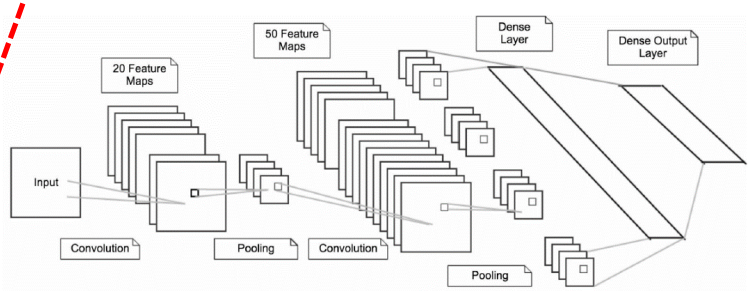
MNIST Dataset – Convolutional Neural Network (CNN) Model

```
from keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.datasets import mnist
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
import numpy as np
import matplotlib.pyplot as plt
```

```
#define the CNN model
class CNN:
  @staticmethod
  def build(input_shape, classes):
    model = Sequential()
    # CONV => RELU => POOL
    model.add(Conv2D(20, kernel_size=5, padding="same",
    input_shape=input_shape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    # CONV => RELU => POOL
    model.add(Conv2D(50, kernel_size=5, border_mode="same"))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    # Flatten => RELU layers
    model.add(Flatten())
    model.add(Dense(500))
    model.add(Activation("relu"))
    # a softmax classifier
    model.add(Dense(classes))
    model.add(Activation("softmax"))
    return model
```

- **Increasing the number of filters learned to 50 in the next layer from 20 in the first layer**
- **Increasing the number of filters in deeper layers is a common technique in deep learning architecture modeling**
- **Flattening the output as input for a Dense layer (fully connected layer)**
- **Fully connected / Dense layer responsible with softmay activation for classification based on learned filters and features**

[1] A. Gulli et al.




```
# initialize the optimizer and model
model = CNN.build(input_shape=INPUT_SHAPE, classes=NB_CLASSES)
model.compile(loss="categorical_crossentropy", optimizer=OPTIMIZER,
metrics=["accuracy"])
```

```
# printout a summary of the model to understand model complexity
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 20, 28, 28)	520
activation_1 (Activation)	(None, 20, 28, 28)	0
max_pooling2d_1 (MaxPooling2	(None, 20, 14, 14)	0
conv2d_2 (Conv2D)	(None, 50, 14, 14)	25050
activation_2 (Activation)	(None, 50, 14, 14)	0
max_pooling2d_2 (MaxPooling2	(None, 50, 7, 7)	0
flatten_1 (Flatten)	(None, 2450)	0
dense_1 (Dense)	(None, 500)	1225500
activation_3 (Activation)	(None, 500)	0
dense_2 (Dense)	(None, 10)	5010
activation_4 (Activation)	(None, 10)	0

```
Total params: 1,256,080
Trainable params: 1,256,080
Non-trainable params: 0
```

MNIST Dataset – Model Parameters & 2D Input Data

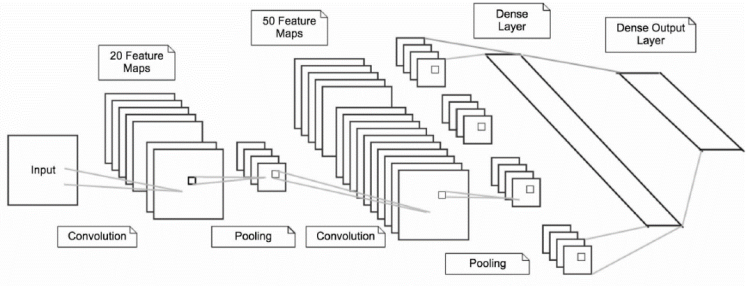
```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
OPTIMIZER = Adam()
VALIDATION_SPLIT=0.2
IMG_ROWS, IMG_COLS = 28, 28 # input image dimensions
NB_CLASSES = 10 # number of outputs = number of digits
INPUT_SHAPE = (1, IMG_ROWS, IMG_COLS)
```

```
# data: shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
K.set_image_dim_ordering("th")
# consider them as float and normalize
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
# we need a 60K x [1 x 28 x 28] shape as input to the CONVNET
X_train = X_train[:, np.newaxis, :, :]
X_test = X_test[:, np.newaxis, :, :]
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
# convert class vectors to binary class matrices
y_train = np_utils.to_categorical(y_train, NB_CLASSES)
y_test = np_utils.to_categorical(y_test, NB_CLASSES)
```

- **OPTIMIZER: Adam - advanced optimization technique that includes the concept of a momentum (a certain velocity component) in addition to the acceleration component of Stochastic Gradient Descent (SGD)**
- **Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients**
- **Adam enables faster convergence at the cost of more computation and is currently recommended as the default algorithm to use (or SGD + Nesterov Momentum)**

[2] D. Kingma et al., 'Adam: A Method for Stochastic Optimization'

- **Compared to the Multi-Output Perceptron and Artificial Neural Networks (ANN) model, the input dataset remains as 2d matricew with 1 x 28 x 28 per image, including also the class vectors that are converted to binary class matrices**



Exercises – Train CNN & Evaluate Output

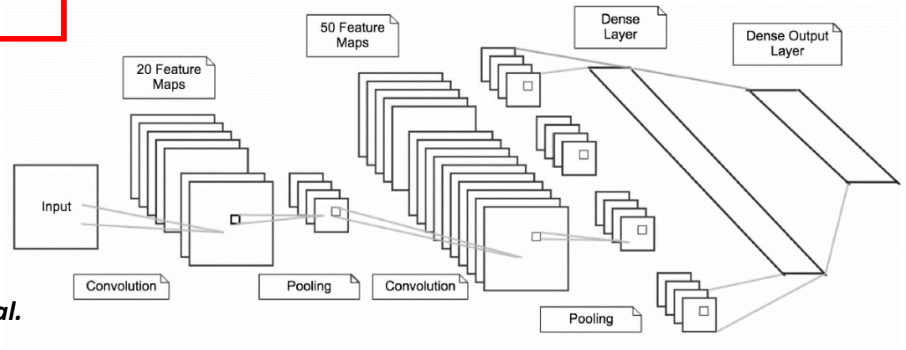
MNIST Dataset – CNN Model Output & Evaluation

```
Epoch 14/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0065 - acc: 0.9980 - val_loss: 0.0346 - val_acc: 0.9921
Epoch 15/20
48000/48000 [==============================] - 4s 89us/step - loss: 0.0030 - acc: 0.9990 - val_loss: 0.0418 - val_acc: 0.9903
Epoch 16/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0057 - acc: 0.9980 - val_loss: 0.0470 - val_acc: 0.9910
Epoch 17/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0043 - acc: 0.9985 - val_loss: 0.0440 - val_acc: 0.9906
Epoch 18/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0046 - acc: 0.9985 - val_loss: 0.0474 - val_acc: 0.9891
Epoch 19/20
48000/48000 [==============================] - 4s 88us/step - loss: 0.0047 - acc: 0.9986 - val_loss: 0.0353 - val_acc: 0.9928
Epoch 20/20
48000/48000 [==============================] - 4s 88us/step - loss: 3.4055e-04 - acc: 1.0000 - val_loss: 0.0374 - val_acc: 0.9927
```

```python
# model evaluation
score = model.evaluate(X_test, y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 1s 70us/step
Test score: 0.0303058747581508
Test accuracy: 0.9936
```

- ✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**
- ✓ **ANN 2 Hidden Layers: ~95,14 % (20 Epochs)**
- ✓ **CNN Deep Learning Model: ~99,36 % (20 Epochs)**



[1] A. Gulli et al.

Why not 100%

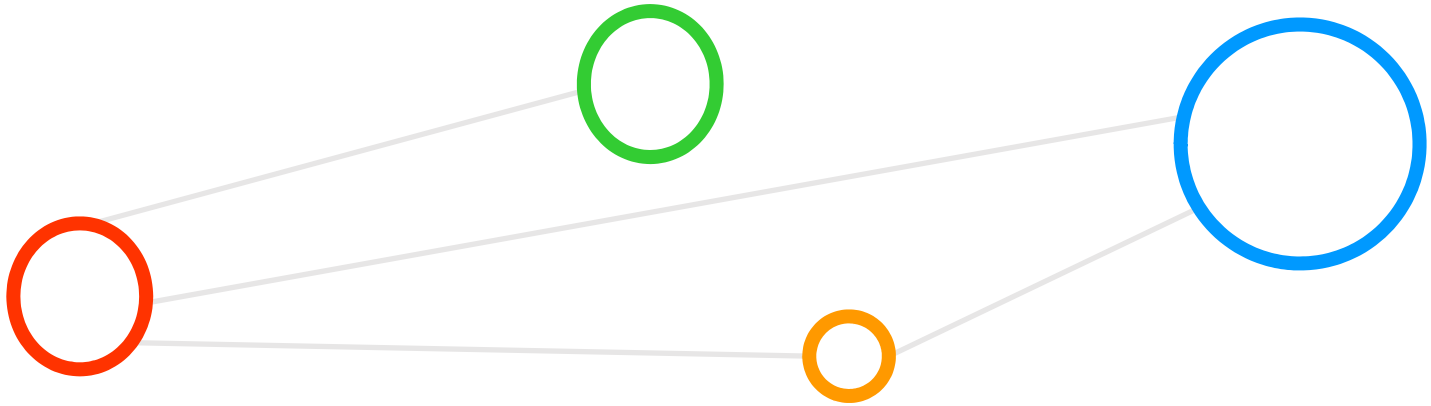
some samples even for a human unrecognizable

Exercises – CNN GPU vs. CPU Execution Examples

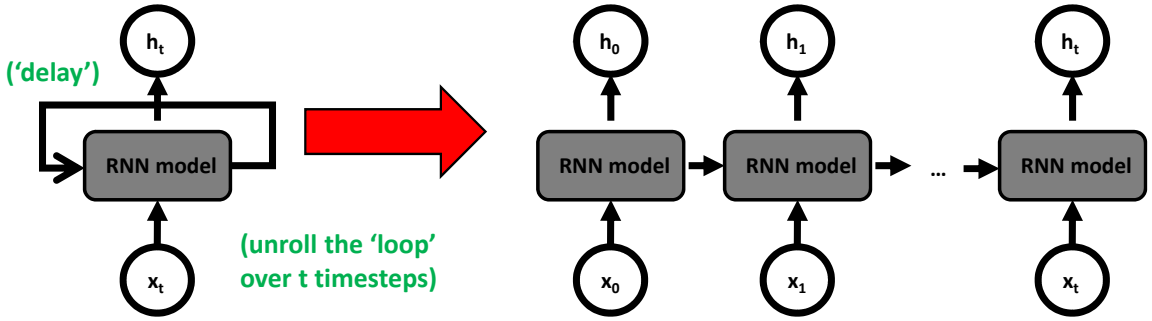
[Video] Deep Learning Revolution

[11] YouTube Video, Deep Learning Revolution

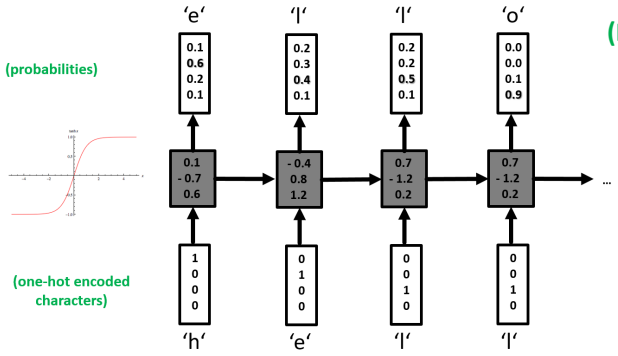
Other Deep Neural Network (DNN) Examples

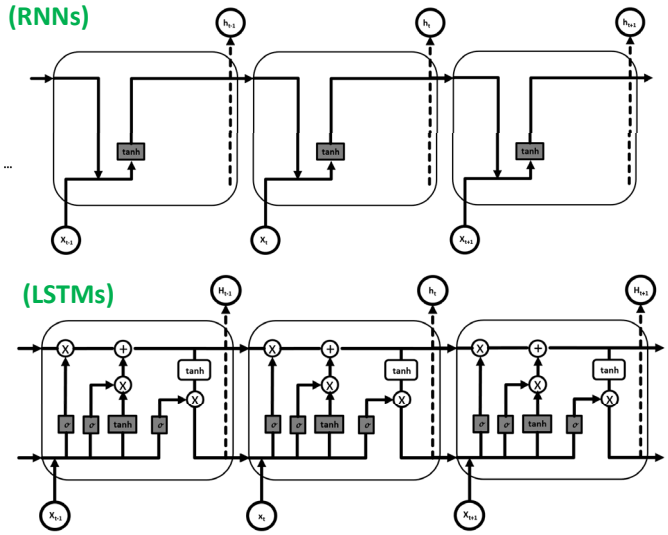


More Complex Deep Learning Model Example: Long Short-Term Memory (LSTM)

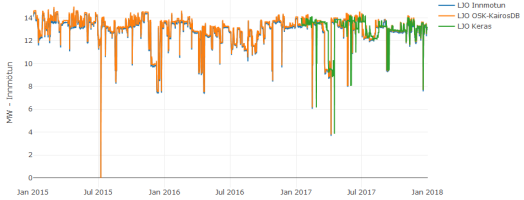


- A Recurrent Neural Network (RNN) consists of cyclic connections that enable the neural network to better model sequence data compared to a traditional feed forward artificial neural network (ANN)
- RNNs consists of 'loops' (i.e. cyclic connections) that allow for information to persist while training
- The repeating RNN model structure is very simple whereby each has only a single layer (e.g. tanh)





(Key challenge: find the right parameters)



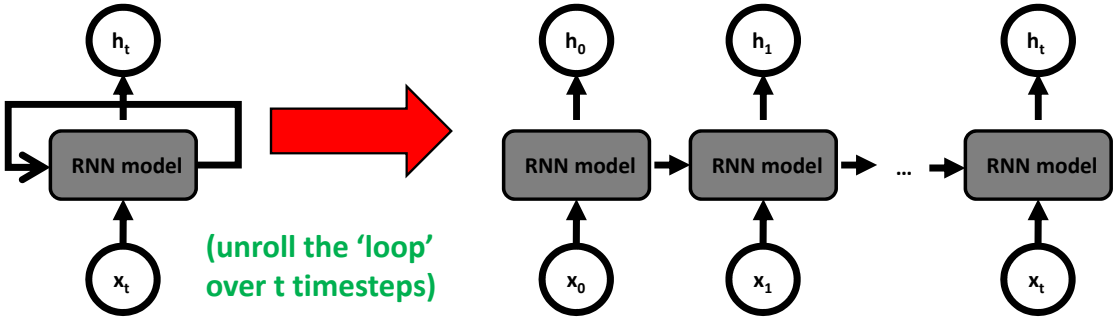
Landsvirkjun
National Power Company of Iceland

- Long Short Term Memory (LSTM) networks are a special kind of Recurrent Neural Networks (RNNs)
- LSTMs learn long-term dependencies in data by remembering information for long periods of time
- The LSTM chain structure consists of four neural network layers interacting in a specific way

Different Useful LSTM Models

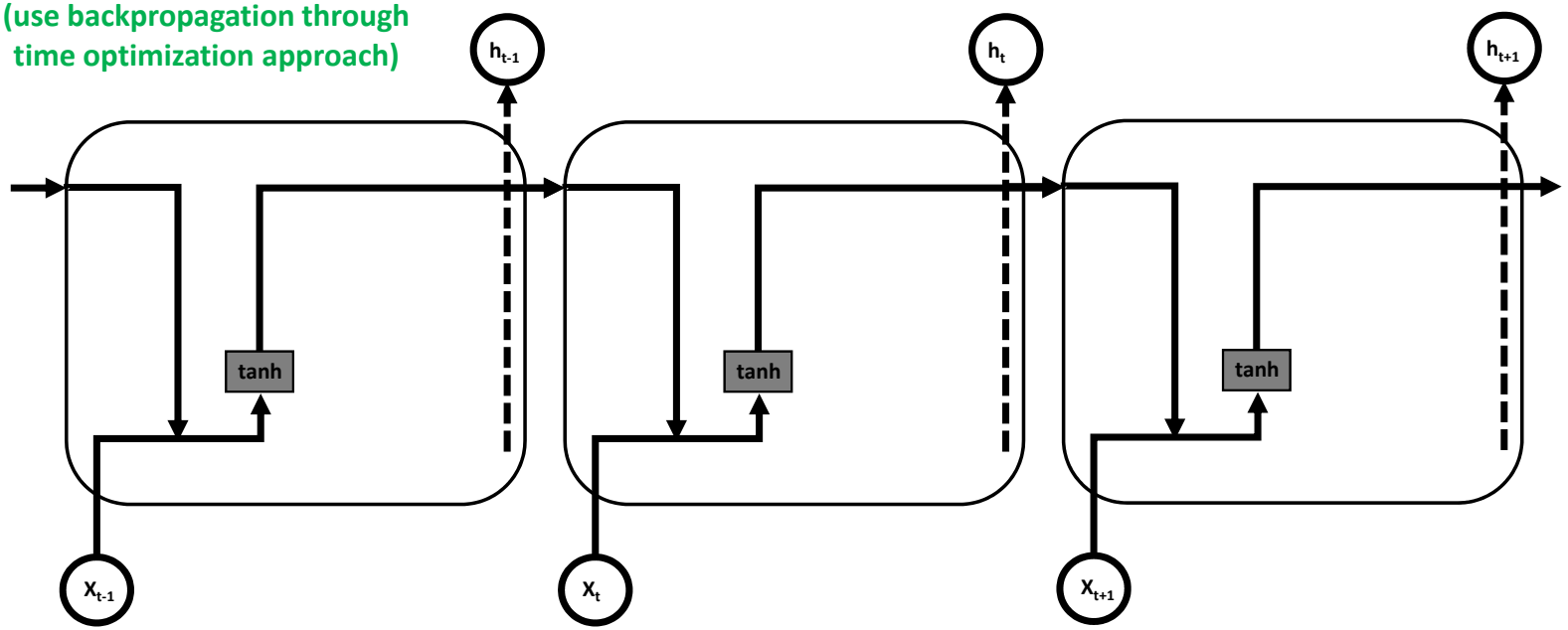
- Standard LSTM
  - Memory cells with single LSTM layer; used in simple network structures
- Stacked LSTM
  - LSTM layers are stacked one on top of another; creating deep networks
- CNN LSTM
  - CNNs to learn features (e.g. images); LSTM for image sequences
- Encoder-Decoder LSTM
  - One LSTM network → encode input; one LSTM network → decode output
- Bidirectional LSTM
  - Input sequences are presented and learned both forward & backwards
- Generative LSTM
  - LSTMs learn the inherent structure relationship in input sequences; then generate new plausible sequences

Recurrent Neural Networks – Basics & Unrolled

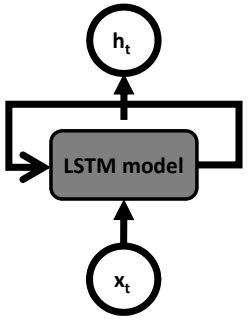


- **A RNN can be viewed as multiple copies of the same network, each passing a message to a successor – this gets clear when 'unrolling the RNN loop'**

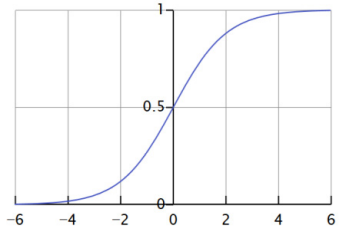
(use backpropagation through time optimization approach)



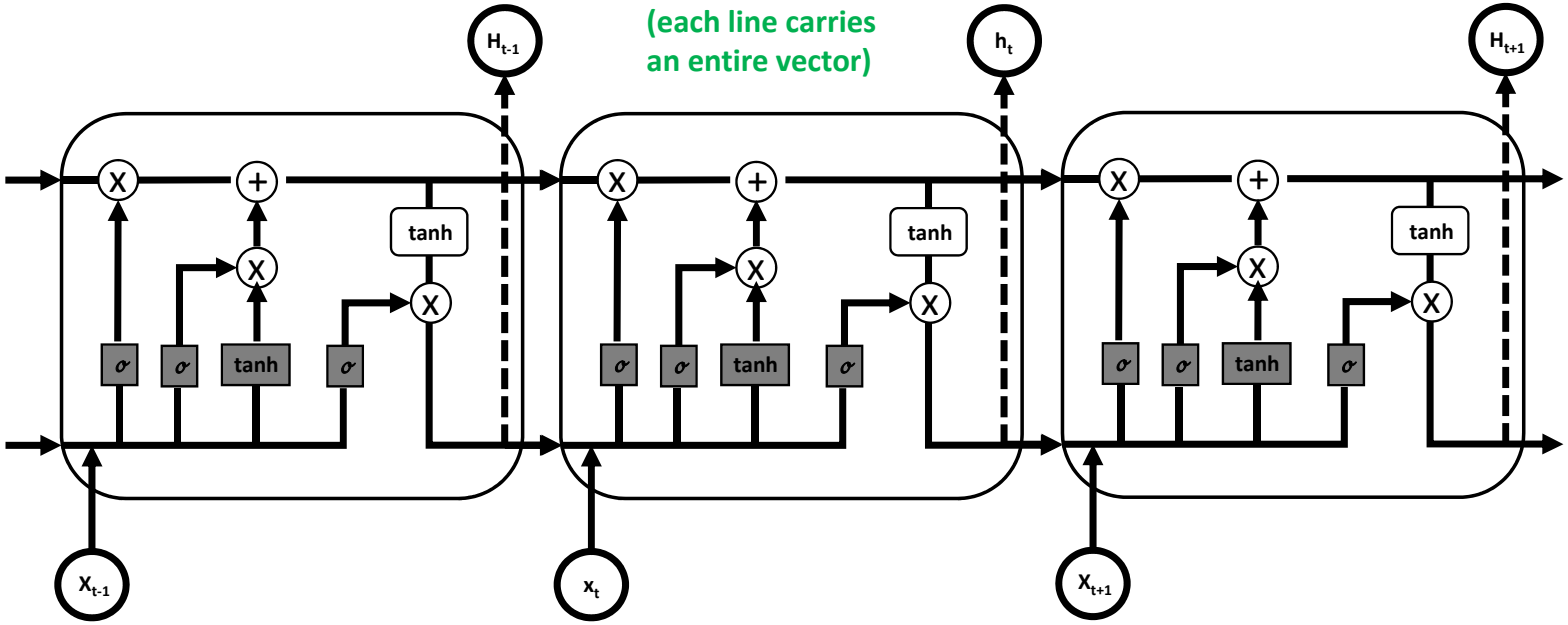
LSTM Model – Basics



- Long Short Term Memory (LSTM) networks are a special kind of Recurrent Neural Networks (RNNs)
- LSTMs learn long-term dependencies in data by remembering information for long periods of time
- The LSTM chain structure consists of four neural network layers interacting in a specific way

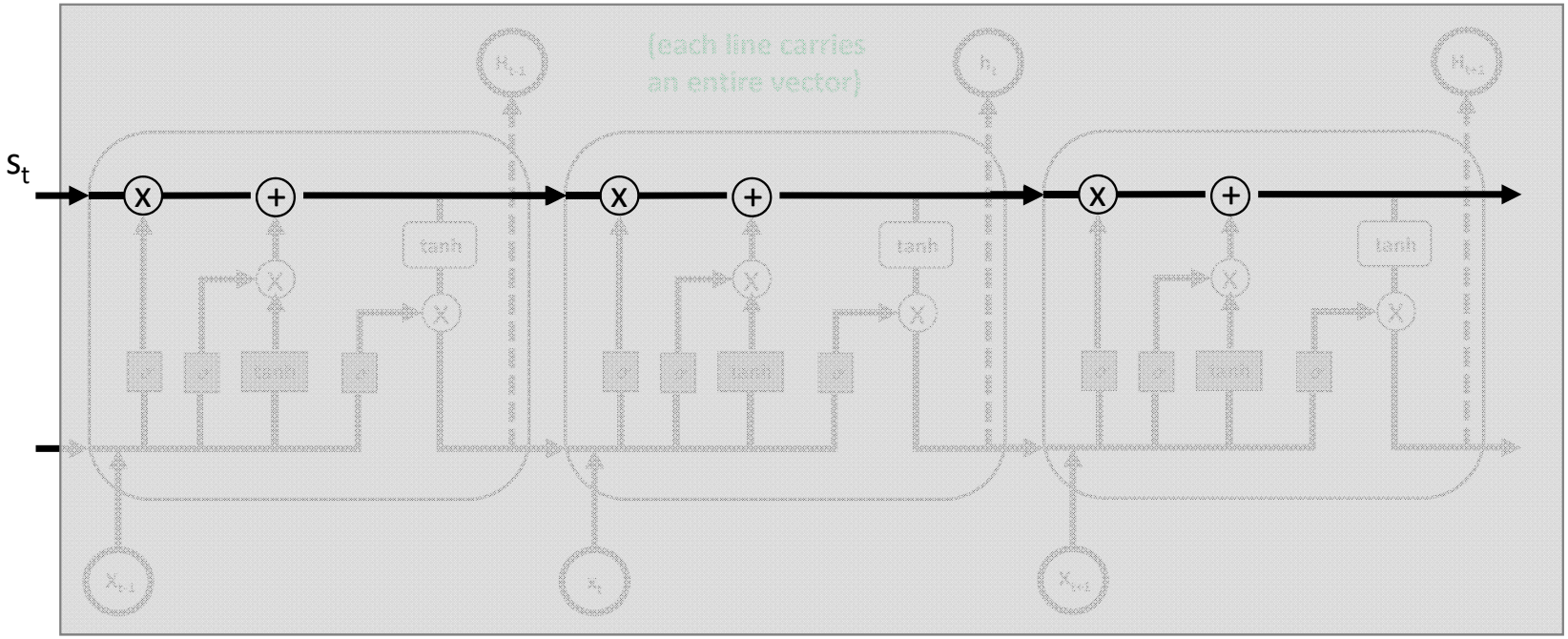


(uses sigmoid σ)



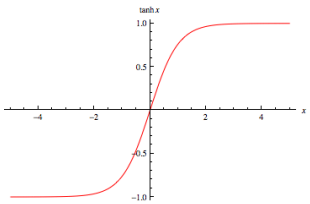
LSTM Model – Memory Cell & Cell State

- **LSTM introduce a ‘memory cell‘ structure into the underlying basic RNN architecture using four key elements: an input gate, a neuron with self-current connection, a forget gate, and an output gate**
- **The data in the LSTM memory cell flows straight down the chain with some linear interactions (x,+)**
- **The cell state s_t can be different at each of the LSTM model steps & modified with gate structures**
- **Linear interactions of the cell state are pointwise multiplication (x) and pointwise addition (+)**
- **In order to protect and control the cell state s_t three different types of gates exist in the structure**



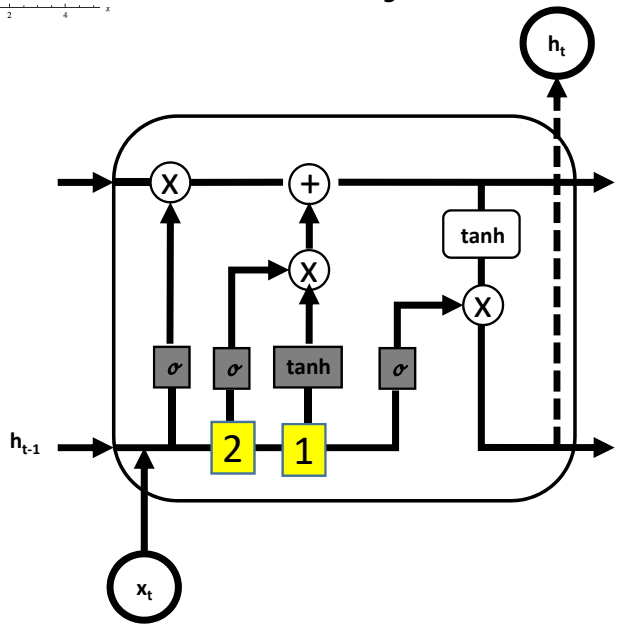
Computing of LSTM Cell – Step 1-2

1. New x_t input together with the output from cell h_{ht-1} are squashed via a tanh layer
   - Outputs between -1 and 1

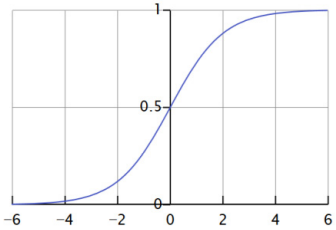


[12] Adventures in Machine Learning

2. New x_t input together with the output from cell h_{ht-1} is passed through the 'input gate'
   - Layer of sigmoid activated nodes whose output is multiplied by squashed input



$$i = \sigma(b^i + x_t U^i + h_{t-1} V^i)$$



(uses sigmoid σ)

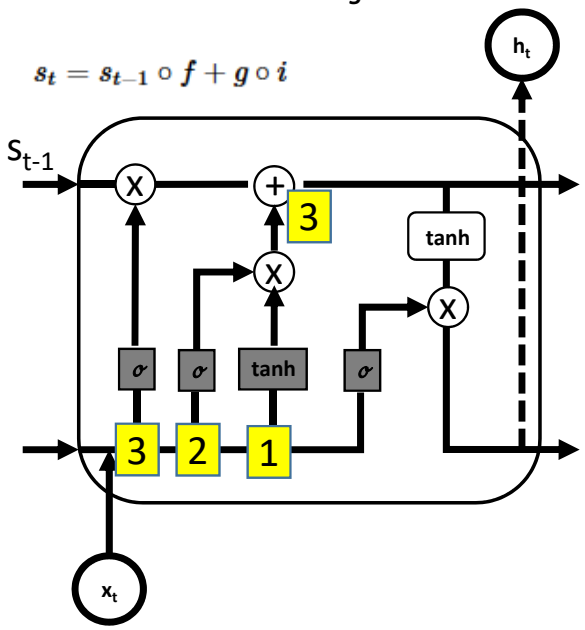
(gate sigmoid σ can act to 'switch off' any elments of the input vector that are not required)

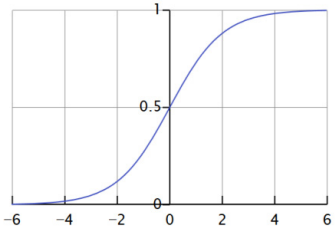
(sigmoid function outputs values between 0 and 1, weights connecting the input to these nodes can be trained to output values close to zero to 'switch off' certain input values – or outputs close to 1 to 'pass through')

Computing of LSTM Cell – Step 3

3. Internal state / forget gate $f = \sigma(b^f + x_t U^f + h_{t-1} V^f)$
   - LSTM cells have internal cell state s_t
   - 'Delay' – lagged one time step: s_{t-1}
   - Added to the input data to create an effective 'layer of recurrence'
   - Addition instead of 'usual' multiplication reduces risk of vanishing gradients
   - The connection to cell state is carefully controlled by a forget gate with sigmoid (works like the input gate)

[12] Adventures in Machine Learning





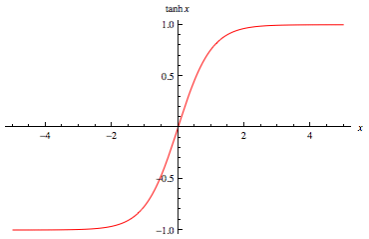
(uses sigmoid σ)

(gate sigmoid σ can act to 'switch off' any elments of the cell state to steer what variables should be remembered or forgotten)

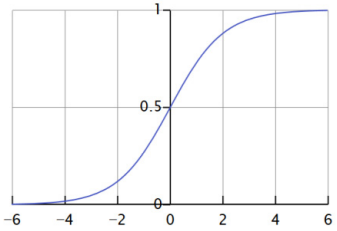
Computing of LSTM Cell – Step 4

4. Output layer & output gate $o = \sigma(b^o + x_t U^o + h_{t-1} V^o)$

- Output layer with tanh squashing function



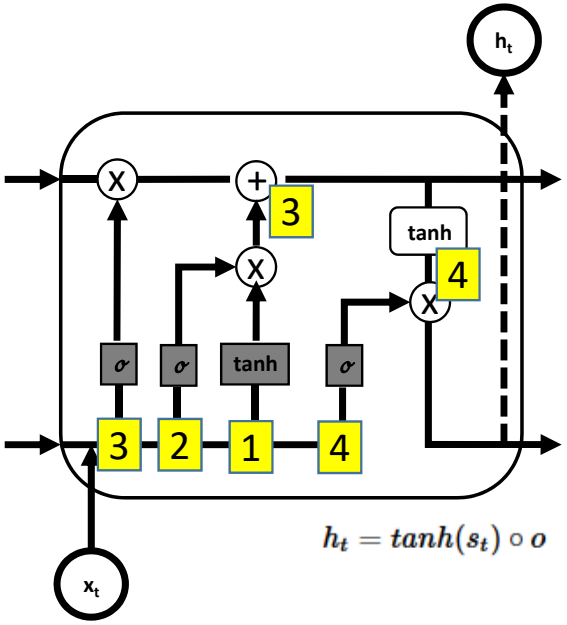
- Output is controlled via output gate with sigmoid activation function



(uses sigmoid σ)

(gate sigmoid σ can learn to determine which values are allowed as an output from the cell)

[12] Adventures in Machine Learning



$h_t = tanh(s_t) \circ o$

Low-Level Tools – Tensorflow

- Tensorflow is an open source library for deep learning models using a flow graph approach
- Tensorflow nodes model mathematical operations and graph edges between the nodes are so-called tensors (also known as multi-dimensional arrays)
- The Tensorflow tool supports the use of CPUs and GPUs (much more faster than CPUs)
- Tensorflow work with the high-level deep learning tool Keras in order to create models fast
- LSTM models are created using tensors & graphs and there are LSTM package contributions

[13] Tensorflow Deep Learning Framework

```
...
lstm = rnn_cell.BasicLSTMCell(lstm_size, state_is_tuple=False)
...
stacked_lstm = rnn_cell.MultiRNNCell([lstm] * number_of_layers,
    state_is_tuple=False)
...
initial_state = state = stacked_lstm.zero_state(batch_size, tf.float32)

for i in range(num_steps):
    # The value of state is updated
    # after processing each batch of words.
    output, state = stacked_lstm(words[:, i], state)

    # The rest of the code.
    # ...

final_state = s
```

- The class BasicLSTMCell() offers a simple LSTM Cell implementation in Tensorflow

High-level Tools – Keras

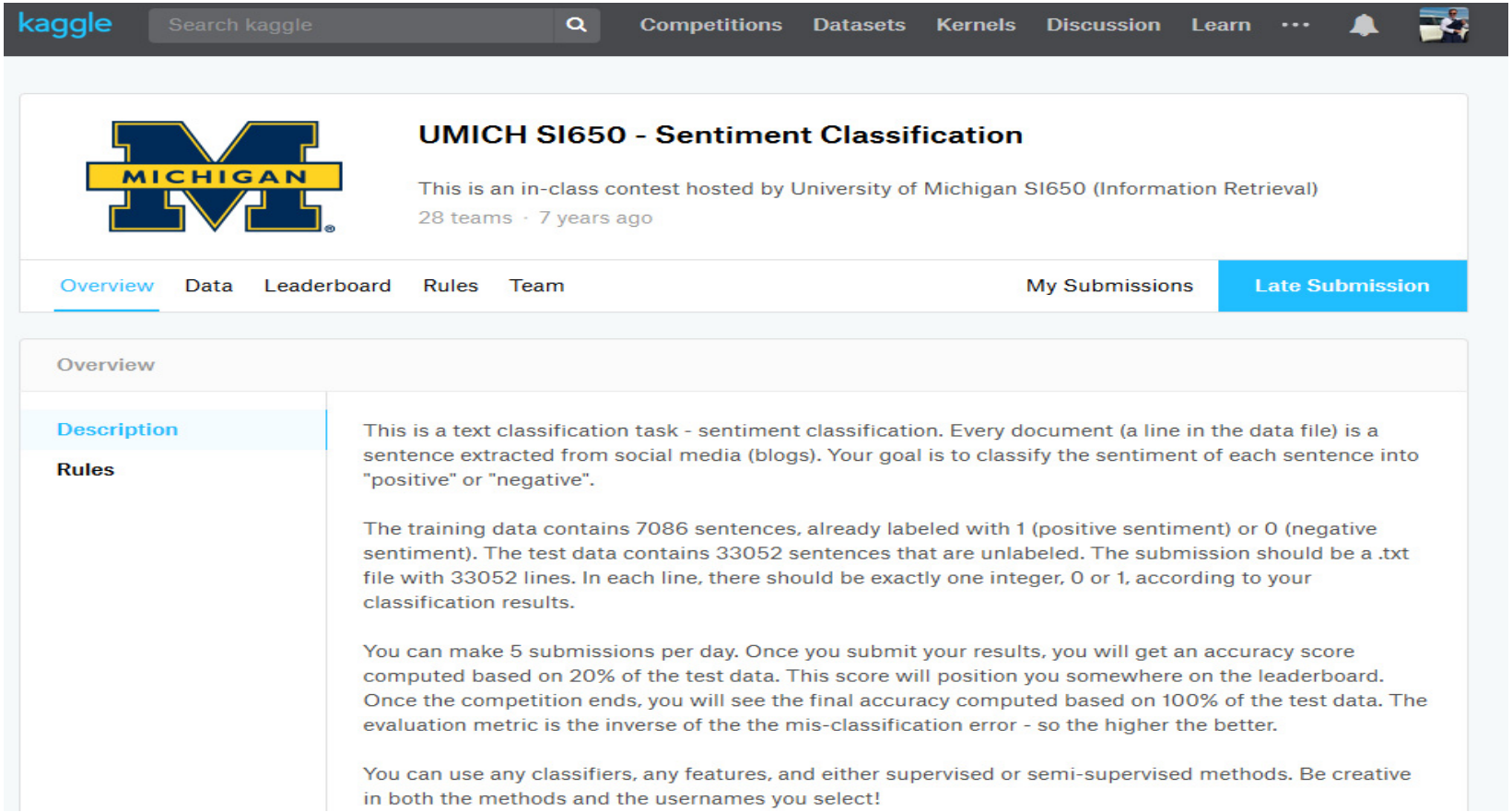
- Keras is a high-level deep learning library implemented in Python that works on top of existing other rather low-level deep learning frameworks like Tensorflow, CNTK, or Theano
- The key idea behind the Keras tool is to enable faster experimentation with deep networks
- Created deep learning models run seamlessly on CPU and GPU via low-level frameworks

```
keras.layers.LSTM(    units,
                      activation='tanh',
                      recurrent_activation='hard_sigmoid',
                      use_bias=True,
                      kernel_initializer='glorot_uniform',
                      recurrent_initializer='orthogonal',
                      bias_initializer='zeros',
                      unit_forget_bias=True,
                      kernel_regularizer=None,
                      recurrent_regularizer=None,
                      bias_regularizer=None,
                      activity_regularizer=None,
                      kernel_constraint=None,
                      recurrent_constraint=None,
                      bias_constraint=None,
                      dropout=0.0, ...)
```

- Tool Keras supports the LSTM model via keras.layers.LSTM() that offers a wide variety of configuration options

[14] Keras Python Deep Learning Library

LSTM Example – Data Repository



[15] Kaggle, UMICH SI650 – Sentiment Classification Data

LSTM Example – Dataset & Application

- Sentiment analysis (many-to-one RNN topology)
  - Input: sentence as sequence of words (i.e. movie ratings texts)
  - Output: Sentiment value (positive/negative movie rating)
  - Application was a former competition (i.e. Kaggle platform overall idea)
  - Goal: Create LSTM network that will learn to predict a correct sentiment
- Small dataset example for tutorial: training & test data available
  - Training samples: 7086 short sentences (labelled) [~440 KB]
  - Test samples: 33052 short sentences[~1.94 MB]
  - Format: label & tab seperated sentence
  - https://www.kaggle.com/c/si650winter11/data

[15] Kaggle, UMICH SI650 – Sentiment Classification Data

LSTM Example – Dataset Exploration

- Create
  directory lstm
- Copy data
  - From the training project folder to your own home directory

```
-bash-4.2$ head training.txt
1       The Da Vinci Code book is just awesome.
1       this was the first clive cussler i've ever read, but even books like Relic, and Da Vinci code were more plausible than this.
1       i liked the Da Vinci Code a lot.
1       i liked the Da Vinci Code a lot.
1       I liked the Da Vinci Code but it ultimatly didn't seem to hold it's own.
1       that's not even an exaggeration ) and at midnight we went to Wal-Mart to buy the Da Vinci Code, which is amazing of course.
1       I loved the Da Vinci Code, but now I want something better and different!..
1       i thought da vinci code was great, same with kite runner.
1       The Da Vinci Code is actually a good movie...
1       I thought the Da Vinci Code was a pretty good book.
```

(labelled training dataset)

```
-bash-4.2$ head testdata.txt
" I don't care what anyone says, I like Hillary Clinton.
have an awesome time at purdue!..
Yep, I'm still in London, which is pretty awesome: P Remind me to post the million and one pictures that I took when I get back to Markham!...
Have to say, I hate Paris Hilton's behavior but I do think she's kinda cute..
i will love the lakers.
I'm so glad I love Paris Hilton, too, or this would be excruciating.
considering most Geico commericals are stupid...
i liked MIT though, esp their little info book(
Before I left Missouri, I thought London was going to be so good and cool and fun and a really great experience and I was really excited.
I still like Tom Cruise.
```

(testing dataset)

LSTM Example – Keras Python Script – Preprocessing

```
from keras.layers.core import Activation, Dense, Dropout, SpatialDropout1D
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
import collections
import matplotlib.pyplot as plt
import nltk
import numpy as np
import os
```

- Import necessary modules, e.g. LSTM for a simple LSTM cell, or Dense for a fully connected layer
- Import good sklearn model selection tools
- Import numpy for as helper tool

```
# obtain punkt if not there already
nltk.download('punkt')
```

- Natural Language Toolkit (NLTK) is for building Python programs working on human language datasets (punkt is tokenizer)

```
# define a data directory
DATA_DIR = "/p/home/jusers/riedel1/jureca"
```

- Location for labeled training data and testset data

→ Note same issue with downloading as with the mnist data – needs first to be executed with python lstm-example.py on login-node – then in interactive session

[16] Deep Learning with Keras

LSTM Example – Keras Python Script – Vocabulary Setup

```
# exploratory analysis
maxlen = 0
word_freqs = collections.Counter()
num_recs = 0
ftrain = open(os.path.join(DATA_DIR, "training.txt"), 'rb')
for line in ftrain:
    label, sentence = line.strip().split('\t')
    words = nltk.word_tokenize(sentence.decode("ascii", "ignore").lower())
    if len(words) > maxlen:
        maxlen = len(words)
    for word in words:
        word_freqs[word] += 1
    num_recs += 1
ftrain.close()
```

- Perform exploratory analysis in order to find out the number of unique words in the whole corpus & how many words are roughly in each sentence

```
# vocabulary setup
MAX_FEATURES = 2000
MAX_SENTENCE_LENGTH = 40
```

- Exploration reveals maxlen: 42 & len(word_freqs): 2313
- Number of words in sentence (maxlen) enables a fixed sequence length & PAD = 0; truncate long ones

```
# vocabulary and indices
vocab_size = min(MAX_FEATURES, len(word_freqs)) + 2
word2index = {x[0]: i+2 for i, x in
enumerate(word_freqs.most_common(MAX_FEATURES))}
word2index["PAD"] = 0
word2index["UNK"] = 1
index2word = {v:k for k, v in word2index.items()}
```

- Creating indices index2word and vice versa
- Out of vocabulary means UNK (unknown)

[16] Deep Learning with Keras

LSTM Example – Keras Python Script – Indices & Padding

```
# input sentence -> word index sequences
X = np.empty((num_recs, ), dtype=list)
y = np.zeros((num_recs, ))
i = 0
ftrain = open(os.path.join(DATA_DIR, "training.txt"), 'rb')
for line in ftrain:
  label, sentence = line.strip().split('\t')
  words = nltk.word_tokenize(sentence.decode("ascii", "ignore").lower())
  seqs = []
  for word in words:
    if word2index.has_key(word):
      seqs.append(word2index[word])
    else:
      seqs.append(word2index["UNK"])
  X[i] = seqs
  y[i] = int(label)
  i += 1
ftrain.close()
```

- Convert input sentences from the training data to word index sequences and add unknown ones as UNK in index

```
# perform padding if needed
X = sequence.pad_sequences(X, maxlen=MAX_SENTENCE_LENGTH)
```

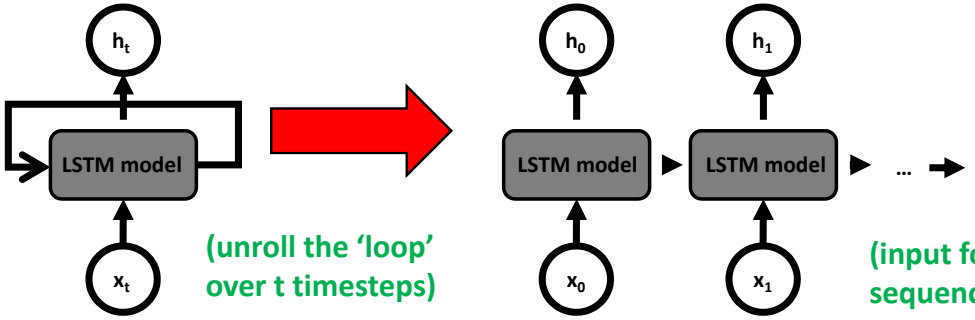
- Perform padding to the maximum sentence length (40)
- Labels are binary (positive/negative sentiment) and do not need padding

```
# split into training and testing
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y,
     test_size=0.2, random_state=42)
```

- Split between training & testing set (ratio rule of thumb 80:20)
- There is another test set put aside for nicely checking out-of-sample

[16] Deep Learning with Keras

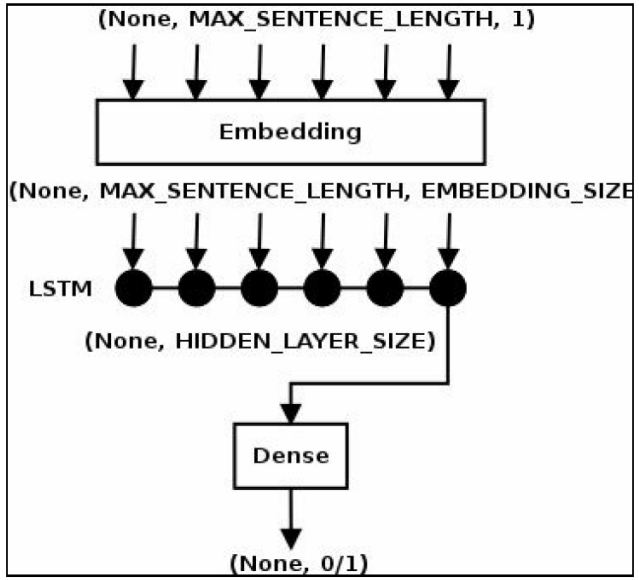
LSTM Example – Modelling & Decisions



(unroll the 'loop' over t timesteps)

(input for each row is a sequence of word indices – sequence length is given by MAX_SENTENCE_LENGTH)

(tensor layout: None X MAX_SENTENCE_LENGTH X 1)



(tensor dimensions: first is None →
indicate that the batch size is currently unknown,
i.e. number of records fed to the network →
defined in runtime using BATCH_SIZE parameter)

(tensor fed to embedding layer →
weights are initialized with small random values & learned
i.e. layer transforms the tensor to a shape of
None X MAX_SENTENCE_LENGTH X EMBEDDING_SIZE)

(output of LSTM is the tensor
None X HIDDEN_LAYER_SIZE, because last tensor can be defined
as return_sequences = False → we just need 0/1 output)

(Dense layer with Sigmoid activation function →
0 – negative review / 1 positive review)

modified from [16] Deep Learning with Keras

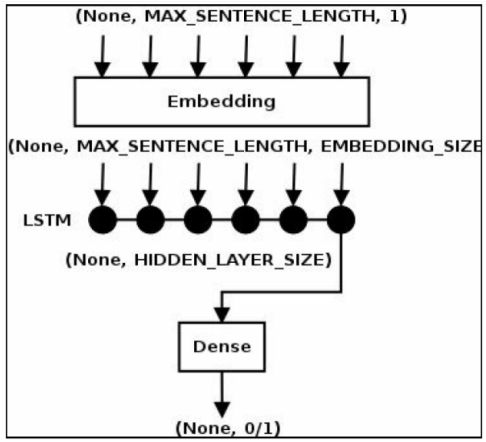
LSTM Example – Keras Python Script – Model & Parameter

```
#hyperparameter
EMBEDDING_SIZE = 128
HIDDEN_LAYER_SIZE = 64
BATCH_SIZE = 32
NUM_EPOCHS = 10
```

- Hyperparameters embedding=128; hidden layers=64; parameter by experimentation

```
# LSTM model
model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_SIZE,
input_length=MAX_SENTENCE_LENGTH))
model.add(SpatialDropout1D(Dropout(0.2)))
model.add(LSTM(HIDDEN_LAYER_SIZE, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1))
model.add(Activation("sigmoid"))
model.compile(loss="binary_crossentropy", optimizer="adam",
  metrics=["accuracy"])
```

- Create first embedding layer with input tensor None X maximum sentence length X 1
- Add regularizer SpatialDropout1D
- Add LSTM cell with hidden layer size 64 with regularizers dropout and recurrent_dropout
- Add Dense layer and Sigmoid activation



- All hyperparameters are tuned experimentally over many runs
- Compile model using binary cross-entropy loss function good for a binary model used here
- Use of Adam optimizer as good general purpose optimizer

[16] Deep Learning with Keras

LSTM Example – Keras Python Script – Train & Evaluate

```
# training
story = model.fit(Xtrain, ytrain, batch_size=BATCH_SIZE, epochs=NUM_EPOCHS,
validation_data=(Xtest, ytest))
```

```
# evaluation
score, acc = model.evaluate(Xtest, ytest, batch_size=BATCH_SIZE)
print("Test score: %.3f, accuracy: %.3f" % (score, acc))
for i in range(5):
  idx = np.random.randint(len(Xtest))
  xtest = Xtest[idx].reshape(1,40)
  ylabel = ytest[idx]
  ypred = model.predict(xtest)[0][0]
  sent = " ".join([index2word[x] for x in xtest[0].tolist() if x != 0])
  print("%.0ft  %dt  %s" % (ypred, ylabel, sent))
```

- **Train the LSTM network for 10 epochs (NUM_EPOCHS) & with batch size 32**
- **Perform validation at each epoch using test data**

- Cf. supervised learning process (day one)
  - Labels existing (not in this unsupervised example)
  - Train model for fixed number of epochs
  - Evaluate model against test dataset (splitted training)
- Home Work: Use model for prediction of the real ‘test-data‘ (not splitted training)
  - Note: real ‘test-data‘ has no labels, aka unseen data

Exercises – Train LSTM Model and Evaluate Output

LSTM Example – Output Interpretation

- Supervised learning problem
  - Check output with ‘more out.txt‘
  - Idea: predicted sentiment should be closed to sentiment labels
  - More epochs/iterations → better quality of the model

```
Train on 5668 samples, validate on 1418 samples
Epoch 1/10

  32/5668 [..............................] - ETA: 35:08 - loss: 0.6938 - acc: 0.4688
  64/5668 [..............................] - ETA: 17:36 - loss: 0.6927 - acc: 0.5312
  96/5668 [..............................] - ETA: 11:45 - loss: 0.6911 - acc: 0.5625
```

(learned well compared to first iteration → one can observe loss decrease and increase in accuracy over multiple epochs)

```
5664/5668 [============================>.] - ETA: 0s - loss: 0.0015 - acc: 0.9995
5668/5668 [==============================] - 15s 3ms/step - loss: 0.0015 - acc: 0.9995 - val_loss: 0.0845 - val_acc: 0.9718
Epoch 10/10

  32/5668 [..............................] - ETA: 13s - loss: 0.0697 - acc: 0.9688
  64/5668 [..............................] - ETA: 13s - loss: 0.0353 - acc: 0.9844
  96/5668 [..............................] - ETA: 13s - loss: 0.0240 - acc: 0.9896
```

```
Test score: 0.072, accuracy: 0.980
1t  1t  the people who are worth it know how much i love the da vinci code .
1t  1t  anyway , thats why i love `` brokeback mountain .
0t  0t  the da vinci code sucked .
0t  0t  this quiz sucks and harry potter sucks ok bye..
1t  1t  because i would like to make friends who like the same things i like ,
```

Exercises – Train LSTM Model and Change Parameters of the Model & Training

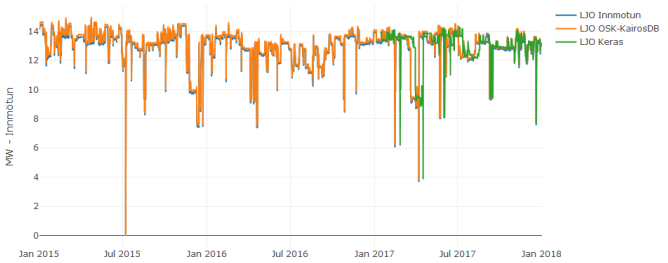
Different Useful LSTM Models – Many other applications

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

```
# design network
model = Sequential()
model.add(LSTM(
    units=config['units'],
    input_shape=(train_X.shape[1], train_X.shape[2])
))
model.add(Dense(1, activation=config['activation']))

model.compile(loss=config['loss'], optimizer=config['optimizer'])

# fit network
print("Fitting model..")

history = model.fit(
    train_X,
    train_y,
    epochs=config['epochs'],
    batch_size=config['batchsize'],
    validation_data=(test_X, test_y),
    verbose=2,
    shuffle=config['shuffle']
)
```

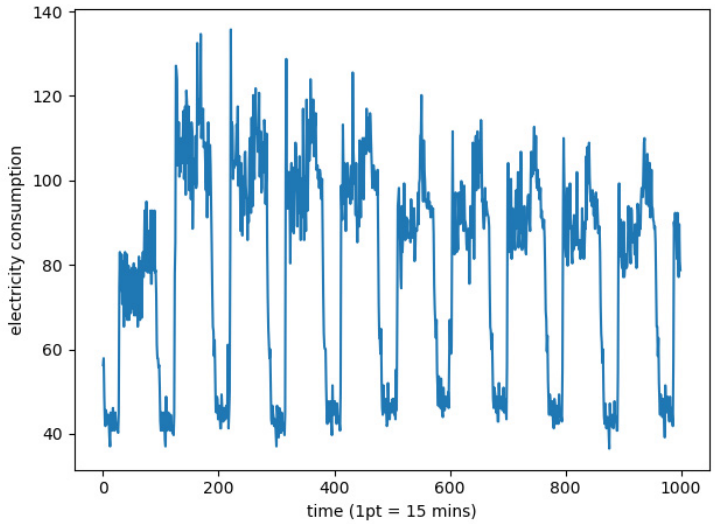
- **LSTM models work quite well to predict power but needs to be trained and tuned for different power stations**
- **Observing that some peaks can not be 'learned'**

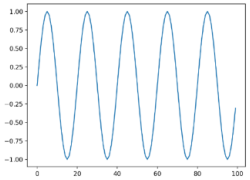
Landsvirkjun
National Power Company of Iceland



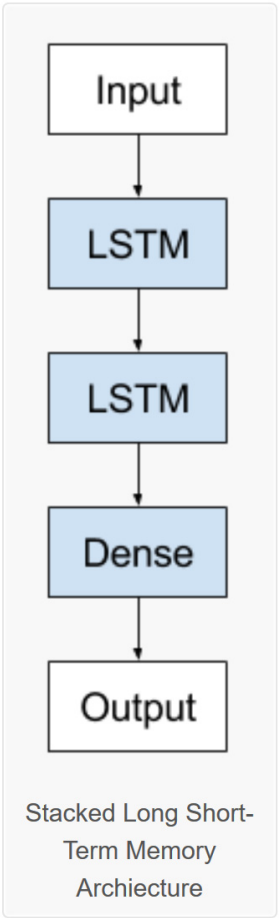
Different Useful LSTM Models – Stacked LSTMs

- E.g. predicting electricity consumption / customer
  - Stacked LSTM cells
  - Periodic elements can take advantage of state
  - Needs to be carefully tuned
  - Requires through use of state more computing
- E.g. damped sine wave prediction
  - Stacked LSTM cells since again periodic character
  - Depending on wave the pattern might be not able to be detected w/o LSTMs





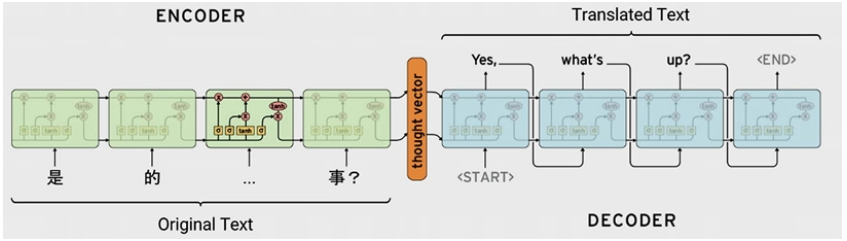
Building a deep RNN by stacking multiple recurrent hidden states on top of each other. This approach potentially allows the hidden state at each level to operate at different timescale

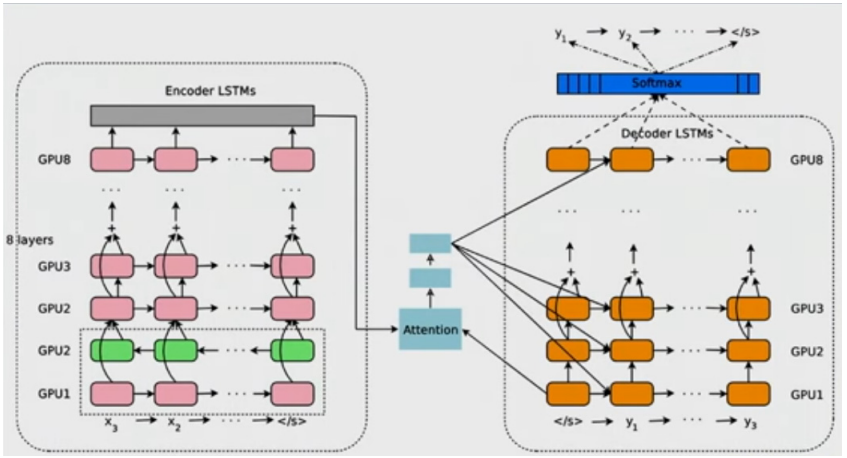


Stacked Long Short-Term Memory Archiecture

Tensorflow – LSTM Google Translate Example & GPUs

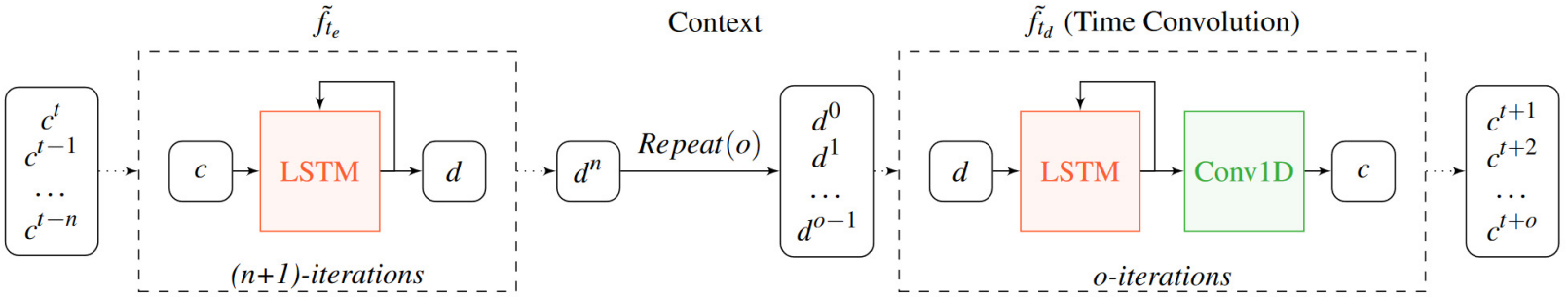
- Use of 2 LSTM networks in a stacked manner
  - Called ‘sequence-2-sequence‘ model
  - Encoder network
  - Decoder network
  - Needs context of sentence (memory) for translation

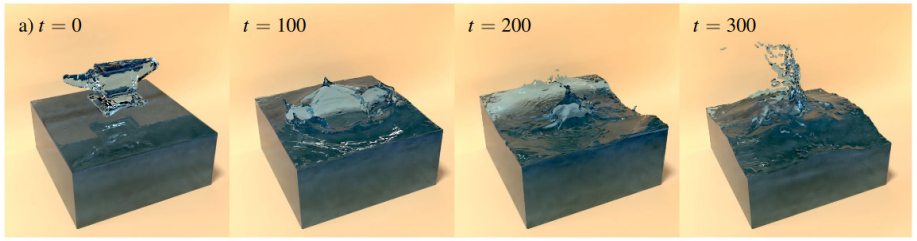


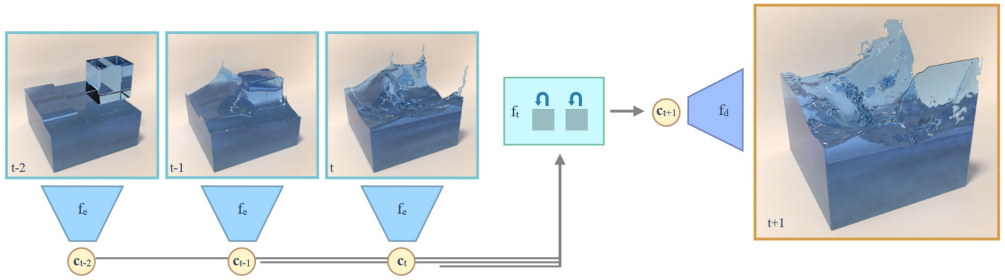


[18] Sequence Models

Combining Simulation Sciences & LSTM Model Approaches







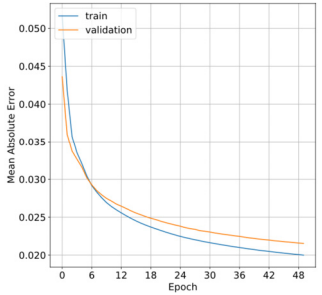
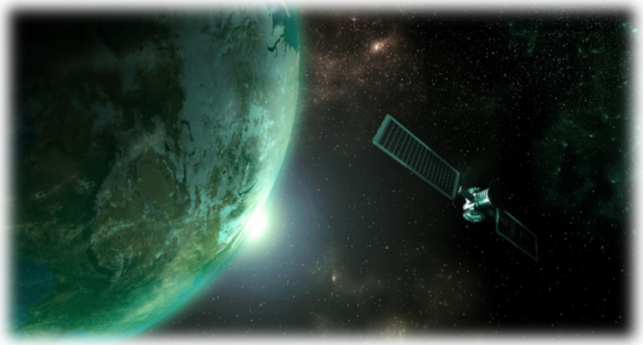
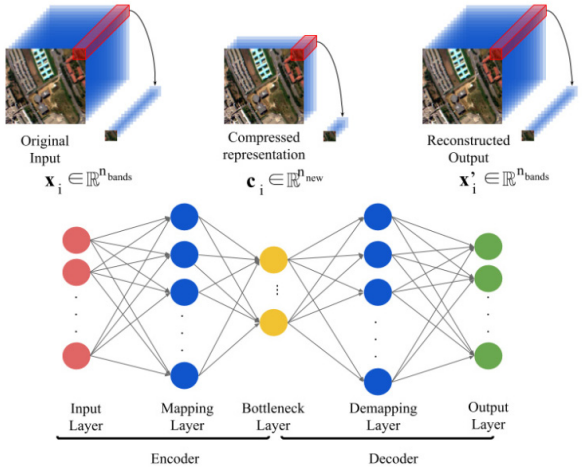


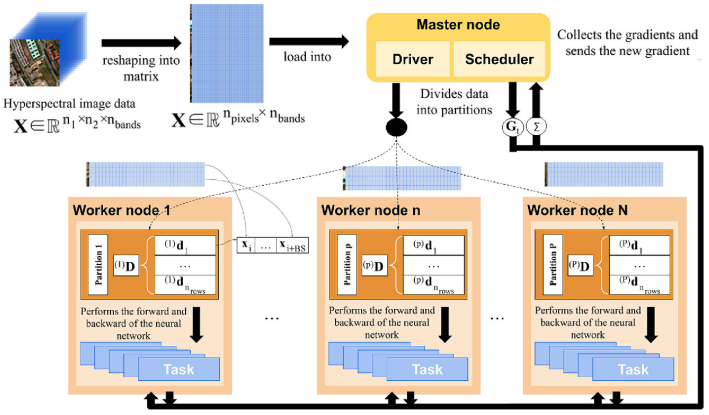
Figure 1: *Our method encodes multiple steps of a simulation field, typically pressure, into a reduced latent representation with a convolutional neural network. A second neural network with LSTM units then predicts the latent space code for one or more future time steps, yielding large reductions in runtime compared to regular solvers.*

[10] Wiewel, S. et al.

More Complex Deep Learning Model Example: Autoencoder Networks



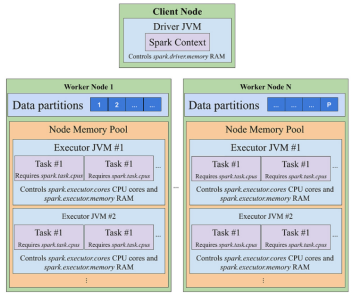




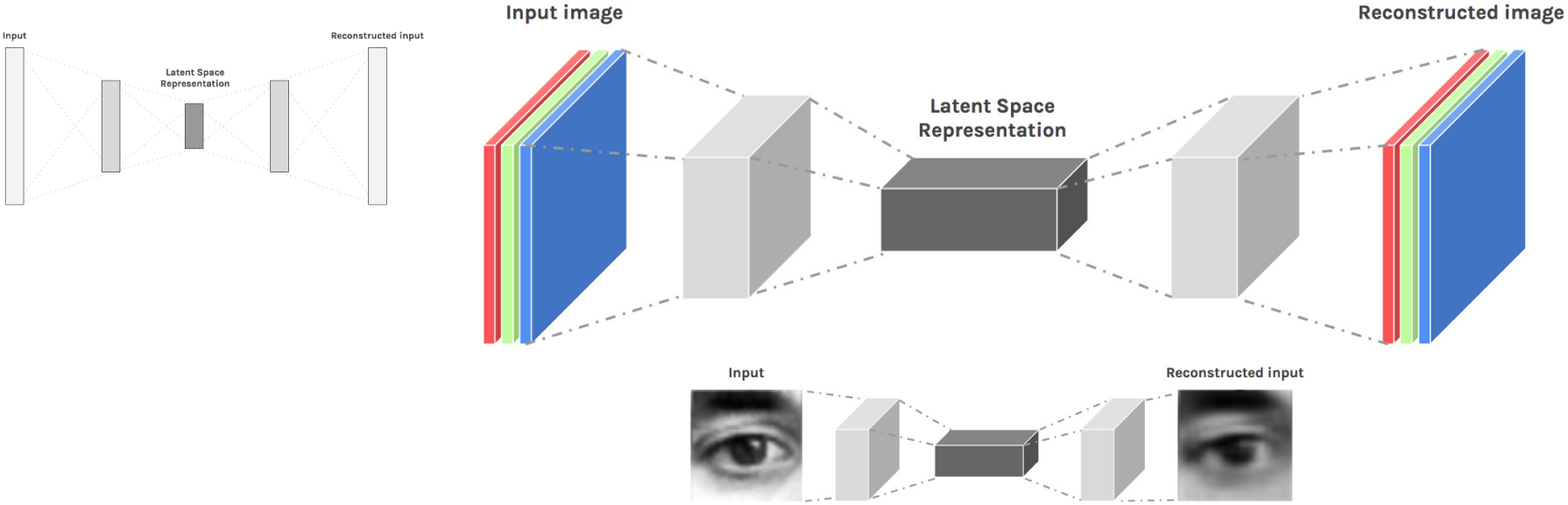
- **Find right set of hyper-parameters and the right neural network architecture for autoencoder is a manual time-consuming and error-prone process**
- **Needs urgently HPC, but a systematic and automated way is required as trying out all options of hyper-parameters and architectures is computationally infeasible**
- **As resolutions of sensors becomes better and more data is available it is likely that the learning model will be increasingly complex in the future that in turn raises demands for automated architecture search and meta-learning approaches**

[4] J. Haut, G. Cavallaro and M. Riedel et al.,
IEEE Transactions on Geoscience and Remote Sensing, 2019

[5] Apache Spark Web page



Autoencoders – Short Overview

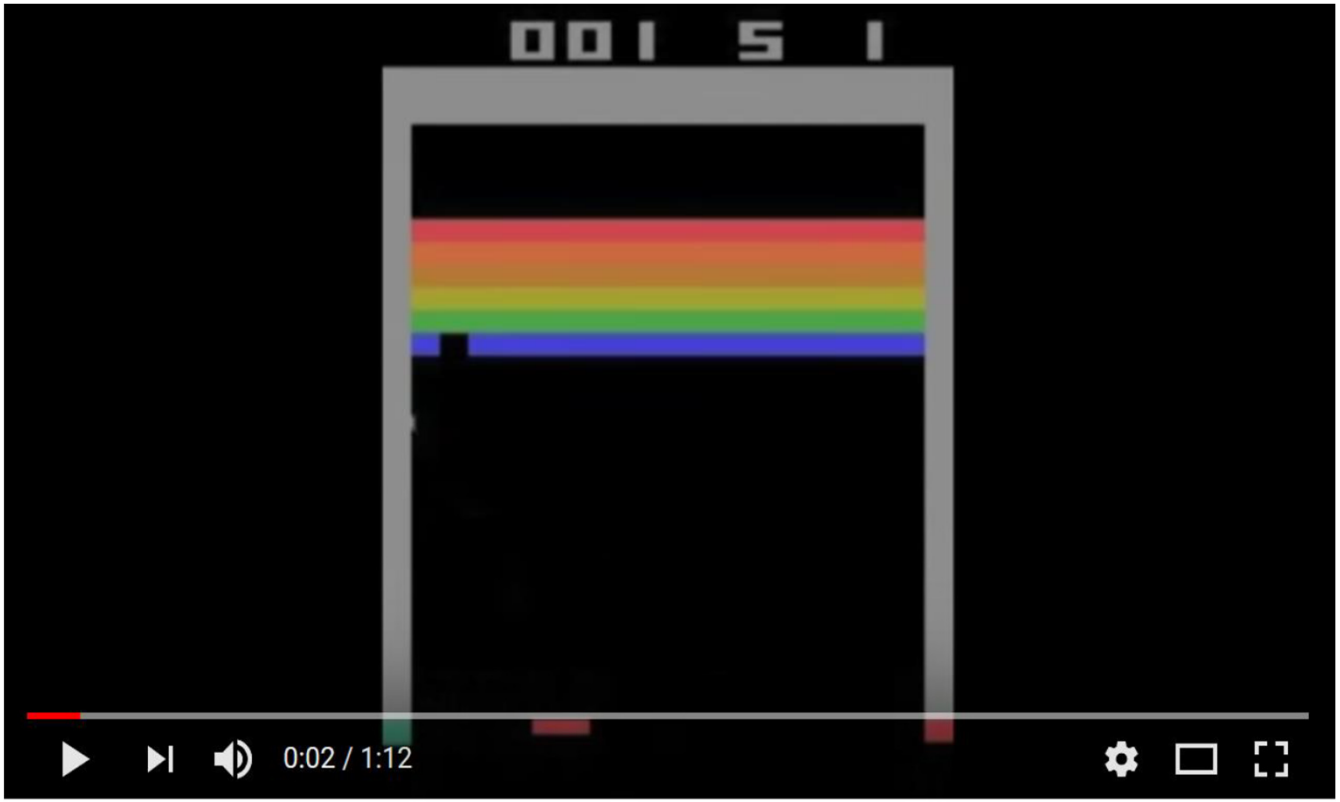


- **Input is Output, here Network is called a Convolutional Autoencoder (CAE)**
- **Apply AE to image for image compression**
- **Convert the input from wide and thin (e.g. 100 x 100 px with 3 channels—RGB) to narrow and thick**
- **helps the network extract visual features from the images, and therefore obtain a much more accurate latent space representation**
- **Reconstruction process uses *upsampling* and convolutions.**

[19] Autoencoders

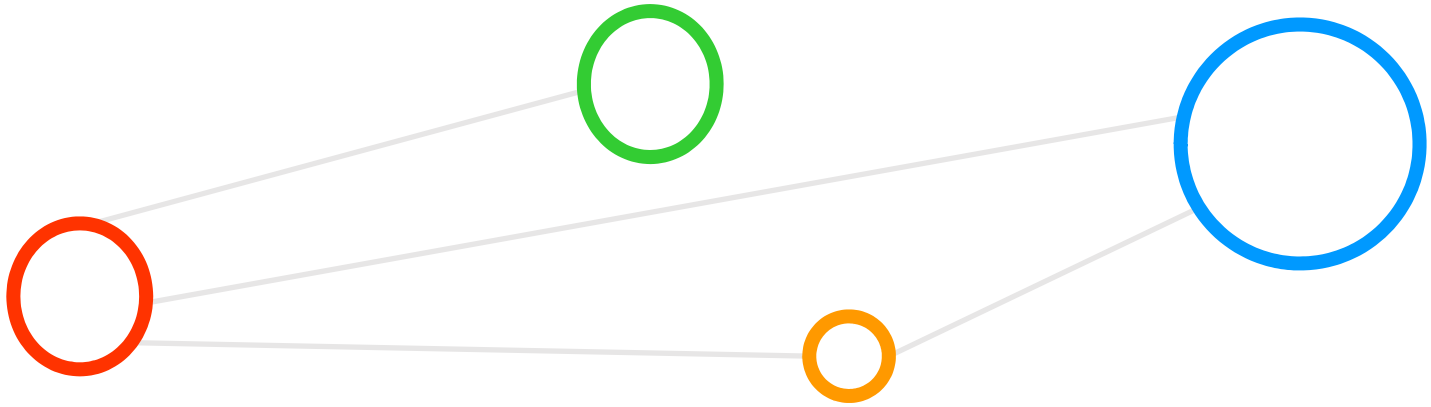
Generative Adversial Networks (GANs)

[Video] DEEP Q Learning



[9] YouTube Video, DQN

Lecture Bibliography



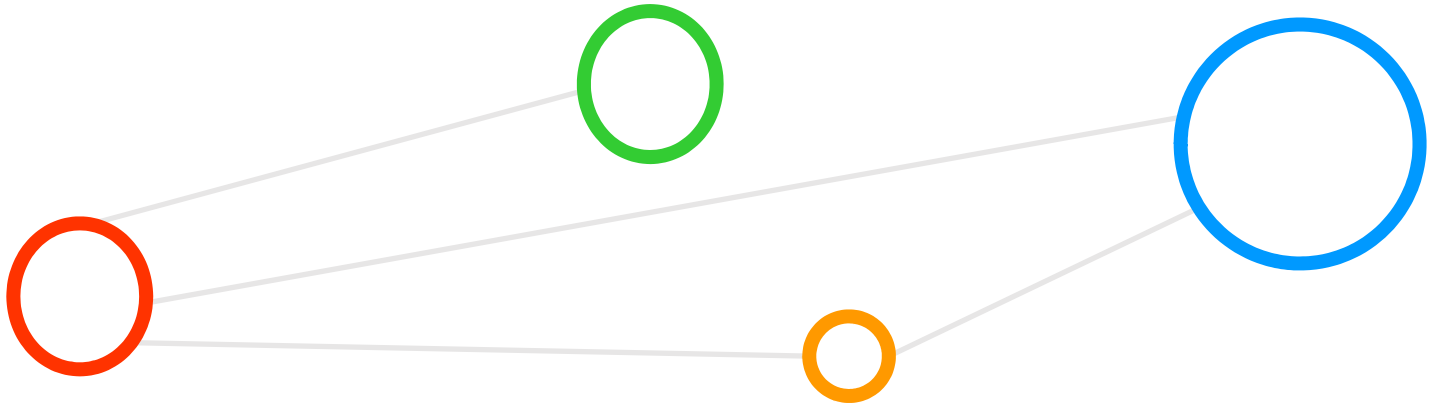
Lecture Bibliography (1)

- [1] A. Gulli and S. Pal, 'Deep Learning with Keras' Book, ISBN-13 9781787128422, 318 pages, Online: https://www.packtpub.com/big-data-and-business-intelligence/deep-learning-keras
- [2] D. Kingma and Jimmy Ba, 'Adam: A Method for Stochastic Optimization', Online: https://arxiv.org/abs/1412.6980
- [3] S.191 MIT Intro to Deep Learning, 'Sequence Modeling with Neural Networks' Online: https://www.youtube.com/watch?v=CznICCPa63Q&t=29s
- [4] Haut, J.M., Gallardo, J.A., Paoletti, M.E., Cavallaro, G., Plaza, J., Plaza, A., Riedel, M.: Cloud Deep Networks for Hyperspectral Image Analysis, IEEE Transactions on Geoscience and Remote Sensing, PP(99):1-17, 2019, Online: https://www.researchgate.net/publication/335181248_Cloud_Deep_Networks_for_Hyperspectral_Image_Analysis
- [5] Apache Spark, Online: https://spark.apache.org/
- [6] K. Hwang, G. C. Fox, J. J. Dongarra, 'Distributed and Cloud Computing', Book, Online: http://store.elsevier.com/product.jsp?locale=en_EU&isbn=9780128002049
- [7] Harley, A.W., An Interactive Node-Link Visualization of Convolutional Neural Networks, Online: http://scs.ryerson.ca/~aharley/vis/conv/flat.html
- [8] A. Rosebrock, 'Get off the deep learning bandwagon and get some perspective', Online: http://www.pyimagesearch.com/2014/06/09/get-deep-learning-bandwagon-get-perspective/
- [9] YouTube Video ,DEEP QN', Online: https://www.youtube.com/watch?v=TmPfTpjtdgg
- [10] YouTube Video, GANs, Online: https://www.youtube.com/watch?v=G06dEcZ-QTg

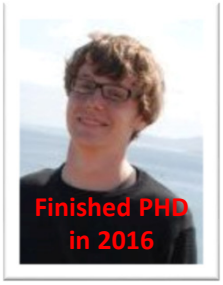
Lecture Bibliography (2)

- [11] YouTube Video, 'Deep Learning Revolution', Online: https://www.youtube.com/watch?v=Dy0hJWltsyE
- [12] Adventures in Machine Learning, Keras LSTM tutorial, Online: http://adventuresinmachinelearning.com/keras-lstm-tutorial/
- [13] Tensorflow Deep Learning Framework, Online: https://www.tensorflow.org/
- [14] Keras Python Deep Learning Library, Online: https://keras.io/
- [15] Kaggle, 'UMICH SI650 – Sentiment Classification', Online: https://www.kaggle.com/c/si650winter11
- [16] Book, Deep Learning with Keras
- [17] Stacked LSTM Architecture, Online: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/
- [18] YouTube Video,'Sequence Models and the RNN API (TensorFlow Dev Summit 2017)', Online: https://www.youtube.com/watch?v=RIR_-Xlbp7s
- [19] Autoencoders, Online: https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694

Acknowledgements



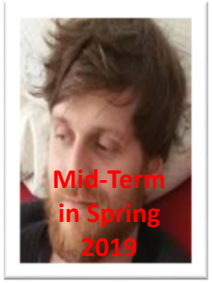
Acknowledgements – High Productivity Data Processing Research Group



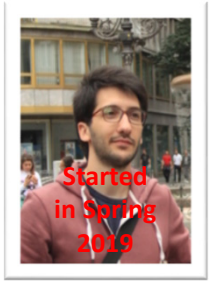
PD Dr.
G. Cavallaro

Senior PhD
Student A.S. Memon

Senior PhD
Student M.S. Memon



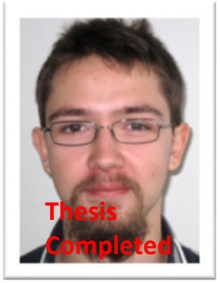
PhD Student
E. Erlingsson



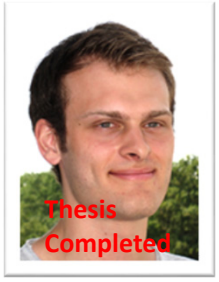
PhD Student
S. Bakarat

PhD Student
R. Sedona

Dr. M. Goetz
(now KIT)



MSc M.
Richerzhagen
(now other division)



MSc
P. Glock
(now INM-1)

MSc
C. Bodenstein
(now
Soccerwatch.tv)

MSc Student
G.S. Guðmundsson
(Landsverkjun)

This research group has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 763558 (DEEP-EST EU Project)


services
measurement
within
fusion
new
funding
Policy-based
concepts
forms
cross-disciplinary
resources
year
like
Computational
Test
E.g
Cross-Disciplinary
methods
EU
Science
High
big
Centre
device
Analysis
climate
Computer
expertise
Computing
nodes
PB
centers
community
disciplines
results
Approaches
construction
Fusion
Enable
Modelling
different
Environment
Key
scientific
important
storage
basis
Infrastructure
files
DKRZ
Use
etc
simulations
ENES
technologies
SMAQ
international
model
engineering
References
formats
brain
Big
Project
Data
MapReduce
create
increasing
via
EUDAT
processing
physical
IPCC
Common
TB
Methods
computing
Using
European
databases
codes
GB
Supercomputing
Work
images
analysis
Large
Summary
data-intensive
Service
Scientific
levels
often
DLCL
Level
Research
stored
Resources
analyze
external
performance
use
JSC
computational
Matters
Concrete
research
e.g
science
HPC
Juelich
system
datasets
acquisition
compute
Landscape
management
System
Climate
modelling
AR5
Hardware
distributed
large-scale
cores
access
hundreds
Nutshell
Services
network
DLCLs
Understanding
structures
Technologies
Earth
Structure
current
Germany
Simulation
directory
project
General
Provide
NASA
Energy
systems
day
Processing
Health