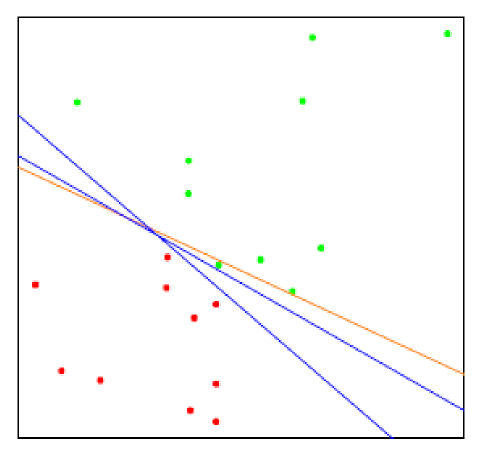


Deep Learning

Introduction to Deep Learning

Dr. – Ing. Morris Riedel

Adjunct Associated Professor

School of Engineering and Natural Sciences, University of Iceland

Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 5

Model Selection and Regularization

June 7th, 2018

Juelich Supercomputing Centre, Germany

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

JÜLICH Forschungszentrum

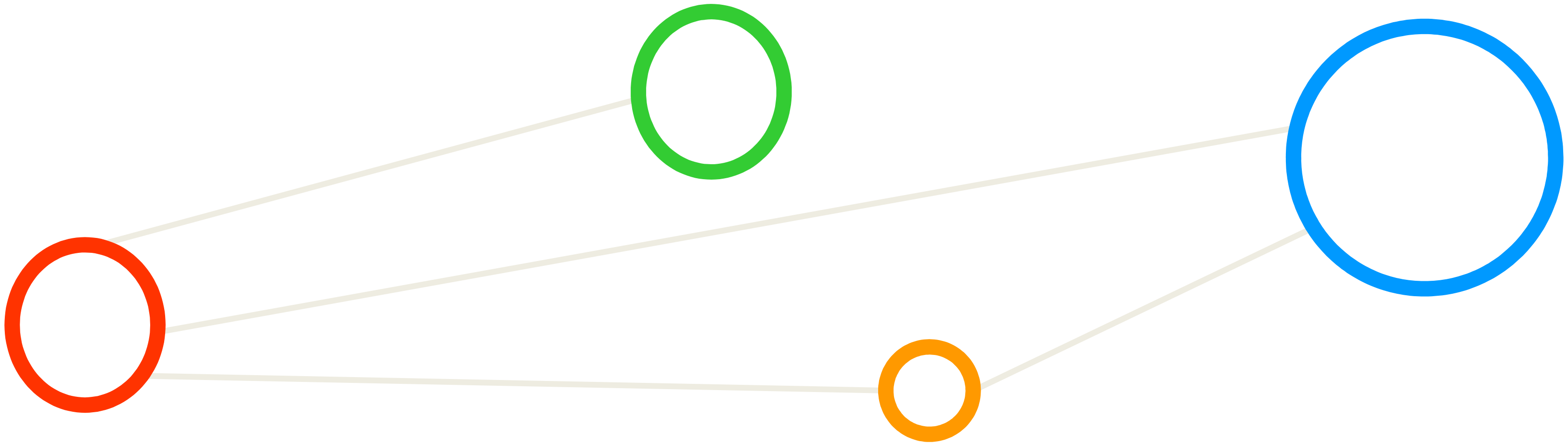
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

DEEP Projects

Outline of the Course

1. Introduction to Deep Learning
2. Fundamentals of Convolutional Neural Networks (CNNs)
3. Deep Learning in Remote Sensing: Challenges
4. Deep Learning in Remote Sensing: Applications
5. Model Selection and Regularization
6. Fundamentals of Long Short-Term Memory (LSTM)
7. LSTM Applications and Challenges
8. Deep Reinforcement Learning

Outline

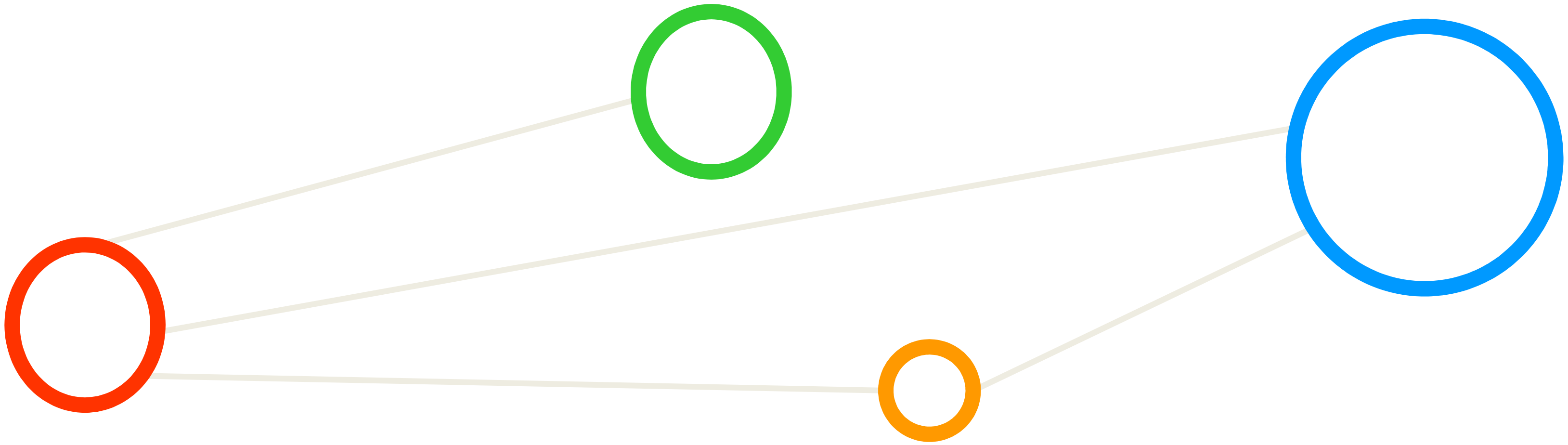


Outline

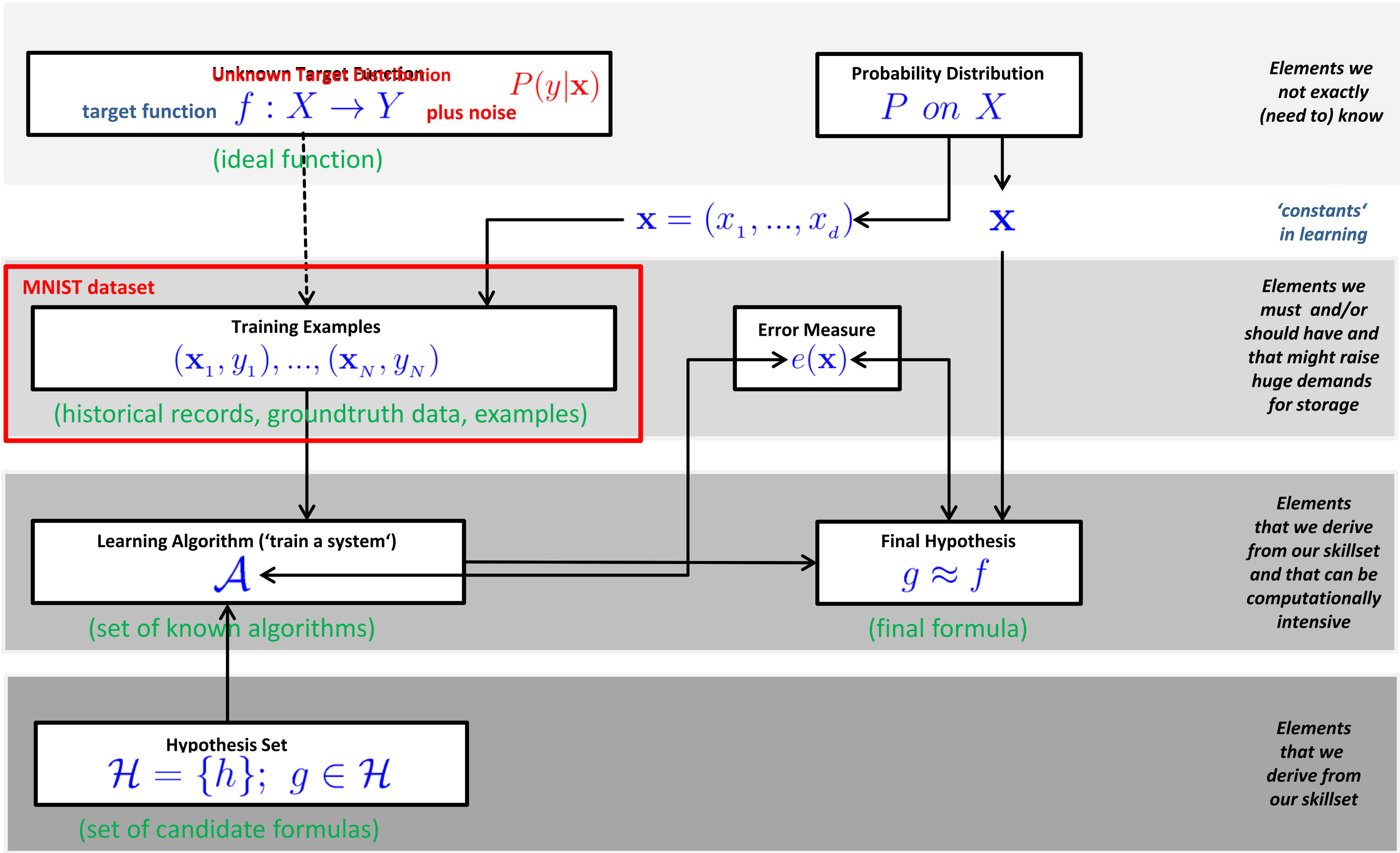
- Model Selection
  - MNIST Dataset Exploration & Normalization
  - Training and Testing Datasets
  - Creating ANN Network Topologies
  - Parameter Hidden Layers & Overfitting
  - Validation Datasets & Splits

- Regularization
  - Problem of Overfitting
  - Overfitting Reasoning
  - Regularization and Validation Counter Approach
  - Regularization Techniques
  - Dropout Regularizer

Model Selection



Supervised Learning – Training Examples

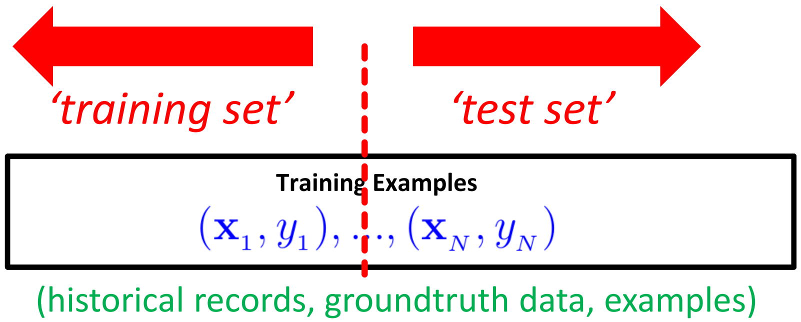


Terminologies & Different Dataset Elements

- Target Function $f : X \to Y$
  - Ideal function that 'explains' the data we want to learn
- Labelled Dataset (samples)
  - 'in-sample' data given to us: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
- Learning vs. Memorizing
  - The goal is to create a system that works well 'out of sample'
  - In other words we want to classify 'future data' (ouf of sample) correct
- Dataset Part One: Training set
  - Used for training a machine learning algorithms
  - Result after using a training set: a trained system
- Dataset Part Two: Test set
  - Used for testing whether the trained system might work well
  - Result after using a test set: accuracy of the trained model

Model Evaluation – Training and Testing Phases

- Different Phases in Learning (cf. day one remote sensing)
  - Training phase is a hypothesis search
  - Testing phase checks if we are on right track (once the hypothesis clear)
- Work on 'training examples'
  - Create two disjoint datasets
  - One used for training only (aka training set)
  - Another used for testing only (aka test set)



  - Exact seperation is rule of thumb per use case (e.g. 10 % training, 90% test)
  - Practice: If you get a dataset take immediately test data away ('throw it into the corner and forget about it during modelling')
  - Reasoning: Once we learned from training data it has an 'optimistic bias'

Learning Approaches – Supervised Learning – Formalization

- Each observation of the predictor measurement(s) has an associated response measurement:
  - Input $\mathbf{x} = x_1, \ldots, x_d$
  - Output $y_i, i = 1, .., n$
  - Data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

Training Examples

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(historical records, groundtruth data, examples)

- Goal: Fit a model that relates the response to the predictors
  - Prediction: Aims of accurately predicting the response for future observations
  - Inference: Aims to better understanding the relationship between the response and the predictors

- **Supervised learning approaches fits a model that related the response to the predictors**
- **Supervised learning approaches are used in classification algorithms such as SVMs**
- **Supervised learning works with data = [input, correct output]**

[1] An Introduction to Statistical Learning

Exercises – Explore MNIST Training & Testing Dataset

Handwritten Character Recognition MNIST Dataset

- Metadata
  - Subset of a larger dataset from US National Institute of Standards (NIST)
  - Handwritten digits including corresponding labels with values 0 to 9
  - All digits have been size-normalized to 28 * 28 pixels
    and are centered in a fixed-size image for direct processing
  - Not very challenging dataset, but good for experiments / tutorials

- Dataset Samples
  - Labelled data (10 classes)
  - Two separate files
    for training and test
  - 60000 training samples (~47 MB)
  - 10000 test samples (~7.8 MB)

MNIST Dataset for the Tutorial

- When working with the dataset
  - Dataset is not in any standard image format like jpg, bmp, or gif
  - One needs to write typically a small program to read and work for them
  - Data samples are stored in a simple file format that is designed for storing vectors and multidimensional matrices (here numpy binary files)
  - The pixels of the handwritten digit images are organized row-wise with pixel values ranging from 0 (white background) to 255 (black foreground)
  - Images contain grey levels as a result of an anti-aliasing technique used by the normalization algorithm that generated this dataset.

```
/homea/hpclab/train001/data/mnist
[train001@jrl09 mnist]$ pwd
/homea/hpclab/train001/data/mnist
[train001@jrl09 mnist]$ ls -al
total 53728
drwxr-xr-x  2 train001 hpclab      512 Jun  6 12:17 .
drwxr-xr-x 10 train001 hpclab      512 Jun  6 12:17 ..
-rw-r-----  1 train001 hpclab  7840080 Jun  6 12:17 x_test.npy
-rw-r-----  1 train001 hpclab 47040080 Jun  6 12:17 x_train.npy
-rw-r-----  1 train001 hpclab    10080 Jun  6 12:17 y_test.npy
-rw-r-----  1 train001 hpclab    60080 Jun  6 12:17 y_train.npy
```

MNIST Dataset – Exploration – One Character Encoding

```
[train001@jrl09 mnist]$ python explore-mnist-training.py
Samples of 28 x 28 pixel matrices reserved for training
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   3   18  18  18  126 136 175 26  166 255 247 127 0   0   0   0
0   0   0   0   0   0   0   0   30  36  94  154 170 253 253 253 253 253 225 172 253 242 195 64  0   0   0   0
0   0   0   0   0   0   0   49  238 253 253 253 253 253 253 253 253 251 93  82  82  56  39  0   0   0   0   0
0   0   0   0   0   0   0   18  219 253 253 253 253 253 198 182 247 241 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   80  156 107 253 253 205 11  0   43  154 0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   14  1   154 253 90  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   139 253 190 2   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   11  190 253 70  0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   35  241 225 160 108 1   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   81  240 253 253 119 25  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   45  186 253 253 150 27  0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   16  93  252 253 187 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   249 253 249 64  0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   46  130 183 253 253 207 2   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   39  148 229 253 253 253 250 182 0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   24  114 221 253 253 253 253 201 78  0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   23  66  213 253 253 253 253 198 81  2   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   18  171 219 253 253 253 253 195 80  9   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   55  172 226 253 253 253 253 244 133 11  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   136 253 253 253 212 135 132 16  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0


Label:
5
```

MNIST Dataset – Exploration Script Training

```
import numpy as np

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")
```

```
print("Samples of 28 x 28 pixel matrices reserved for training")

# function for showing a character
def character_show(character):
    for y in character:
        row = ""
        for x in y:
            row += '{0: <4}'.format(x)
        print row
```

```
# view first 10 characters
for i in range (0,9):
  character_show(X_train[i])
  print("\n")
  print("Label:")
  print(Y_train[i])
  print("\n")
```

- Loading MNIST training datasets (X) with labels (Y) stored in a binary numpy format
- Format is 28 x 28 pixel values with grey level from 0 (white background) to 255 (black foreground)

- Small helper function that prints row-wise one ‘hand-written‘ character with the grey levels stored in training dataset
- Should reveal the nature of the number (aka label)

- Loop of the training dataset and the testing dataset (e.g. first 10 characters as shown here)
- At each loop interval the ‘hand-written‘ character (X) is printed in ‘matrix notation‘ & label (Y)

Exercises – Execute Script to Explore MNIST Training Dataset

```
[train001@jrl09 mnist]$ python /homea/hpclab/train001/tools/mnist/explore-mnist-training.py
```

MNIST Dataset – Exploration – Selected Training Samples

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 3 18 18 18 126 136 175 26 166 255 247 127 0 0 0 0
0 0 0 0 0 0 0 0 30 36 94 154 170 253 253 253 253 253 225 172 253 242 195 64 0 0 0 0
0 0 0 0 0 0 0 49 238 253 253 253 253 253 253 253 253 251 93 82 82 56 39 0 0 0 0 0
0 0 0 0 0 0 0 18 219 253 253 253 253 253 198 182 247 241 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 80 156 107 253 253 205 11 0 43 154 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 14 1 154 253 90 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 139 253 190 2 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 11 190 253 70 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 35 241 225 160 108 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 81 240 253 253 119 25 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 45 186 253 253 150 27 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 93 252 253 187 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 249 253 249 64 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 46 130 183 253 253 207 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 39 148 229 253 253 253 250 182 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 24 114 221 253 253 253 253 201 78 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 23 66 213 253 253 253 253 198 81 2 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 18 171 219 253 253 253 253 195 80 9 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 55 172 226 253 253 253 253 244 133 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 136 253 253 253 212 135 132 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Label:
5
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 51 159 253 159 50 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 48 238 252 252 252 237 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 54 227 253 252 239 233 252 57 6 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 10 60 224 252 253 252 202 84 252 253 122 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 163 252 252 252 253 252 252 96 189 253 167 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 51 238 253 253 190 114 253 228 47 79 255 168 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 48 238 252 252 179 12 75 121 21 0 0 253 243 50 0 0 0 0 0
0 0 0 0 0 0 0 0 38 165 253 233 208 84 0 0 0 0 0 0 253 252 165 0 0 0 0 0
0 0 0 0 0 0 0 7 178 252 240 71 19 28 0 0 0 0 0 0 253 252 195 0 0 0 0 0
0 0 0 0 0 0 0 57 252 252 63 0 0 0 0 0 0 0 0 0 253 252 195 0 0 0 0 0
0 0 0 0 0 0 0 198 253 190 0 0 0 0 0 0 0 0 0 0 255 253 196 0 0 0 0 0
0 0 0 0 0 0 76 246 252 112 0 0 0 0 0 0 0 0 0 0 253 252 148 0 0 0 0 0
0 0 0 0 0 0 85 252 230 25 0 0 0 0 0 0 0 0 7 135 253 186 12 0 0 0 0 0
0 0 0 0 0 0 85 252 223 0 0 0 0 0 0 0 0 7 131 252 225 71 0 0 0 0 0 0
0 0 0 0 0 0 85 252 145 0 0 0 0 0 0 0 48 165 252 173 0 0 0 0 0 0 0 0
0 0 0 0 0 0 86 253 225 0 0 0 0 0 0 114 238 253 162 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 85 252 249 146 48 29 85 178 225 253 223 167 56 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 85 252 252 252 229 215 252 252 252 196 130 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 28 199 252 252 253 252 252 233 145 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 25 128 252 253 252 141 37 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Label:
0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 67 232 39 0 0 0 0 0
0 0 0 0 62 81 0 0 0 0 0 0 0 0 0 0 0 0 0 0 120 180 39 0 0 0 0 0
0 0 0 0 126 163 0 0 0 0 0 0 0 0 0 0 0 0 0 2 153 210 40 0 0 0 0 0
0 0 0 0 220 163 0 0 0 0 0 0 0 0 0 0 0 0 0 27 254 162 0 0 0 0 0 0
0 0 0 0 222 163 0 0 0 0 0 0 0 0 0 0 0 0 0 183 254 125 0 0 0 0 0 0
0 0 0 46 245 163 0 0 0 0 0 0 0 0 0 0 0 0 0 198 254 56 0 0 0 0 0 0
0 0 0 120 254 163 0 0 0 0 0 0 0 0 0 0 0 0 23 231 254 29 0 0 0 0 0 0
0 0 0 159 254 120 0 0 0 0 0 0 0 0 0 0 0 0 163 254 216 16 0 0 0 0 0 0
0 0 0 159 254 67 0 0 0 0 0 0 0 0 0 14 86 178 248 254 91 0 0 0 0 0 0 0
0 0 0 159 254 85 0 0 0 47 49 116 144 150 241 243 234 179 241 252 40 0 0 0 0 0 0 0
0 0 0 150 253 237 207 207 207 253 254 250 240 198 143 91 28 5 233 250 0 0 0 0 0 0 0 0
0 0 0 0 119 177 177 177 177 177 98 56 0 0 0 0 0 102 254 220 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 254 137 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 254 57 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 254 57 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 255 94 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 254 96 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 254 153 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 169 255 153 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 96 254 153 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Label:
4
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 13 25 100 122 7 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 33 151 208 252 252 252 146 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 40 152 244 252 253 224 211 252 232 40 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 15 152 239 252 252 252 216 31 37 252 252 60 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 96 252 252 252 252 217 29 0 37 252 252 60 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 181 252 252 220 167 30 0 0 77 252 252 60 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 26 128 58 22 0 0 0 0 100 252 252 60 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 157 252 252 60 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 110 121 122 121 202 252 194 3 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 10 53 179 253 253 255 253 253 228 35 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 54 227 252 243 228 170 242 252 252 231 117 6 0 0 0 0 0 0 0
0 0 0 0 0 0 0 6 78 252 252 125 59 0 18 208 252 252 252 252 87 7 0 0 0 0 0 0
0 0 0 0 0 0 5 135 252 252 180 16 0 21 203 253 247 129 173 252 252 184 66 49 49 0 0 0
0 0 0 0 0 3 136 252 241 106 17 0 53 200 252 216 65 0 14 72 163 241 252 252 223 0 0 0
0 0 0 0 0 105 252 242 88 18 73 170 244 252 126 29 0 0 0 0 0 89 180 180 37 0 0 0
0 0 0 0 0 231 252 245 205 216 252 252 252 124 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 207 252 252 252 252 178 116 36 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 13 93 143 121 23 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Label:
2
```

Exercises – Modify Script to Explore MNIST Testing Dataset

```
[train001@jrl09 mnist]$ cp /homea/hpclab/train001/tools/mnist/explore-mnist-training.py .
```

MNIST Dataset – Exploration Script Testing (one solution)

```
import numpy as np

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

print("Samples of 28 x 28 pixel matrices reserved for testing")

# function for showing a character
def character_show(character):
    for y in character:
        row = ""
        for x in y:
            row += '{0: <4}'.format(x)
        print row

# view first 10 characters
for i in range (0,9):
  character_show(X_test[i])
  print("\n")
  print("Label:")
  print(Y_test[i])
  print("\n")
```

MNIST Dataset – Reshape & Normalization

```
import numpy as np

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

# reshape for the neural network 28 x 28 = 784
RESHAPED= 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
# specify type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# data output: number of samples
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# data output: number of values / sample
print(X_train.shape[1], 'input pixel values per train samples')
print(X_test.shape[1], 'input pixel values per test samples')

# data output: character
print(X_train[0])
```

- **Loading MNIST training datasets (X) and testing datasets (Y) stored in a binary numpy format with labels for X and Y**
- **Format is 28 x 28 pixel values with grey level from 0 (white background) to 255 (black foreground)**

- **Reshape from 28 x 28 matrix of pixels to 784 pixel values considered to be the input for the neural networks later**
- **Normalization is added for mathematical convenience since the computing with numbers get easier (not too large)**

Exercises – Execute Script to Reshape MNIST Datasets

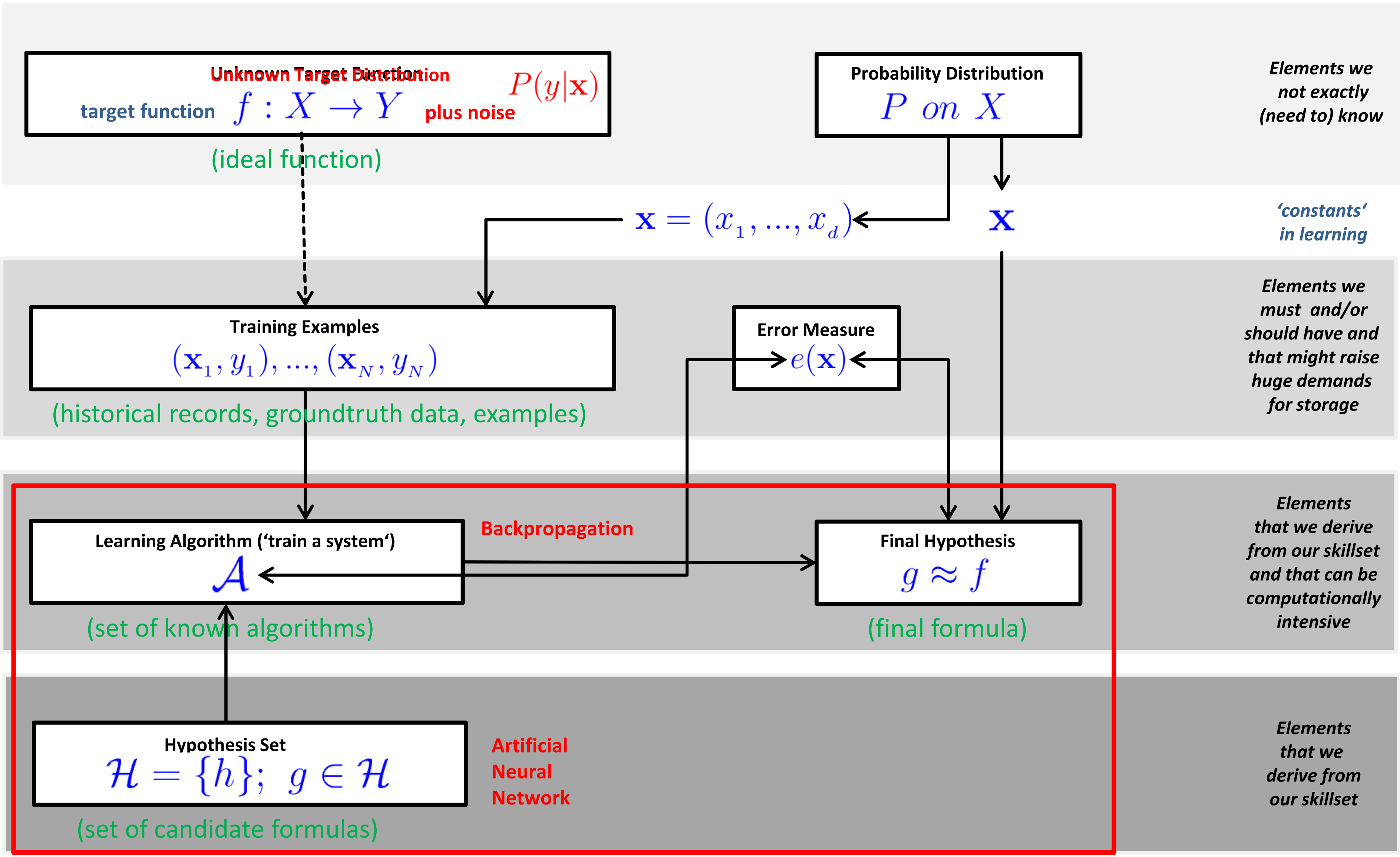
```
[train001@jrl09 mnist]$ cp mnist-reshape.py ~
```

MNIST Dataset – Reshape & Normalization – Example

```
[train001@jrl09 mnist]$ python mnist-reshape.py
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333336 0.6862745  0.10196079 0.6509804  1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117648 0.36862746 0.6039216
 0.6666667  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235295 0.6745098  0.99215686 0.9490196  0.7647059  0.2509804
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215687
 0.93333334 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.9843137  0.3647059  0.32156864
 0.32156864 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.07058824 0.85882354 0.99215686
 0.99215686 0.99215686 0.99215686 0.99215686 0.7764706  0.7137255
 0.96862745 0.94509804 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.3137255  0.6117647  0.41960785 0.99215686
 0.99215686 0.8039216  0.04313726 0.         0.16862746 0.6039216
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

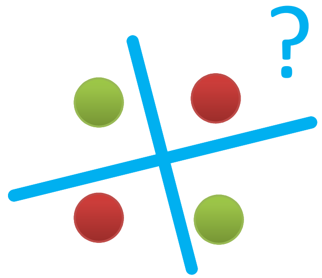
(numbers are between 0 and 1)

Supervised Learning – Training Examples



Artificial Neural Network (ANN) – cf. Day One

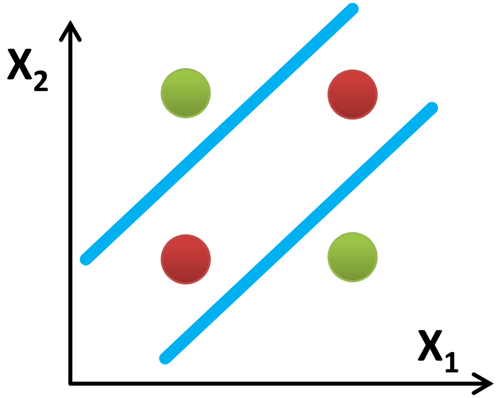
- Simple perceptrons fail: 'not linearly seperable'



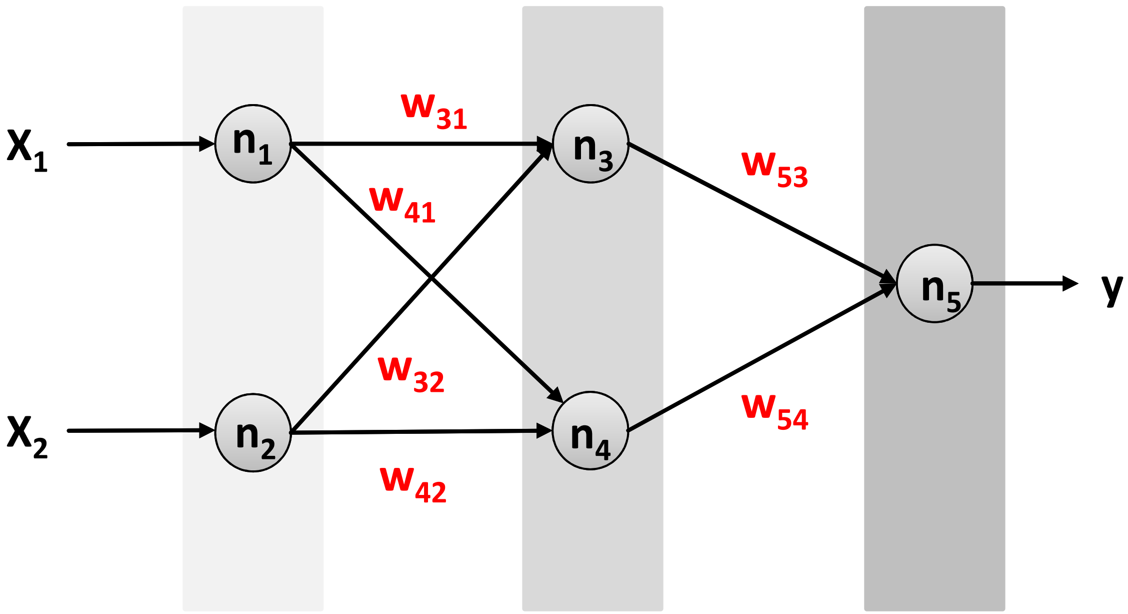
X_1	X_2	Y
0	0	-1
1	0	1
0	1	1
1	1	-1

Labelled Data Table

(Idea: instances can be classified using two lines at once to model XOR)



Decision Boundary



Two-Layer, feed-forward Artificial Neural Network topology

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.Dense(units,
                   activation=None,
                   use_bias=True,
                   kernel_initializer='glorot_uniform',
                   bias_initializer='zeros',
                   kernel_regularizer=None,
                   bias_regularizer=None,
                   activity_regularizer=None,
                   kernel_constraint=None,
                   bias_constraint=None)

keras.optimizers.SGD(lr=0.01,
                     momentum=0.0,
                     decay=0.0,
                     nesterov=False)
```

- **Tool Keras supports inherently the creation of artificial neural networks using Dense layers and optimizers (e.g. SGD)**
- **Includes regularization (e.g. weight decay) or momentum**

Keras

[2] Keras Python Deep Learning Library

ANN – MNIST Dataset – Create ANN Blueprint

✓ Data Preprocessing done (i.e. data normalization, reshape, etc.)

1. Define a neural network topology
   - Which layers are required?
   - Think about input layer need to match the data – what data we had?
   - Maybe hidden layers?
   - Think Dense layer – Keras?
   - Think about final Activation as Softmay (cf. Day One) → output probability
2. Compile the model → model representation for Tensorflow et al.
   - Think about what loss function you want to use in your problem?
   - What is your optimizer strategy, e.g. SGD (cf. Day One)
3. Fit the model → the model learning takes place
   - How long you want to train (e.g. NB_EPOCHS)
   - How much samples are involved (e.g. BATCH_SIZE)

Exercises – Create a Simple ANN Model – One Dense

ANN – MNIST Dataset – Parameters & Data Normalization

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

# ANN parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'

# n x 28 x 28 pixel training data
X_train = np.load("/homea/hpclab/train001/data/mnist/x_train.npy")

# n x 1 training labels
Y_train = np.load("/homea/hpclab/train001/data/mnist/y_train.npy")

# n x 28 x 28 pixel testing data
X_test = np.load("/homea/hpclab/train001/data/mnist/x_test.npy")

# n x 1 testing labels
Y_test = np.load("/homea/hpclab/train001/data/mnist/y_test.npy")

# reshape for the neural network 28 x 28 = 784
RESHAPED= 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
# specify type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# data output: number of samples
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# data output: number of values / sample
print(X_train.shape[1], 'input pixel values per train samples')
print(X_test.shape[1], 'input pixel values per test samples')
```

- **NB_CLASSES: 10 Class Problem**
- **NB_EPOCH: number of times the model is exposed to the training set – at each iteration the optimizer adjusts the weights so that the objective function is minimized**
- **BATCH_SIZE: number of training instances taken into account before the optimizer performs a weight update**
- **OPTIMIZER: Stochastic Gradient Descent ('SGD') – only one training sample/iteration**

- **Data load shuffled between training and testing set in files**
- **Data preparation, e.g. X_train is 60000 samples / rows of 28 x 28 pixel values that are reshaped in 60000 x 784 including type specification (i.e. float32)**
- **Data normalization: divide by 255 – the max intensity value to obtain values in range [0,1]**

ANN – MNIST Dataset – A Simple Model

- The Sequential() Keras model is a linear pipeline (aka 'a stack') of various neural network layers including Activation functions of different types (e.g. softmax)

- Dense() represents a fully connected layer used in ANNs that means that each neuron in a layer is connected to all neurons located in the previous layer

- The non-linear Activation function 'softmax' represents a generalization of the sigmoid function – it squashes an n-dimensional vector of arbitrary real values into a n-dimenensional vector of real values in the range of 0 and 1 – here it aggregates 10 answers provided by the Dense layer with 10 neurons

```
# convert label vectors to binary matrices of classes
Y_train = np_utils.to_categorical(Y_train, NB_CLASSES)
Y_test = np_utils.to_categorical(Y_test, NB_CLASSES)

# simple ANN model
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
model.summary()

# compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE)

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

$$L_i = -\Sigma_j t_{i,j} \log(p_{i,j})$$

- Loss function is a multi-class logarithmic loss: target is *ti,j* and prediction is *pi,j*

- Train the model ('fit')

ANN – MNIST Dataset – Job Script

```
[train001@jrl09 scripts]$ cp submit_train_ann_mnist.sh ~
```

```
#!/bin/bash -x
#SBATCH--nodes=1
#SBATCH--ntasks=1
#SBATCH--output=mnist_out.%j
#SBATCH--error=mnist_err.%j
#SBATCH--time=01:00:00
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=ANN_mnist

#SBATCH--partition=gpus
#SBATCH--gres=gpu:1

#SBATCH--reservation=deep_learning

### location executable
MNIST=/homea/hpclab/train001/tools/mnist/mnist-simple-ann.py

module restore dl_tutorial_2

### submit
python $MNIST
```

ANN – MNIST Dataset – Job Submit & Check Output

```
[train001@jrl09 scripts]$ sbatch submit_train_ann_mnist.sh
Submitted batch job 5522445
[train001@jrl09 scripts]$ pwd
/homea/hpclab/train001/scripts
```

```
[train001@jrl09 scripts]$ more mnist_out.5522445
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200

  128/60000 [..............................] - ETA: 5:56 - loss: 2.5313 - acc: 0.0625
 2816/60000 [>.............................] - ETA: 16s - loss: 2.3530 - acc: 0.1225
 5888/60000 [=>............................] - ETA: 7s - loss: 2.2351 - acc: 0.2040
 8960/60000 [===>..........................] - ETA: 5s - loss: 2.1290 - acc: 0.2907
```

Model Evaluation – Testing Phase & Confusion Matrix

- Model is fixed
  - Model is just used with the testset
  - Parameters are set
- Evaluation of model performance
  - Counts of test records that are incorrectly predicted
  - Counts of test records that are correctly predicted
  - E.g. create confusion matrix for a two class problem

Counting per sample		**Predicted Class**	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(serves as a basis for further performance metrics usually used)

Model Evaluation – Testing Phase & Performance Metrics

Counting per sample		Predicted Class	
		Class = 1	Class = 0
Actual Class	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

(100% accuracy in learning often points to problems using machine learning methos in practice)

- Accuracy (usually in %)

$$Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$$

- Error rate

$$Error\ rate = \frac{number\ of\ wrong\ predictions}{total\ number\ of\ predictions}$$

ANN – MNIST Dataset – A Simple Model – Output

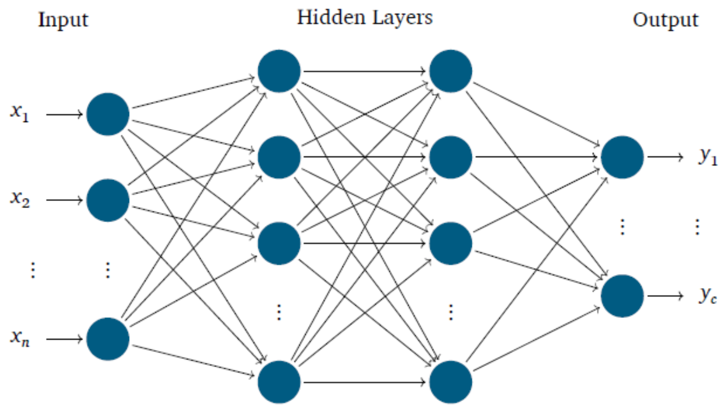
```
[train001@jrl09 scripts]$ tail mnist_out.5522445
   32/10000 [..............................] - ETA: 3s
 1504/10000 [===>..........................] - ETA: 0s
 3008/10000 [========>.....................] - ETA: 0s
 4544/10000 [============>.................] - ETA: 0s
 5952/10000 [================>.............] - ETA: 0s
 7392/10000 [====================>.........] - ETA: 0s
 8768/10000 [=========================>....] - ETA: 0s
10000/10000 [==============================] - 0s 36us/step
('Test score: ', 0.27273561475276695)
('Test accuracy: ', 0.9228)
```

ANN – MNIST Dataset – Extend ANN Blueprint

- ✓ Data Preprocessing done (i.e. data normalization, reshape, etc.)
- ✓ Initial ANN topology existing
- ✓ Initial setup of model works (create, compile, fit)

- Extend the neural network topology
  - Which layers are required?
  - Think about input layer need to match the data – what data we had?
  - Maybe hidden layers?
  - How many hidden layers?
  - What activation function for which layer?
  - Think Dense layer – Keras?
  - Think about final Activation as Softmay (cf. Day One) → output probability

Exercises – Add Two Hidden Layers



ANN – MNIST Dataset – Add Two Hidden Layers

- A hidden layer in an ANN can be represented by a fully connected Dense layer in Keras by just specifying the number of hidden neurons in the hidden layer

- The non-linear Activation function 'relu' represents a so-called Rectified Linear Unit (ReLU) that only recently became very popular because it generates good experimental results in ANNs and more recent deep learning models – it just returns 0 for negative values and grows linearly for only positive values

```
# simple ANN model
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()

# compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE)

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

ANN 2 Hidden – MNIST Dataset – Job Script

```
[train001@jrl09 scripts]$ cp submit_train_ann_mnist.sh ~
```

```
#!/bin/bash -x
#SBATCH--nodes=1
#SBATCH--ntasks=1
#SBATCH--output=mnist_out.%j
#SBATCH--error=mnist_err.%j
#SBATCH--time=01:00:00
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=ANN_mnist_2hidden

#SBATCH--partition=gpus
#SBATCH--gres=gpu:1

#SBATCH--reservation=deep_learning

### location executable
MNIST=/homea/hpclab/train001/tools/mnist/mnist-ann-2hidden.py

module restore dl_tutorial_2

### submit
python $MNIST
```

ANN – MNIST Dataset – Job Submit & Check Output

```
[train001@jrl06 scripts]$ sbatch submit_train_ann_2hidden_mnist.sh
Submitted batch job 5522545
```

```
[train001@jrl06 scripts]$ more ann-2hidden-mnist_out.5522545
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
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
_________________________________________________________________
Epoch 1/200

  128/60000 [..............................] - ETA: 6:02 - loss: 2.3471 - acc: 0.0781
 2560/60000 [>.............................] - ETA: 18s - loss: 2.3044 - acc: 0.0902
 4992/60000 [=>............................] - ETA: 9s - loss: 2.2580 - acc: 0.1332
 7552/60000 [==>...........................] - ETA: 6s - loss: 2.2119 - acc: 0.2080
```

ANN 2 Hidden – MNIST Dataset – Output

```
 5632/10000 [===============>..............] - ETA: 0s
 7040/10000 [===================>.........] - ETA: 0s
 8448/10000 [========================>.....] - ETA: 0s
 9824/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 0s 37us/step
('Test score: ', 0.0748017713768l167)
('Test accuracy: ', 0.9777)
```

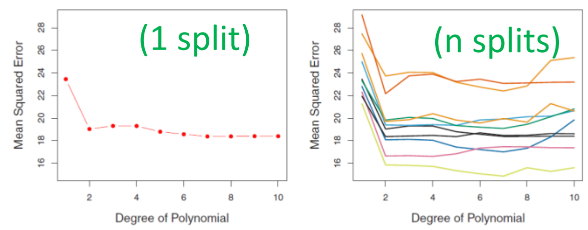
Validation & Model Selection – Terminology

- **The ‘Validation technique‘ should be used in all machine learning or data mining approaches**
- **Model assessment is the process of evaluating a models performance**
- **Model selection is the process of selecting the proper level of flexibility for a model**

modified from [4] ‘An Introduction to Statistical Learning’

- ‘Training error‘
  - Calculated when learning from data (i.e. dedicated training set)
- ‘Test error’
  - Average error resulting from using the model with ‘new/unseen data‘
  - ‘new/unseen data‘ was not used in training (i.e. dedicated test set)
  - In many practical situations, a dedicated test set is not really available
- ‘Validation Set‘
  - Split data into training & validation set
- ‘Variance‘ & ‘Variability‘
  - Result in different random splits (right)

(split creates a two subsets of comparable size)



Validation Technique – Formalization & Goal

- **Validation is a very important technique to estimate the out-of-sample performance of a model**
- **Main utility of regularization & validation is to control or avoid overfitting via model selection**

- Regularization & Validation
  - Approach: introduce a 'overfit penalty' that relates to model complexity
  - Problem: Not accurate values: 'better smooth functions'

(regularization uses a term that captures the overfit penalty)

$$E_{out}(h) = E_{in}(h) + \textbf{overfit penalty}$$

(minimize both to be better proxy for E_{out})

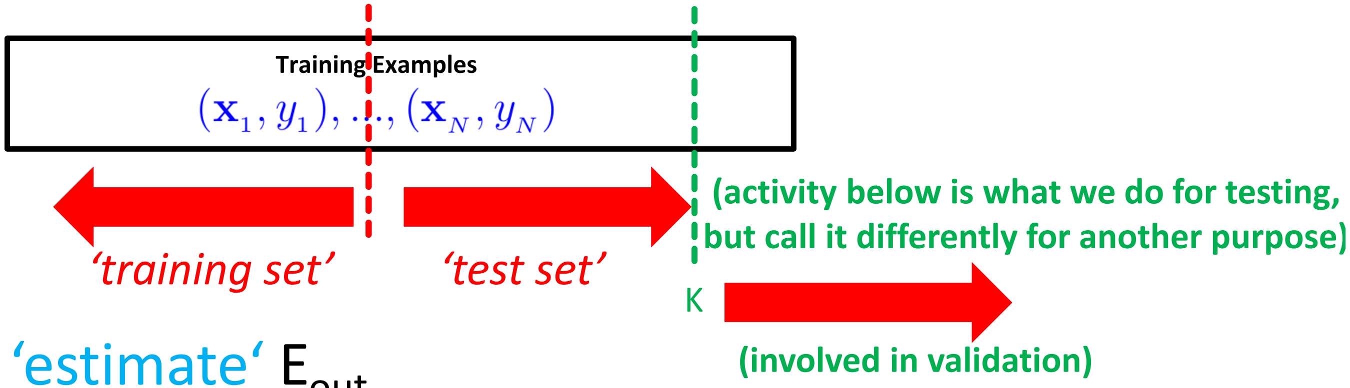
(validation estimates this quantity)

(regularization estimates this quantity)

- Validation

  (measuring E_{out} is not possible as this is an unknown quantity, another quantity is needed that is measurable that at least estimates it)
  - Goal 'estimate the out-of-sample error' **(establish a quantity known as validation error)**
  - Distinct activity from training and testing **(testing also tries to estimate the E_{out})**

Validation Technique – Pick one point & Estimate E_{out}



- Understanding 'estimate' E_{out}
  - On one out-of-sample point $(\mathbf{x}, y)$ the error is $e(h(\mathbf{x}), y)$
  - E.g. use squared error: $e(h(\mathbf{x}), f(\mathbf{x})) = (h(\mathbf{x}) - f(\mathbf{x}))^2$
    $e(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2$
  - Use this quantity as estimate for E_{out} **(poor estimate)**
  - Term 'expected value' to formalize (probability theory)

(Taking into account the theory of Lecture 1 with probability distribution on X etc.)

Probability Distribution
P on X

$\mathbf{x} = (x_1, \ldots, x_d)$

(aka 'random variable')

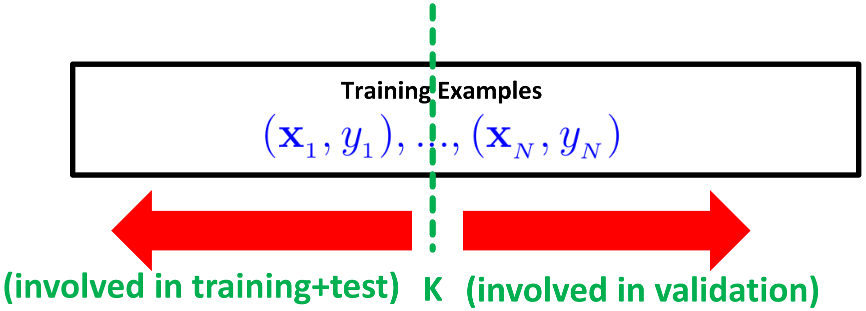
$$\mathbb{E}[e(h(\mathbf{x}), y)] = E_{out}(h)$$ **(aka the long-run average value of repetitions of the experiment)**

(one point as unbiased estimate of E_{out} that can have a high variance leads to bad generalization)

Validation Technique – Validation Set

- **Validation set consists of data that has been not used in training to estimate true out-of-sample**
- **Rule of thumb from practice is to take 20% (1/5) for validation of the learning model**

- Solution for high variance in expected values $\mathbb{E}[e(h(\mathbf{x}), y)] = E_{out}(h)$
  - Take a 'whole set' instead of just one point $(\mathbf{x}, y)$ for validation



(we need points not used in training to estimate the out-of-sample performance)

(we do the same approach with the testing set, but here different purpose)

  - Idea: K data points for validation

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_K, y_K)$ **(validation set)**

$$E_{val}(h) = \frac{1}{K} \sum_{k=1}^{K} e(h(\mathbf{x})_k, y_k)$$ **(validation error)**

  - Expected value to 'measure' the out-of-sample error

(expected values averaged over set)

$$\mathbb{E}[E_{val}(h)] = \frac{1}{K} \sum_{k=1}^{K} \mathbb{E}[e(h(\mathbf{x})_k, y_k)] = E_{out}$$

  - 'Reliable estimate' if K is large

(on rarely used validation set, otherwise data gets contaminated)

(this gives a much better (lower) variance than on a single point given K is large)

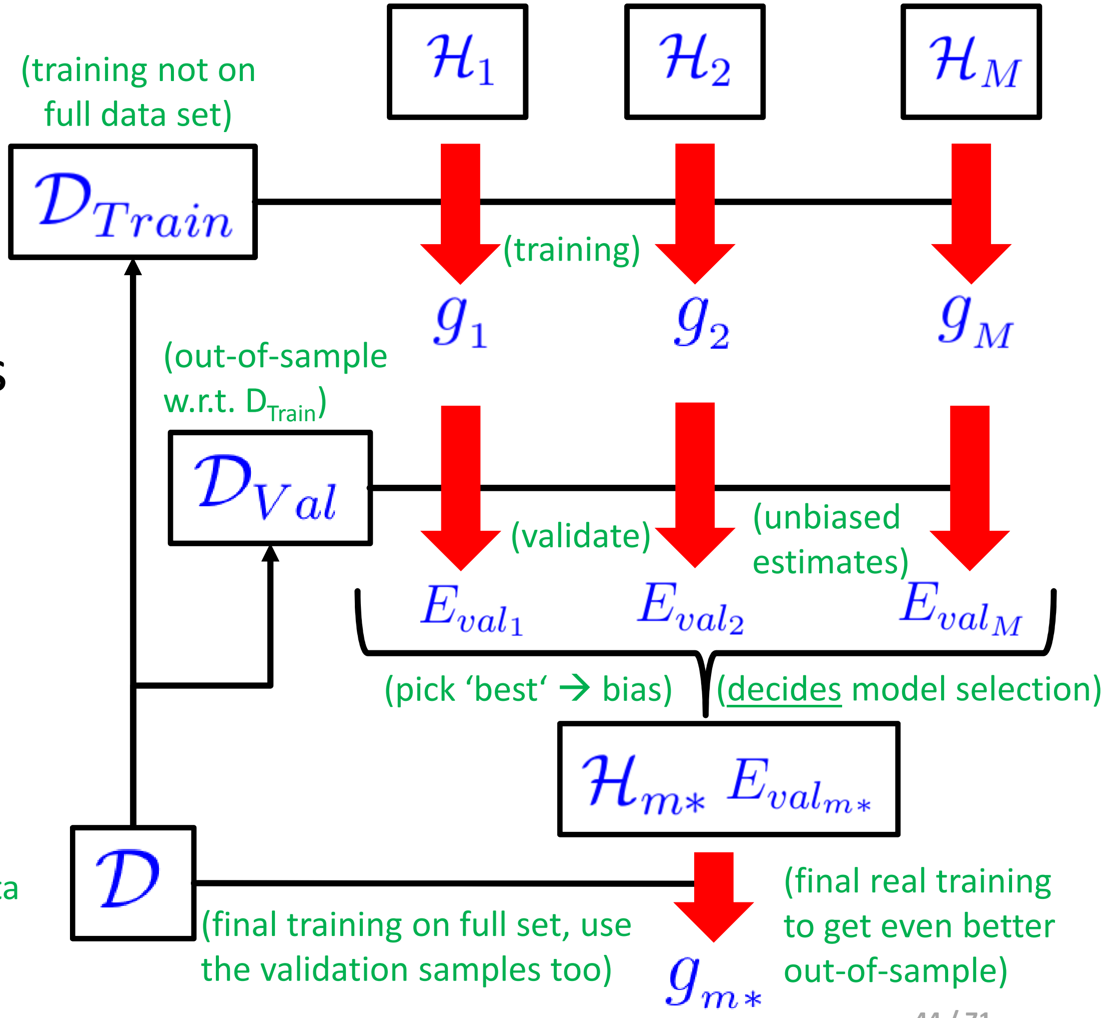
Validation Technique – Model Selection Process

- **Model selection is choosing (a) different types of models or (b) parameter values inside models**
- **Model selection takes advantage of the validation error in order to decide → 'pick the best'**

Hypothesis Set
$\mathcal{H} = \{h\};\ g \in \mathcal{H}$

(set of candidate formulas across models)

- Many different models
  Use validation error to perform select decisions
- Careful consideration:
  - 'Picked means decided' hypothesis has already bias (→ contamination)
  - Using $\mathcal{D}_{Val}$ M times



Final Hypothesis
$g_{m*} \approx f$

(test this on unseen data good, but depends on availability in practice)

Exercises – Add 1/5th for Validation

ANN 2 Hidden 1/5 Validation – MNIST Dataset

- If there is enough data available one rule of thumb is to take 1/5 (0.2) 20% of the datasets for validation only
- Validation data is used to perform model selection (i.e. parameter / topology decisions)

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils

# ANN parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'
VALIDATION_SPLIT = 0.2 # 1/5 for validation
```

- The validation split parameter enables an easy validation approach during the model training (aka fit)
- Expectations should be a higher accuracy for unseen data since training data is less biased when using validation for model decisions (check statistical learning theory)
- VALIDATION_SPLIT: Float between 0 and 1
- Fraction of the training data to be used as validation data
- The model fit process will set apart this fraction of the training data and will not train on it
- Intead it will evaluate the loss and any model metrics on the validation data at the end of each epoch.

```
# compilation
model.compile(loss='categorical_crossentropy', optimizer=OPTIMIZER, metrics=['accuracy'])

# fit the model
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split = VALIDATION_SPLIT)
```

ANN 2 Hidden 1/5 Validation – MNIST Dataset – Job Script

```
#!/bin/bash -x
#SBATCH--nodes=1
#SBATCH--ntasks=1
#SBATCH--output=mnist_out.%j
#SBATCH--error=mnist_err.%j
#SBATCH--time=01:00:00
#SBATCH--mail-user=m.riedel@fz-juelich.de
#SBATCH--mail-type=ALL
#SBATCH--job-name=ANN_mnist_2hidden_val

#SBATCH--partition=gpus
#SBATCH--gres=gpu:1

#SBATCH--reservation=deep_learning

### location executable
MNIST=/homea/hpclab/train001/tools/mnist/mnist-ann-2hidden-val.py

module restore dl_tutorial_2

### submit
python $MNIST
```

ANN – MNIST Dataset – Job Submit & Check Output

```
[train001@jrl06 scripts]$ sbatch submit_train_ann_2hidden_val_mnist.sh
Submitted batch job 5522552
```

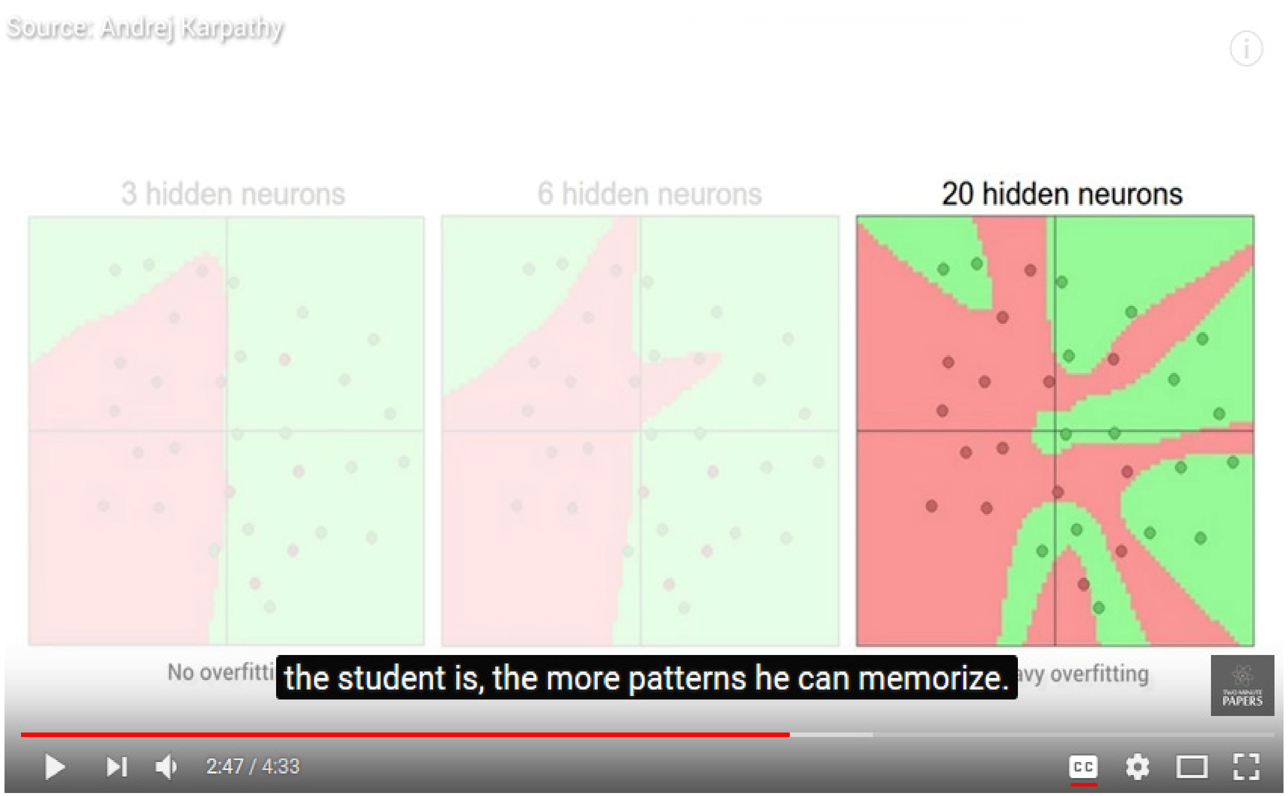
```
[train001@jrl06 scripts]$ more ann-2hidden-val-mnist_out.5522545
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
Layer (type)                 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
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200

  128/48000 [..............................] - ETA: 4:53 - loss: 2.3388 - acc: 0.0391
```

ANN 2 Hidden – 1/5 Validation – MNIST Dataset – Output

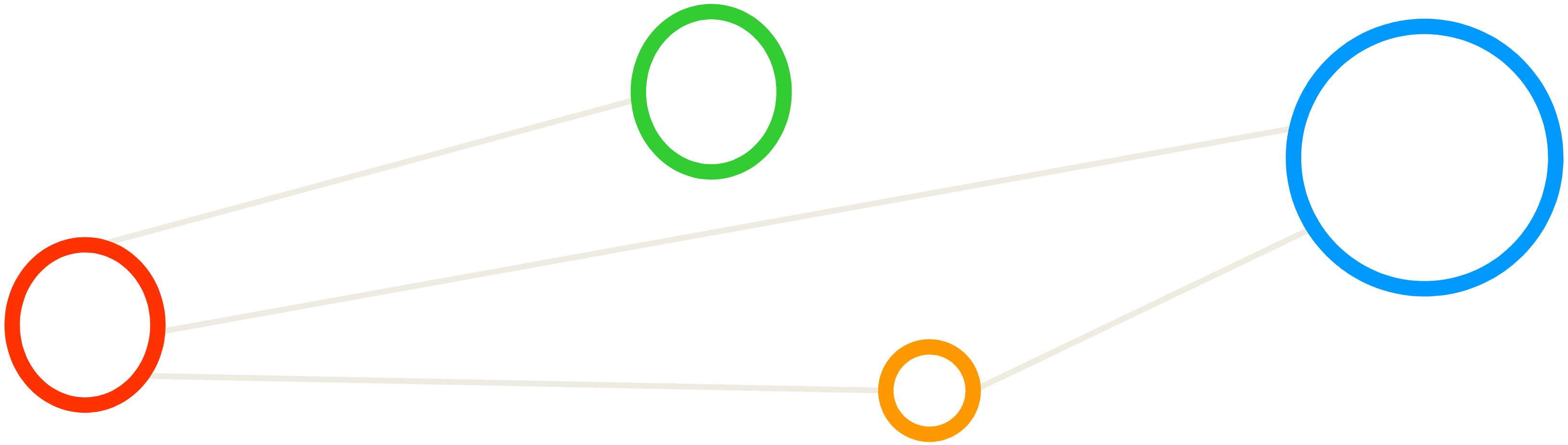
```
 6784/10000 [==================>..........] - ETA: 0s
 8192/10000 [======================>......] - ETA: 0s
 9568/10000 [==========================>..] - ETA: 0s
10000/10000 [==============================] - 0s 37us/step
('Test score: ', 0.07833538340910454)
('Test accuracy: ', 0.9772)
```

[Video] Overfitting in Deep Neural Networks



[4] Overfitting and Regularization For Deep Learning, YouTube

Regularization



Remote Sensing - Experimental Setup – Growing Parameter

- CNN Setup
  - Table overview
- HPC Machines used
  - Systems JURECA and JURON
- GPUs
  - NVIDIA Tesla K80 (JURECA)
  - NVIDIA Tesla P100 (JURON)
  - While Using MathWorks' Matlab for the data
- Frameworks
  - Keras library (2.0.6) was used
  - Tensorflow (0.12.1 on Jureca, 1.3.0rc2 on Juron) as back-end
  - Automated usage of the GPU's of these machines via Tensorflow

Feature	Representation / Value
Conv. Layer Filters	48, 32, 32
Conv. Layer Filter size	(3,3,5), (3,3,5), (3,3,5)
Dense Layer Neurons	128, 128
Optimizer	SGD
Loss Function	mean squared error
Activation Functions	ReLU
Training Epochs	600
Batch Size	50
Learning Rate	1
Learning Rate Decay	5×10^{-6}

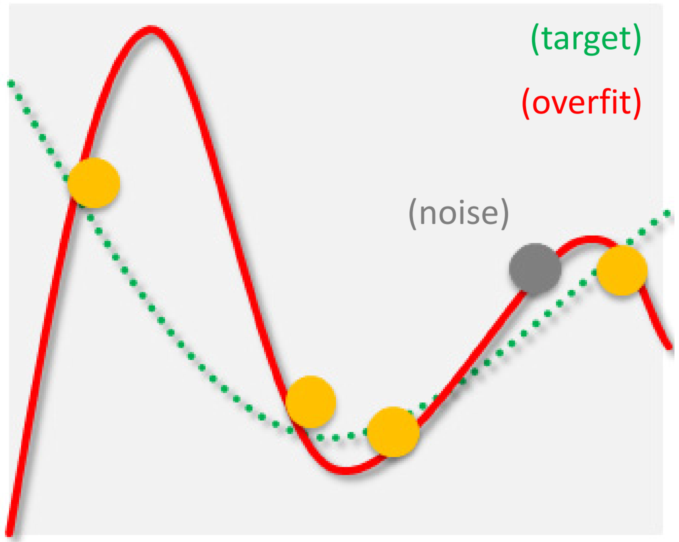
(adding regularization values adds even more complexity in finding the right parameters)

(having the validation with the full grid search of all parameters and all combinations is quite compute – intensive → ~infeasable)

Challenge Two – Problem of Overfitting

- **<u>Over</u>fitting refers to fit the data too well – more than is warranted – thus may misguide the learning**
- **Overfitting is not just 'bad generalization' - e.g. the VC dimension covers noiseless & noise targets**
- **Theory of Regularization are approaches against overfitting and prevent it using different methods**

- Key problem: noise in the target function leads to overfitting
  - Effect: 'noisy target function' and its noise misguides the fit in learning
  - There is always 'some noise' in the data
  - Consequence: poor target function ('distribution') approximation
- Example: Target functions is second order polynomial (i.e. parabola)
  - Using a higher-order polynomial fit
  - Perfect fit: low $E_{in}(g)$, but large $E_{out}(g)$



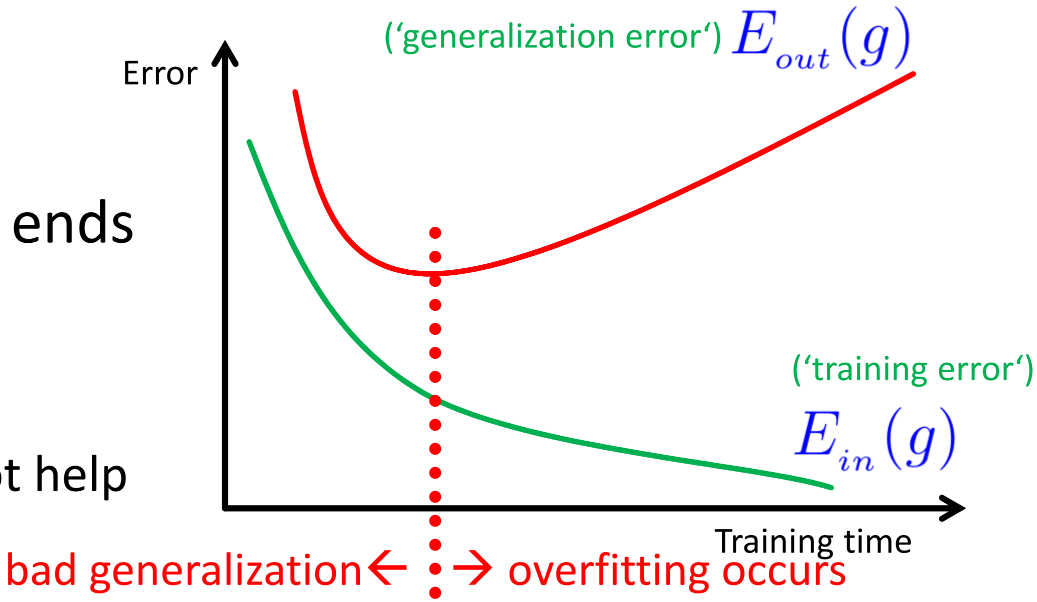
(but simple polynomial works good enough)
('over': here meant as 4th order, a 3rd order would be better, 2nd best)

Problem of Overfitting – Clarifying Terms

- **A good model must have low training error (E_{in}) and low generalization error (E_{out})**
- **Model overfitting is if a model fits the data too well (E_{in}) with a poorer generalization error (E_{out}) than another model with a higher training error (E_{in})**

[1] Introduction to Data Mining

- Overfitting & Errors
  - $E_{in}(g)$ goes down
  - $E_{out}(g)$ goes up
- 'Bad generalization area' ends
  - Good to reduce $E_{in}(g)$
- 'Overfitting area' starts
  - Reducing $E_{in}(g)$ does not help
  - Reason 'fitting the noise'

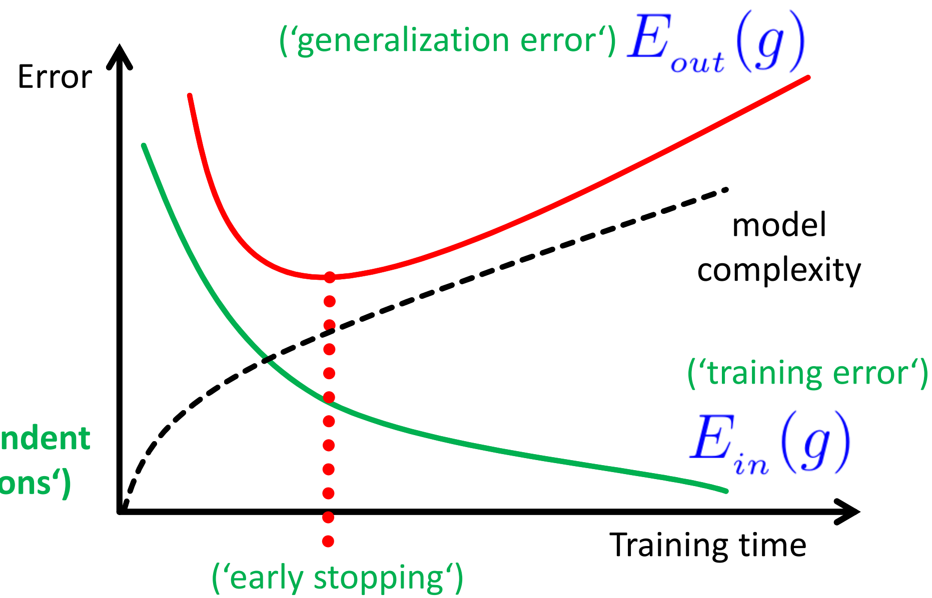


- **The two general approaches to prevent overfitting are (1) regularization and (2) validation**

Problem of Overfitting – Model Relationships

- Review 'overfitting situations'
  - When comparing 'various models' and related to 'model complexity'
  - Different models are used, e.g. 2^{nd} and 4^{th} order polynomial
  - Same model is used with e.g. two different instances
    (e.g. two neural networks but with different parameters)
- Intuitive solution
  - Detect when it happens
  - 'Early stopping regularization term' to stop the training
  - Early stopping method (later)

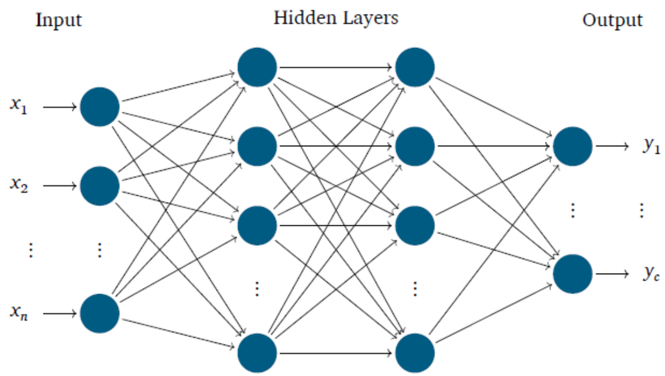
('model complexity measure: the VC analysis was independent of a specific target function – bound for all target functions')

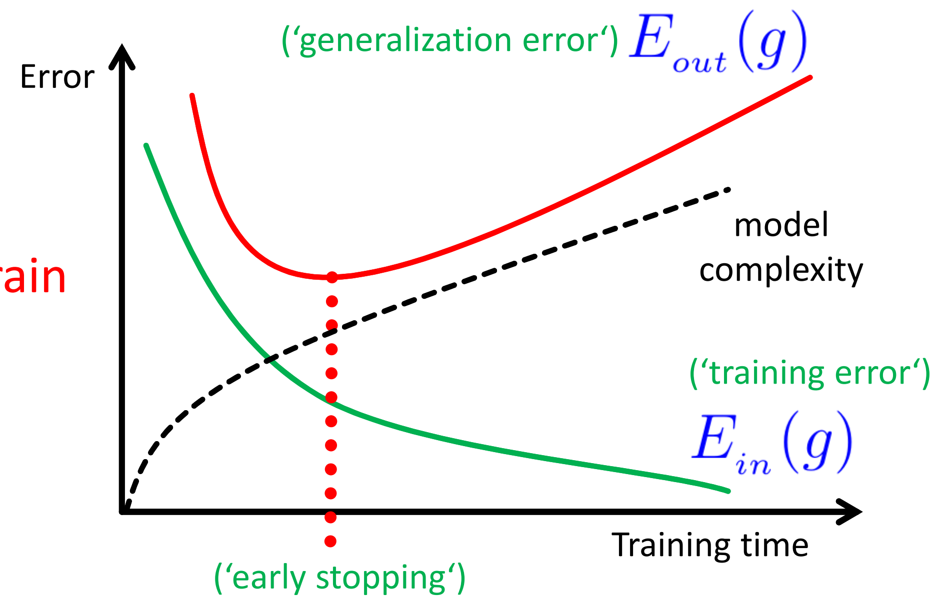


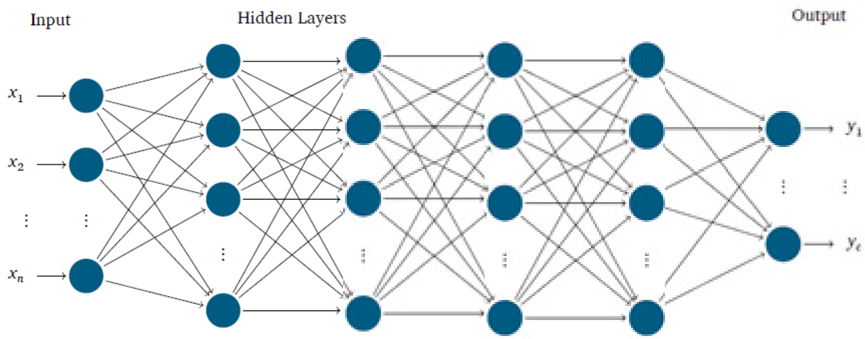
- **'Early stopping' approach is part of the theory of regularization, but based on validation methods**

Problem of Overfitting – ANN Model Example

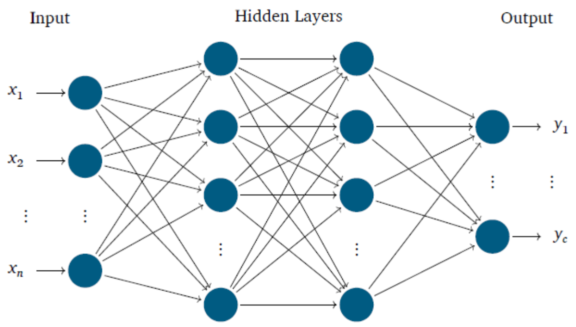
- Two Hidden Layers
  - Good accuracy and works well
  - Model complexity seem to match the application & data
- Four Hidden Layers
  - Accuracy goes down
  - $E_{in}(g)$ goes down
  - $E_{out}(g)$ goes up
  - Significantly more weights to train
  - Higher model complexity

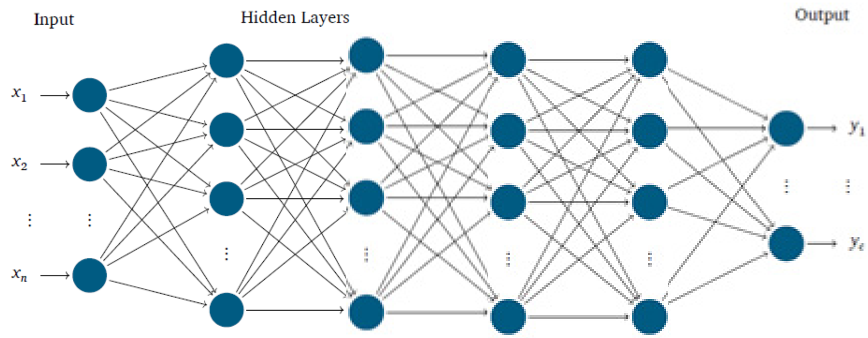






Exercises - Add more Hidden Layers – Accuracy?

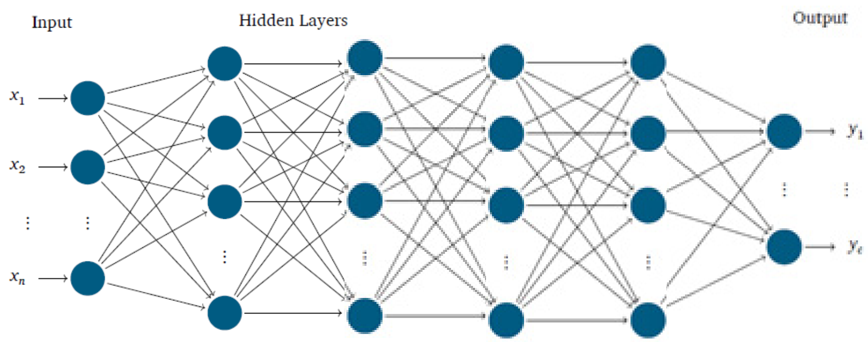




Exercises – Add more Hidden Layers – Growth Parameter

```
[train001@jrl06 scripts]$ more ann-4hidden-val-mnist_out.5522545
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
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, 128)               16512
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 151,306
Trainable params: 151,306
Non-trainable params: 0
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200
```

Exercises - Add more Hidden Layers – 4 Hidden Layers




```
[train001@jrl06 scripts]$ tail ann-4hidden-val-mnist_out.5522545
 1248/10000 [==>...........................] - ETA: 0s
 2464/10000 [======>.......................] - ETA: 0s
 3744/10000 [==========>...................] - ETA: 0s
 4992/10000 [=============>................] - ETA: 0s
 6272/10000 [=================>............] - ETA: 0s
 7520/10000 [=====================>........] - ETA: 0s
 8768/10000 [=========================>....] - ETA: 0s
10000/10000 [==============================] - 0s 40us/step
('Test score: ', 0.12568975675739202)
('Test accuracy: ', 0.9746)
```

Problem of Overfitting – Noise Term Revisited

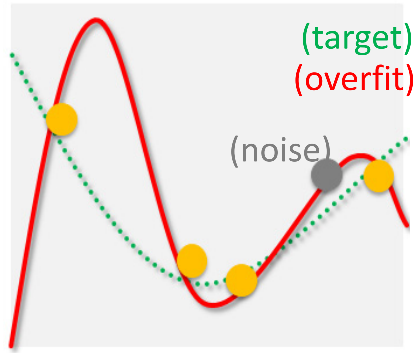
- ‘(Noisy) Target function‘ is not a (deterministic) function
  - Getting with ‘same x in‘ the ‘same y out‘ is not always given in practice
  - Idea: Use a ‘target distribution‘ instead of ‘target function‘

Unknown Target Distribution $P(y|\mathbf{x})$

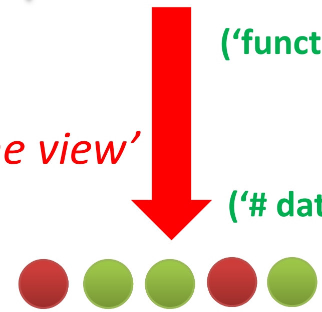
target function $f: X \rightarrow Y$ plus noise

(ideal function)

- **Fitting some noise in the data is the basic reason for overfitting and harms the learning process**
- **Big datasets tend to have more noise in the data so the overfitting problem might occur even more intense**



- ‘Different types of some noise‘ in data
  - Key to understand overfitting & preventing it
  - ‘Shift of view‘: refinement of noise term
  - Learning from data: ‘matching properties of # data‘



Problem of Overfitting – Stochastic Noise

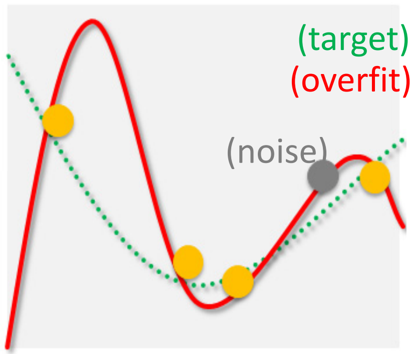
- Stoachastic noise is a part 'on top of' each learnable function
  - Noise in the data that can not be captured and thus not modelled by f
  - Random noise : aka 'non-deterministic noise'
  - Conventional understanding established early in this course
  - Finding a 'non-existing pattern in noise not feasible in learning'

Unknown Target Distribution

target function $f : X \rightarrow Y$ plus noise $P(y|\mathbf{x})$

(ideal function)

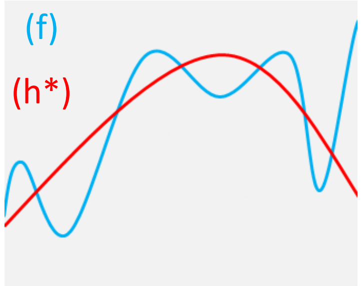
- Practice Example
  - Random fluctuations and/or measurement errors in data
  - Fitting a pattern that not exists 'out-of-sample'
  - Puts learning progress 'off-track' and 'away from f'



- **Stochastic noise here means noise that can't be captured, because it's just pure 'noise as is' (nothing to look for) – aka no pattern in the data to understand or to learn from**

Problem of Overfitting – Deterministic Noise

- Part of target function f that H can not capture: $f(\mathbf{x}) - h^*(\mathbf{x})$
  - Hypothesis set H is limited so best h* can not fully approximate f
  - h* approximates f, but fails to pick certain parts of the target f
  - 'Behaves like noise', existing even if data is 'stochastic noiseless'
- Different 'type of noise' than stochastic noise
  - Deterministic noise depends on $\mathcal{H}$ **(determines how much more can be captured by h*)**
  - E.g. same f, and more sophisticated $\mathcal{H}$: noise is smaller (stochastic noise remains the same, nothing can capture it)
  - Fixed for a given $\mathbf{x}$, clearly measurable (stochastic noise may vary for values of $\mathbf{x}$)



(learning deterministic noise is outside the ability to learn for a given h*)

- **Deterministic noise here means noise that can't be captured, because it is a limited model (out of the league of this particular model), e.g. 'learning with a toddler statistical learning theory'**

Problem of Overfitting – Impacts on Learning

- **The higher the degree of the polynomial (cf. model complexity), the more degrees of freedom are existing and thus the more capacity exists to overfit the training data**

- Understanding deterministic noise & target complexity
  - Increasing target complexity increases deterministic noise (at some level)
  - Increasing the number of data N decreases the deterministic noise
- Finite N case: $\mathcal{H}$ tries to fit the noise
  - Fitting the noise straightforward (e.g. Perceptron Learning Algorithm)
  - Stochastic (in data) and deterministic (simple model) noise will be part of it
- Two 'solution methods' for avoiding overfitting
  - Regularization: 'Putting the brakes in learning', e.g. early stopping (more theoretical, hence 'theory of regularization')
  - Validation: 'Checking the bottom line', e.g. other hints for out-of-sample (more practical, methods on data that provides 'hints')

High-level Tools – Keras – Regularization Techniques

- 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.Dropout(rate,
                     noise_shape=None,
                     seed=None)
```

- Dropout is randomly setting a fraction of input units to 0 at each update during training time, which helps prevent overfitting (using parameter rate)

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
kernel_regularizer=regularizers.l2(0.01),
activity_regularizer=regularizers.l1(0.01)))
```

- L2 regularizers allow to apply penalties on layer parameter or layer activity during optimization itself – therefore the penalties are incorporated in the loss function during optimization

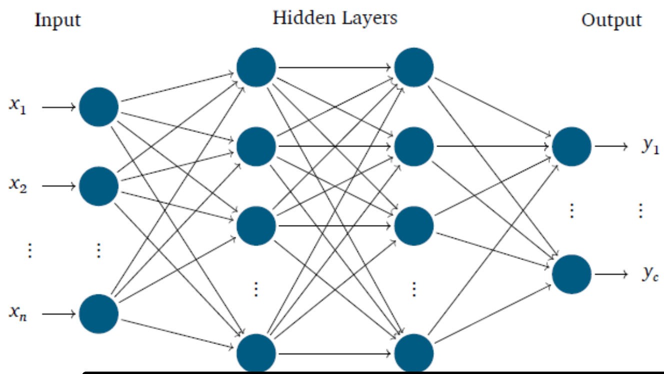
Keras *[5] Keras Python Deep Learning Library*

Exercises – Underfitting & Add Dropout Regularizer

- Run with 20 Epochs first (not trained enough); then 250 Epochs
  - Training accuracy should be above the test accuracy – otherwise ‘underfitting‘

ANN – MNIST Dataset – Add Weight Dropout Regularizer

```
# ANN parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 128
OPTIMIZER = 'SGD'
VALIDATION_SPLIT = 0.2 # 1/5 for validation
DROPOUT = 0.3
```




```
# simple ANN model
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
```

- **A Dropout() regularizer randomly drops with ist dropout probability some of the values propagated inside the Dense network hidden layers improving accuracy again**
- **Our standard model is already modified in the python script but needs to set the DROPOUT rate**
- **A Dropout() regularizer randomly drops with ist dropout probability some of the values propagated inside the Dense network hidden layers improving accuracy again**

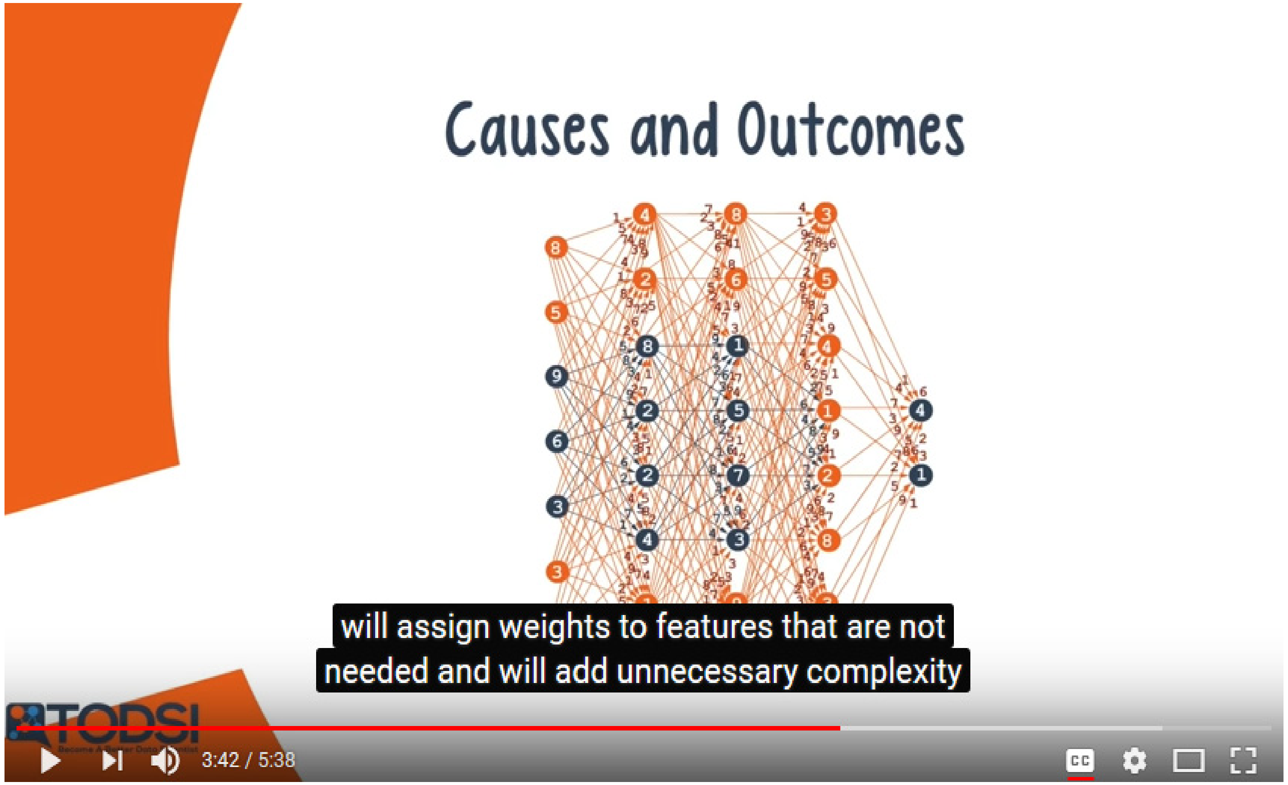
(compare with CNN models, day one ~99%)

ANN – MNIST - DROPOUT

```
[train001@jrl06 scripts]$ more mnist_out.5522587
(60000, 'train samples')
(10000, 'test samples')
(784, 'input pixel values per train samples')
(784, 'input pixel values per test samples')
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (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
_________________________________________________________________
Train on 48000 samples, validate on 12000 samples
Epoch 1/200
```

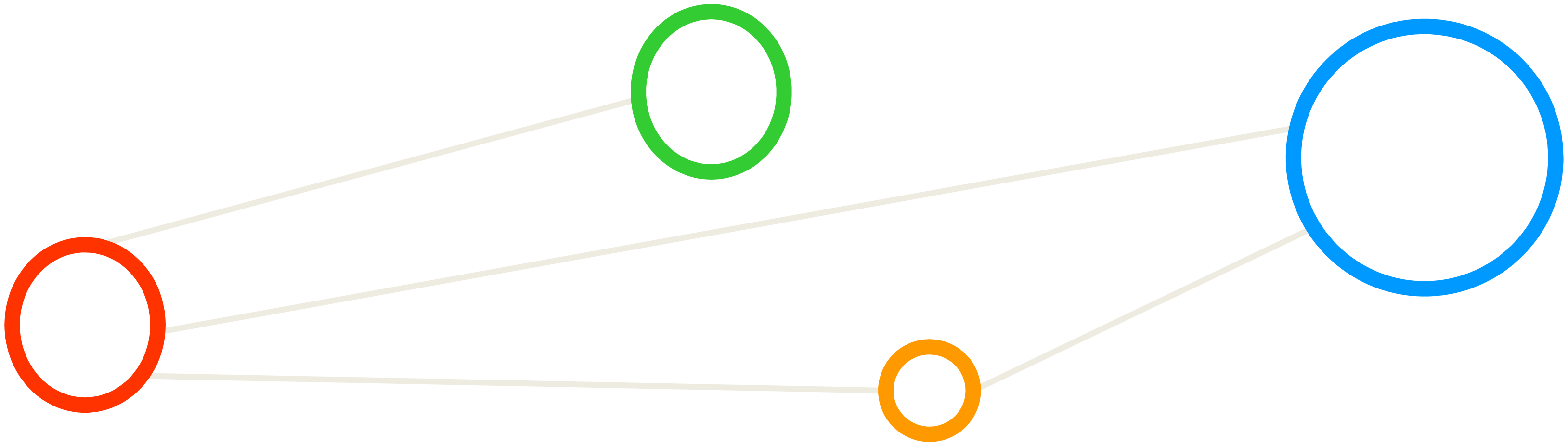
```
[train001@jrl06 scripts]$ tail mnist_out.5522587
 1088/10000 [==>...........................] - ETA: 0s
 2464/10000 [======>.......................] - ETA: 0s
 3840/10000 [==========>...................] - ETA: 0s
 5184/10000 [==============>...............] - ETA: 0s
 6560/10000 [==================>...........] - ETA: 0s
 7936/10000 [======================>.......] - ETA: 0s
 9344/10000 [===========================>..] - ETA: 0s
10000/10000 [==============================] - 0s 39us/step
('Test score: ', 0.07293171860824223)
('Test accuracy: ', 0.9779)
```

[Video] Overfitting in Deep Neural Networks



[3] How good is your fit?, YouTube

Lecture Bibliography



Lecture Bibliography

- [1] An Introduction to Statistical Learning with Applications in R,
  Online: http://www-bcf.usc.edu/~gareth/ISL/index.html
- [2] Keras Python Deep Learning Library,
  Online: https://keras.io/
- [3] YouTube Video, 'How good is your fit? - Ep. 21 (Deep Learning SIMPLIFIED)',
  Online: https://www.youtube.com/watch?v=cJA5IHIIL30
- [4] YouTube Video, 'Overfitting and Regularization For Deep Learning | Two Minute Papers #56',
  Online: https://www.youtube.com/watch?v=6aF9sJrzxaM


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