Neural Networks & Deep Learning

PARALLEL & SCALABLE MACHINE LEARNING & DEEP LEARNING

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

LECTURE 2

HDSLEE HELMHOLTZ SCHOOL FOR DATA SCIENCE IN LIFE | EARTH | ENERGY

HiDA HELMHOLTZ Information & Data Science Academy

@Morris Riedel @MorrisRiedel @MorrisRiedel

Artificial Neural Network Learning Model & Backpropagation

November 04, 2020

Online Lecture

SEE EUROPEAN OPEN SCIENCE CLOUD EOSC NORDIC EuroHPC Joint Undertaking ADMIRE EURO

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 HELMHOLTZAI ARTIFICIAL INTELLIGENCE COOPERATION UNIT

Review of Lecture 1 – Introduction to ML & Perceptron Learning Model

non-linear activation function

linear combination of input data

$$\hat{y} = g\left(1 * w_0 + \sum_{i=1}^{m} x_i * w_i\right)$$

Output

Bias

Sum

$$\hat{y} = g\left(w_0 + X^T \mathbf{w}\right)$$

Constants

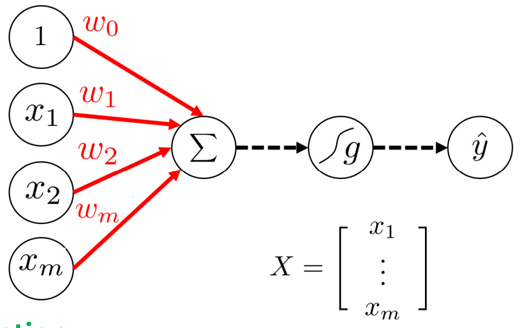
$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Input Data

Trainable Weights

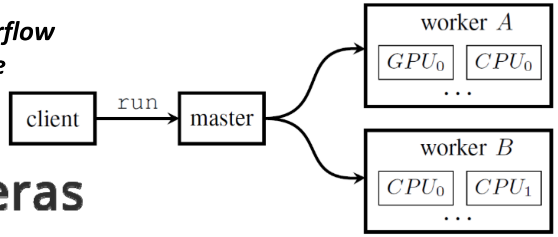
```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   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
```

necessary reshaping & normalization



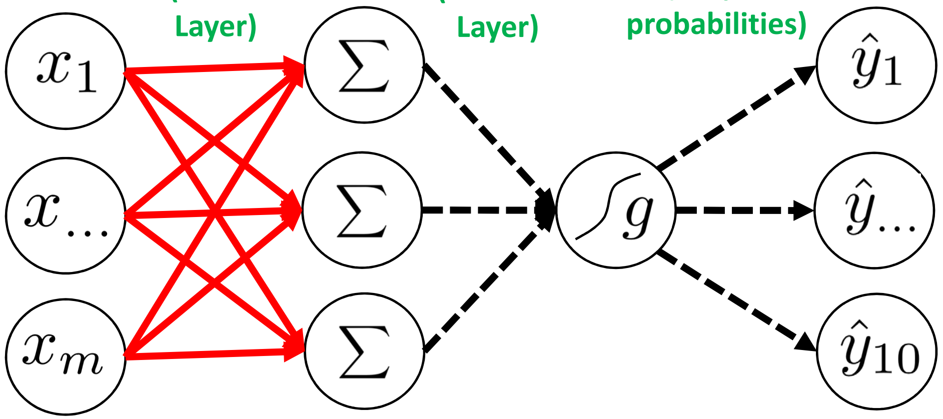
$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

[5] Tensorflow Web page



[6] Keras Web page

✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**



(Dense Layer)

(Softmax Layer)

(output probabilities)

(input m = 784)

(10 neurons sum with 10 bias)

(softmax activation)

(NB_CLASSES = 10)

Outline of the Course

1. Introduction to Machine Learning & Perceptron Learning Model
2. Artificial Neural Network Learning Model & Backpropagation
3. Deep Learning & Convolutional Neural Network Learning Model
4. Using Artificial Neural Networks & Convolutional Neural Networks

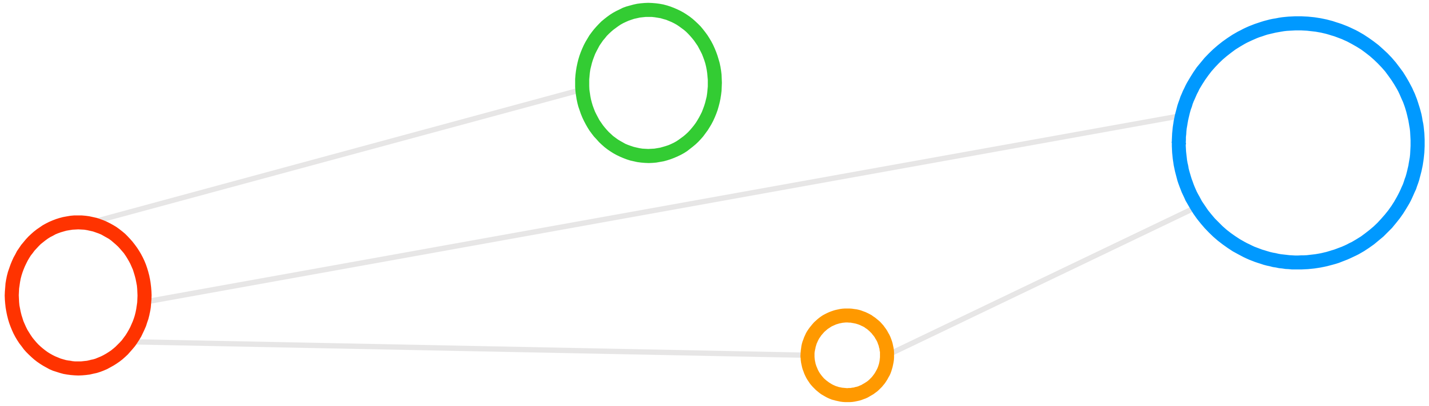
- 🟩 Practical Topics
- 🟦 Theoretical / Conceptual Topics

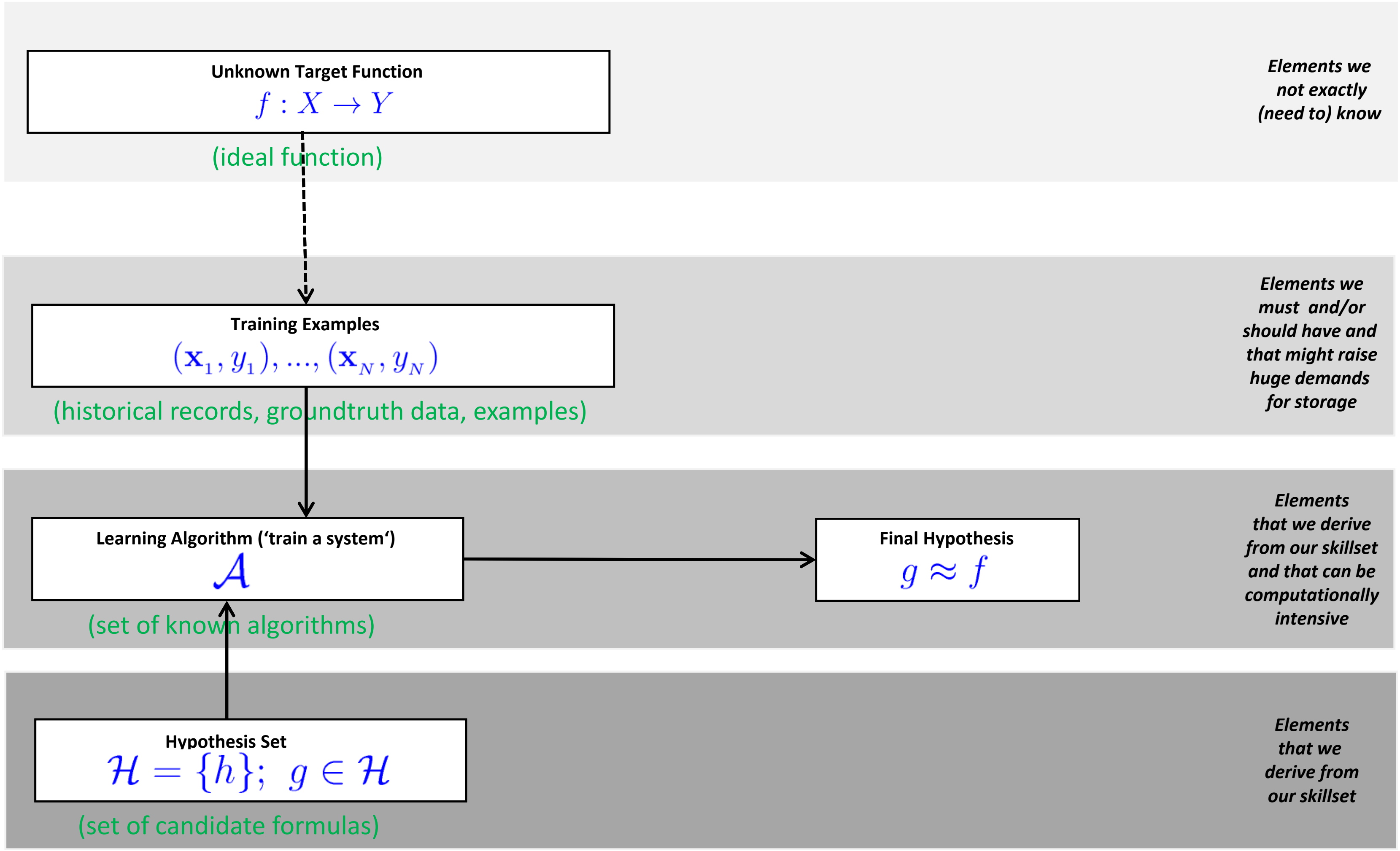
Outline

- Supervised Learning & Statistical Learning Theory
  - Formalization of Supervised Learning & Mathematic Building Blocks Continued
  - Understanding Statistical Learning Theory Basics & PAC Learning
  - Infinite Learning Model & Union Bound
  - Hoeffding Inequality & Vapnik – Chervonenkis (VC) Inequality & Dimension
  - Understanding the Relationship of Number of Samples & Model Complexity

- Artificial Neural Networks & Backpropagation
  - Conceptual Idea of a Multi-Layer Perceptron
  - Artificial Neural Networks (ANNs) & Backpropagation
  - Problem of Overfitting & Different Types of Noise
  - Validation for Model Selection as another Technique against Overfitting
  - Regularization as Technique against Overfitting

Supervised Learning & Statistical Learning Theory





Unknown Target Function

$f : X \to Y$

(ideal function)

Elements we not exactly (need to) know

Training Examples

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

(historical records, groundtruth data, examples)

Elements we must and/or should have and that might raise huge demands for storage

Learning Algorithm (‘train a system‘)

$\mathcal{A}$

(set of known algorithms)

Final Hypothesis

$g \approx f$

Elements that we derive from our skillset and that can be computationally intensive

Hypothesis Set

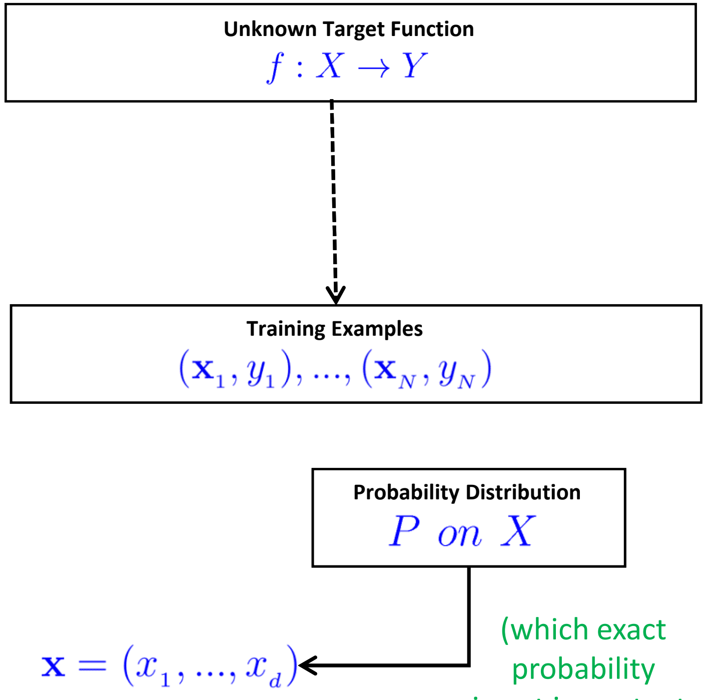
$\mathcal{H} = \{h\}; \; g \in \mathcal{H}$

(set of candidate formulas)

Elements that we derive from our skillset

Feasibility of Learning – Probability Distribution

- Predict output from future input
  (fitting existing data is not enough)
  - In-sample ‘1000 points‘ fit well
  - Possible: Out-of-sample >= ‘1001 point‘ doesn‘t fit very well
  - Learning ‘any target function‘ is not feasible (can be anything)
- Assumptions about ‘future input‘
  - Statement is possible to define about the data outside the in-sample data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
  - All samples (also future ones) are derived from same ‘unknown probability‘ distribution $P \ on \ X$



(which exact probability is not important, but should not be completely random)

- **Statistical Learning Theory assumes an unknown probability distribution over the input space X**

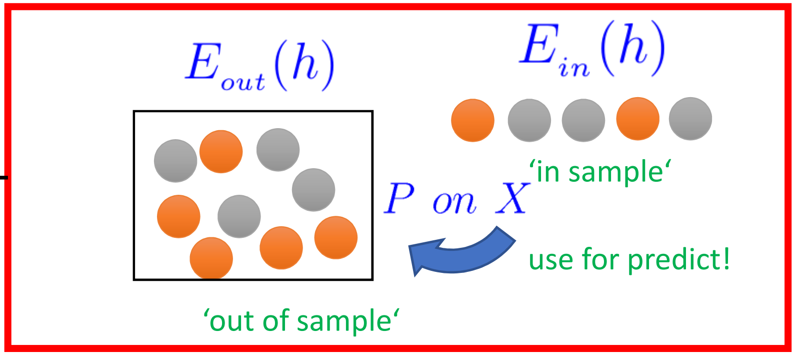
Feasibility of Learning – In Sample vs. Out of Sample

- Given ‘unknown‘ probability $P \ on \ X$
  - Given large sample N for $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
  - There is a probability of ‘picking one point or another‘
  - ‘Error on in sample‘ is known quantity (using labelled data): $E_{in}(h)$
  - ‘Error on out of sample‘ is unknown quantity: $E_{out}(h)$
  - In-sample frequency is likely close to out-of-sample frequency

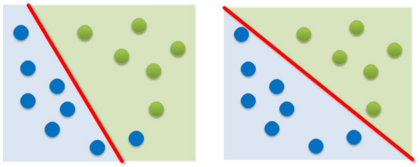
- **Statistical Learning Theory part that enables that learning is feasible in a probabilistic sense (P on X)**

E_{in} tracks E_{out}

depend on which hypothesis h out of M different ones



$\mathcal{H} = \{h_1, ..., h_m\};$



$$E_{in}(h) \approx E_{out}(h)$$

use $E_{in}(h)$ as a proxy thus the other way around in learning

$$E_{out}(h) \approx E_{in}(h)$$

Feasibility of Learning – Union Bound & Factor M

- Assuming no overlaps in hypothesis set
  - Apply very ‘poor‘ mathematical rule ‘union bound‘
  - (Note the usage of g instead of h, we need to visit all)

Final Hypothesis

$g \approx f$

Think if E_{in} deviates from E_{out} with more than tolerance Є it is a ‘bad event‘ in order to apply union bound

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <= \Pr\ [\ |\ E_{in}(h_1) - E_{out}(h_1)\ | > \epsilon$$

$$\text{or}\ \ |\ E_{in}(h_2) - E_{out}(h_2)\ | > \epsilon\ \ \ldots$$

$$\text{or}\ \ |\ E_{in}(h_M) - E_{out}(h_M)\ | > \epsilon\ \ ]$$

‘visiting M different hypothesis‘

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <= \sum_{m=1}^{M} \Pr\ [\ |\ E_{in}(h_m) - E_{out}(h_m)\ | > \epsilon\ ]$$

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <= \sum_{m=1}^{M} 2e^{-2\epsilon^2 N}$$

fixed quantity for each hypothesis obtained from Hoeffdings Inequality

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <= 2Me^{-2\epsilon^2 N}$$

problematic: if M is too big we loose the link between the in-sample and out-of-sample

- **The union bound means that (for any countable set of m ‘events‘) the probability that at least one of the events happens is not greater that the sum of the probabilities of the m individual ‘events‘**

Feasibility of Learning – Modified Hoeffding‘s Inequality

- Errors in-sample $E_{in}(g)$ track errors out-of-sample $E_{out}(g)$
  - Statement is made being ‘Probably Approximately Correct (PAC)‘
  - Given M as number of hypothesis of hypothesis set $\mathcal{H}$
  - ‘Tolerance parameter‘ in learning ϵ
  - Mathematically established via ‘modified Hoeffdings Inequality‘:
    (original Hoeffdings Inequality doesn‘t apply to multiple hypothesis)

‘Approximately‘ ‘Probably‘

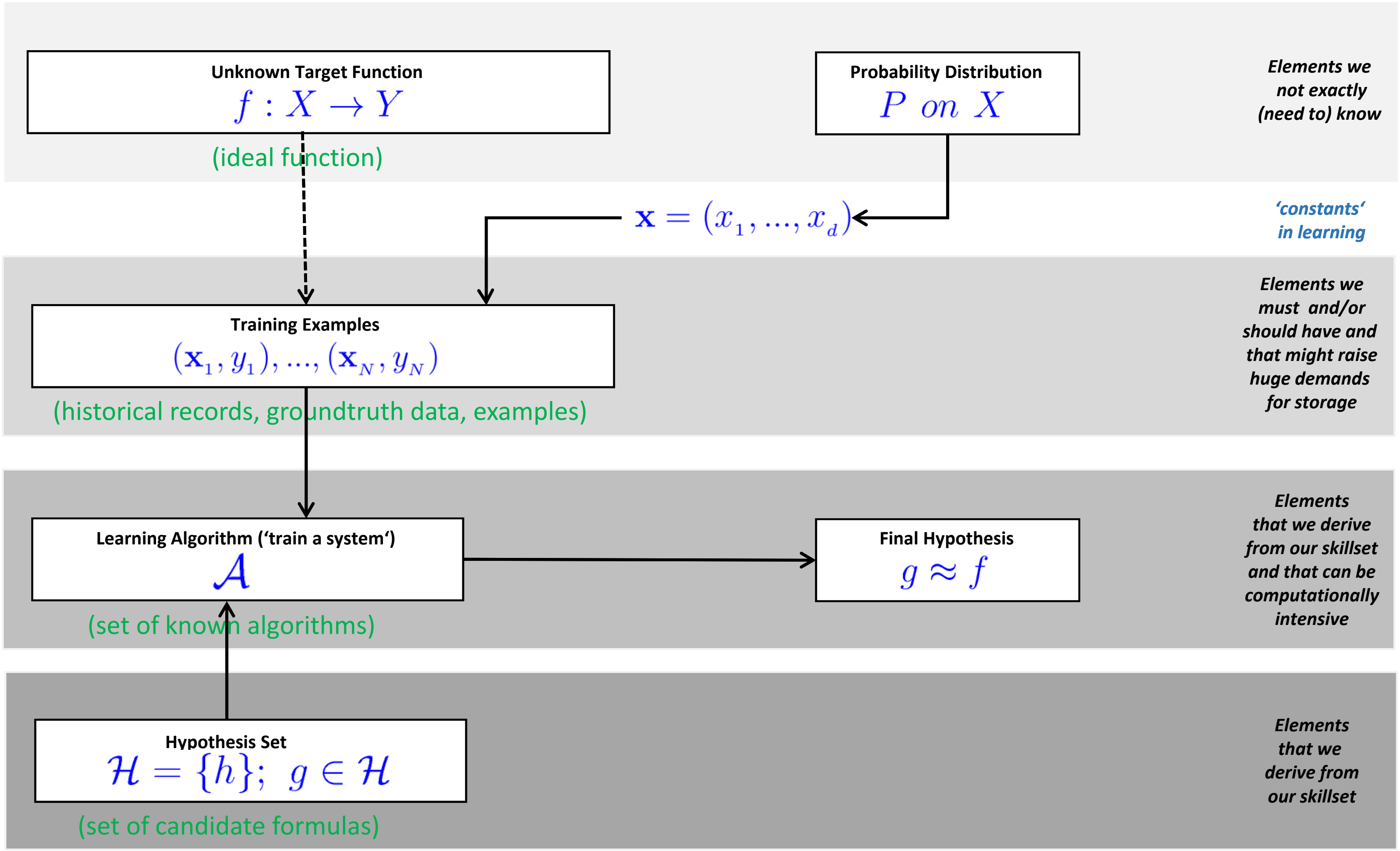
$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

‘Probability that E_{in} deviates from E_{out} by more than the tolerance Є is a small quantity depending on M and N‘

- Theoretical ‘Big Data‘ Impact → more N → better learning
  - The more samples N the more reliable will track $E_{in}(g)$ $E_{out}(g)$ well
  - (But: the ‘quality of samples‘ also matter, not only the number of samples)
  - For supervised learning also the ‘label‘ has a major impact in learning (later)

[1] Valiant, ‘A Theory of the Learnable’, 1984

- **Statistical Learning Theory part describing the Probably Approximately Correct (PAC) learning**



Unknown Target Function

$f : X \rightarrow Y$

(ideal function)

Probability Distribution

$P \text{ on } X$

Elements we not exactly (need to) know

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

'constants' in learning

Training Examples

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

(historical records, groundtruth data, examples)

Elements we must and/or should have and that might raise huge demands for storage

Learning Algorithm ('train a system')

$\mathcal{A}$

(set of known algorithms)

Final Hypothesis

$g \approx f$

Elements that we derive from our skillset and that can be computationally intensive

Hypothesis Set

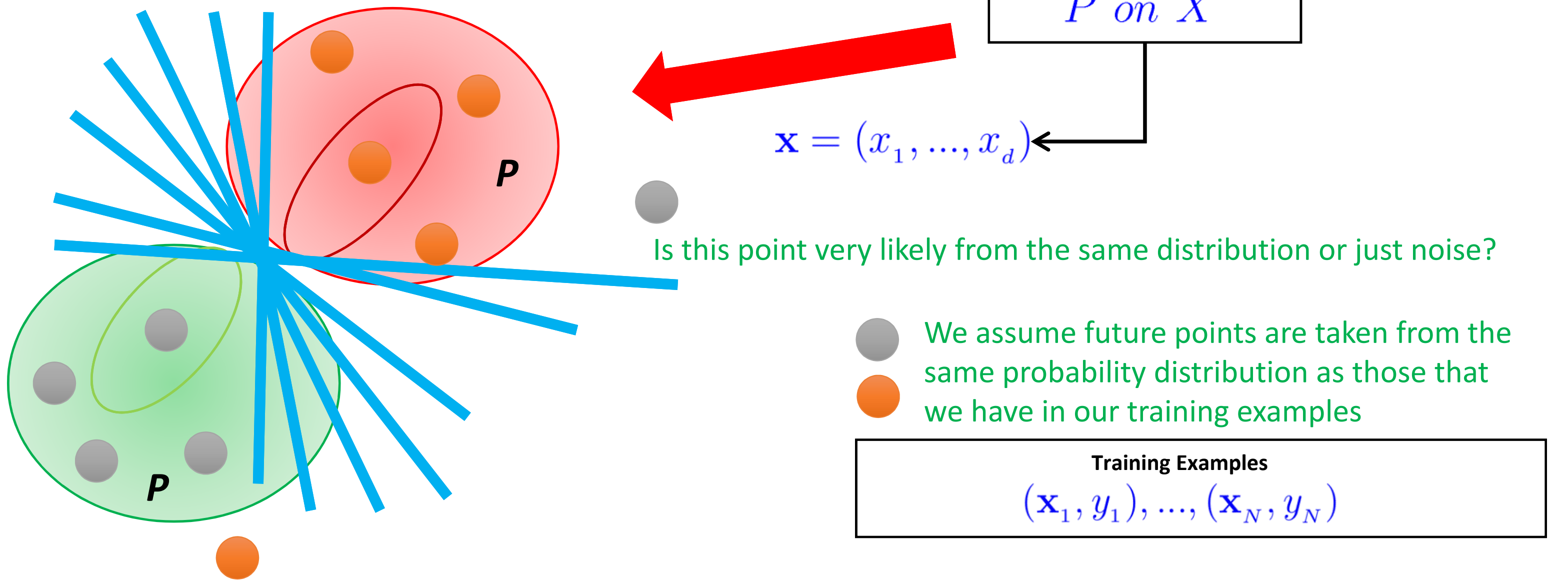
$\mathcal{H} = \{h\}; \;\; g \in \mathcal{H}$

(set of candidate formulas)

Elements that we derive from our skillset

Mathematical Building Blocks (4) – Our Linear Example

(infinite M decision boundaries depending on f)



Probability Distribution

$P \text{ on } X$

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

Is this point very likely from the same distribution or just noise?

We assume future points are taken from the same probability distribution as those that we have in our training examples

Training Examples

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

Is this point very likely from the same distribution or just noise?

(we help here with the assumption for the samples)

(we do not solve the M problem here)

$$\Pr \;[\; |\; E_{in}(g) - E_{out}(g) \;|\; > \epsilon \;]\; <= \; 2Me^{-2\epsilon^2 N}$$

(counter example would be for instance a random number generator, impossible to learn this!)

Statistical Learning Theory – Error Measure & Noisy Targets

- Question: How can we learn a function from (noisy) data?
- ‘Error measures‘ to quantify our progress, the goal is: $h \approx f$
  - Often user-defined, if not often ‘squared error‘:

$$e(h(\mathbf{x}), f(\mathbf{x})) = (h(\mathbf{x}) - f(\mathbf{x}))^2$$

  - E.g. ‘point-wise error measure‘

Error Measure
α

- ‘(Noisy) Target function‘ is not a (deterministic) function

(e.g. think movie rated now and in 10 years from now)

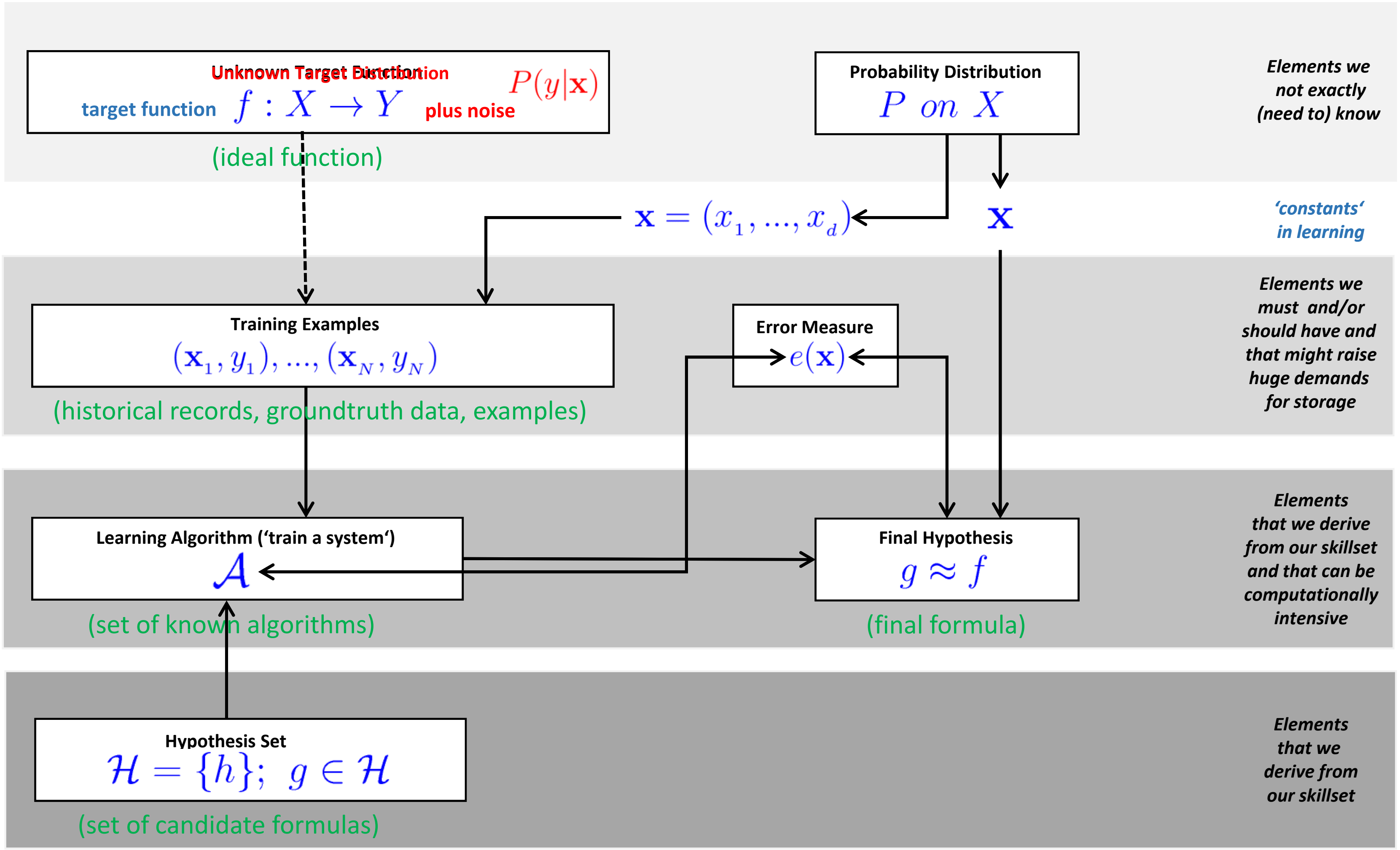
  - Getting with ‘same x in‘ the ‘same y out‘ is not always given in practice
  - Problem: ‘Noise‘ in the data that hinders us from learning
  - Idea: Use a ‘target distribution‘ instead of ‘target function‘
  - E.g. credit approval (yes/no)

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

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

(ideal function)

- Statistical Learning Theory refines the learning problem of learning an unknown target distribution



Unknown Target Distribution

Unknown Target Function

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

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

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

Hypothesis Set

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

(set of candidate formulas)

Elements that we derive from our skillset

Mathematical Building Blocks (5) – Our Linear Example

- Iterative Method using (labelled) training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$

(one point at a time is picked)

1. Pick one misclassified training point where:

$$sign(\mathbf{w}^T \mathbf{x}_n) \neq y_n$$

Error Measure
α

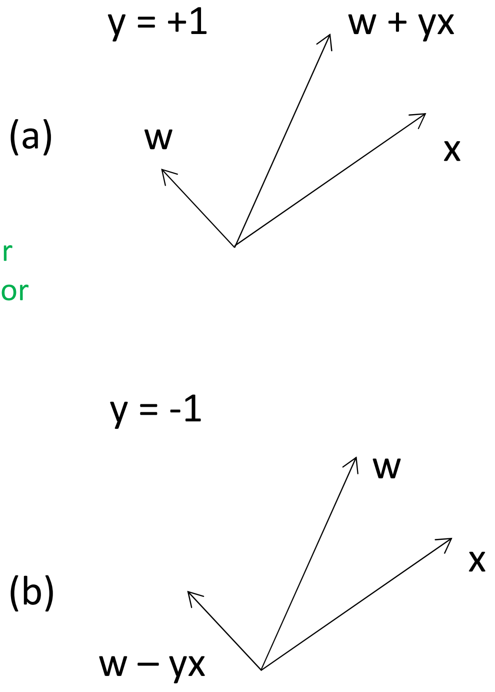
2. Update the weight vector:

(a) adding a vector or
(b) subtracting a vector

$$\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n$$

(y_n is either +1 or -1)

Error Measure
α



- Terminates when there are no misclassified points

(converges only with linearly seperable data)

Training and Testing – Influence on Learning

- Mathematical notations
  - Testing follows: (hypothesis clear) $\Pr\left[\,|\,E_{in}(g) - E_{out}(g)\,| > \epsilon\,\right] <= 2 \quad e^{-2\epsilon^2 N}$
  - Training follows: (hypothesis search) $\Pr\left[\,|\,E_{in}(g) - E_{out}(g)\,| > \epsilon\,\right] <= 2Me^{-2\epsilon^2 N}$
- Practice on ‘training examples‘

(e.g. student exam training on examples to get E_{in} ‚down‘, then test via exam)

  - Create two disjoint datasets
  - One used for training only (aka training set)
  - Another used for testing only (aka test set)

Training Examples

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

(historical records, groundtruth data, examples)

- Training & Testing are different phases in the learning process
  - Concrete number of samples in each set often influences learning

Theory of Generalization – Initial Generalization & Limits

- Learning is feasible in a probabilistic sense
  - Reported final hypothesis – using a ‘generalization window‘ on $E_{out}(g)$
  - Expecting ‘out of sample performance‘ tracks ‘in sample performance‘
  - Approach: $E_{in}(g)$ acts as a ‘proxy‘ for $E_{out}(g)$

$$E_{out}(g) \approx E_{in}(g)$$

This is not full learning – rather ‘good generalization‘ since the quantity $E_{out}(g)$ is an unknown quantity

- Reasoning
  - Above condition is not the final hypothesis condition:
  - More similiar like $E_{out}(g)$ approximates 0
    (out of sample error is close to 0 if approximating f)
  - $E_{out}(g)$ measures how far away the value is from the ‘target function’
  - Problematic because $E_{out}(g)$ is an unknown quantity (cannot be used…)
  - The learning process thus requires ‘two general core building blocks‘

Final Hypothesis

$g \approx f$

Theory of Generalization – Learning Process Reviewed

- ‘Learning Well‘
  - Two core building blocks that achieve $E_{out}(g)$ approximates 0
- First core building block
  - Theoretical result using Hoeffdings Inequality $E_{out}(g) \approx E_{in}(g)$
  - Using $E_{out}(g)$ directly is not possible – it is an unknown quantity
- Second core building block

(try to get the ‘in-sample‘ error lower)

  - Practical result using tools & techniques to get $E_{in}(g) \approx 0$
  - e.g. linear models with the Perceptron Learning Algorithm (PLA)
  - Using $E_{in}(g)$ is possible – it is a known quantity – ‘so lets get it small‘
  - Lessons learned from practice: in many situations ‘close to 0‘ impossible

- **Full learning means that we can make sure that $E_{out}(g)$ is close enough to $E_{in}(g)$ [from theory]**
- **Full learning means that we can make sure that $E_{in}(g)$ is small enough [from practical techniques]**

Complexity of the Hypothesis Set – Infinite Spaces Problem

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

theory helps to find a way to deal
with infinite M hypothesis spaces

- Tradeoff & Review
  - Tradeoff between Є, M, and the 'complexity of the hypothesis space H'
  - Contribution of detailed learning theory is to 'understand factor M'
- M Elements of the hypothesis set $\mathcal{H}$ M elements in H here
  - Ok if N gets big, but problematic if M gets big → bound gets meaningless
  - E.g. classification models like perceptron, support vector machines, etc.
  - Challenge: those classification models have continous parameters
  - Consequence: those classification models have infinite hypothesis spaces
  - Aproach: despite their size, the models still have limited expressive power

- **Many elements of the hypothesis set H have continous parameter with infinite M hypothesis spaces**

Factor M from the Union Bound & Hypothesis Overlaps

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <= \Pr\ [\ |\ E_{in}(h_1) - E_{out}(h_1)\ | > \epsilon$$

$$\text{or}\ \ |\ E_{in}(h_2) - E_{out}(h_2)\ | > \epsilon\ \ \ldots$$

$$\text{or}\ \ |\ E_{in}(h_M) - E_{out}(h_M)\ | > \epsilon\ \ ]$$

assumes no overlaps, all probabilities happen disjointly

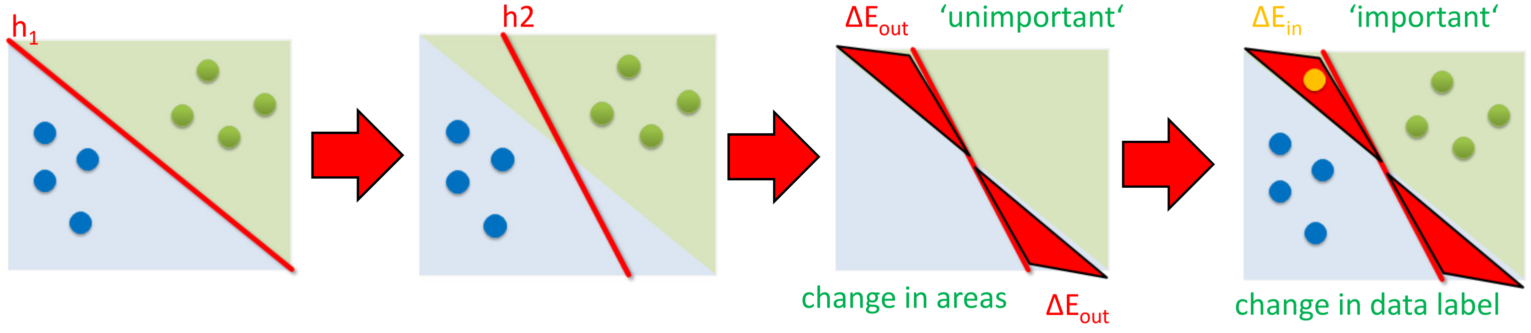
$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

takes no overlaps of M hypothesis into account

- Union bound is a 'poor bound', ignores correlation between h
  - Overlaps are common: the interest is shifted to data points changing label

$$|\ E_{in}(h_1) - E_{out}(h_1)\ | \approx\ \ |\ E_{in}(h_2) - E_{out}(h_2)\ |$$

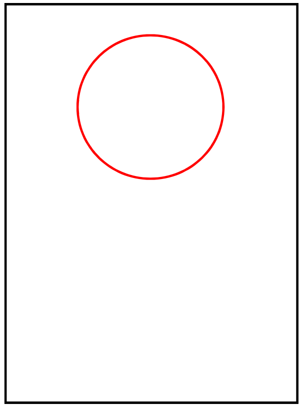
(at least very often, indicator to reduce M)



- Statistical Learning Theory provides a quantity able to characterize the overlaps for a better bound

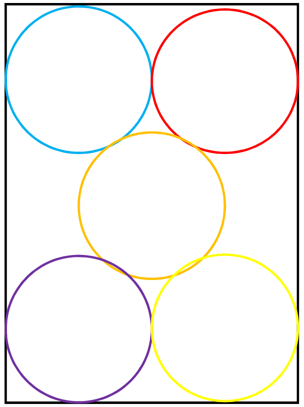
Replacing M & Large Overlaps

(Hoeffding Inequality)



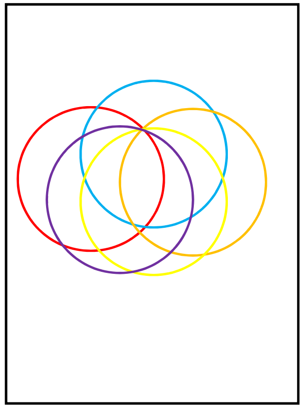
(valid for 1 hypothesis)

(Union Bound)



(valid for M hypothesis, worst case)

(towards Vapnik Chervonenkis Bound)



(valid for m (N) as growth function)

- Characterizing the overlaps is the idea of a ‘growth function‘
  - Number of dichotomies: Number of hypothesis but on finite number N of points

$$\mathbf{m}_{\mathcal{H}}(N) = \max_{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N} |\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N)|$$

  - Much redundancy: Many hypothesis will reports the same dichotomies

- **The mathematical proofs that m_H(N) can replace M is a key part of the theory of generalization**

Complexity of the Hypothesis Set – VC Inequality

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <=\ 2Me^{-2\epsilon^2 N}$$

$$\mathbf{m}_{\mathcal{H}}(N) = \max_{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N} |\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)|$$

- Vapnik-Chervonenkis (VC) Inequality
  - Result of mathematical proof when replacing M with growth function m
  - 2N of growth function to have another sample (2 x $E_{in}(h)$, no $E_{out}(h)$)

$$\Pr\ [\ |\ E_{in}(g) - E_{out}(g)\ | > \epsilon\ ]\ <=\ 4m_{\mathcal{H}}(2N)e^{-1/8\epsilon^2 N}$$

(characterization of generalization)

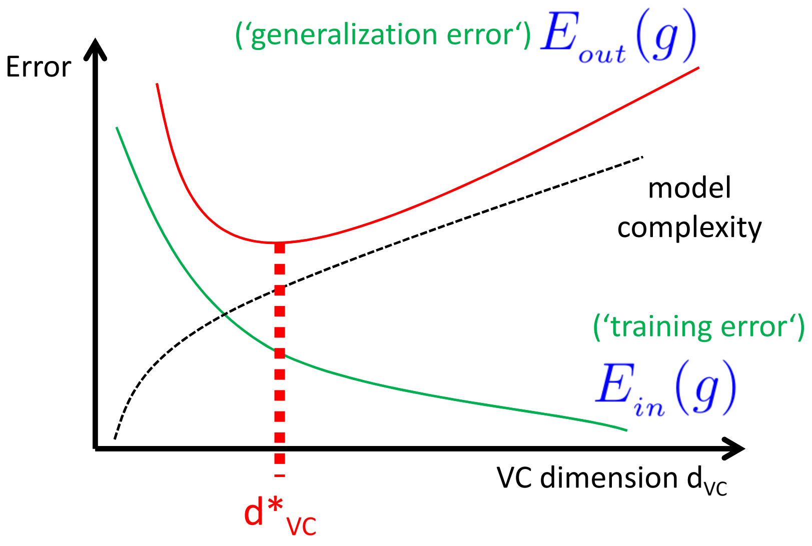
  - In Short – finally : We are able to learn and can generalize ‘ouf-of-sample‘

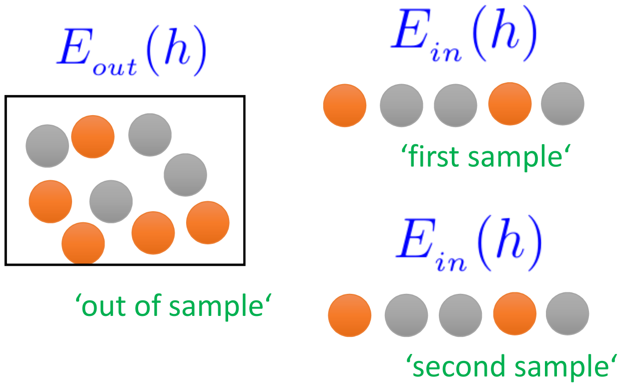
- **The Vapnik-Chervonenkis Inequality is the most important result in machine learning theory**
- **The mathematial proof brings us that M can be replaced by growth function (no infinity anymore)**
- **The growth function is dependent on the amount of data N that we have in a learning problem**

Complexity of the Hypothesis Set – VC Dimension & Model Complexity

- Vapnik-Chervonenkis (VC) Dimension over instance space X
  - VC dimension gets a ‘generalization bound‘ on all possible target functions

Issue: unknown to ‘compute‘ – VC solved this using the growth function on different samples





idea: ‘first sample‘ frequency close to ‘second sample‘ frequency

- **Complexity of Hypothesis set H can be measured by the Vapnik-Chervonenkis (VC) Dimension d_{VC}**
- **Ignoring the model complexity d_{VC} leads to situations where $E_{in}(g)$ gets down and $E_{out}(g)$ gets up**

Different Models – Hypothesis Set & Model Capacity

Hypothesis Set

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

$$\mathcal{H} = \{h_1, ..., h_m\};$$

(all candidate functions derived from models and their parameters)

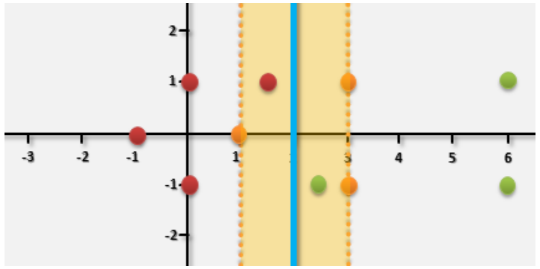
- Choosing from various model approaches $h_1, \ldots, h_m$ is a different hypothesis
- Additionally a change in model parameters of $h_1, \ldots, h_m$ means a different hypothesis too
- The model capacity characterized by the VC Dimension helps in choosing models
- Occam's Razor rule of thumb: 'simpler model better' in any learning problem, not too simple!

'select one function' that best approximates

Final Hypothesis

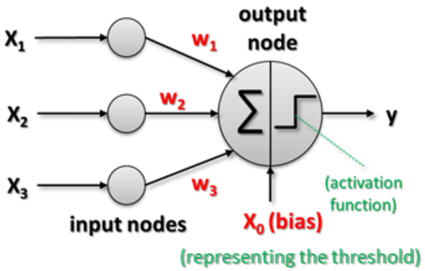
$$g \approx f$$

h_1



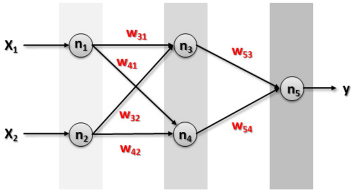
(e.g. support vector machine model)

h_2



(e.g. linear perceptron model)

h_m

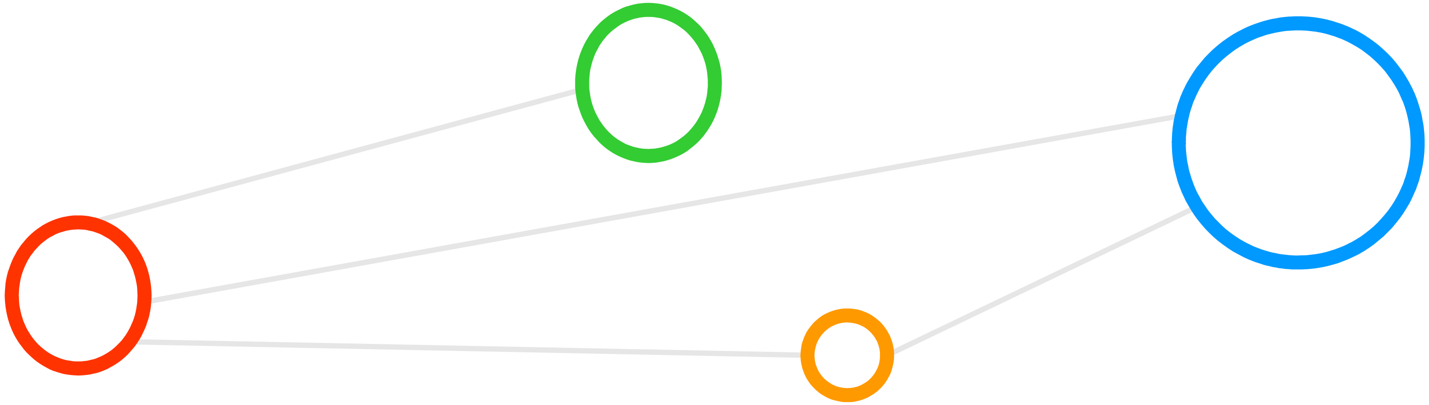


(e.g. artificial neural network model)

[Video] Prevent Overfitting for better Generalization

[2] YouTube Video, Stop Overfitting

Artificial Neural Networks & Backpropagation



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)

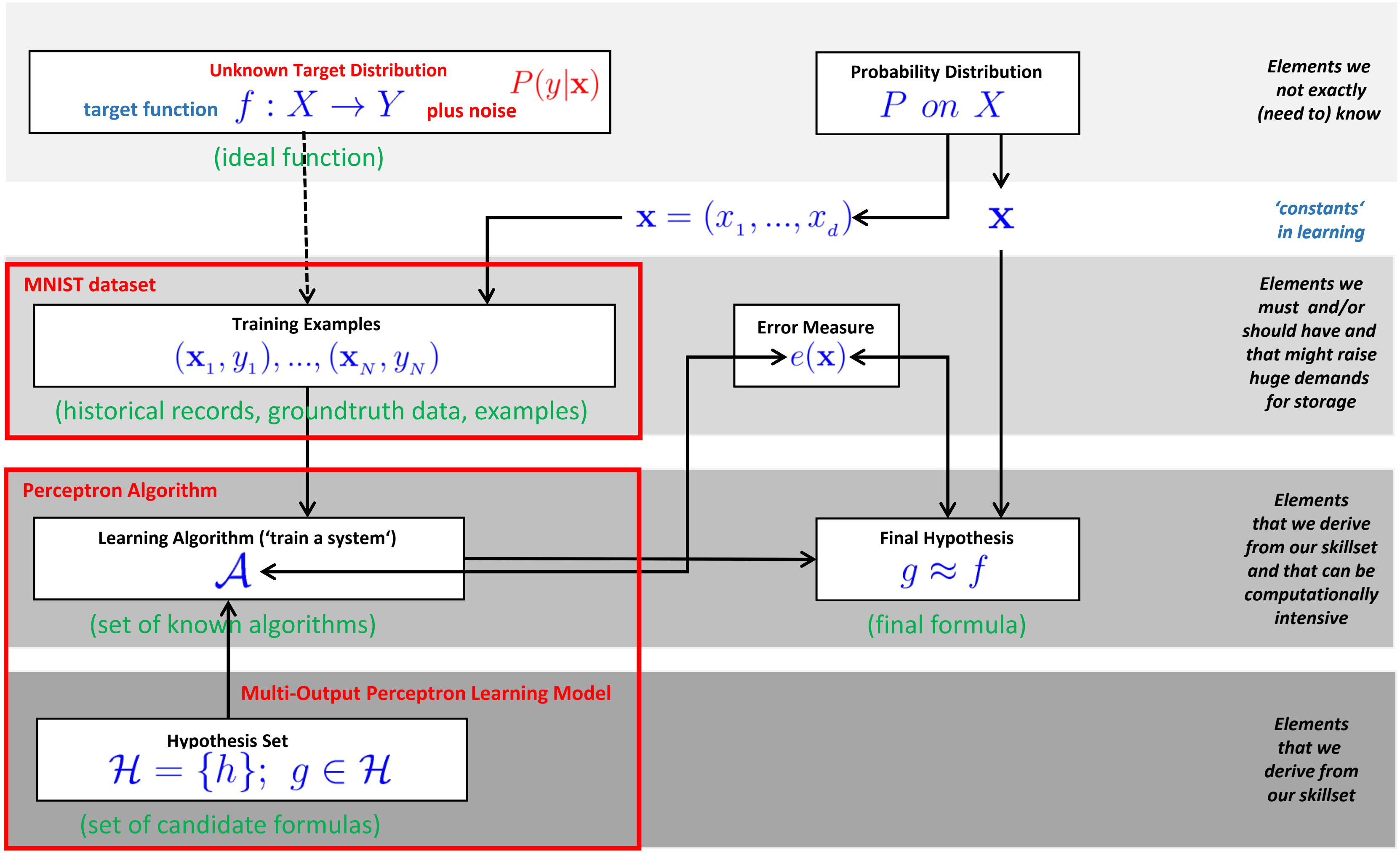
Model Evaluation – Testing Phase & Performance Metrics

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

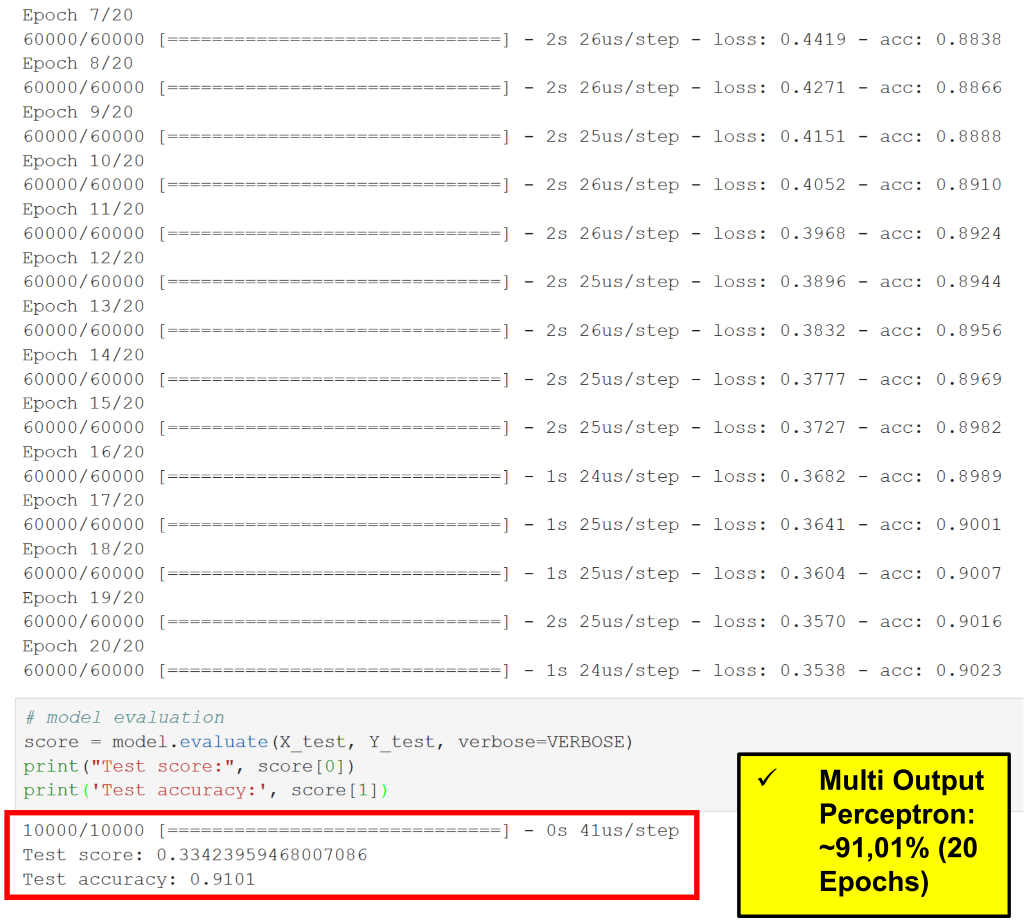
(100% accuracy in learning often points to problems using machine learning methos in practice)

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

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


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
Perceptron 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
Multi-Output Perceptron Learning Model
Hypothesis Set
$\mathcal{H} = \{h\};\ g \in \mathcal{H}$
(set of candidate formulas)
Elements that we derive from our skillset

MNIST Dataset – A Multi Output Perceptron Model – Revisited (cf. Lecture 3)

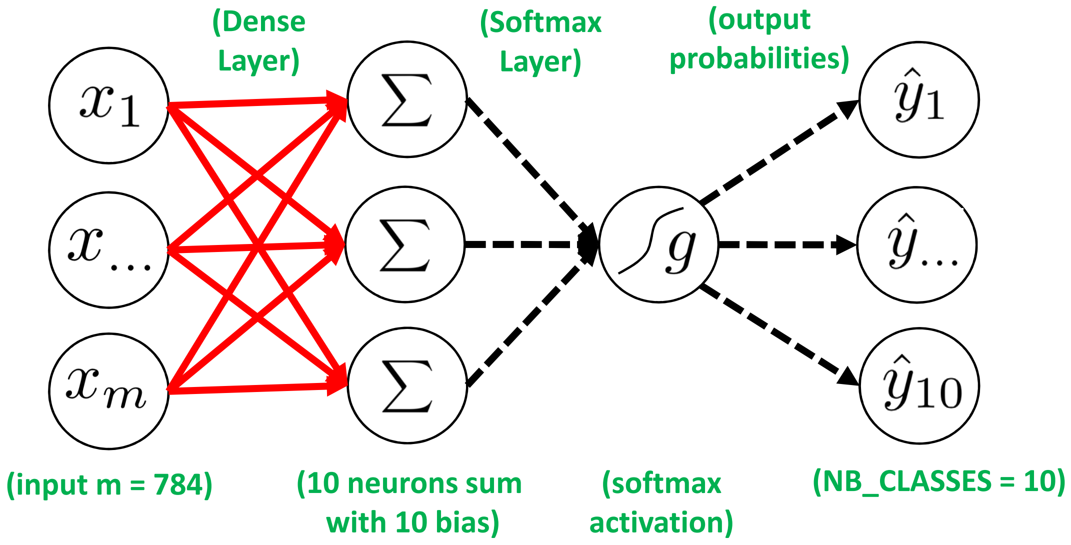


```
Epoch 7/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4419 - acc: 0.8838
Epoch 8/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4271 - acc: 0.8866
Epoch 9/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.4151 - acc: 0.8888
Epoch 10/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4052 - acc: 0.8910
Epoch 11/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3968 - acc: 0.8924
Epoch 12/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3896 - acc: 0.8944
Epoch 13/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3832 - acc: 0.8956
Epoch 14/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3777 - acc: 0.8969
Epoch 15/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3727 - acc: 0.8982
Epoch 16/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3682 - acc: 0.8989
Epoch 17/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.3641 - acc: 0.9001
Epoch 18/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.3604 - acc: 0.9007
Epoch 19/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3570 - acc: 0.9016
Epoch 20/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3538 - acc: 0.9023
```

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

```
10000/10000 [==============================] - 0s 41us/step
Test score: 0.33423959468007086
Test accuracy: 0.9101
```

✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**



(Dense Layer)

(Softmax Layer)

(output probabilities)

(input m = 784)

(10 neurons sum with 10 bias)

(softmax activation)

(NB_CLASSES = 10)

- **How to improve the model design by extending 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 (e.g. maybe ReLU)?**
- **Think Dense layer – Keras?**
- **Think about final Activation as Softmax → output probability**

Different Models – Hypothesis Set & Choosing a Model with more Capacity

Hypothesis Set

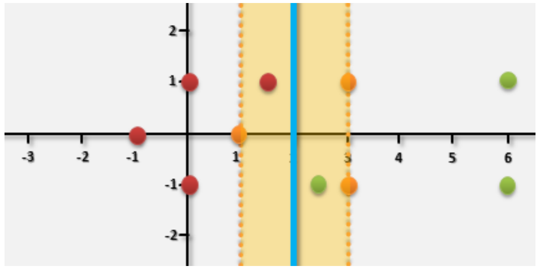
$\mathcal{H} = \{h\};\ g \in \mathcal{H}$

$$\mathcal{H} = \{h_1, \ldots, h_m\};$$

(all candidate functions derived from models and their parameters)

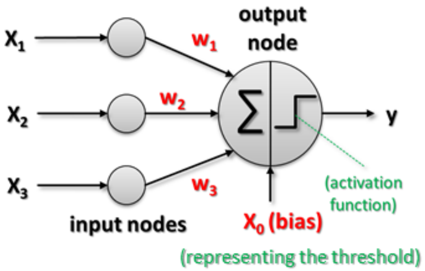
- Choosing from various model approaches $h_1, \ldots, h_m$ is a different hypothesis
- Additionally a change in model parameters of $h_1, \ldots, h_m$ means a different hypothesis too
- The model capacity characterized by the VC Dimension helps in choosing models
- Occam‘s Razor rule of thumb: ‘simpler model better‘ in any learning problem, not too simple!

h_1



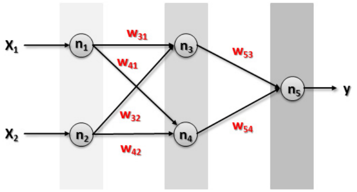
(e.g. support vector machine model)

h_2



(e.g. linear perceptron model)

h_m



(e.g. artificial neural network model)

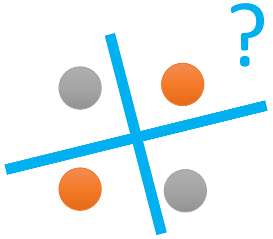
‘select one function‘ that best approximates

Final Hypothesis

$g \approx f$

Artificial Neural Network (ANN)

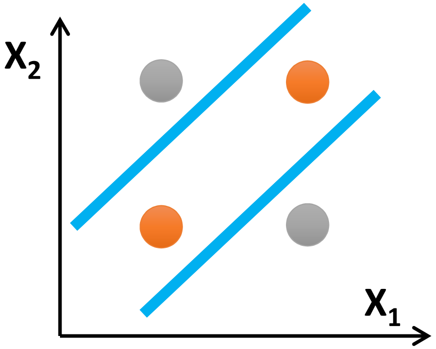
- Simple perceptrons fail: ‘not linearly seperable’



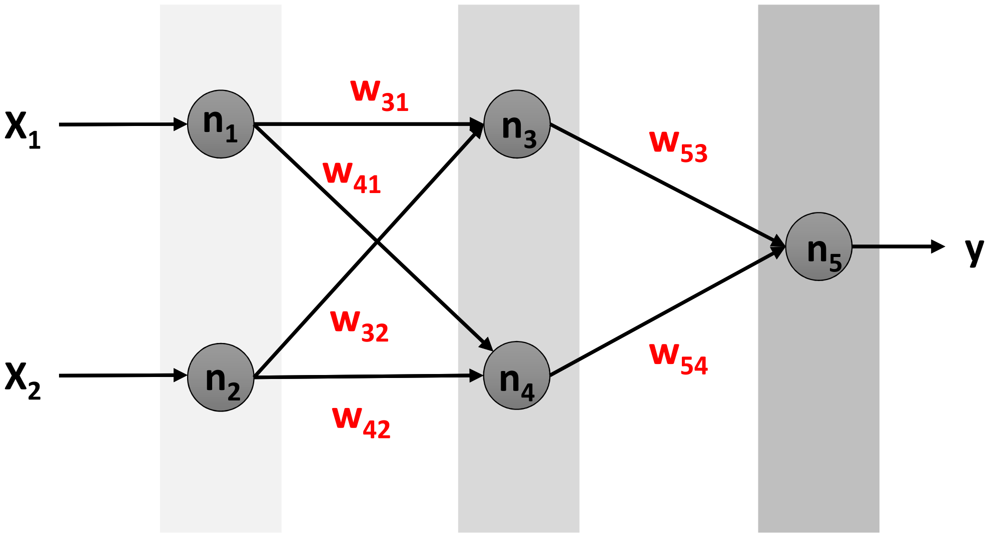
X_1	X_2	Y
0	*0*	*-1*
1	*0*	*1*
0	*1*	*1*
1	*1*	*-1*

Labelled Data Table

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

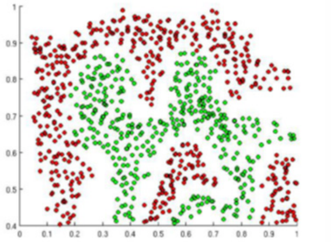


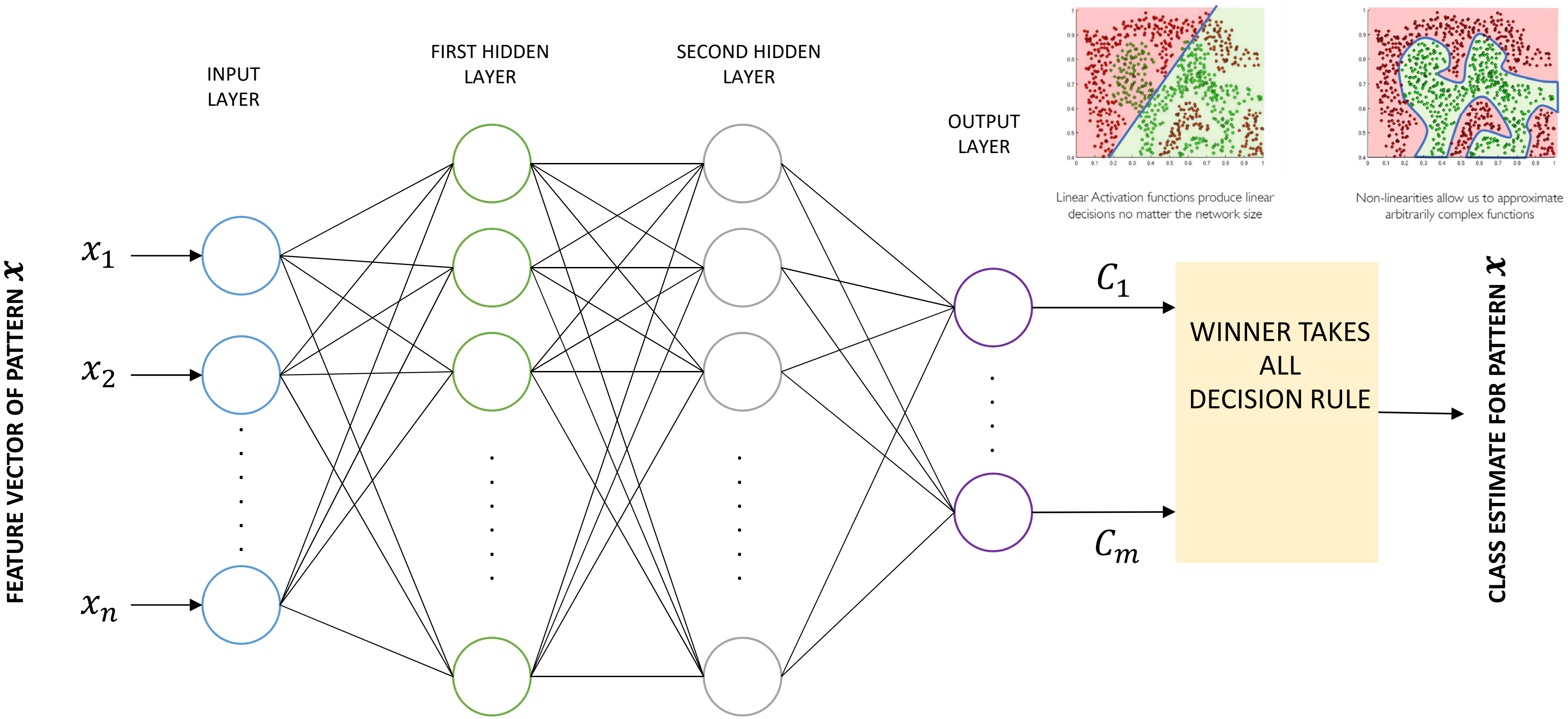
Decision Boundary



Two-Layer, feed-forward Artificial Neural Network topology

Multi-Layer Perceptron (MLP) using Non-linearities




INPUT LAYER
FIRST HIDDEN LAYER
SECOND HIDDEN LAYER
OUTPUT LAYER
FEATURE VECTOR OF PATTERN x
x_1
x_2
x_n
C_1
C_m
WINNER TAKES ALL DECISION RULE
CLASS ESTIMATE FOR PATTERN x

Linear Activation functions produce linear decisions no matter the network size

Non-linearities allow us to approximate arbitrarily complex functions

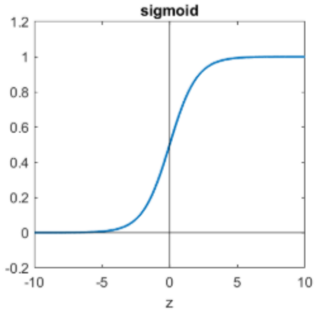
- **Forward interconnection of several layers of perceptrons**
- **MLPs can be used as universal approximators**
- **In classification problems, they allow modeling nonlinear discriminant functions**
- **Interconnecting neurons aims at increasing the capability of modeling complex input-output relationships**

[8] MIT Deep Learning

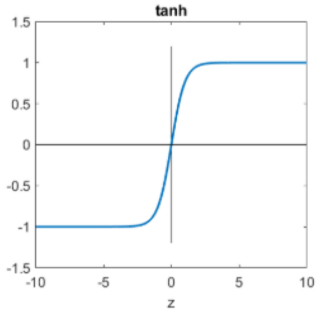
Activation Functions to Choose From

- Facts
  - The choice of the architecture and the activation function plays a key role in the definition of the network
  - Each activation function takes a single number and performs a certain fixed mathematical operation on it

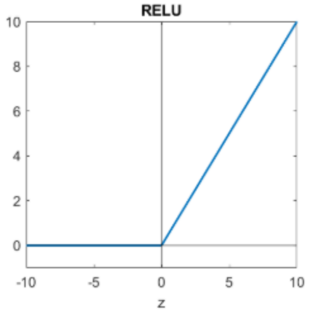
[9] Understanding Neural Networks



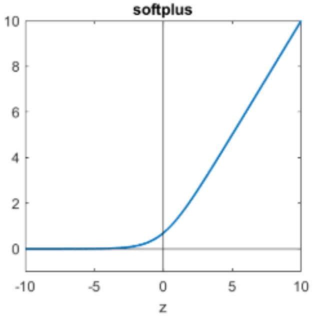
$$h(z) = \frac{1}{1 + e^{-z}}$$



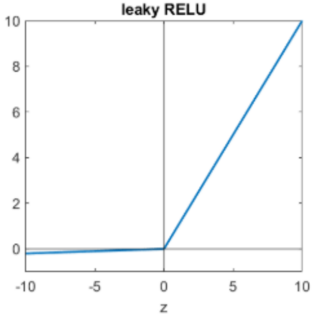
$$h(z) = \tanh z$$



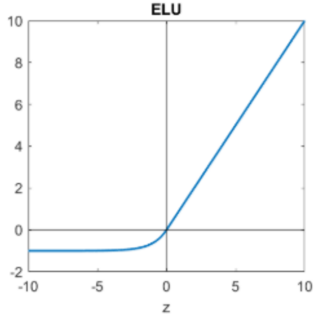
$$h(z) = \max(z, 0)$$



$$h(z) = \log(1 + e^{z})$$

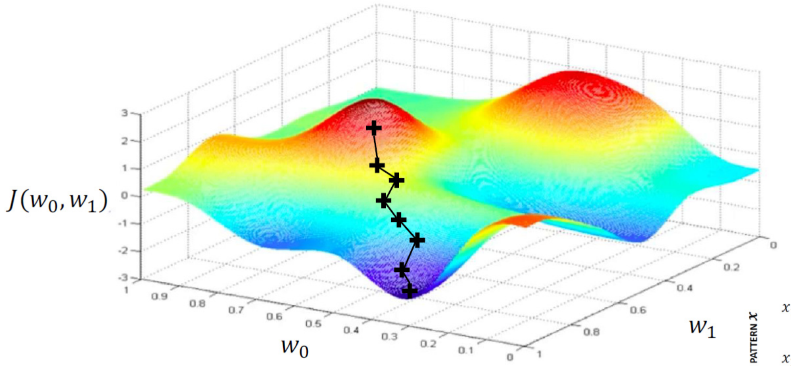


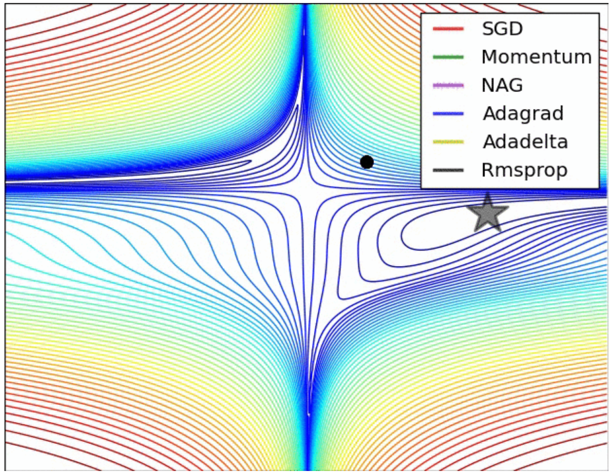
$$h(z) = \max(\mathrm{z}, \mathrm{z}\alpha)$$
$$0 < \alpha < 1$$

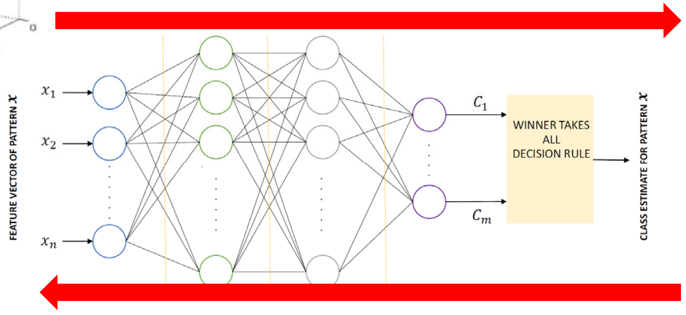


$$h(z) = \begin{cases} z, z > 0 \\ \alpha(e^{z} - 1)z \leq 0 \end{cases}$$

Backpropagation Algorithm using Optimization







FEATURE VECTOR OF PATTERN x
x_1
x_2
x_n
C_1
C_m
WINNER TAKES ALL DECISION RULE
CLASS ESTIMATE FOR PATTERN x

```
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3.     Pick batch of B data points
4.     Compute gradient $\frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5.     Update weights $\boldsymbol{W} := \boldsymbol{W} - \eta \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights
```

TOTAL NUMBER OF TRAINING SAMPLES

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - d_i)^2$$

OUTPUT VALUE OBTAINED BY THE MLP FOR THE i-th SAMPLE

DESIRED OUTPUT (TARGET) VALUE FOR THE i-th SAMPLE

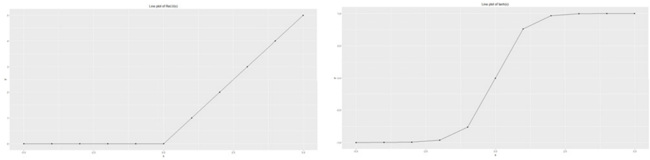
MNIST Dataset – Add Two Hidden Layers for Artificial Neural Network (ANN)

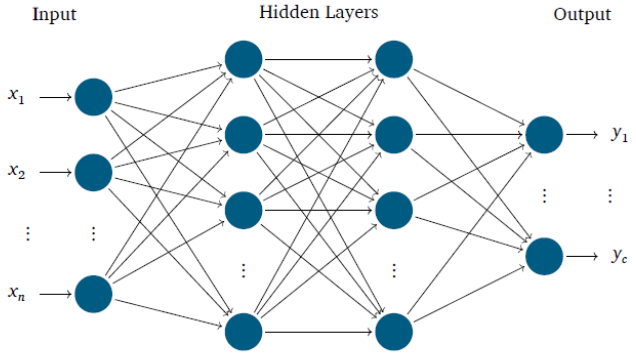
- All parameter value remain the same as before
- We add N_HIDDEN as parameter in order to set 128 neurons in one hidden layer – this number is a hyperparameter that is not directly defined and needs to be find with parameter search

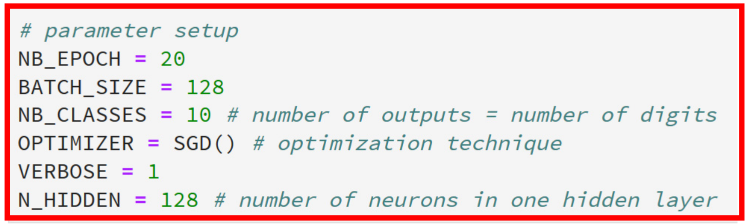
[3] big-data.tips, ‘Relu Neural Network’

[4] big-data.tips, ‘tanh’

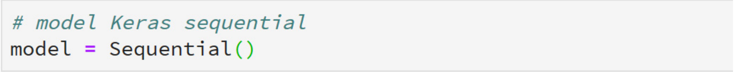
(activation functions ReLU & Tanh)



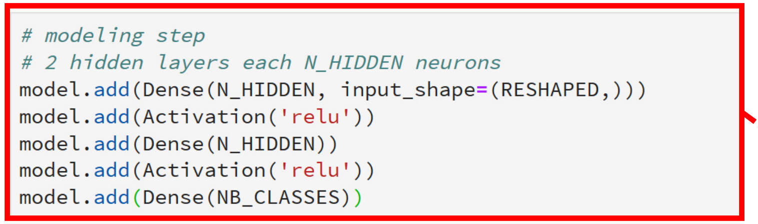




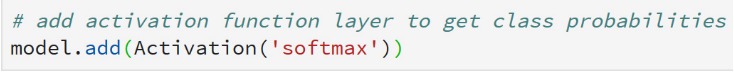
```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
N_HIDDEN = 128 # number of neurons in one hidden layer
```



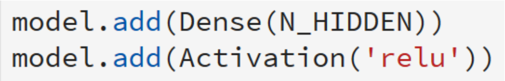
```
# model Keras sequential
model = Sequential()
```



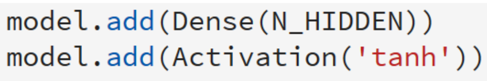
```
# modeling step
# 2 hidden layers each N_HIDDEN neurons
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
```



```
# add activation function layer to get class probabilities
model.add(Activation('softmax'))
```

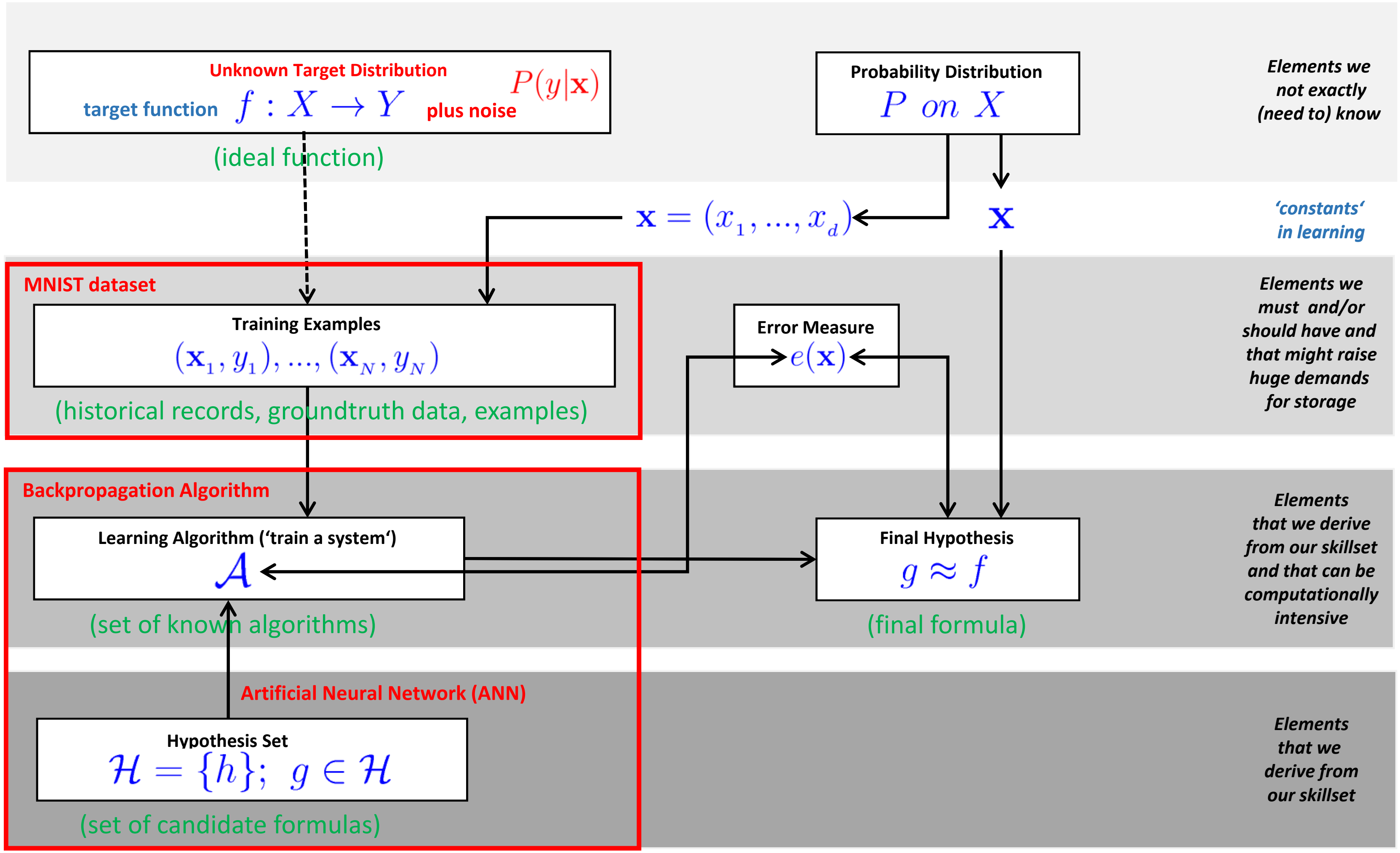


```
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
```



```
model.add(Dense(N_HIDDEN))
model.add(Activation('tanh'))
```

- 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
- 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



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

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

$\mathbf{x}$

'constants' in learning

MNIST dataset

Training Examples

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

(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)

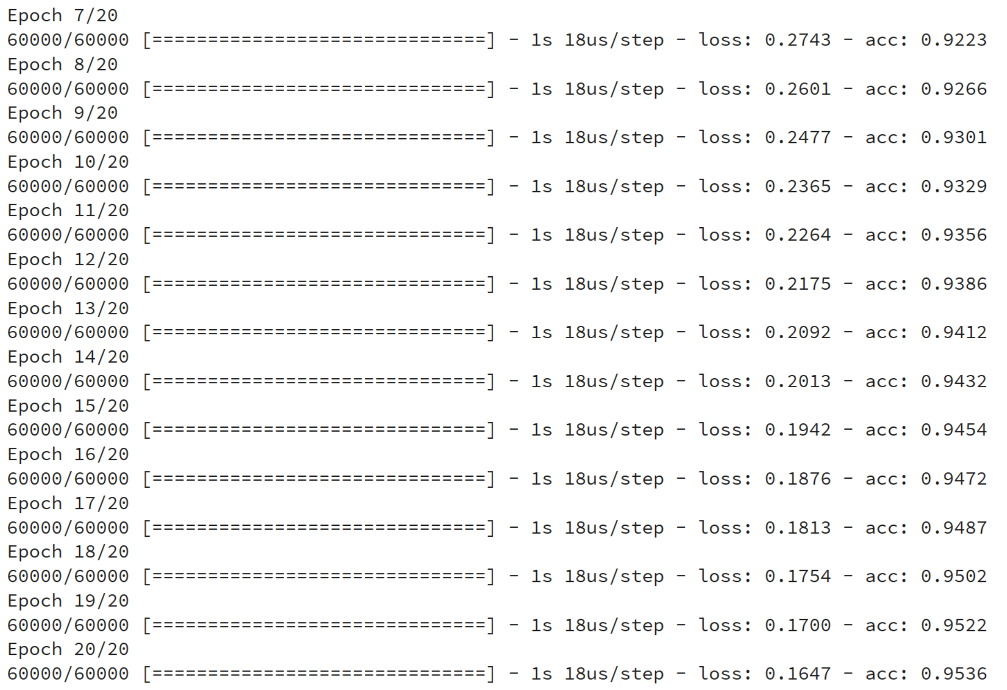
Hypothesis Set

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

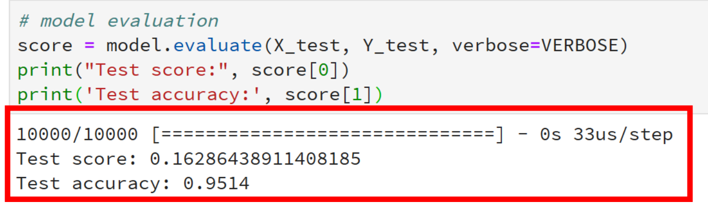
(set of candidate formulas)

Elements that we derive from our skillset

MNIST Dataset – ANN Model Parameters & Output Evaluation



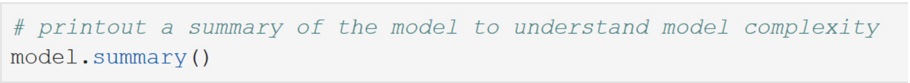
```
Epoch 7/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2743 - acc: 0.9223
Epoch 8/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2601 - acc: 0.9266
Epoch 9/20
60000/60000 [==============================] - 1s 18us/step - loss: 0.2477 - acc: 0.9301
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
```



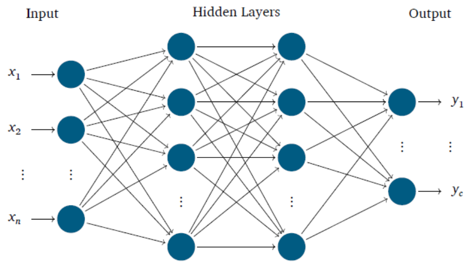
```
# model evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

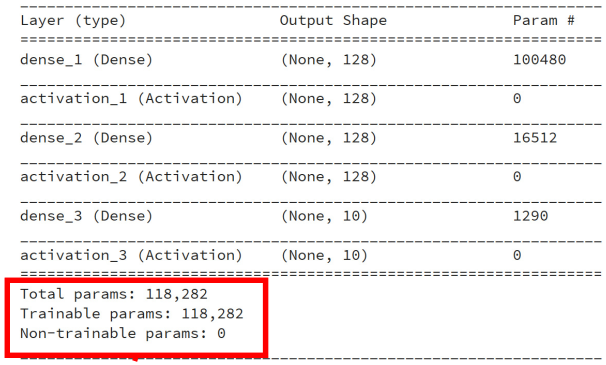
```
10000/10000 [==============================] - 0s 33us/step
Test score: 0.16286438911408185
Test accuracy: 0.9514
```

- ✓ 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()
```



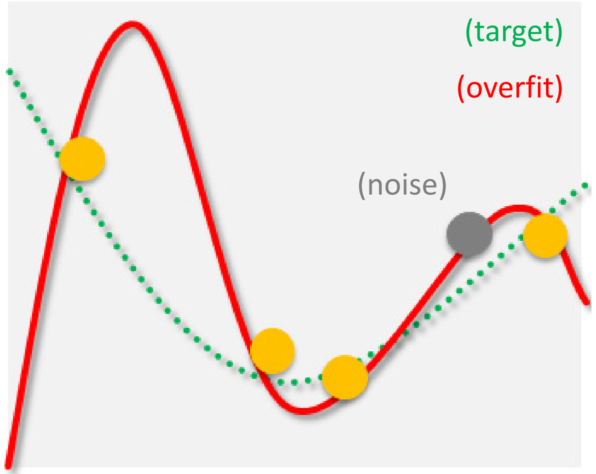


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
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)

Machine Learning Challenges – Problem of Overfitting

- 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 $E_{out}(g)$

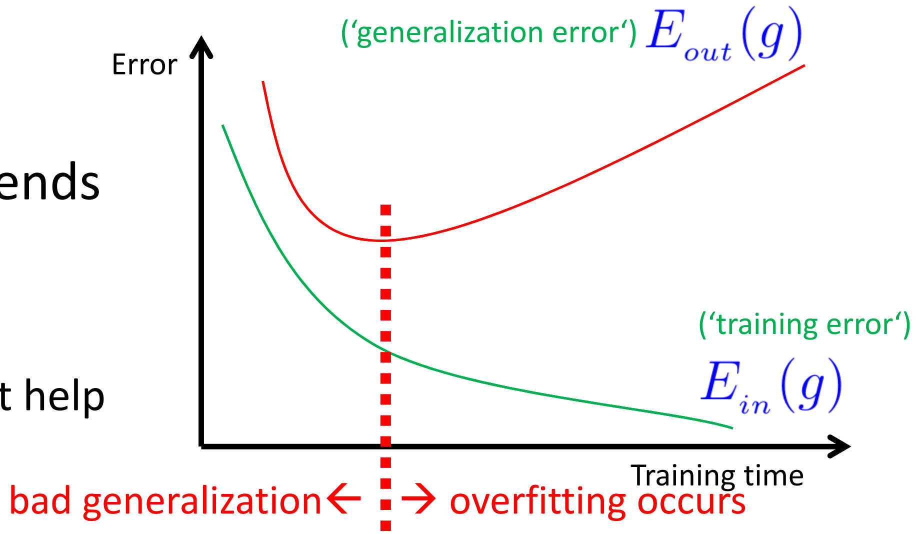


(but simple polynomial works good enough)
(‘over‘: here meant as 4th order,
a 3rd order would be better, 2nd best)

- **Overfitting 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**

Problem of Overfitting – Clarifying Terms

- 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'

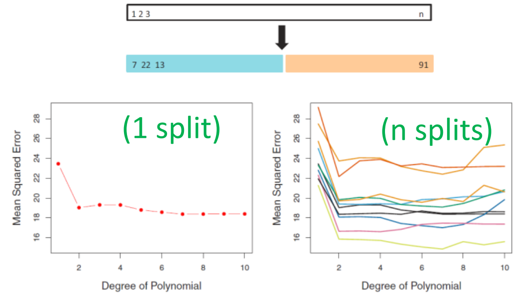


- A good model must have low training error (E_{in}) and low generalization error (E_{out})
- Model overfitting is if a model fits the data too well (E_{in}) with a poorer generalization error (E_{out}) than another model with a higher training error (E_{in})
- The two general approaches to prevent overfitting are (1) validation and (2) regularization

Validation & Model Selection – Terminology

- ‘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)



- **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**

Validation Technique – Formalization & Goal

- 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)

$$\underset{\substack{\uparrow \\ \text{(validation estimates this quantity)}}}{E_{out}(h)} = E_{in}(h) + \underset{\substack{\uparrow \\ \text{(regularization estimates this quantity)}}}{\textbf{overfit penalty}}$$

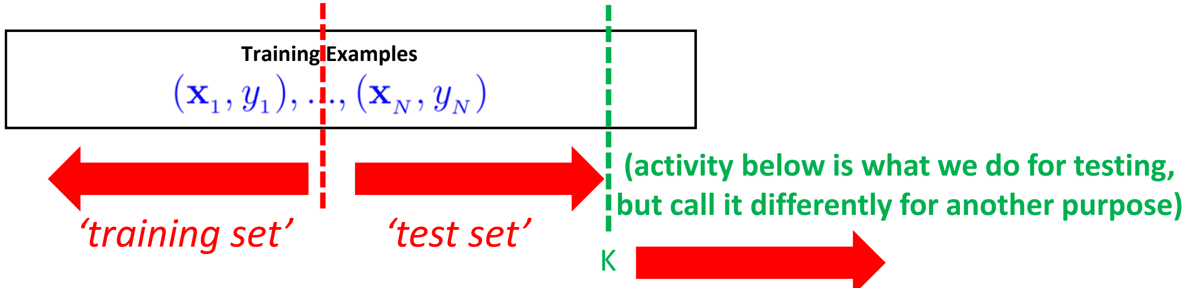
(minimize both to be better proxy for E_{out})

(measuring E_{out} is not possible as this is an unknown quantity, another quantity is needed that is measurable that at least estimates it)

- Validation
  - 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 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

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



(activity below is what we do for testing, but call it differently for another purpose)

(involved in validation)

- 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')

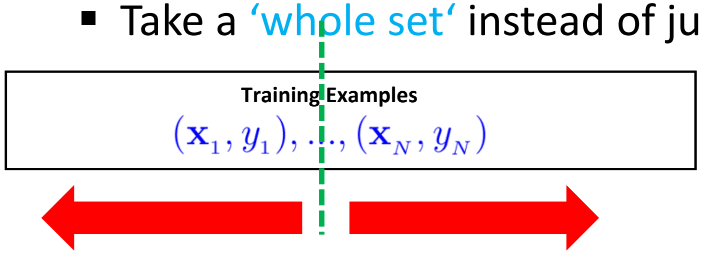
$$\mathbb{E}[e(h(\mathbf{x}), y)] = E_{out}(h)$$

(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

- 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



(involved in training+test) K (involved in 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

$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_K, y_K)$ (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)

- ‘Reliable estimate‘ if K is large $\mathbb{E}[E_{val}(h)] = \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}[e(h(\mathbf{x})_k, y_k)] = E_{out}$

(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)

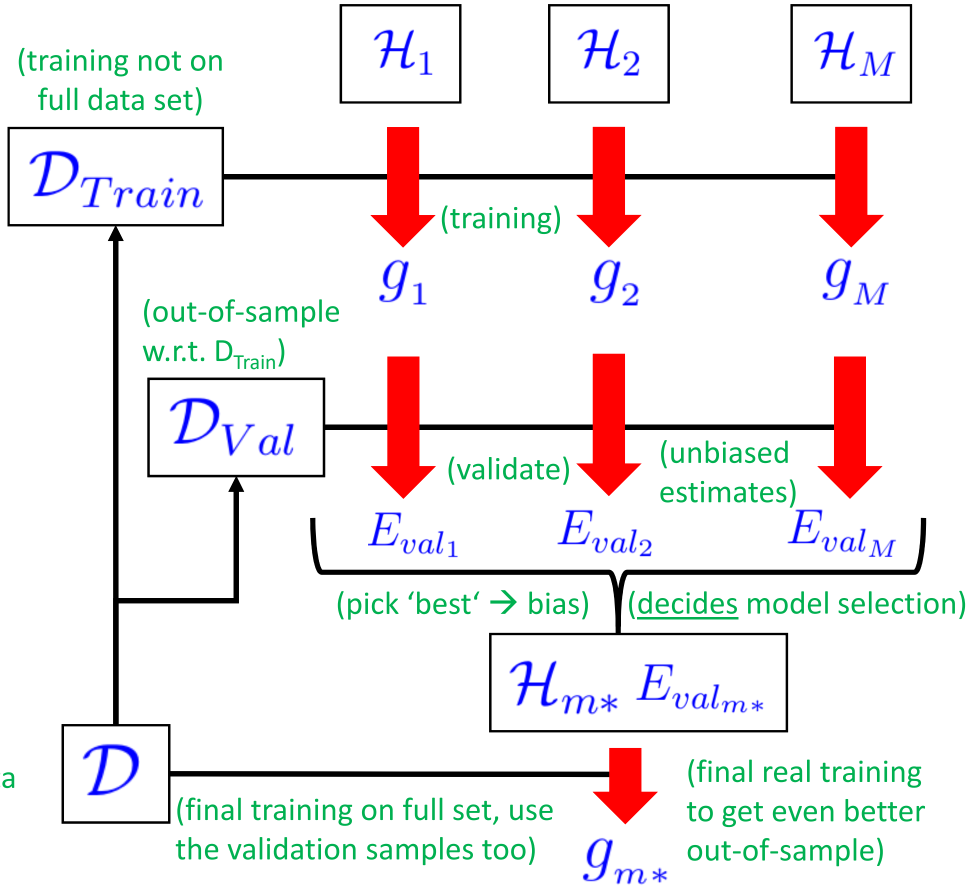
- Validation set consists of data that has been not used in training to estimate true out-of-sample
- Rule of thumb from practice is to take 20% (1/5) for validation of the learning model

Validation Technique – Model Selection Process

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



(training not on full data set)
$\mathcal{H}_1$
$\mathcal{H}_2$
$\mathcal{H}_M$
$\mathcal{D}_{Train}$
(training)
g_1
g_2
g_M
(out-of-sample w.r.t. D_{Train})
$\mathcal{D}_{Val}$
(validate)
(unbiased estimates)
E_{val_1}
E_{val_2}
E_{val_M}
(pick ‘best‘ → bias)
(decides model selection)
$\mathcal{H}_{m*}\ E_{val_{m*}}$
$\mathcal{D}$
(final training on full set, use the validation samples too)
g_{m*}
(final real training to get even better out-of-sample)

- **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‘**

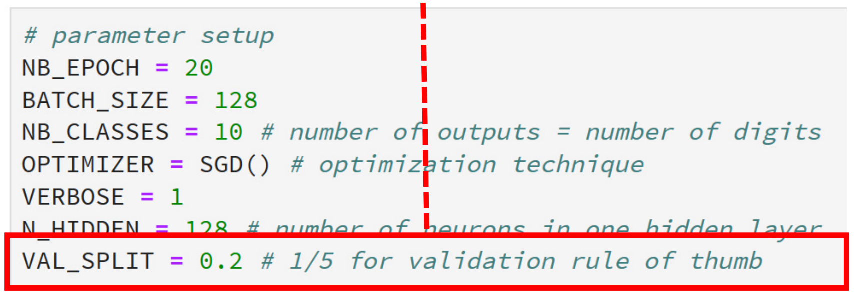
Final Hypothesis
$g_{m*} \approx f$

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

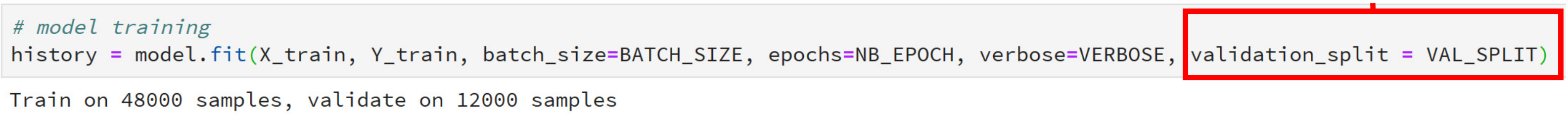
ANN 2 Hidden 1/5 Validation – MNIST Dataset

- 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)

- 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.



```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
N_HIDDEN = 128 # number of neurons in one hidden layer
VAL_SPLIT = 0.2 # 1/5 for validation rule of thumb
```

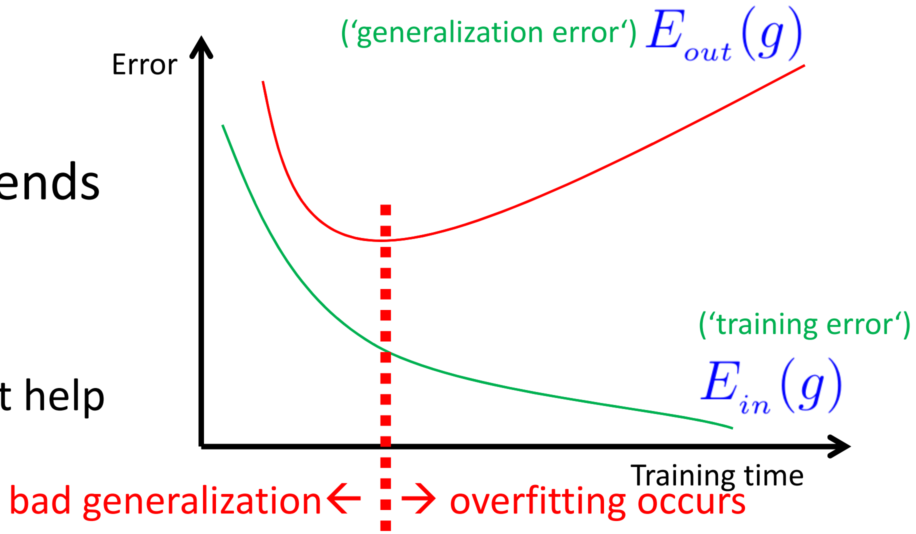


```
# model training
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=NB_EPOCH, verbose=VERBOSE, validation_split = VAL_SPLIT)
```

```
Train on 48000 samples, validate on 12000 samples
```

Problem of Overfitting – Clarifying Terms – Revisited

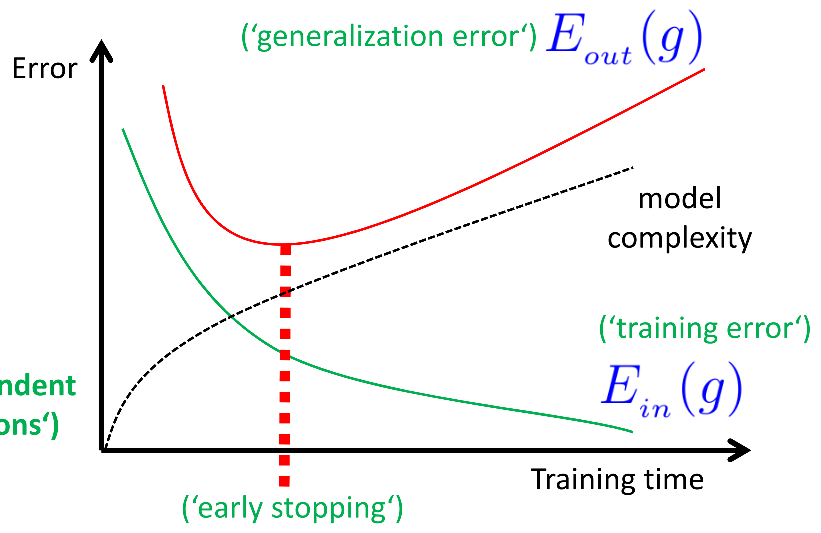
- Overfitting & Errors
  - $E_{in}(g)$ goes down
  - $E_{out}(g)$ goes up
- ‘Bad generalization area‘ ends
  - Good to reduce $E_{in}(g)$
- ‘Overfitting area‘ starts
  - Reducing $E_{in}(g)$ does not help
  - Reason ‘fitting the noise‘



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

Problem of Overfitting – Model Relationships

- Review ‘overfitting situations‘
  - When comparing ‘various models‘ and related to ‘model complexity‘
  - Different models are used, e.g. 2nd and 4th 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

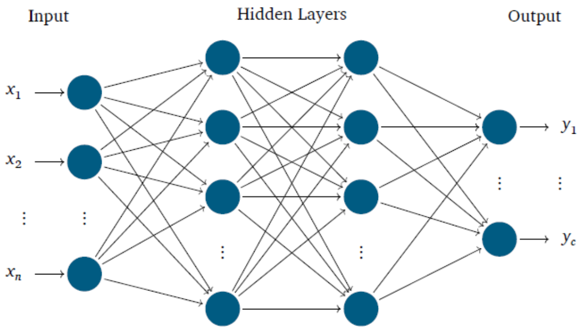


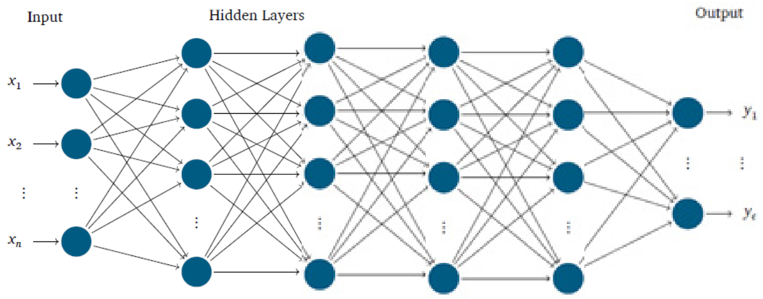
(‘model complexity measure: the VC analysis was independent of a specific target function – bound for all target functions‘)

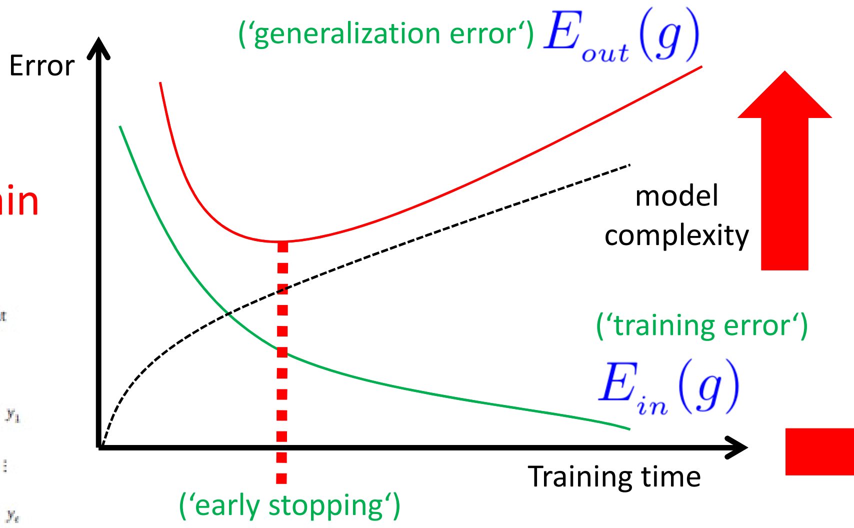
- **‘Early stopping‘ approach is part of the theory of regularization, but based on validation methods**

Problem of Overfitting – ANN Model Example possible towards 99% Accuracy?

- 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



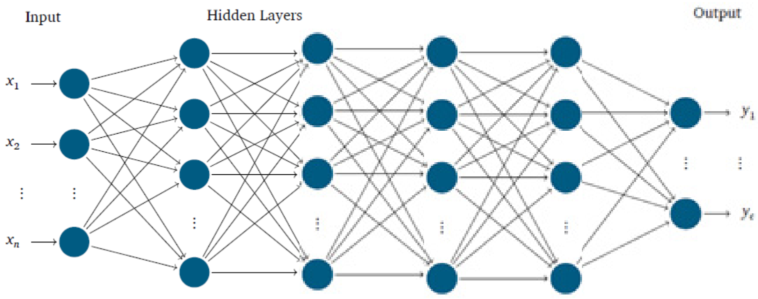


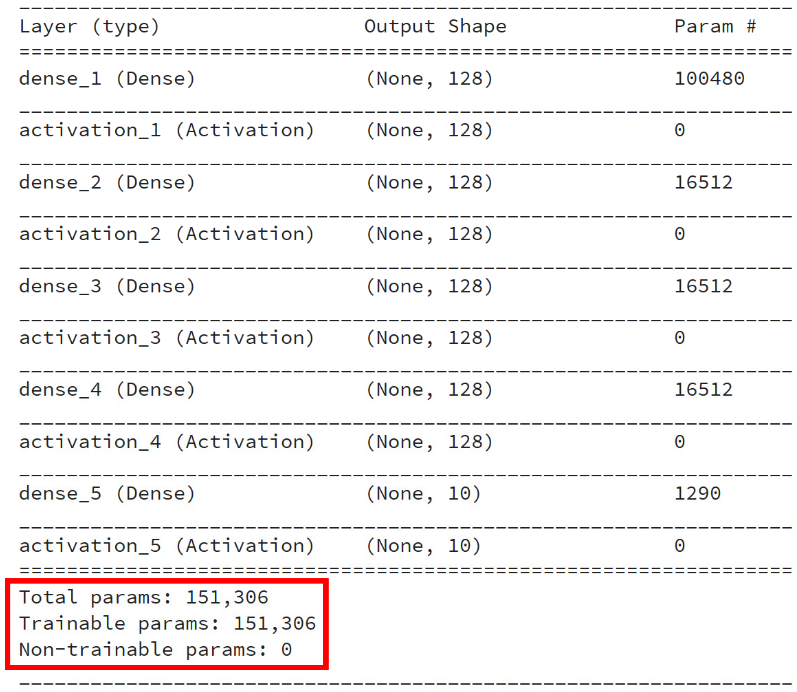


- 1st possible Change: Adding more layers means more model complexity
- 2nd possible change: Longer training time to enable better learning
- Questions remains: will it be useful to get towards 99% accuracy?

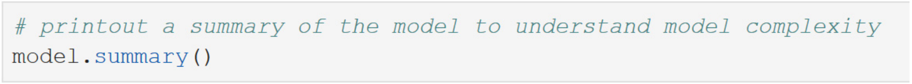
MNIST Dataset & Model Summary & Parameters

- Four Hidden Layers
  - Each hidden layers has 128 neurons



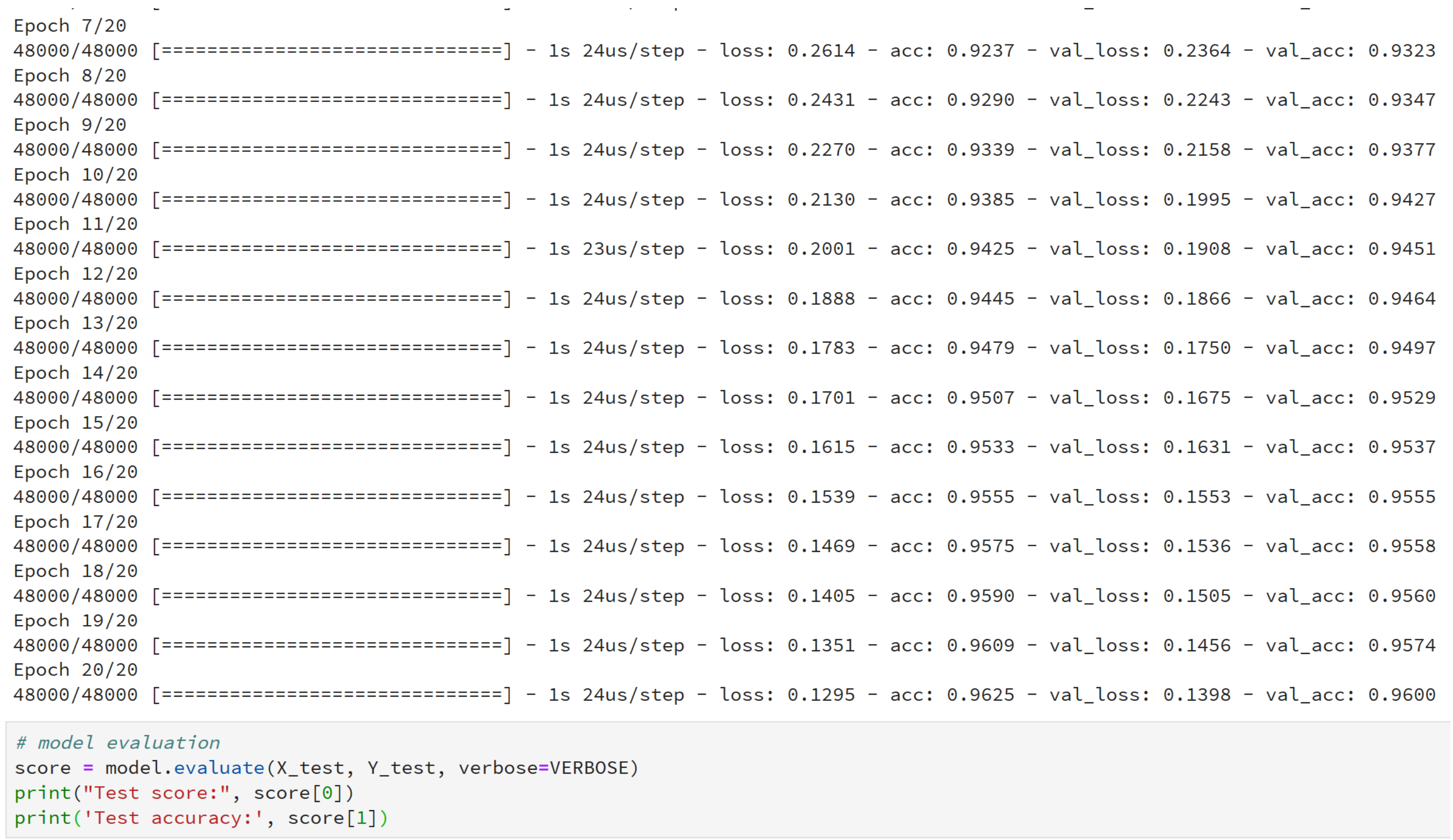


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
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
_________________________________________________________________
```



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

Exercises - Add more Hidden Layers – 4 Hidden Layers



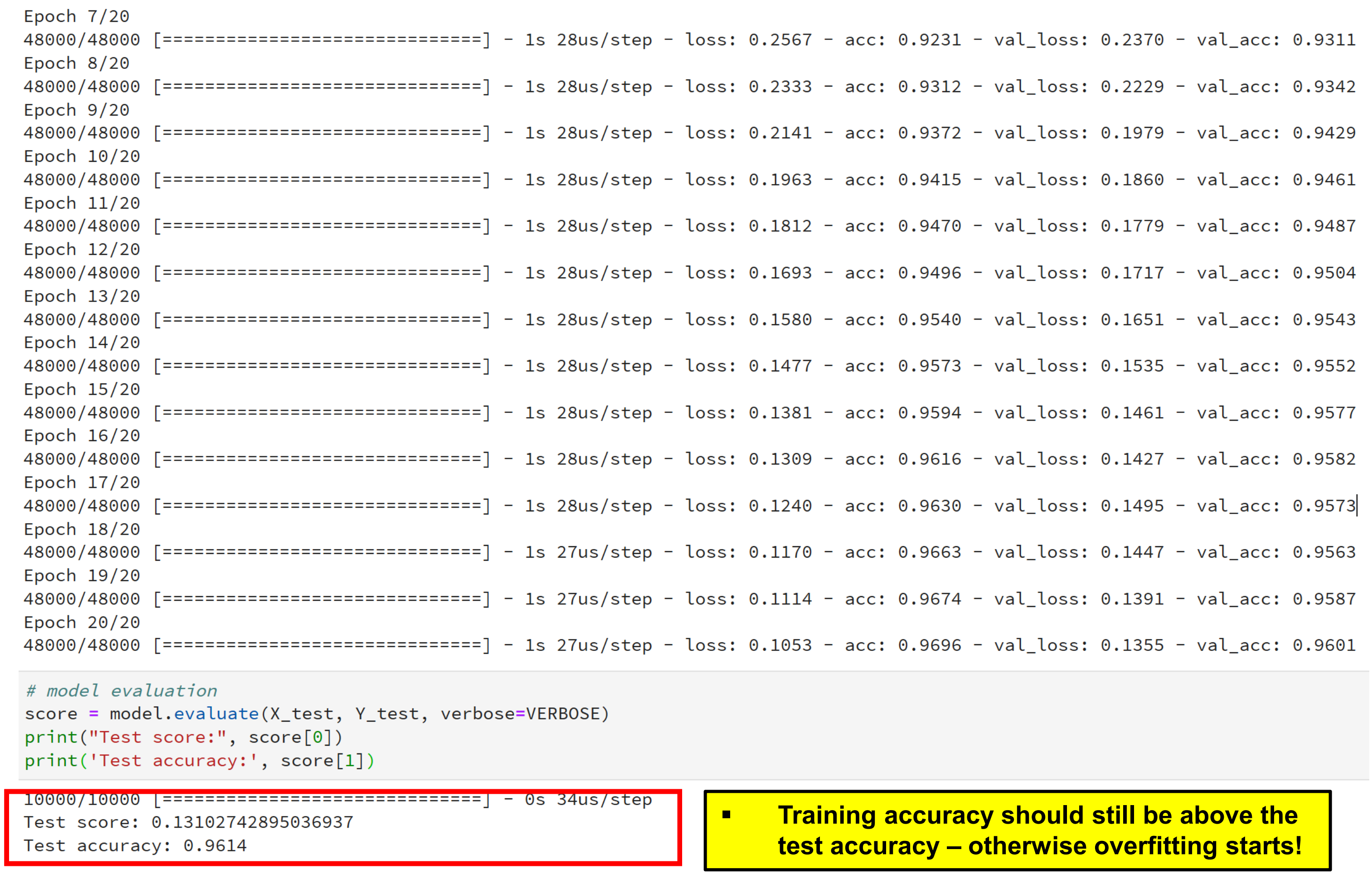
```
Epoch 7/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.2614 - acc: 0.9237 - val_loss: 0.2364 - val_acc: 0.9323
Epoch 8/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.2431 - acc: 0.9290 - val_loss: 0.2243 - val_acc: 0.9347
Epoch 9/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.2270 - acc: 0.9339 - val_loss: 0.2158 - val_acc: 0.9377
Epoch 10/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.2130 - acc: 0.9385 - val_loss: 0.1995 - val_acc: 0.9427
Epoch 11/20
48000/48000 [==============================] - 1s 23us/step - loss: 0.2001 - acc: 0.9425 - val_loss: 0.1908 - val_acc: 0.9451
Epoch 12/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1888 - acc: 0.9445 - val_loss: 0.1866 - val_acc: 0.9464
Epoch 13/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1783 - acc: 0.9479 - val_loss: 0.1750 - val_acc: 0.9497
Epoch 14/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1701 - acc: 0.9507 - val_loss: 0.1675 - val_acc: 0.9529
Epoch 15/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1615 - acc: 0.9533 - val_loss: 0.1631 - val_acc: 0.9537
Epoch 16/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1539 - acc: 0.9555 - val_loss: 0.1553 - val_acc: 0.9555
Epoch 17/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1469 - acc: 0.9575 - val_loss: 0.1536 - val_acc: 0.9558
Epoch 18/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1405 - acc: 0.9590 - val_loss: 0.1505 - val_acc: 0.9560
Epoch 19/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1351 - acc: 0.9609 - val_loss: 0.1456 - val_acc: 0.9574
Epoch 20/20
48000/48000 [==============================] - 1s 24us/step - loss: 0.1295 - acc: 0.9625 - val_loss: 0.1398 - val_acc: 0.9600
```

```python
# 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: 0.13893915132246912
Test accuracy: 0.9571
```

- **Training accuracy should still be above the test accuracy – otherwise overfitting starts!**

Exercises - Add more Hidden Layers – 6 Hidden Layers



```
Epoch 7/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.2567 - acc: 0.9231 - val_loss: 0.2370 - val_acc: 0.9311
Epoch 8/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.2333 - acc: 0.9312 - val_loss: 0.2229 - val_acc: 0.9342
Epoch 9/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.2141 - acc: 0.9372 - val_loss: 0.1979 - val_acc: 0.9429
Epoch 10/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1963 - acc: 0.9415 - val_loss: 0.1860 - val_acc: 0.9461
Epoch 11/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1812 - acc: 0.9470 - val_loss: 0.1779 - val_acc: 0.9487
Epoch 12/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1693 - acc: 0.9496 - val_loss: 0.1717 - val_acc: 0.9504
Epoch 13/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1580 - acc: 0.9540 - val_loss: 0.1651 - val_acc: 0.9543
Epoch 14/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1477 - acc: 0.9573 - val_loss: 0.1535 - val_acc: 0.9552
Epoch 15/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1381 - acc: 0.9594 - val_loss: 0.1461 - val_acc: 0.9577
Epoch 16/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1309 - acc: 0.9616 - val_loss: 0.1427 - val_acc: 0.9582
Epoch 17/20
48000/48000 [==============================] - 1s 28us/step - loss: 0.1240 - acc: 0.9630 - val_loss: 0.1495 - val_acc: 0.9573
Epoch 18/20
48000/48000 [==============================] - 1s 27us/step - loss: 0.1170 - acc: 0.9663 - val_loss: 0.1447 - val_acc: 0.9563
Epoch 19/20
48000/48000 [==============================] - 1s 27us/step - loss: 0.1114 - acc: 0.9674 - val_loss: 0.1391 - val_acc: 0.9587
Epoch 20/20
48000/48000 [==============================] - 1s 27us/step - loss: 0.1053 - acc: 0.9696 - val_loss: 0.1355 - val_acc: 0.9601
```

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

```
10000/10000 [==============================] - 0s 34us/step
Test score: 0.13102742895036937
Test accuracy: 0.9614
```

- **Training accuracy should still be above the test accuracy – otherwise overfitting starts!**

Problem of Overfitting – Noise Term Revisited

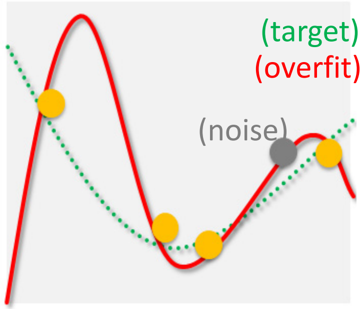
- ‘(Noisy) Target function‘ is not a (deterministic) function
  - Getting with ‘same x in‘ the ‘same y out‘ is not always given in practice
  - Idea: Use a ‘target distribution‘ instead of ‘target function‘

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‘

(‘function view‘)

‘shift the view‘

(‘# data view‘)

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'

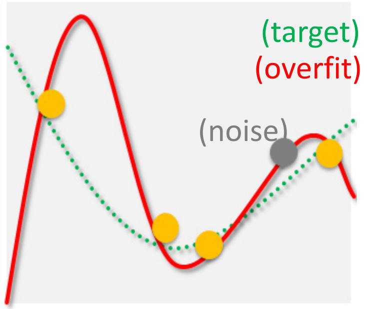
Unknown Target Distribution $P(y|\mathbf{x})$

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

(ideal function)

- 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

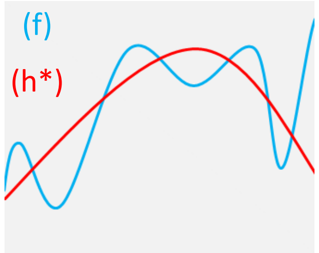
- Practice Example
  - Random fluctuations and/or measurement errors in data
  - Fitting a pattern that not exists 'out-of-sample'
  - Puts learning progress 'off-track' and 'away from f'



Problem of Overfitting – Deterministic Noise

- 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'

- 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*)

Problem of Overfitting – Impacts on Learning

- 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‘)

- **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**

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)
```

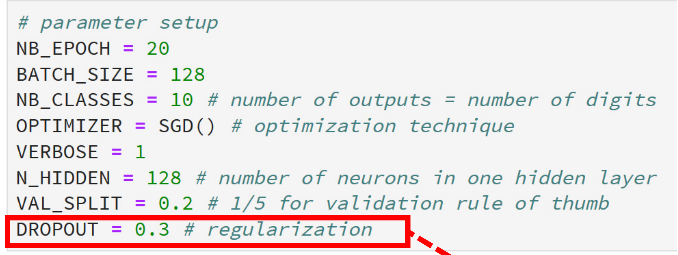
- Dropout is randomly setting a fraction of input units to 0 at each update during training time, which helps prevent overfitting (using parameter rate)

```
from keras import regularizers

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

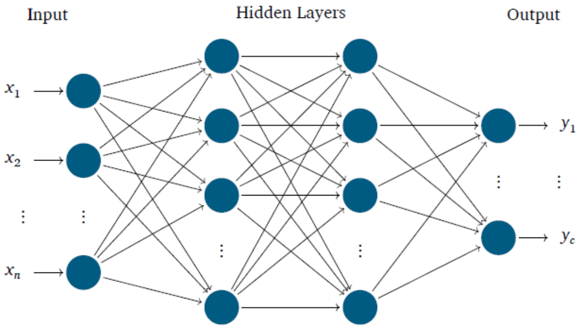
- L2 regularizers allow to apply penalties on layer parameter or layer activity during optimization itself – therefore the penalties are incorporated in the loss function during optimization

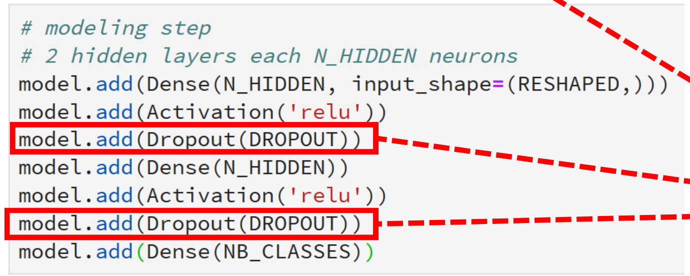
K Keras

ANN – MNIST Dataset – Add Weight Dropout Regularizer



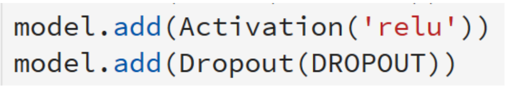
```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
N_HIDDEN = 128 # number of neurons in one hidden layer
VAL_SPLIT = 0.2 # 1/5 for validation rule of thumb
DROPOUT = 0.3 # regularization
```





```
# modeling step
# 2 hidden layers each N_HIDDEN neurons
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))
```

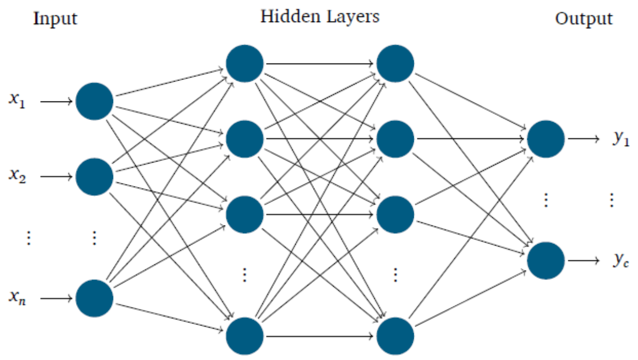
- A Dropout() regularizer randomly drops with ist dropout probability some of the values propagated inside the Dense network hidden layers improving accuracy again
- Our standard model is already modified in the python script but needs to set the DROPOUT rate
- A Dropout() regularizer randomly drops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

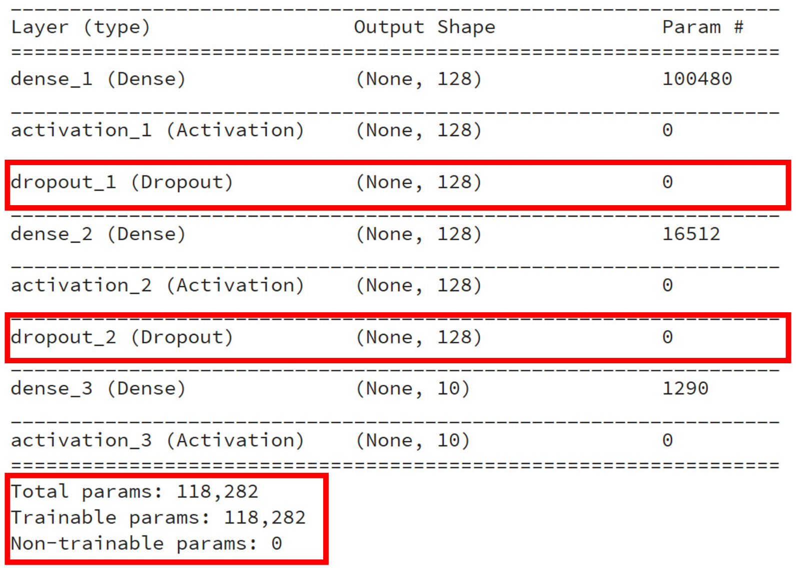


```
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
```

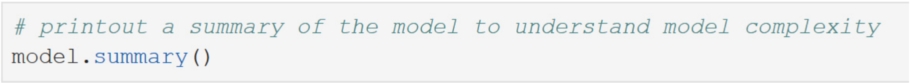
MNIST Dataset & Model Summary & Parameters

- Only two Hidden Layers but with Dropout
  - Each hidden layers has 128 neurons



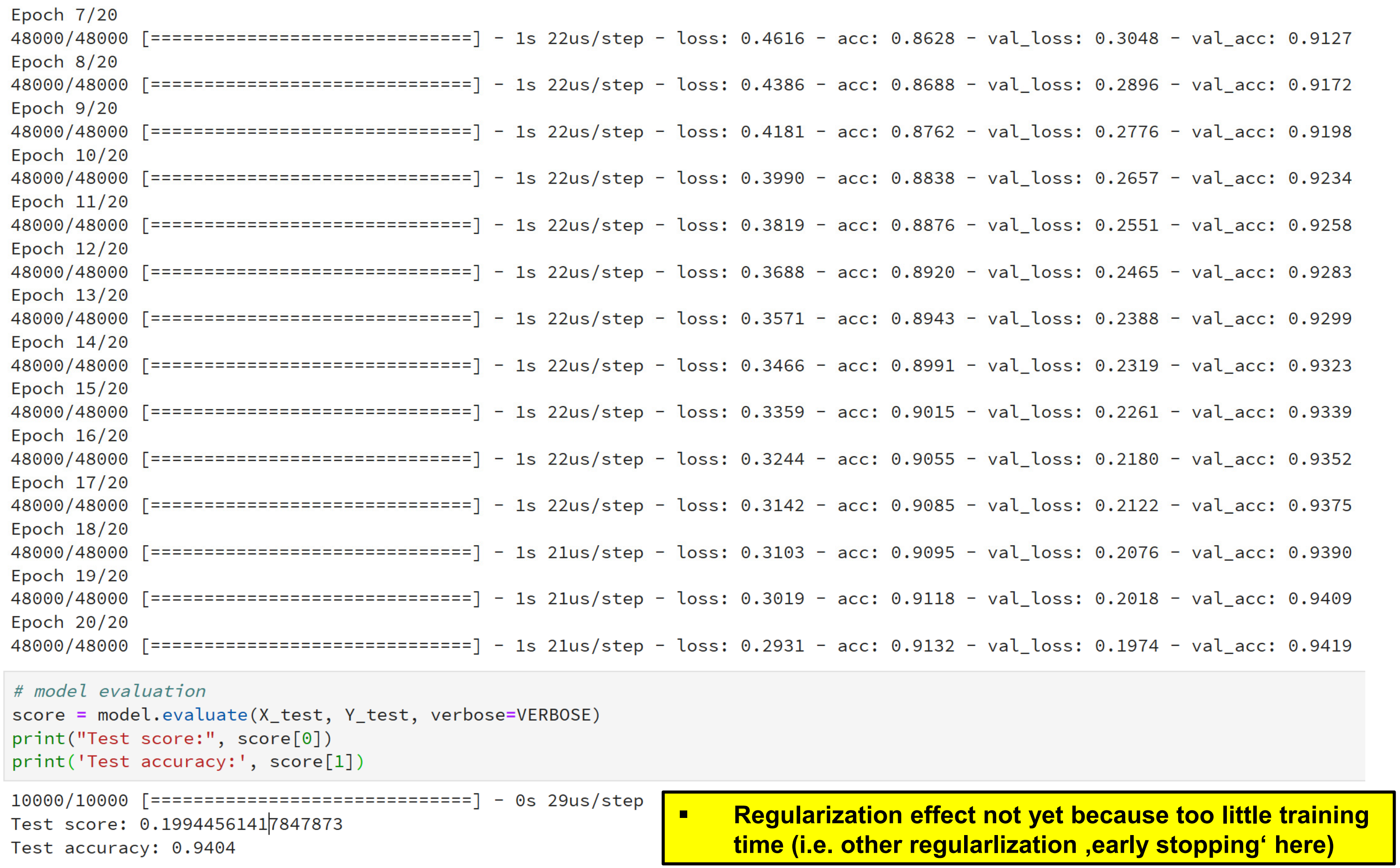


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 128)               100480    
_________________________________________________________________
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
```



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

ANN – MNIST – DROPOUT (20 Epochs)



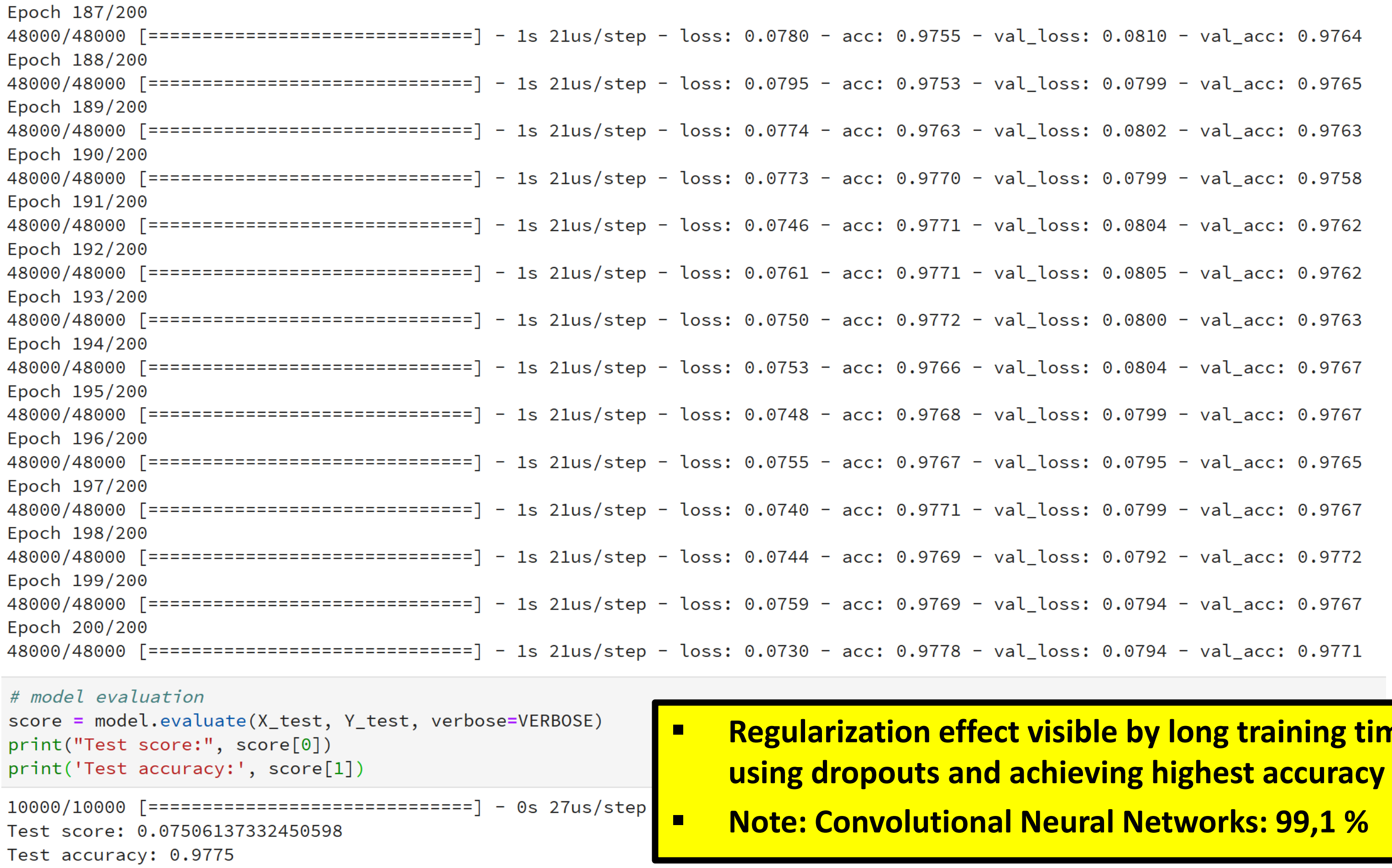
```
Epoch 7/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.4616 - acc: 0.8628 - val_loss: 0.3048 - val_acc: 0.9127
Epoch 8/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.4386 - acc: 0.8688 - val_loss: 0.2896 - val_acc: 0.9172
Epoch 9/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.4181 - acc: 0.8762 - val_loss: 0.2776 - val_acc: 0.9198
Epoch 10/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3990 - acc: 0.8838 - val_loss: 0.2657 - val_acc: 0.9234
Epoch 11/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3819 - acc: 0.8876 - val_loss: 0.2551 - val_acc: 0.9258
Epoch 12/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3688 - acc: 0.8920 - val_loss: 0.2465 - val_acc: 0.9283
Epoch 13/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3571 - acc: 0.8943 - val_loss: 0.2388 - val_acc: 0.9299
Epoch 14/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3466 - acc: 0.8991 - val_loss: 0.2319 - val_acc: 0.9323
Epoch 15/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3359 - acc: 0.9015 - val_loss: 0.2261 - val_acc: 0.9339
Epoch 16/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3244 - acc: 0.9055 - val_loss: 0.2180 - val_acc: 0.9352
Epoch 17/20
48000/48000 [==============================] - 1s 22us/step - loss: 0.3142 - acc: 0.9085 - val_loss: 0.2122 - val_acc: 0.9375
Epoch 18/20
48000/48000 [==============================] - 1s 21us/step - loss: 0.3103 - acc: 0.9095 - val_loss: 0.2076 - val_acc: 0.9390
Epoch 19/20
48000/48000 [==============================] - 1s 21us/step - loss: 0.3019 - acc: 0.9118 - val_loss: 0.2018 - val_acc: 0.9409
Epoch 20/20
48000/48000 [==============================] - 1s 21us/step - loss: 0.2931 - acc: 0.9132 - val_loss: 0.1974 - val_acc: 0.9419
```

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

```
10000/10000 [==============================] - 0s 29us/step
Test score: 0.19944561417847873
Test accuracy: 0.9404
```

- **Regularization effect not yet because too little training time (i.e. other regularlization ‚early stopping' here)**

ANN – MNIST – DROPOUT (200 Epochs)



```
Epoch 187/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0780 - acc: 0.9755 - val_loss: 0.0810 - val_acc: 0.9764
Epoch 188/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0795 - acc: 0.9753 - val_loss: 0.0799 - val_acc: 0.9765
Epoch 189/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0774 - acc: 0.9763 - val_loss: 0.0802 - val_acc: 0.9763
Epoch 190/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0773 - acc: 0.9770 - val_loss: 0.0799 - val_acc: 0.9758
Epoch 191/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0746 - acc: 0.9771 - val_loss: 0.0804 - val_acc: 0.9762
Epoch 192/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0761 - acc: 0.9771 - val_loss: 0.0805 - val_acc: 0.9762
Epoch 193/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0750 - acc: 0.9772 - val_loss: 0.0800 - val_acc: 0.9763
Epoch 194/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0753 - acc: 0.9766 - val_loss: 0.0804 - val_acc: 0.9767
Epoch 195/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0748 - acc: 0.9768 - val_loss: 0.0799 - val_acc: 0.9767
Epoch 196/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0755 - acc: 0.9767 - val_loss: 0.0795 - val_acc: 0.9765
Epoch 197/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0740 - acc: 0.9771 - val_loss: 0.0799 - val_acc: 0.9767
Epoch 198/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0744 - acc: 0.9769 - val_loss: 0.0792 - val_acc: 0.9772
Epoch 199/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0759 - acc: 0.9769 - val_loss: 0.0794 - val_acc: 0.9767
Epoch 200/200
48000/48000 [==============================] - 1s 21us/step - loss: 0.0730 - acc: 0.9778 - val_loss: 0.0794 - val_acc: 0.9771
```

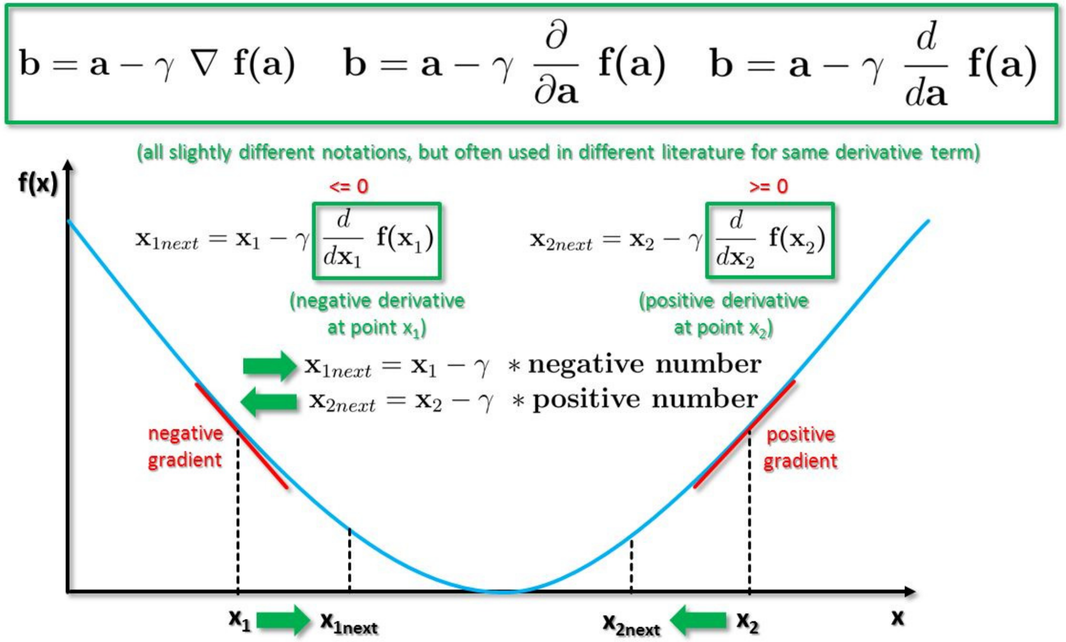
```
# model evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 0s 27us/step
Test score: 0.07506137332450598
Test accuracy: 0.9775
```

- **Regularization effect visible by long training time using dropouts and achieving highest accuracy**
- **Note: Convolutional Neural Networks: 99,1 %**

MNIST Dataset & SGD Method – Changing Optimizers is another possible tuning

- **Gradient Descent (GD) uses all the training samples available for a step within a iteration**
- **Stochastic Gradient Descent (SGD) converges faster: only one training samples used per iteration**



$$b = a - \gamma \nabla f(a) \quad b = a - \gamma \frac{\partial}{\partial a} f(a) \quad b = a - \gamma \frac{d}{da} f(a)$$

(all slightly different notations, but often used in different literature for same derivative term)

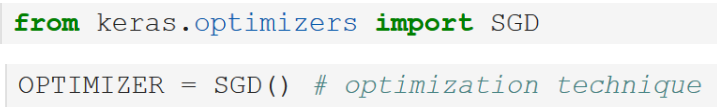
$$x_{1next} = x_1 - \gamma \overset{<= 0}{\boxed{\frac{d}{dx_1} f(x_1)}} \qquad x_{2next} = x_2 - \gamma \overset{>= 0}{\boxed{\frac{d}{dx_2} f(x_2)}}$$

(negative derivative at point x_1)

(positive derivative at point x_2)

$$x_{1next} = x_1 - \gamma \; * \text{negative number}$$

$$x_{2next} = x_2 - \gamma \; * \text{positive number}$$

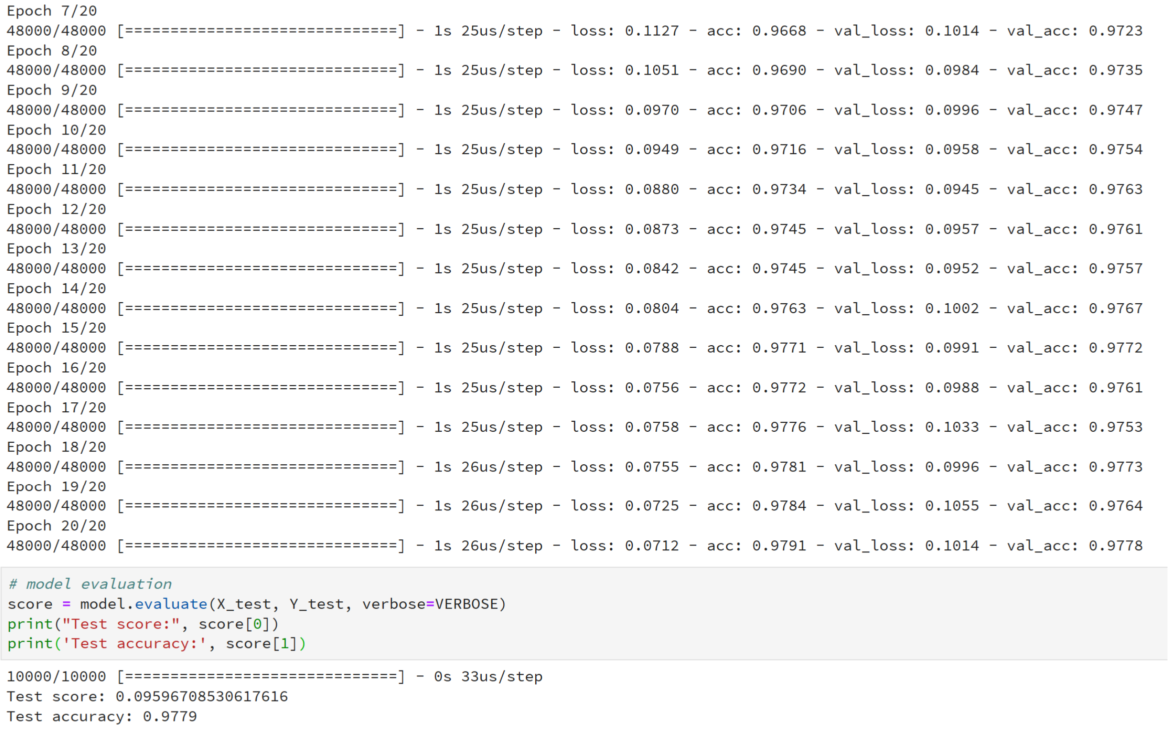


```
from keras.optimizers import SGD

OPTIMIZER = SGD() # optimization technique
```

[7] Big Data Tips, Gradient Descent

MNIST Dataset & RMSprop & Adam Optimization Methods

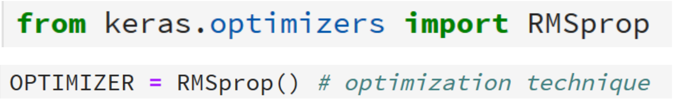
- **RMSProp is an advanced optimization technique that in many cases enable earlier convergence**
- **Adam includes a concept of momentum (i.e. veloctity) in addition to the acceleration of SGD**



```
Epoch 7/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.1127 - acc: 0.9668 - val_loss: 0.1014 - val_acc: 0.9723
Epoch 8/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.1051 - acc: 0.9690 - val_loss: 0.0984 - val_acc: 0.9735
Epoch 9/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0970 - acc: 0.9706 - val_loss: 0.0996 - val_acc: 0.9747
Epoch 10/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0949 - acc: 0.9716 - val_loss: 0.0958 - val_acc: 0.9754
Epoch 11/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0880 - acc: 0.9734 - val_loss: 0.0945 - val_acc: 0.9763
Epoch 12/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0873 - acc: 0.9745 - val_loss: 0.0957 - val_acc: 0.9761
Epoch 13/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0842 - acc: 0.9745 - val_loss: 0.0952 - val_acc: 0.9757
Epoch 14/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0804 - acc: 0.9763 - val_loss: 0.1002 - val_acc: 0.9767
Epoch 15/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0788 - acc: 0.9771 - val_loss: 0.0991 - val_acc: 0.9772
Epoch 16/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0756 - acc: 0.9772 - val_loss: 0.0988 - val_acc: 0.9761
Epoch 17/20
48000/48000 [==============================] - 1s 25us/step - loss: 0.0758 - acc: 0.9776 - val_loss: 0.1033 - val_acc: 0.9753
Epoch 18/20
48000/48000 [==============================] - 1s 26us/step - loss: 0.0755 - acc: 0.9781 - val_loss: 0.0996 - val_acc: 0.9773
Epoch 19/20
48000/48000 [==============================] - 1s 26us/step - loss: 0.0725 - acc: 0.9784 - val_loss: 0.1055 - val_acc: 0.9764
Epoch 20/20
48000/48000 [==============================] - 1s 26us/step - loss: 0.0712 - acc: 0.9791 - val_loss: 0.1014 - val_acc: 0.9778
```

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

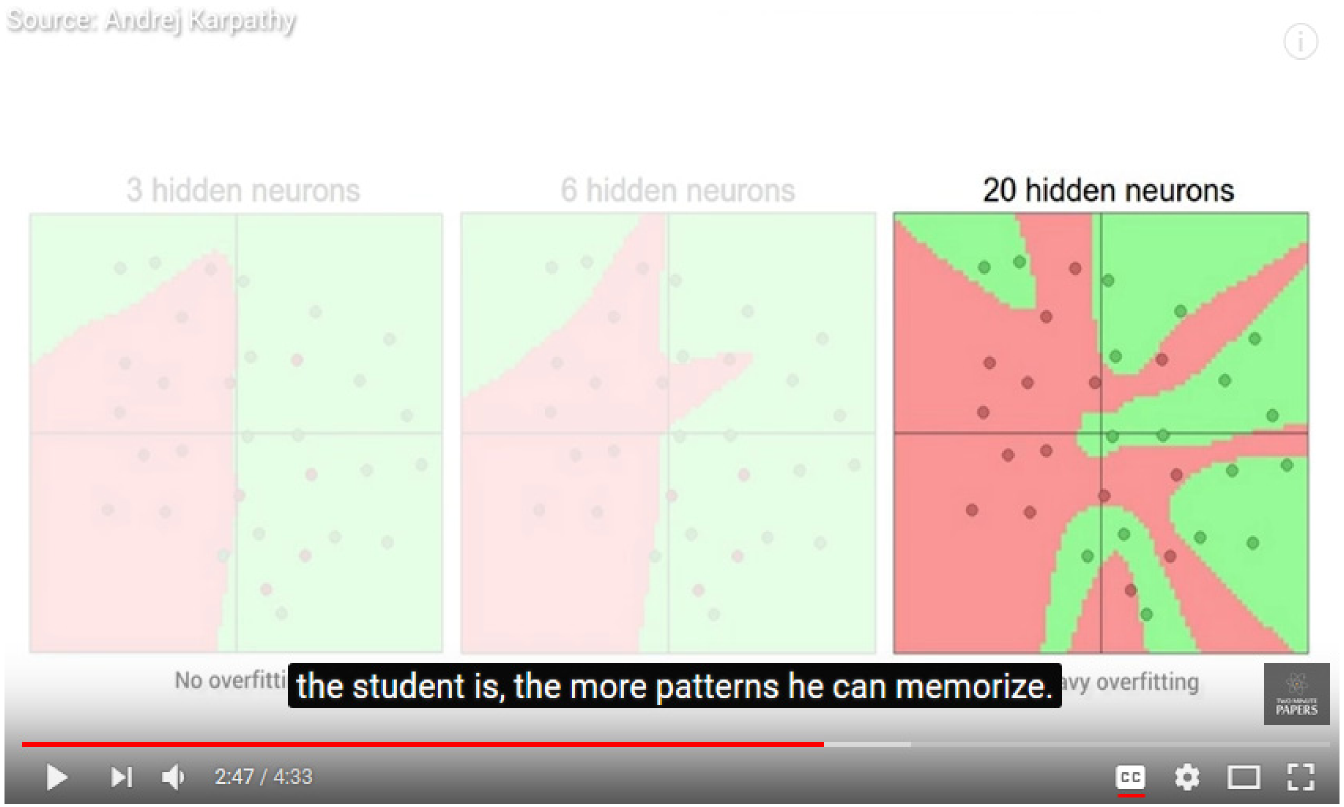
```
10000/10000 [==============================] - 0s 33us/step
Test score: 0.09596708530617616
Test accuracy: 0.9779
```



```python
from keras.optimizers import RMSprop
```

```python
OPTIMIZER = RMSprop() # optimization technique
```

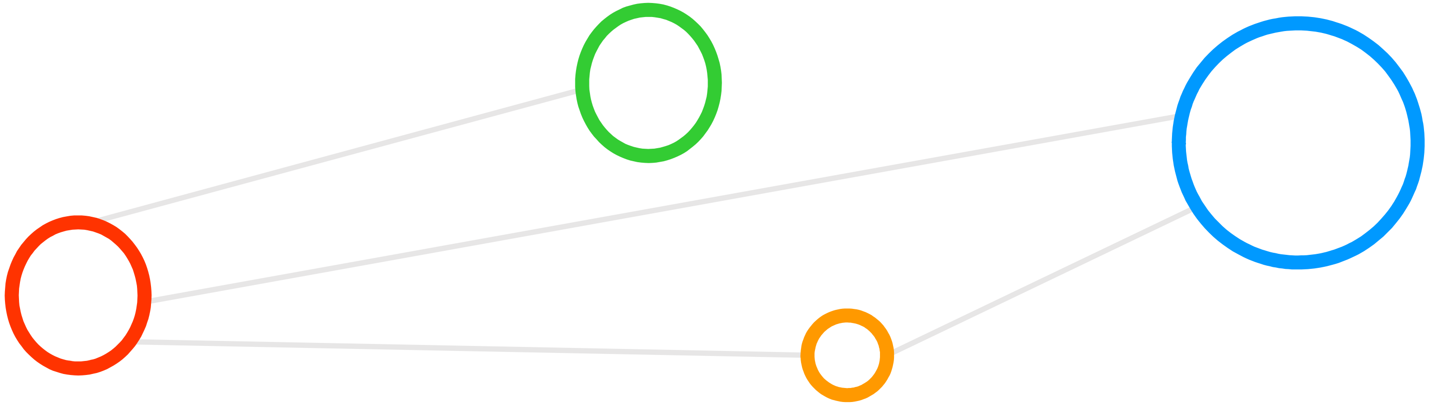
[Video] Overfitting in Deep Neural Networks



the student is, the more patterns he can memorize.

[7] YouTube Video, Overfitting and Regularization For Deep Learning

Lecture Bibliography



Lecture Bibliography

- [1] Leslie G. Valiant, ‘*A Theory of the Learnable*’, Communications of the ACM 27(11):1134–1142, 1984, Online: https://people.mpi-inf.mpg.de/~mehlhorn/SeminarEvolvability/ValiantLearnable.pdf
- [2] Udacity, ‘Overfitting‘, Online: https://www.youtube.com/watch?v=CxAxRCv9WoA
- [3] www.big-data.tips, ‘Relu Neural Network‘, Online: http://www.big-data.tips/relu-neural-network
- [4] www.big-data.tips, ‘tanh‘, Online: http://www.big-data.tips/tanh
- [5] Tensorflow, Online: https://www.tensorflow.org/
- [6] Keras Python Deep Learning Library, Online: https://keras.io/
- [6] www.big-data.tips, ‘Gradient Descent, Online: http://www.big-data.tips/gradient-descent
- [7] YouTube Video, ‘Overfitting and Regularization For Deep Learning | Two Minute Papers #56’, Online: https://www.youtube.com/watch?v=6aF9sJrzxaM
- [8] MIT 6.S191: Introduction to Deep Learning, Online: http://introtodeeplearning.com/
- [9] Understanding the Neural Network, Online: http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2019/www/hwnotes/HW1p1.html

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