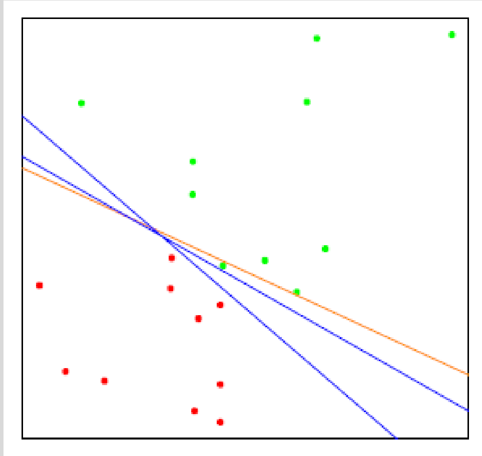


Deep Learning

Using a Convolutional Neural Network

Dr. – Ing. Morris Riedel

Adjunct Associated Professor

School of Engineering and Natural Sciences, University of Iceland

Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 4

Convolutional Neural Networks Challenges

December 1st, 2017

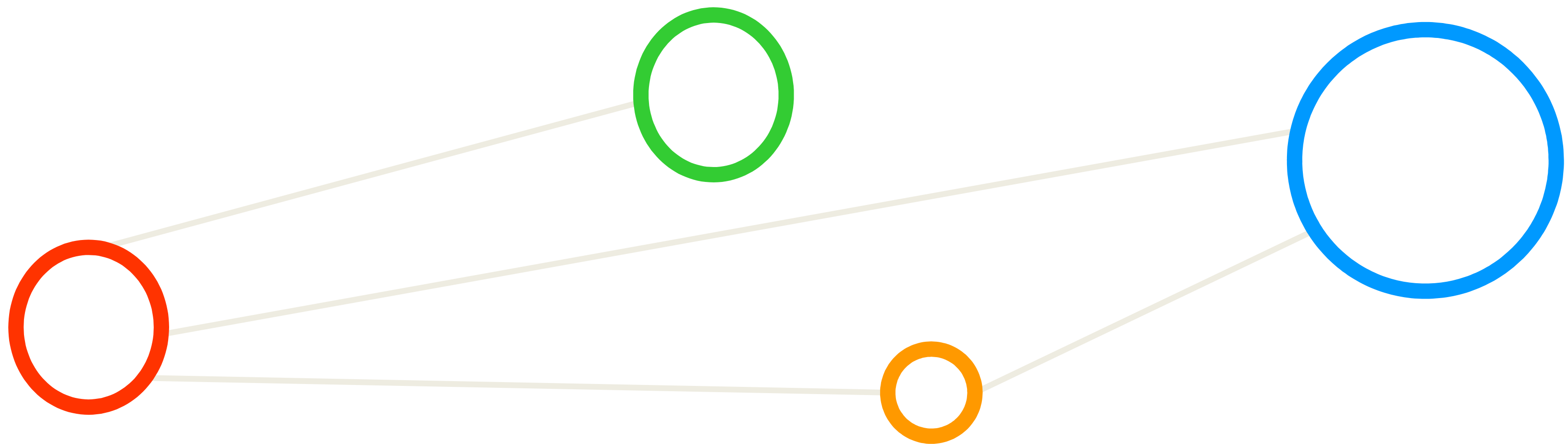
Ghent, Belgium

UNIVERSITY OF ICELAND

SCHOOL OF ENGINEERING AND NATURAL SCIENCES

FACULTY OF INDUSTRIAL ENGINEERING,
MECHANICAL ENGINEERING AND COMPUTER SCIENCE

Outline



Outline of the Course

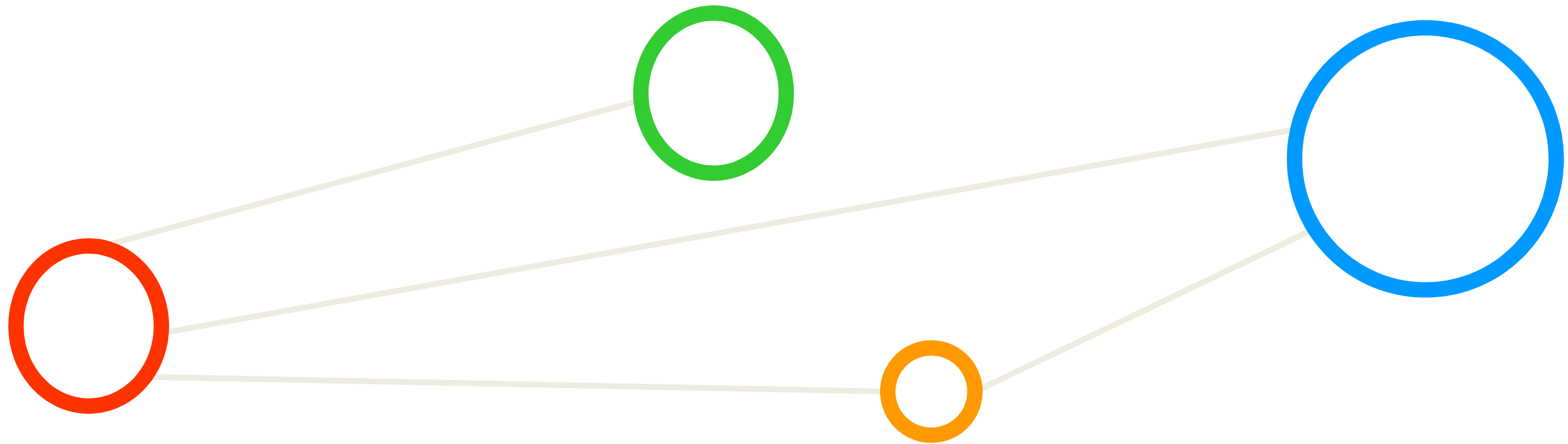
1. Deep Learning Fundamentals & GPGPUs
2. Convolutional Neural Networks & Tools
3. Convolutional Neural Network Applications
4. Convolutional Neural Network Challenges
5. Transfer Learning Technique
6. Other Deep Learning Models & Summary

Outline

- Regularization
  - Overfitting as Key Challenge
  - Fitting Noise & Noise Types
  - ANN & CNN Examples
  - Regularization Approaches
  - Weight Dropout and L2 Examples

- Validation and Model Selection
  - Many Parameters of Model Selection
  - Validation Approaches
  - ANN & CNN Examples
  - Complexity in Finding the right Parameters

Regularization

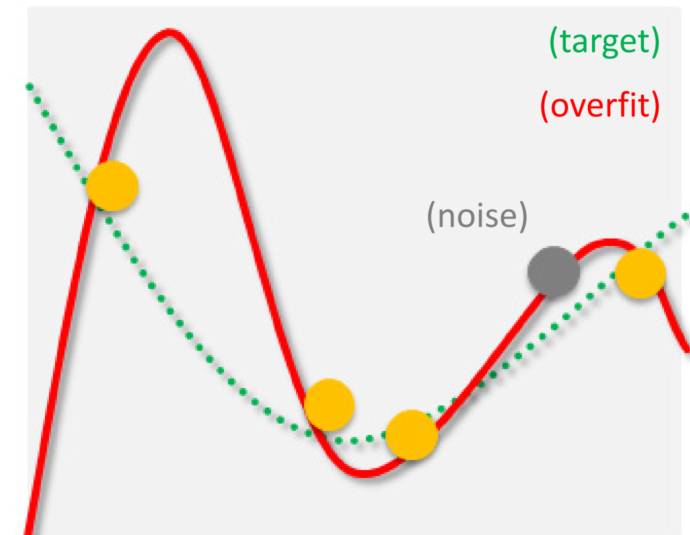


Exercises

Challenge Two – Problem of Overfitting

- 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

- 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 $E_{out}(g)$



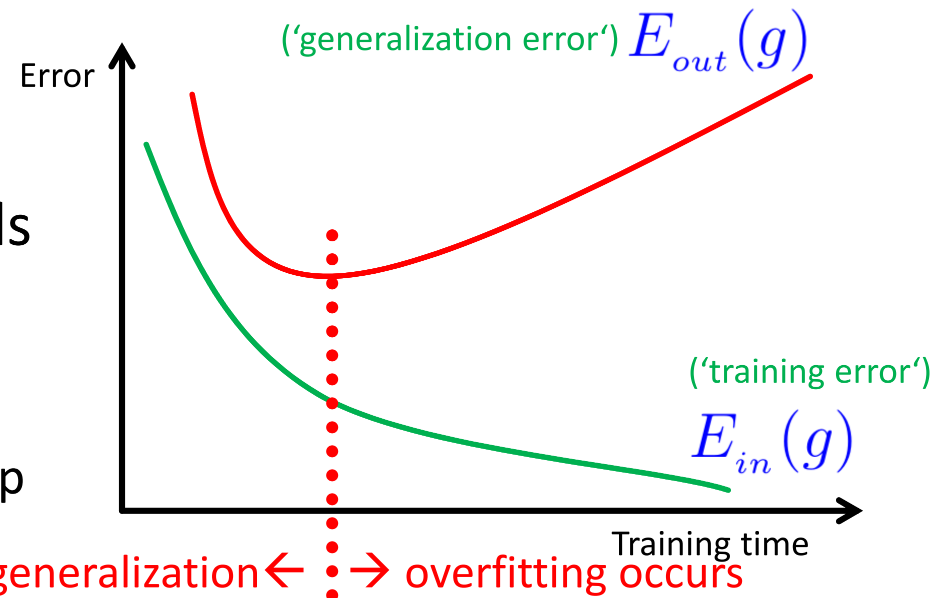
(but simple polynomial works good enough)
('over': here meant as 4th order, a 3rd order would be better, 2nd best)

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'

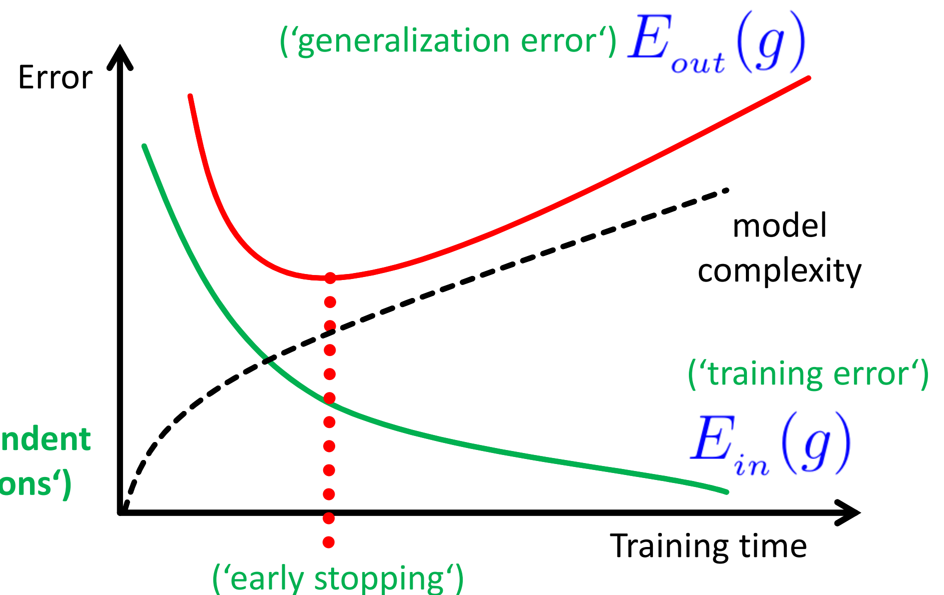


- The two general approaches to prevent overfitting are (1) regularization and (2) validation

Problem of Overfitting – Model Relationships

- Review ‘overfitting situations‘
  - When comparing ‘various models‘ and related to ‘model complexity‘
  - Different models are used, e.g. 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 (later)

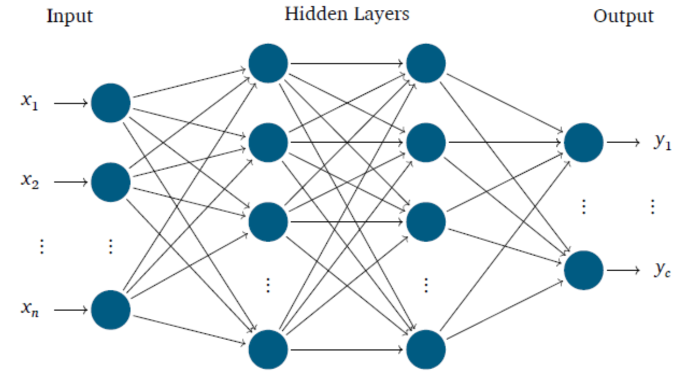
(‘model complexity measure: the VC analysis was independent of a specific target function – bound for all target functions‘)

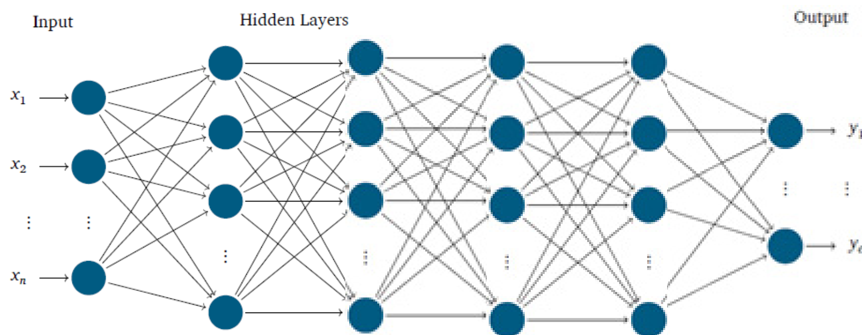


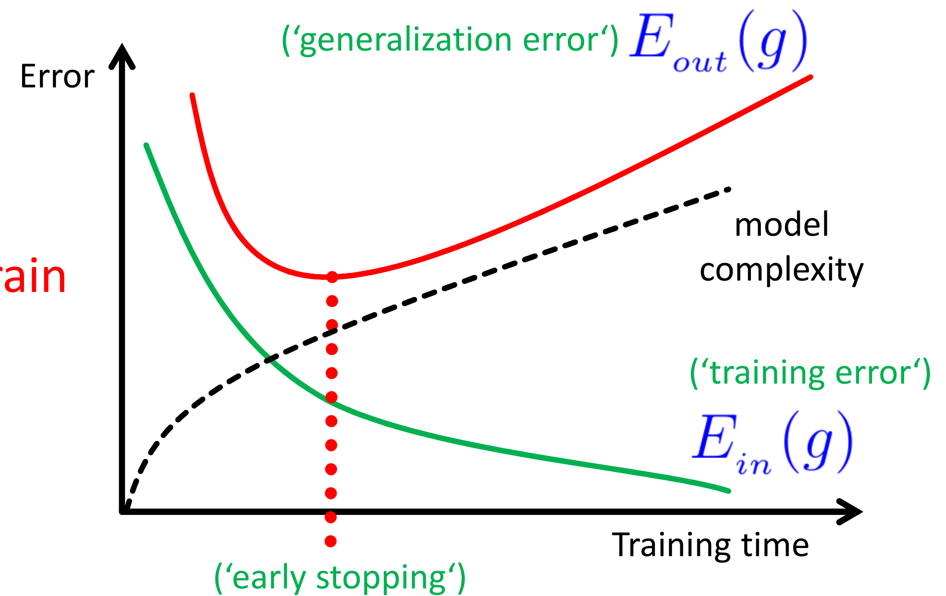
- **‘Early stopping‘ approach is part of the theory of regularization, but based on validation methods**

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



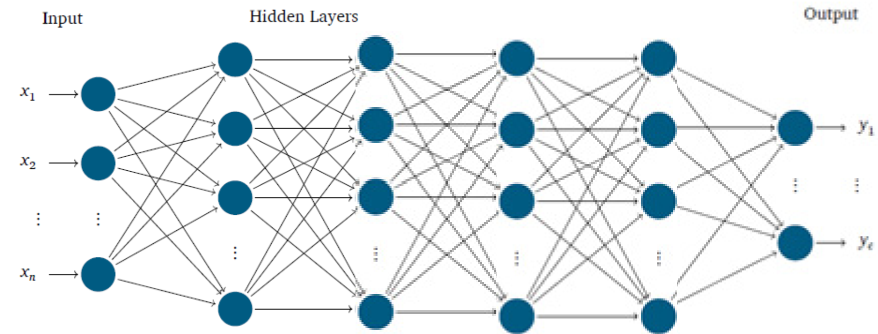




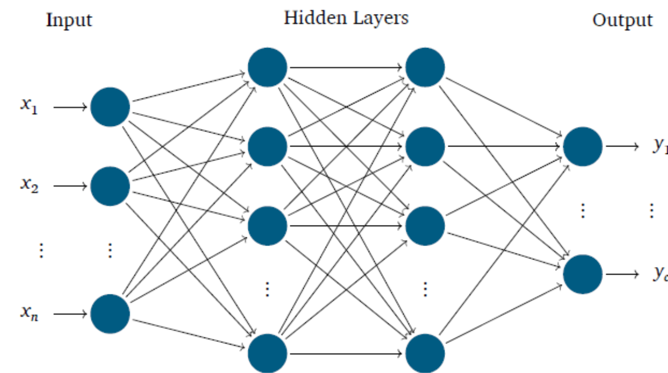
Exercises

ANN – MNIST Dataset – Add Two More 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




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

# convert 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)

# ANN model with hidden layers
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, validation_split=VALIDATION_SPLIT)

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



Problem of Overfitting – Noise Term Revisited

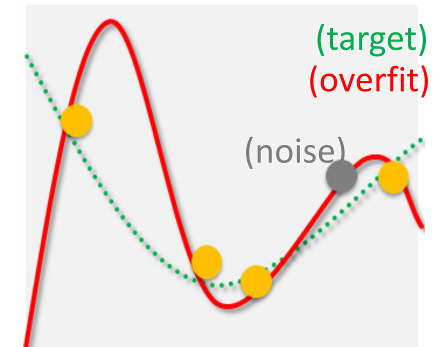
- '(Noisy) Target function' is not a (deterministic) function
  - Getting with 'same x in' the 'same y out' is not always given in practice
  - Idea: Use a 'target distribution' instead of 'target function'

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

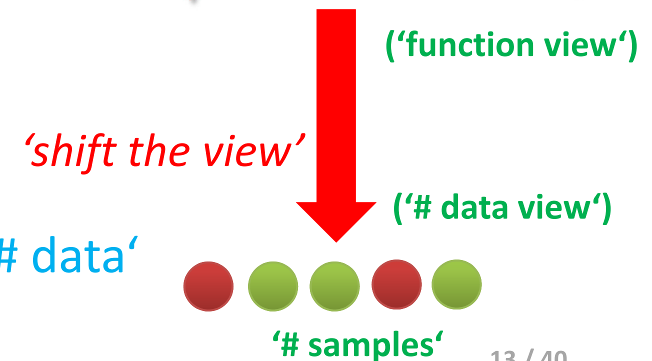
target function $f : X \to Y$ plus noise

(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

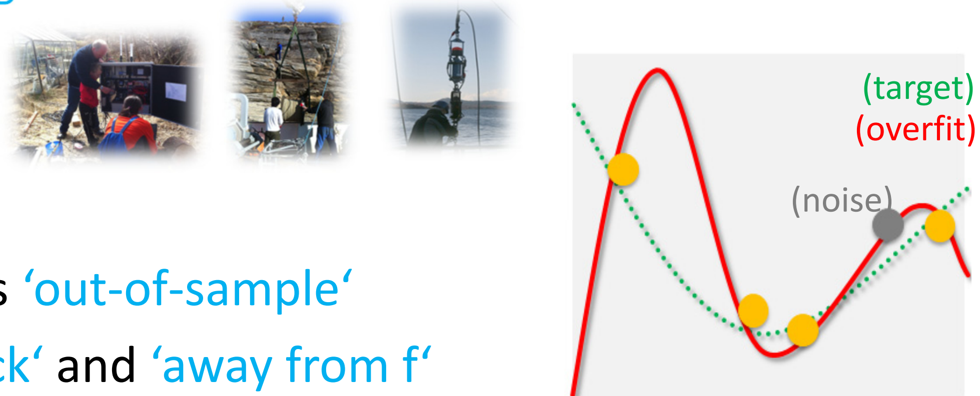
- Stoachastic noise is a part 'on top of' each learnable function
  - Noise in the data that can not be captured and thus not modelled by f
  - Random noise : aka 'non-deterministic noise'
  - Conventional understanding established early in this course
  - Finding a 'non-existing pattern in noise not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

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

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

(ideal function)

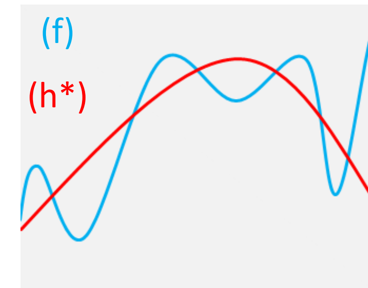
- Practice Example
  - Random fluctuations and/or measurement errors in data
  - Fitting a pattern that not exists 'out-of-sample'
  - Puts learning progress 'off-track' and 'away from f'



- **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 lost function that the network optimizes

Keras

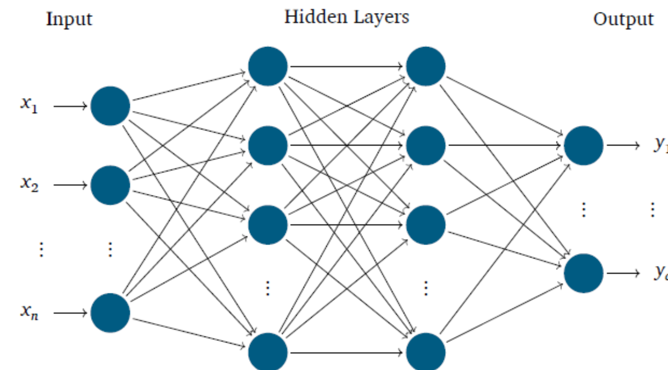
[5] Keras Python Deep Learning Library

Exercises – Underfitting & 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

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

# parameters
NB_CLASSES = 10
NB_EPOCH = 200
BATCH_SIZE = 128
VERBOSE = 1
N_HIDDEN = 64
OPTIMIZER = 'SGD'
VALIDATION_SPLIT = 0.2
DROPOUT = 0.3

# ANN model with hidden layers
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()

# 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)

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



- **A Dropout() regularizer randomly drops with ist dropout probability some of the values propagated inside the Dense network hidden layers improving accuracy again**

Exercises – Regularizers in non-Standard CNN Model

Remote Sensing Dataset – CNN Script – Dropout Regularizer

```
# standard model with dropout
def build_model_standard_dropout(input_shape, activation, num_classes, dropout_frac):
    model = Sequential()
    model.add(Conv3D(48, kernel_size=(3, 3, 5), activation=activation, input_shape=input_shape))
    model.add(Dropout(dropout_frac))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format=None))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation))
    model.add(Dropout(dropout_frac))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format=None))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation))
    model.add(Dropout(dropout_frac))
    model.add(MaxPooling3D(pool_size=(1, 1, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation=activation))
    model.add(Dropout(dropout_frac))
    model.add(Dense(128, activation=activation))
    model.add(Dense(num_classes, activation='softmax'))
    return model

#  -   -   -   -   -   -   -   -   set all the parameters  -   -   -
num_classes = 16  #58
channels = 220
window_size = 9
batch_size = 50
epochs = 1000 #2000
learning_rate = 1
momentum = 0
decay = 0.000005
reg_factor = 0.0          # for L2-regularization (see other models at the end of the program)
dropout_frac = 0.3        # for dropout            (see other models at the end of the program)
activation = 'relu'
loss = 'mean_squared_error'
optimizer = optimizers.SGD(lr=learning_rate, momentum=momentum, decay=decay)
```

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

Remote Sensing Dataset – CNN Script – L2 Regularizer

```python
# standard model with L2-regularization
def build_model_standard_L2(input_shape, activation, num_classes, reg_factor):
    model = Sequential()
    model.add(Conv3D(48, kernel_size=(3, 3, 5), activation=activation, kernel_regularizer=regularizers.l2(reg_factor), input_shape=input_shape))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format=None))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation, kernel_regularizer=regularizers.l2(reg_factor)))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format=None))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation, kernel_regularizer=regularizers.l2(reg_factor)))
    model.add(MaxPooling3D(pool_size=(1, 1, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation=activation, kernel_regularizer=regularizers.l2(reg_factor)))
    model.add(Dense(128, activation=activation, kernel_regularizer=regularizers.l2(reg_factor)))
    model.add(Dense(num_classes, activation='softmax'))
    return model
```

- **Our standard model is already modified in the python script but needs to set the reg_factor**
- **L1 regularization (also known as lasso): The complexity of the model is expressed as the sum**
- **L2 regularization (also known as ridge): The complexity of the model is expressed as the sum of the squares of the weights**
- **Elastic net regularization: The complexity of the model is captured by a combination of the two preceding techniques**

```python
#   -   -   -   -   -   -   -   -   set all the parameters   -   -   -   -   -
num_classes = 16  #58
channels = 220
window_size = 9
batch_size = 50
epochs = 1000 #2000
learning_rate = 1
momentum = 0
decay = 0.000005
reg_factor = 0.01        # for L2-regularization (see other models at the end
dropout_frac = 0.0       # for dropout           (see other models at the end
activation = 'relu'
loss = 'mean_squared_error'
optimizer = optimizers.SGD(lr=learning_rate, momentum=momentum, decay=decay)
```

[Video] Overfitting in Deep Neural Networks

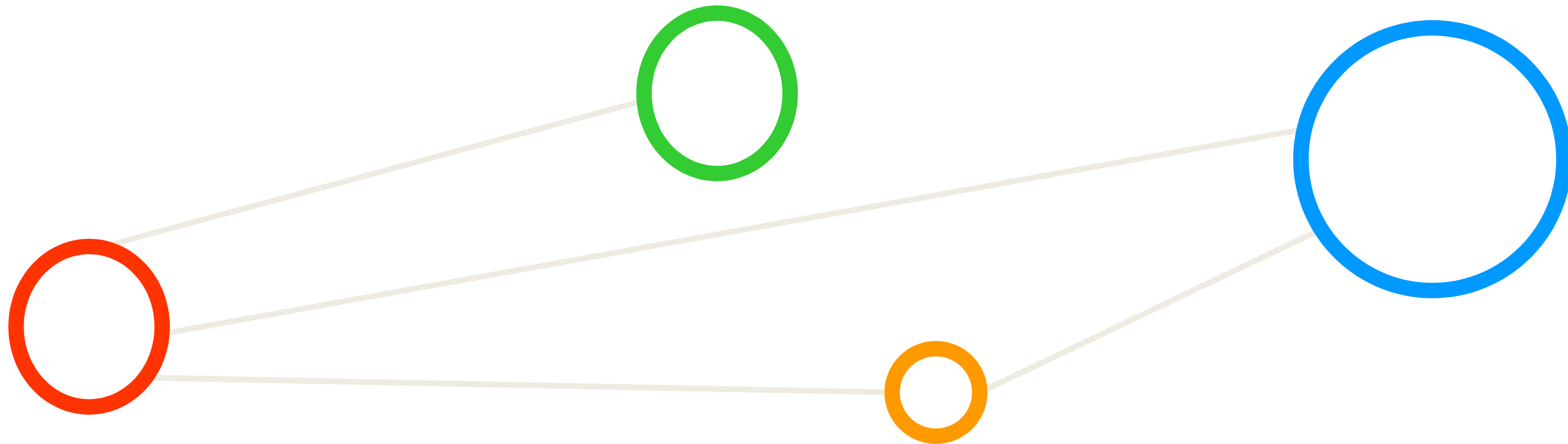
Causes and Outcomes

will assign weights to features that are not
needed and will add unnecessary complexity

3:42 / 5:38

[2] How good is your fit?, YouTube

Validation and Model Selection



MNIST Dataset – CNN Python Script – Validation Data

```
# parameters
NB_CLASSES = 10
NB_EPOCH = 20
BATCH_SIZE = 128
VERBOSE = 1
OPTIMIZER = 'Adam'
VALIDATION_SPLIT = 0.2
IMG_ROWS, IMG_COLS = 28, 28
INPUT_SHAPE = (1, IMG_ROWS, IMG_COLS)

# dataset 28 x 28 pixels
(X_train, y_train), (X_test, y_test) = mnist.load_data()
K.set_image_dim_ordering("th")
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalization
X_train /= 255
X_test /= 255

# input convnet
X_train = X_train[:, np.newaxis, :, :]
X_test = X_test[:, np.newaxis, :, :]

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

# convert 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 CNN model
model = CNN.build(input_shape=INPUT_SHAPE, classes=NB_CLASSES)

# 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)

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

- **Rule of thumb: Take 1/5 of validation, aka 20%**
- **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.**

MNIST Dataset – CNN Model – Output using Validation

```
[vsc42544@gligar01 deeplearning]$ head KERAS_MNIST_CNN.o1179880
60000 train samples
10000 test samples
Train on 48000 samples, validate on 12000 samples
Epoch 1/20

  128/48000 [..............................] - ETA: 10:06 - loss: 2.2997 - acc: 0.1250
  256/48000 [..............................] - ETA: 7:46 - loss: 2.2578 - acc: 0.1992
  384/48000 [..............................] - ETA: 6:58 - loss: 2.2127 - acc: 0.2083
  512/48000 [..............................] - ETA: 6:35 - loss: 2.1632 - acc: 0.2598
  640/48000 [..............................] - ETA: 6:20 - loss: 2.0934 - acc: 0.3234
```

```
[vsc42544@gligar01 deeplearning]$ tail KERAS_MNIST_CNN.o1179880
 9824/10000 [============================>.] - ETA: 0s
 9856/10000 [============================>.] - ETA: 0s
 9888/10000 [============================>.] - ETA: 0s
 9920/10000 [============================>.] - ETA: 0s
 9952/10000 [============================>.] - ETA: 0s
 9984/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 41s 4ms/step
Test score: 0.0483192791523
Test accuracy: 0.99
Working directory was /user/scratch/gent/vsc425/vsc42544/KERAS_MNIST_CNN_1179880.master19.golett.gent.vsc
```

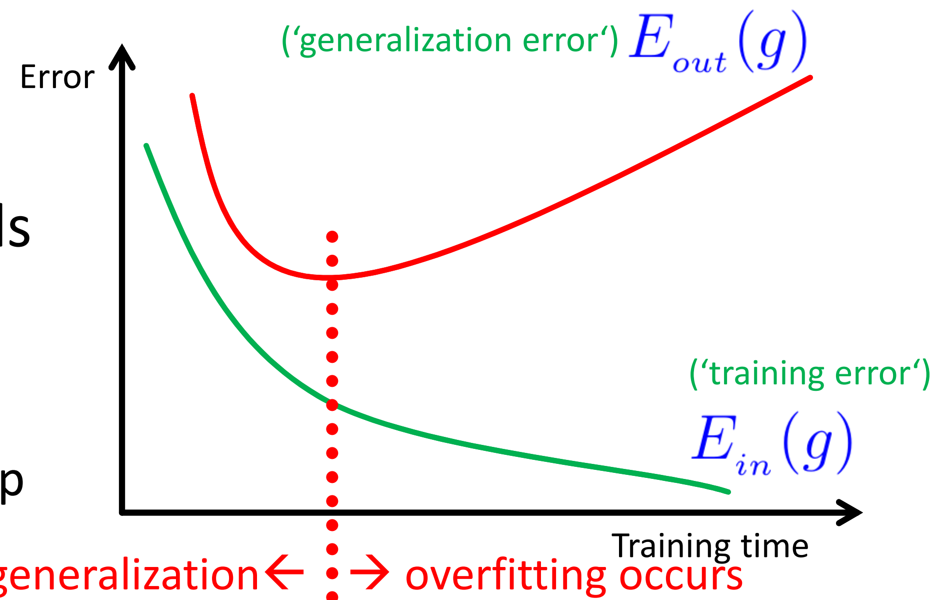
Exercises – Change the Validation to 80% - What happens?

Problem of Overfitting – Clarifying Terms – Revisited

- **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 <u>(2) validation</u>**

(Decisions about the model are related to the problem of overfitting – need another method to 'select model well')

Problem of Overfitting – Impacts on Learning Revisited

- **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. with linear regression)
  - 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')

(Decisions about the model are related to the model complexity – need another method to 'select model well')

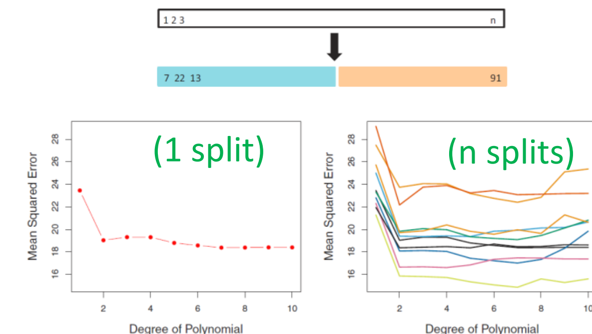
Validation & Model Selection – Terminology

- **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)

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

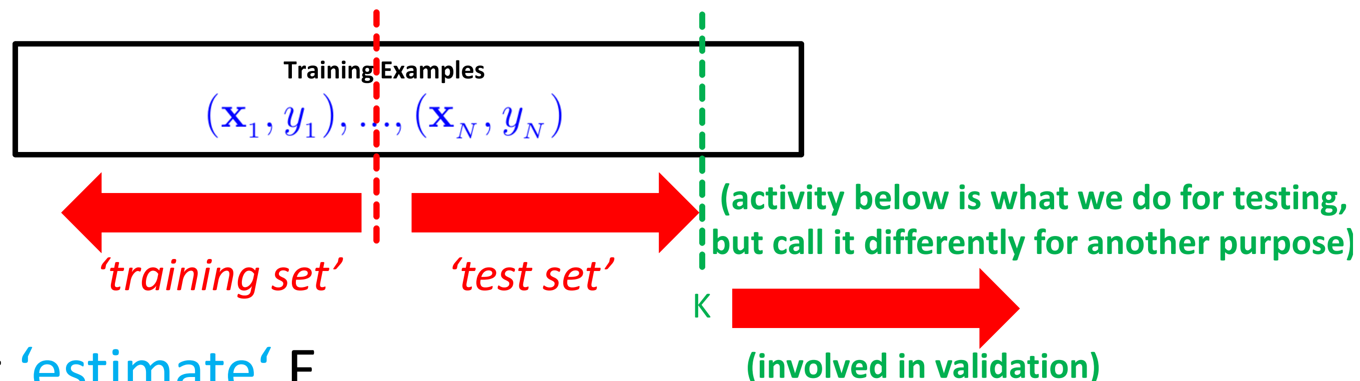
(minimize both to be better proxy for E_{out})

↑ **(validation estimates this quantity)** — points to $E_{out}(h)$

↑ **(regularization estimates this quantity)** — points to overfit penalty

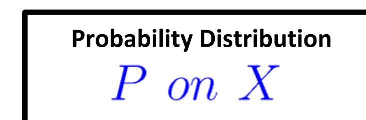
- Validation **(measuring E_{out} is not possible as this is an unknown quantity, another quantity is needed that is measurable that at least estimates it)**
  - Goal 'estimate the out-of-sample error' **(establish a quantity known as validation error)**
  - Distinct activity from training and testing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



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

(aka 'random variable')

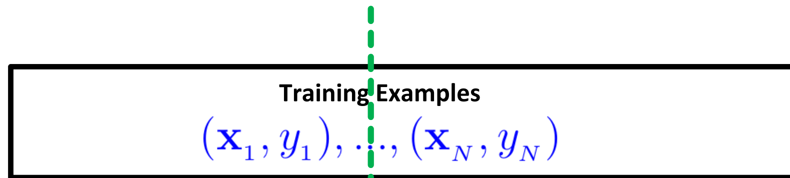
$\mathbb{E}[e(h(\mathbf{x}), y)] = E_{out}(h)$ **(aka the long-run average value of repetitions of the experiment)**

(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



(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

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

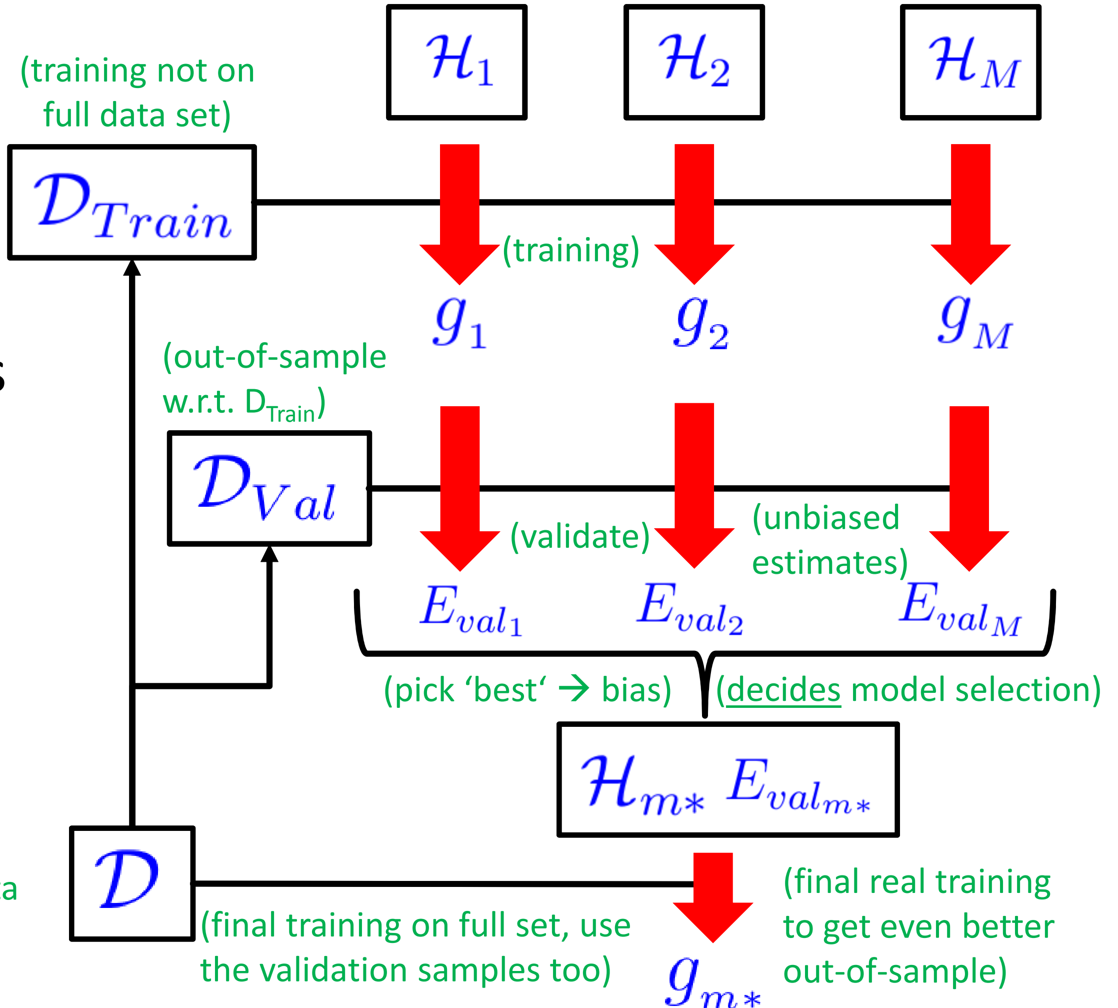
Validation Technique – Model Selection Process

- 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)

Remote Sensing - Experimental Setup @ JSC – Revisited

- 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}

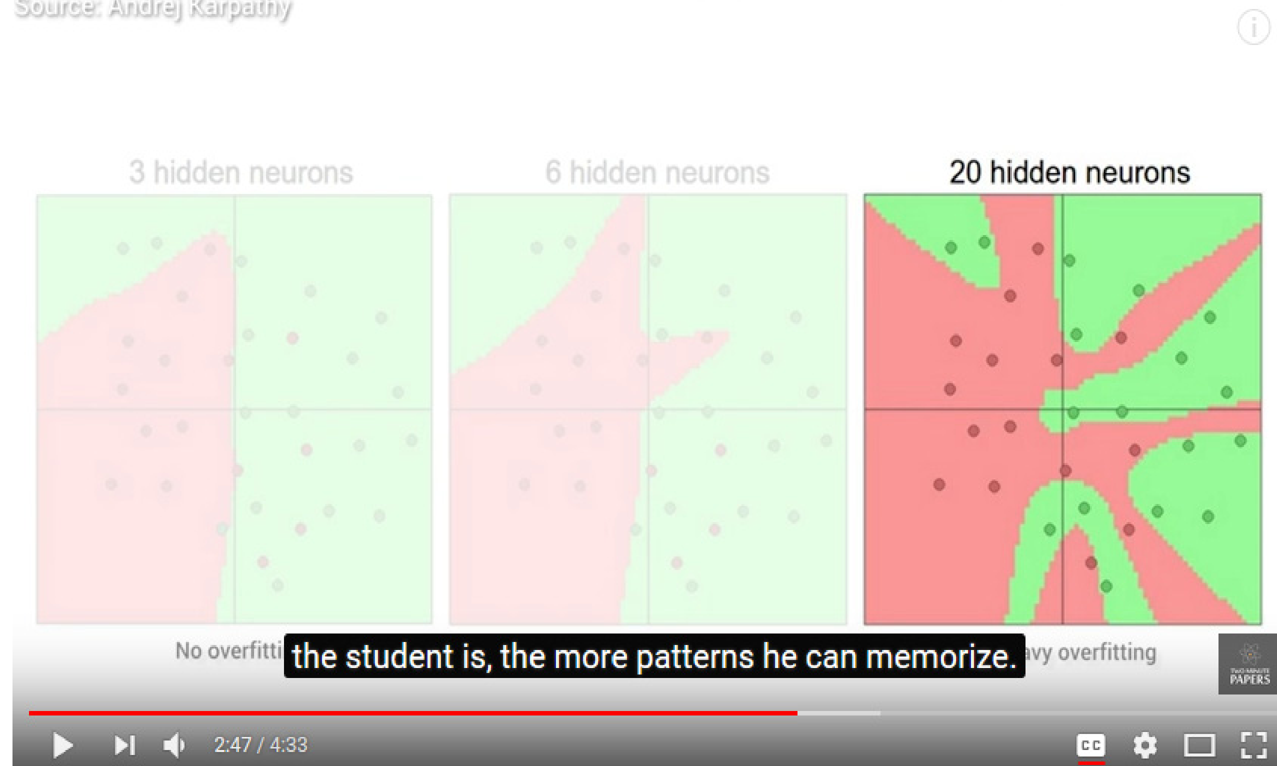
(adding regularization values adds even more complexity in finding the right parameters)

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

Remote Sensing Data – Group Exercises L2 Values / Dropouts

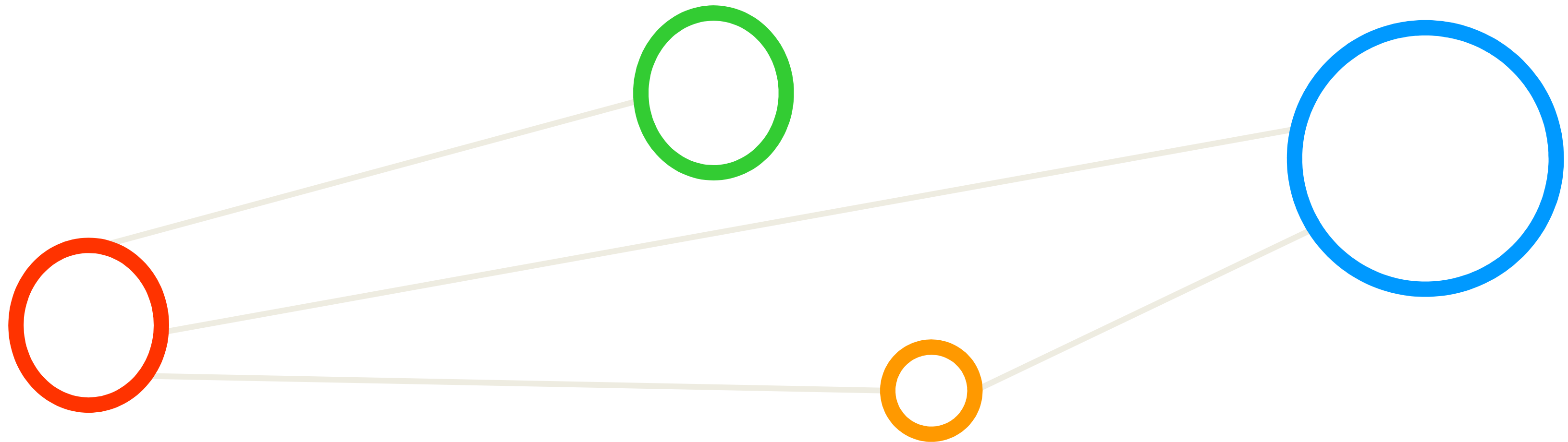
- Add Validation (20%) and different L2 / dropouts per group
  - Which group performs best?

[Video] Overfitting in Deep Neural Networks



*[3] O*verfitting and Regularization For Deep Learning*, YouTube*

Lecture Bibliography



Lecture Bibliography

- [1] Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Addison Wesley, ISBN 0321321367, English, ~769 pages, 2005
- [2] YouTube Video, ‘How good is your fit? - Ep. 21 (Deep Learning SIMPLIFIED)‘, Online: https://www.youtube.com/watch?v=cJA5IHIIL30
- [3] YouTube Video, ‘Overfitting and Regularization For Deep Learning | Two Minute Papers #56’, Online: https://www.youtube.com/watch?v=6aF9sJrzxaM
- [4] An Introduction to Statistical Learning with Applications in R, Online: http://www-bcf.usc.edu/~gareth/ISL/index.html
- [5] Keras Python Deep Learning Library, Online: https://keras.io/

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