Machine Learning

Rudolf Mayer

April 8th, 2025

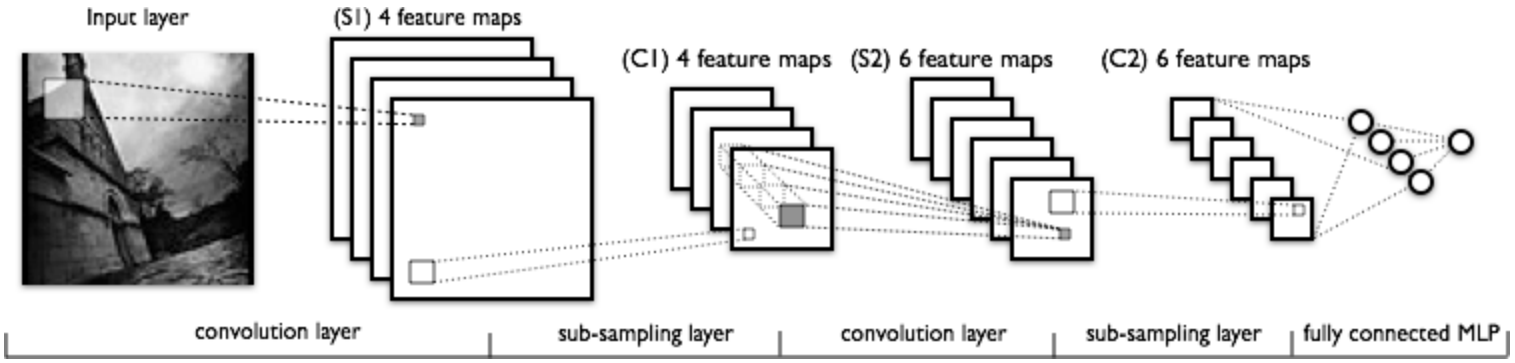
Outline

- Short Recap
- Deep Learning: Convolutional Neural Networks
- Transfer Learning
- Ensemble Learning

Outline

- Short Recap
- Deep Learning: Convolutional Neural Networks
- Transfer Learning
- Ensemble Learning

Recap: Convolutional Neural Network (CNN)

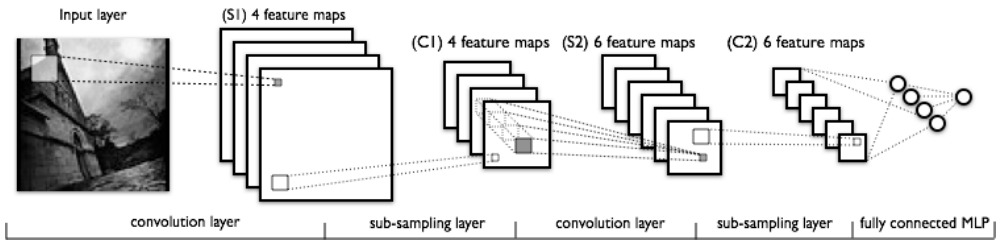


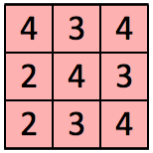
- Convolutional layer:
  - Perform 2D convolution of 2D input with multiple, **learned** 2D kernels
    - I.e. Matrix values are randomly initialised
    - Optimised via back-propagation
- Subsampling layer
  - Replace 2D patch by maximum ("max-pooling") or average
- Fully-connected layer
  - Compute weighted sums of its input with multiple sets of learned coefficients

1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$
0	0	$1_{\times 0}$	$1_{\times 1}$	$0_{\times 0}$
0	1	$1_{\times 1}$	$0_{\times 0}$	$0_{\times 1}$

4	3	4
2	4	3
2	3	4

Recap: Convolutional Neural Network (CNN)

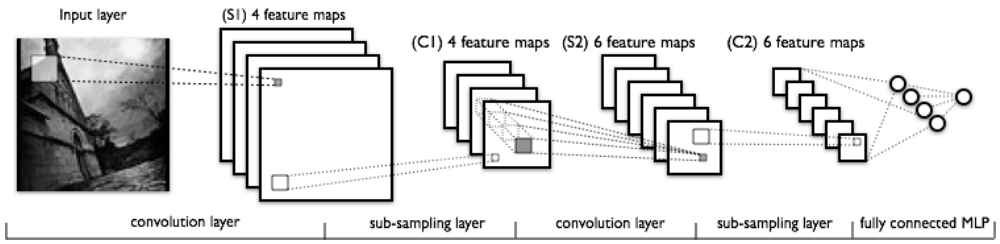


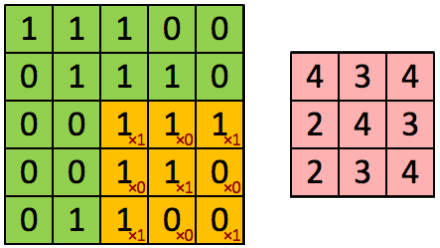


- Kernel, filter, feature map, ...?
  - Often used interchangeably for the 2D-matrix* we multiply with
- Convolution?
  - The mathematical operation, also the output (Feature map)
- Pooling?
  - Selects the maximum value, which means?
  - ➔ The spot where the kernel matched the input the best
    - Within the region covered by the pooling operator. Size?
  - Average pooling? ➔ how well the kernel fits in average
- Tensor?
  - Scalar: 0D tensor; Array: 1D tensor; matrix: 2D tensor; 3+D matrix: ND tensor

* Could also be 1D vector, or other shape

Recap: Convolutional Neural Network (CNN)

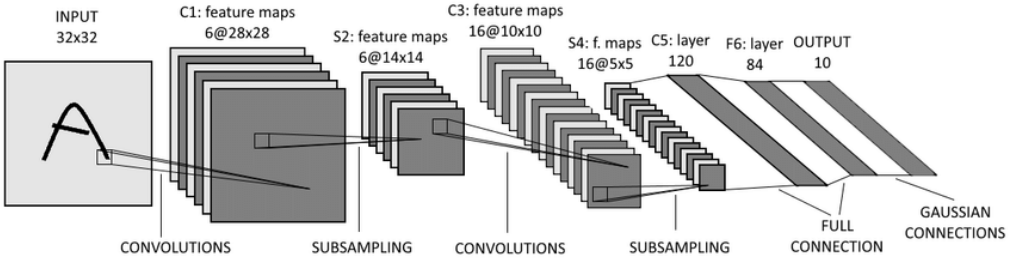




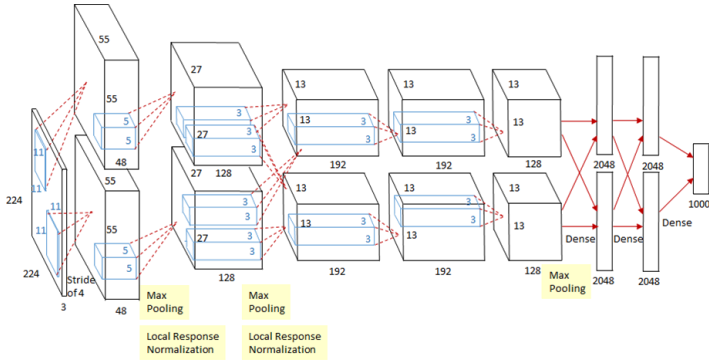
- So, what does a CNN do?
  - Reduces the number of inputs (e.g. pixels)
    - By convolution, but mostly subsampling (pooling)
  - Tolerates shifts in the location
    - By not directly learning a mapping from pixel position to output
  - Uses correlations of pixel patterns
    - Considers regions of image, not single pixels
      - Each cell in feature map corresponds to group of neighbouring pixels
    - Uses the same filters in all regions

Recap: Popular Architectures

- LeNet: Yann LeCunn, 1998
  - 14,000 citations
  - 14,704 parameters

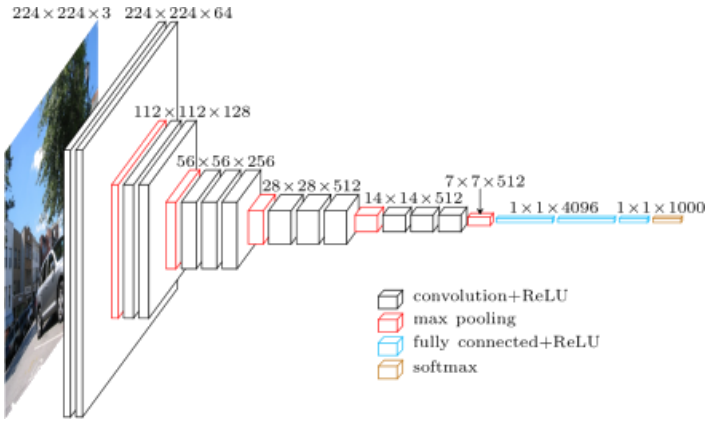


- AlexNet: Krizhevsky et al. 2012
  - 30,000 citations
  - 60 million parameters

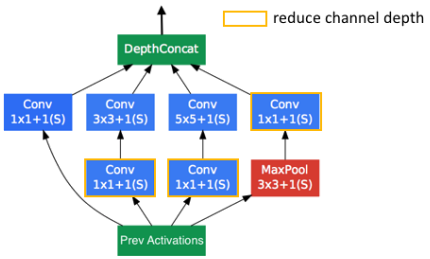


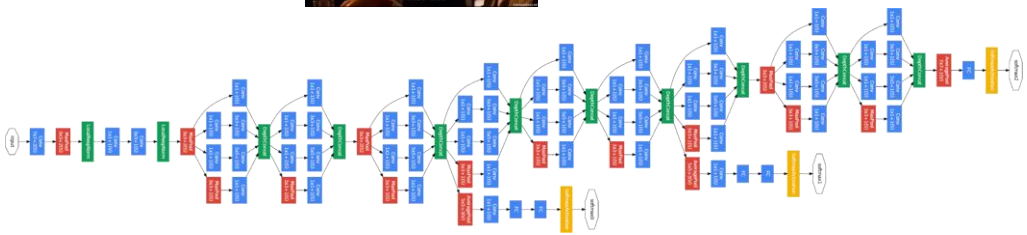
Recap: Popular Architectures

- VGG16 / 19, 2014
  - 138 million parameters



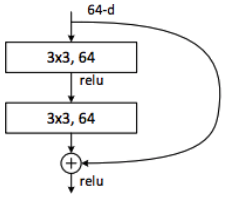
- GoogLeNet (Inception) 2014
  - 22 layers, auxiliary FC layers
  - ~5/13 million parameters (v1/v3)

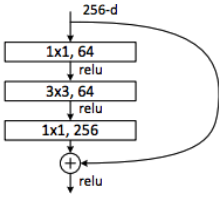


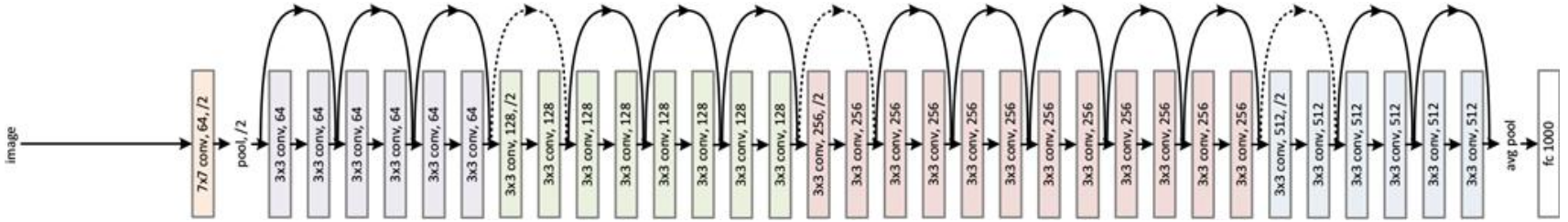


Recap: Popular Architectures

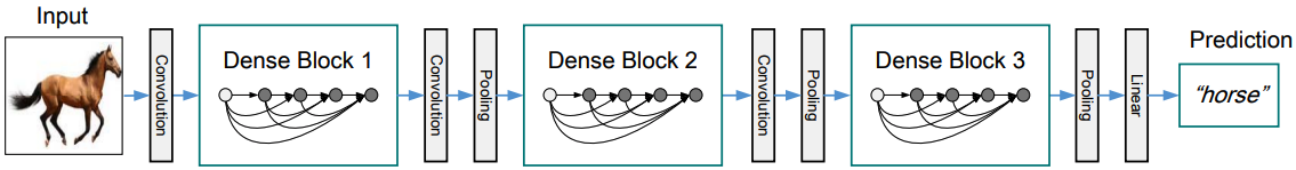
- ResNet, Microsoft, 2015
  - ~ 25/45/60 million parameters
    (ResNet 50/101/152)







- DenseNet, Facebook, 2017
  - 0.8/15/50 million parameters (DenseNet-100&k=12/250&k=24/190& k=40)



Recap: Neural Network Design Choices

- *Very **time** consuming for very large networks!*
  - "Efficient Neural Architecture Search via Parameter Sharing". Hieu Pham et al., 2018

Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**

Outline

- Short Recap
- Deep Learning: Data Augmentation
- Transfer Learning
- Ensemble Learning

Data Augmentation

Goal: Increase **invariance** to **irrelevant properties**
(CNNs are generally not invariant to translations!)

cat facing left

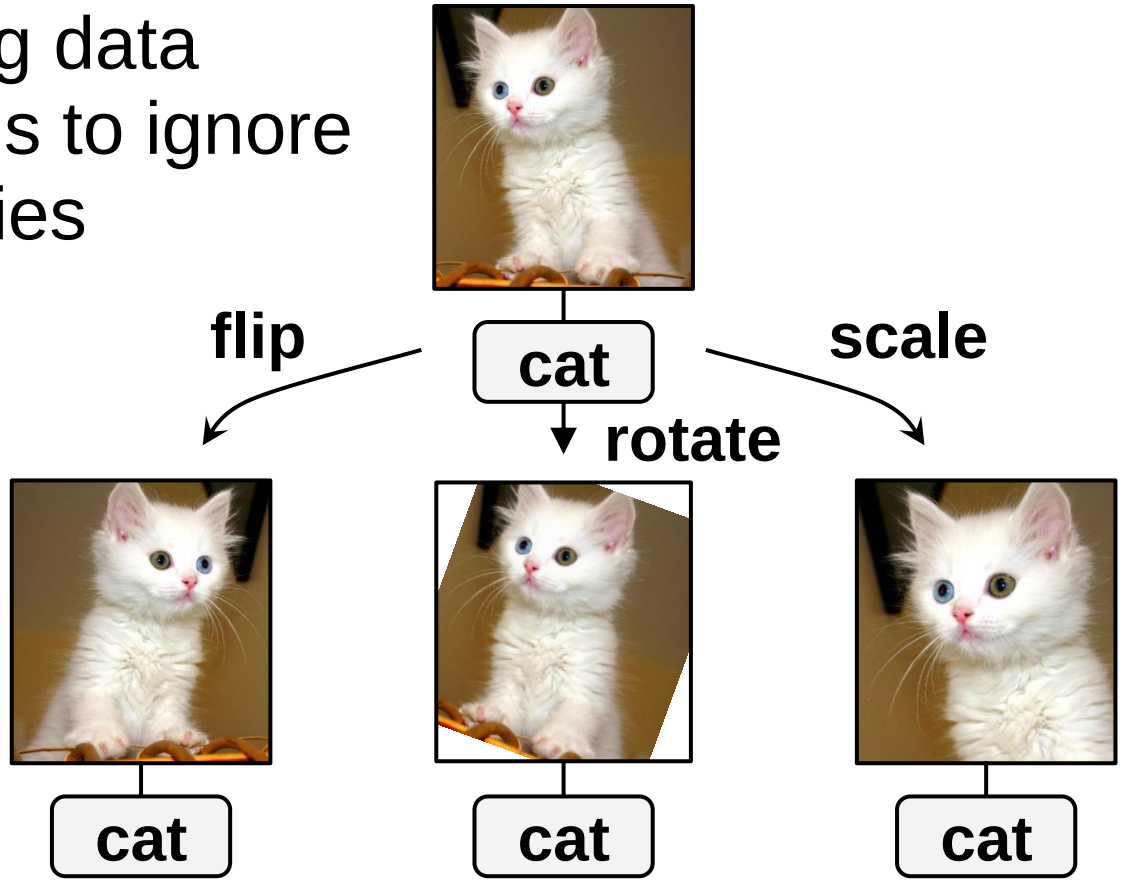
cat facing right

Possible solutions:

- Collect more (more diverse) training data
- Design invariant features or invariant model
- **Data augmentation**

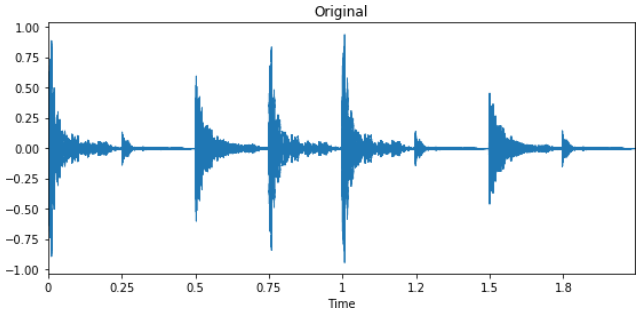
Data Augmentation

- Transform training data
  ➔ classifier learns to ignore irrelevant properties
- E.g. orientation,
  rotation,
  scale,

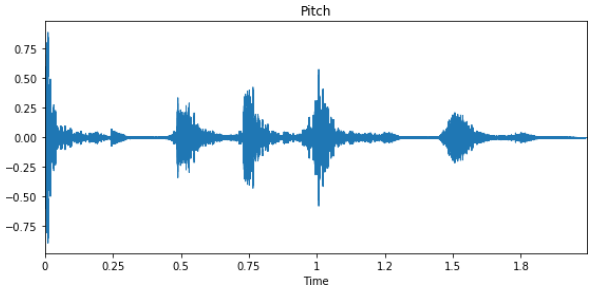


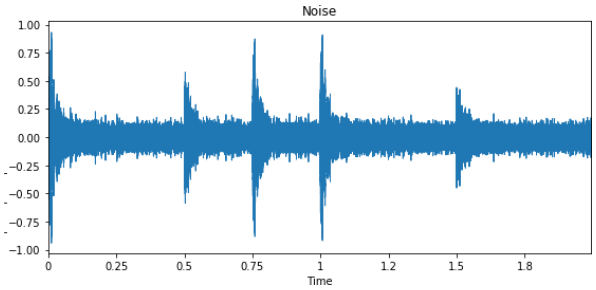
- Also: color, contrast, etc.

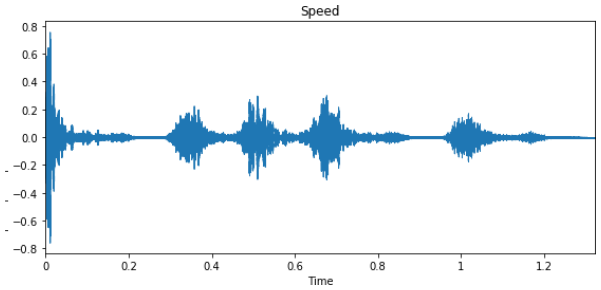
Data Augmentation



- Audio: time stretching, pitch shifting, noise, …







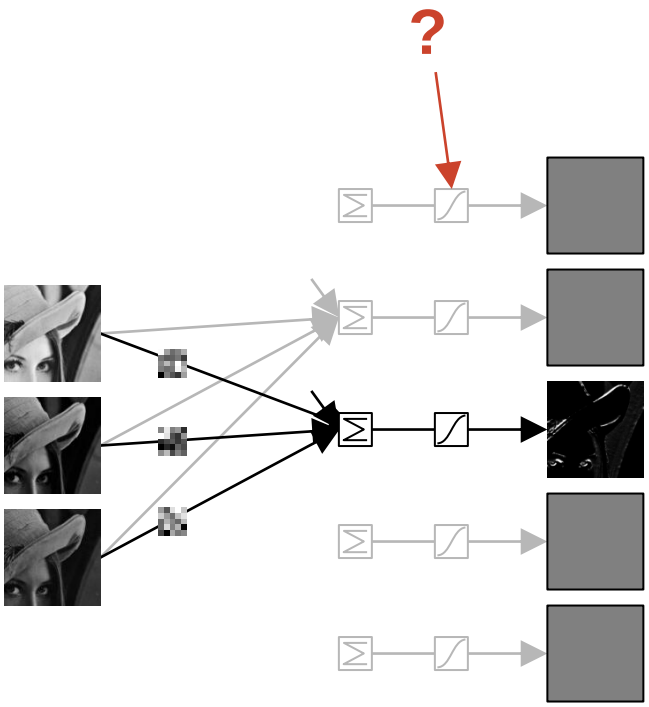
- Other modalities (e.g. video): similar operations
- Side effects:
  - Regularization: Less likely to learn data “by heart”
  - Increases apparent training set size
    - Can train larger model
    - Can use smaller dataset

Outline

- Short Recap
- Deep Learning: Training Tweaks
- Transfer Learning
- Ensemble Learning

Activation Functions

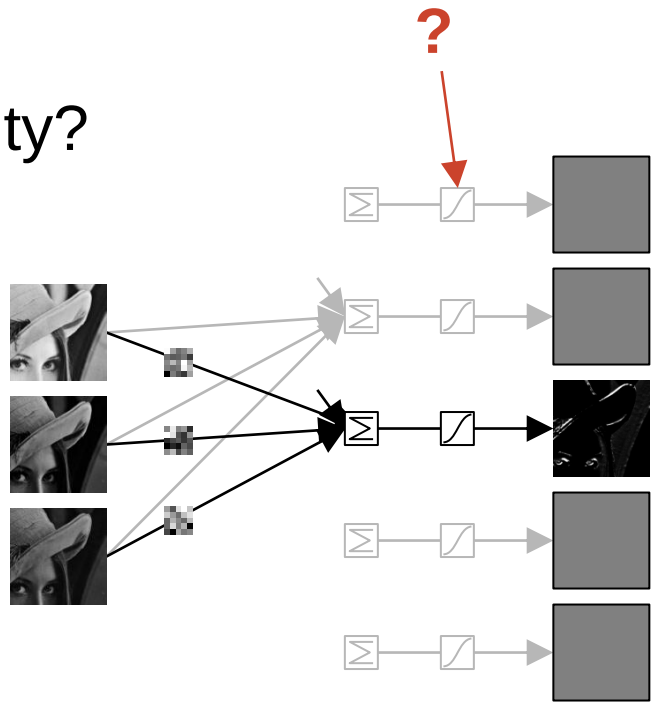
What shall the learnable function look like?



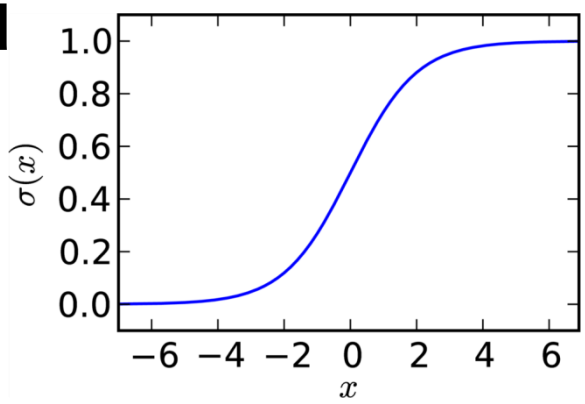
Activation Functions

What shall the learnable function look like?

- Specifically, what kind of nonlinearity?



Sigmoid

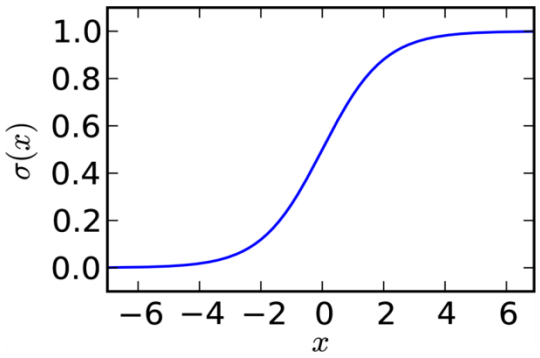


Problem:

saturates for large inputs (small slope ➔ weak gradient)

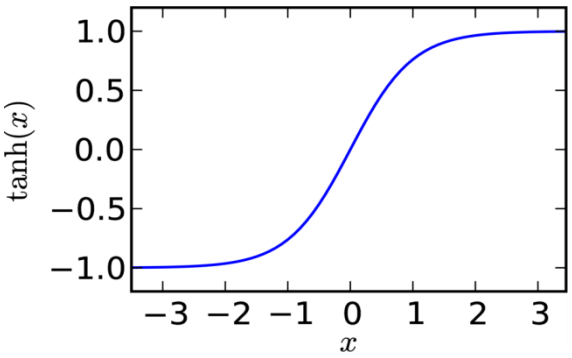
has nonzero mean (slows learning)

Linear Rectifier (ReLU)



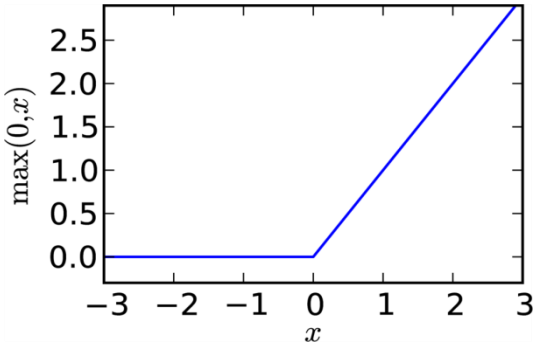
Sigmoid

saturates for large inputs (small slope, weak gradient)



TanH

saturates for large inputs (small slope, weak gradient!)

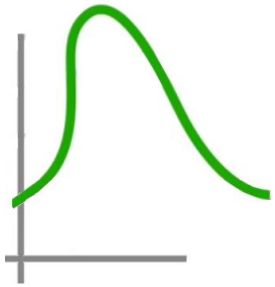


ReLU

Higher gradients also for larger inputs
➔ Avoids vanishing gradient

Benefits:
no saturation
low computational costs

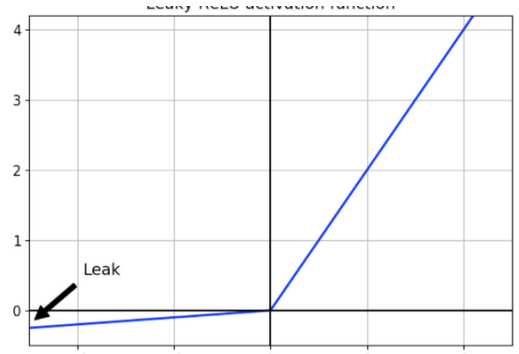
Shape of learned function?



Softplus

Variants of ReLU:

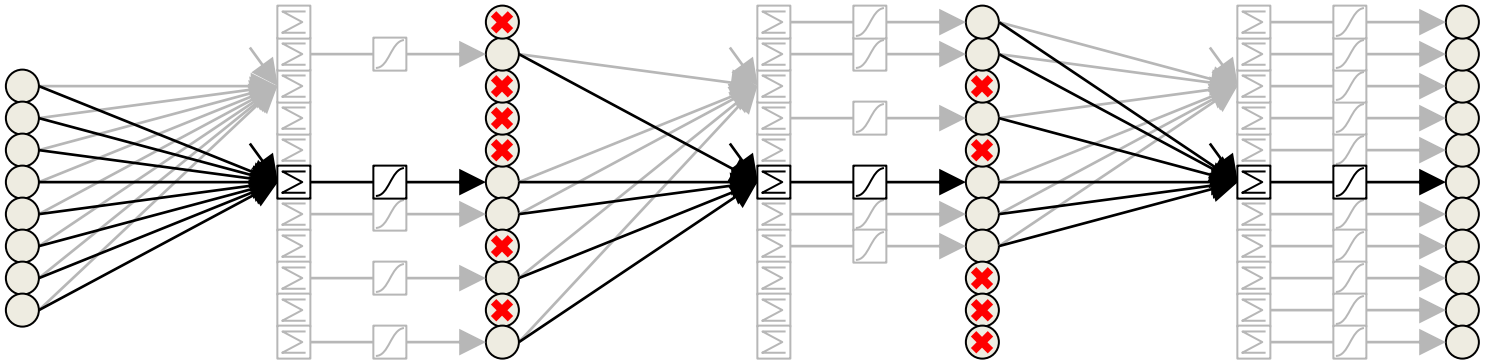
- Leaky Rectifier (LReLU)
  - small slope for <0
- Parametric Rectifier (PReLU), …



ICML 2013: Rectifier Nonlinearities Improve Neural Network Acoustic Models

Dropout

Randomly “delete” a part of the units
for each training example (e.g. 10% or 50%)



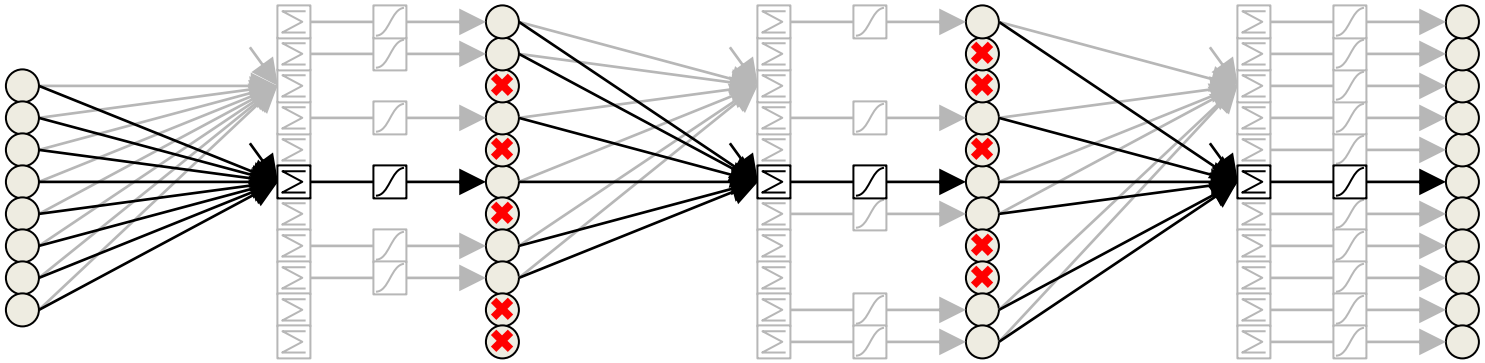
Makes the units more robust, prevents *co-adaptation of feature detectors (kernels)*

(they cannot rely on all inputs and neighbours being present)

Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Dropout

Randomly “delete” a part of the units
for each training example (e.g. 10% or 50%)



Makes the units more robust, prevents *co-adaptation*
(they cannot rely on all inputs and neighbours being present)

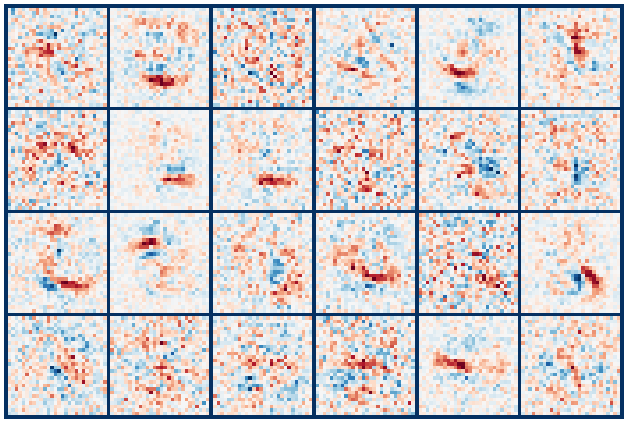
Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Dropout Example

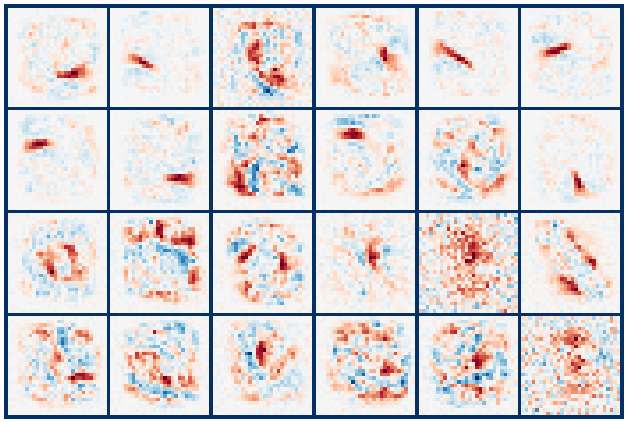
E.g. randomly delete half of the units for each training example.

MNIST digit recognition:

First-layer features after training:



No dropout:
noisy features, overfit to training set



20% input, 50% hidden dropout:
cleaner global features, more general

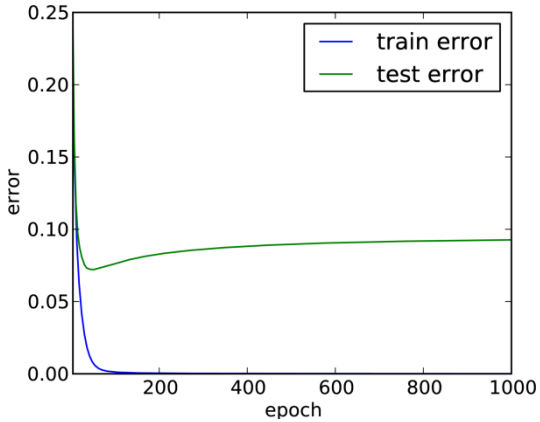
Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Dropout Example

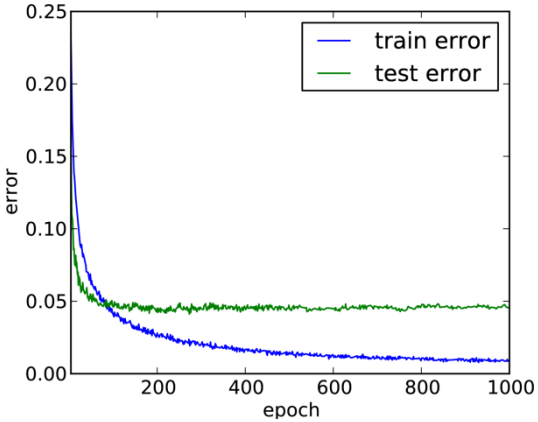
E.g. randomly delete half of the units for each training example.

MNIST digit recognition:

Train/Test error:



No dropout:
quick overfitting, 169 test errors

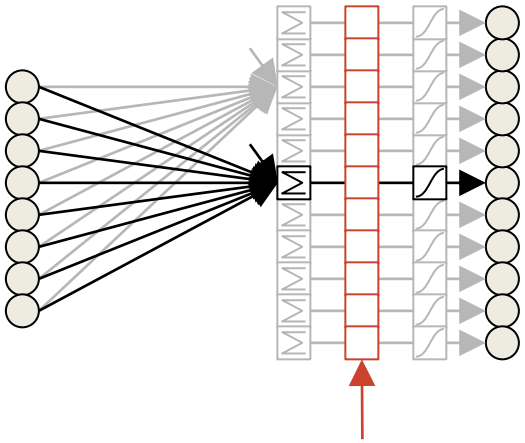


20% input, 50% hidden dropout:
test error plateaus, 99 test errors

Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Batch Normalisation

- Weight gradients depend on the unit activations
  - Very small or very large activations lead to small/large gradients
    - For ***input*** units, **standardizing** the **training data** avoids this
  - **Batch normalisation does the same for hidden units**
    - Even as mean/variance (distribution) changes (internal covariate shift) during training
    - Computationally cheap
  - Also works as regulariser!



batch normalisation

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Optimiser

- The method by which we find “optimal” values for all parameters (weights, biases, ...)
- Optimal: ideally for the unlabelled data, but approximated by the training data
  - Compute loss between predicted and target outputs
  - Adapt weights accordingly

- Gradient Descent (GD) the most popular method
  - Have already discussed batch vs stochastic vs. mini-batch gradient descent

  ➔ Many more variations proposed ...

Mini-batch Gradient Descent

- Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

  - Randomly select one sample to compute loss / error
  - Take small step in direction of negative gradient
  - Repeat until all samples have been processed (one epoch)

- Mini-batch: instead of update after single training sample: gradient averaged over mini-batch of samples (often 2^x instances)
  - Sweet spot between
    - Online learning (single example, very noisy gradient estimate, can only take small steps) and
    - Batch learning (all examples, too much computation per update step)

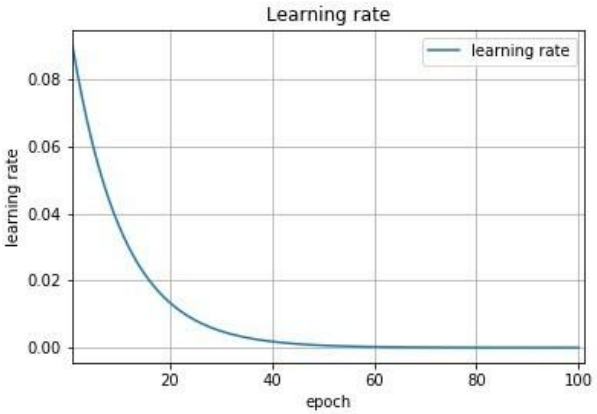
Gradient Descent

- How to set the learning rate α?
  - The learning rate is a hyperparameter of great influence
  - It's not easy to guess, so often many values are tried
  - To allow for better convergence, the learning rate can be adjusted over time using a decay function

Time decay: $\alpha_t = \frac{\alpha_0}{t}$

Exponential decay: $\alpha_t = \alpha_0 * \exp(-t * k)$

Exponential decayfunction



Gradient Descent

- Several alternatives to (stochastic) gradient descent proposed
- Aim: solve issue of (speed of) convergence
  - Gradient descent converges slowly to the optimum in low curvature regions
  - **Momentum** is a simple approach to speed up the optimizer
  - Might also help to avoid local optima

- Very active research field, new approaches frequently published
- Two selected approaches:
  - Momentum
  - Adaptive Moment Estimation (Adam)

Gradient Descent: Momentum

- Stochastic Gradient Descent (SGD) with Momentum:

$$v = \beta v - \alpha \frac{\partial L}{\partial \theta}$$

$$\Theta_{t+1} = \theta_t - v$$

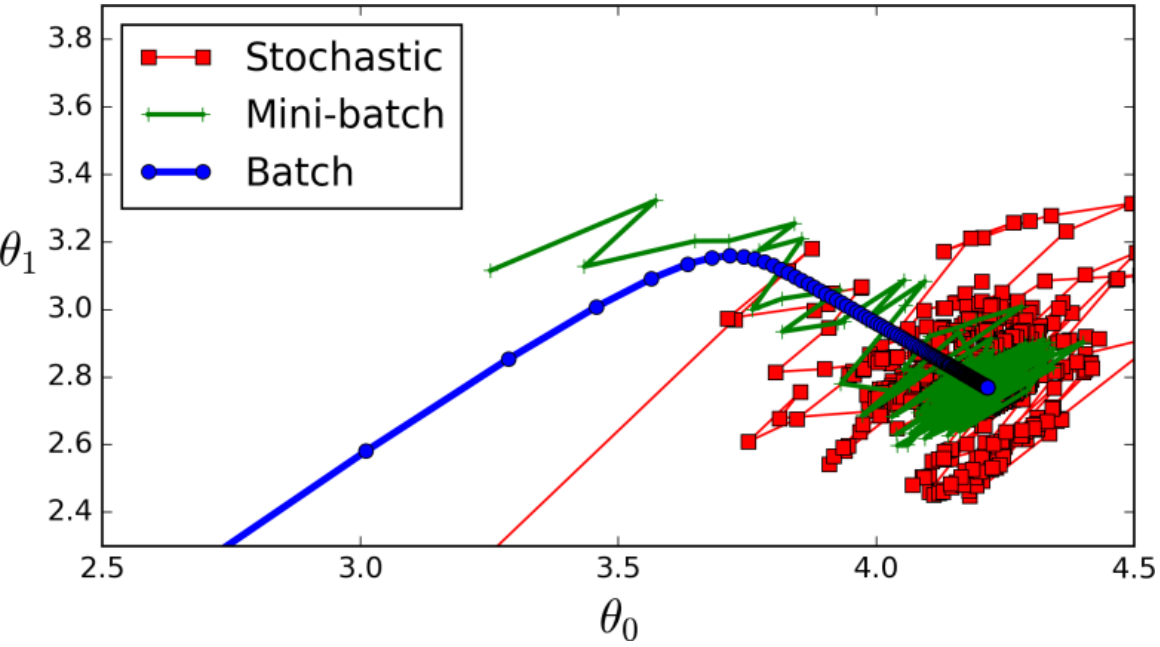
- Analogy: Ball rolling downhill (along the cost function)
  - Builds up momentum if subsequent gradients point into same direction ➔ doesn't slow down immediately when it's flat
  - Dampen velocity according to friction coefficient ß (e.g., 0.9)
  - Increase velocity in direction of negative gradient
  - Move according to velocity
  - Issues:
    - Often results in oscillations and instability in high-curvature regions
    - How to pick the damping coefficient?

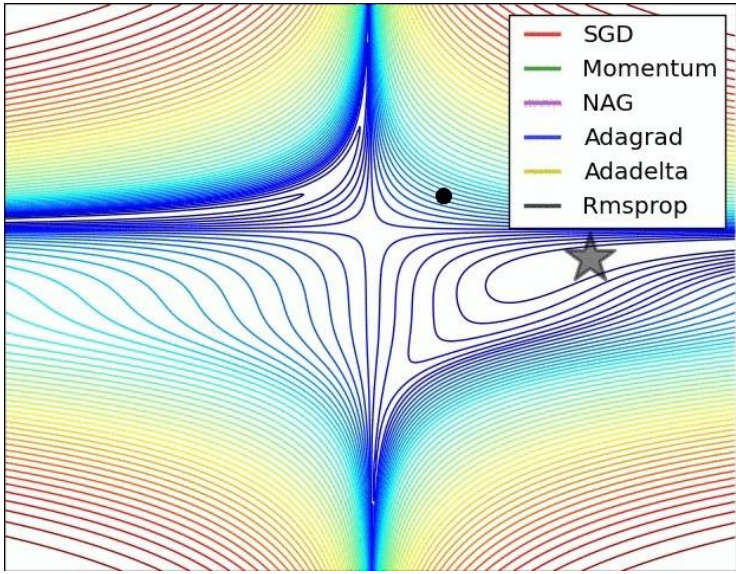
Adam (Adaptive Moment Estimation)

- Optimization methods that adapt the learning rate **for each parameter**
  - Compute velocity (first moment): exponential moving average over past gradients (as before)
  - Compute second moment estimate: exponential moving average over past gradient magnitudes
  - Move according to velocity, divided by second moment

- Intuition: counter notoriously **small** gradients by **upscaling**, and **large** gradients by **downscaling**
  - Separately for each weight

Optimization: Many Flavours





Outline

- Short Recap
- Deep Learning: Initial Summary
- Transfer Learning
- Ensemble Learning

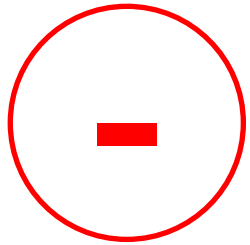
Deep Learning: first conclusions

- Deep Learning is a branch of machine learning where the learnable function is “deep”
  - A stack of (nonlinear) functions
- Neural Networks are ***decades old***
  - Main enablers for deep learning: faster computers (+ GPUs) and bigger datasets
    - plus some practical tweaks that nobody would have found without fast computers & large datasets
  - Added a few more elements, such as convolutions, pooling, etc...
- Very Active area of research!

Deep Learning: first conclusions

- Deep Neural Networks
  - Benefitted a lot from recent progress in hardware
  - Are applicable to a wide range of tasks
  - Require less domain knowledge (for feature design)
  - Often outperform “hand-crafting” or feature design (images!)

- Potential drawbacks:
  - Open up new design spaces (network architecture, training data, training method) which may be less well understood
  - Large networks usually require large data sets (and lots of computing power / electricity)
  - A lot of experimenting & parameter tweaking needed
  - Trained network is often hard to analyze (“black box”)



ML success stories: Image Classification

- Surpassing human-level performance on some tasks, e.g. ImageNet classification

Delving Deep into Rectifiers:
Surpassing Human-Level Performance on ImageNet Classification

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun

Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

- Why / how?

Software Frameworks

Python:

“low”-level:

- Theano
- TensorFlow (Google)

high-level:

- Keras
- Lasagne
- Blocks
- Theanets
- Pretty Tensor (Google)
- ...

C++ / others*:

- Caffe
- Torch
- cuda-convnet2
- Chainer
- MXNet
- CNTK (Microsoft)
- Leaf (AutumnAI)
- ...

* (with bindings for other languages)

Hardware: GPU recommended! Most libraries support GPUs.

Longevity of ML Code

ML/DL Frameworks in 2017

Python:

"low"-level:

- Theano — Development stopped after v1.0, Nov 2017
- TensorFlow (Google) — TF1 vs TF2 ....

high-level:

- Keras — Not cross-backend anymore, only TensorFlow
- Lasagne — Inactive since mid 2018
- Blocks — Builds on Theano, same developers
- Theanets — Builds on Theano, inactive since 2018
- Pretty Tensor (Google) — Inactive since 2017
- ...

- **PyTorch (Facebook)**

C++ / others:

- Caffe — Inactive since 2017
- Torch — Inactive, merged with Caffee2 → PyTorch
- cuda-convnet2 — Inactive since 2017?
- Chainer — Supporting Torch since 2017
- MXNet (Apache)
- CNTK (Microsoft) — Inactive since 2019
- Leaf (AutumnAI) — Inactive since 2018
- ...

- **Azure AI** (MicroSoft)
- **ONNX (MicroSoft) (Open Neural Network Exchange)**

Deep Learning – still to come

- Transfer Learning
- Recurrent Neural Networks
- Explainable AI
- Robustness / Attacks on Deep Learning
- ....

More Deep Learning in Vienna

- Vienna Deep Learning Meetup
  - ~ Once per month
  - Mix of academic and industry talks
    - Google Developers on Tensorflow, ...
    - Seznam.cz/ (“Czech Google”)
    - Stability.ai ➔ Stable Diffusion
  - Very popular (~100-150 participants)

  - http://meetup.com/Vienna-Deep-Learning-Meetup/
    - It’s for free **&** (most of the times) you will get drinks and snacks ☺
    - And meet other people working in the field!

(They are back to face2face meetups ☺)

Outline

- Short Recap
- Deep Learning: Convolutional Neural Networks
- Transfer Learning
- Ensemble Learning

Neural Networks: Design Choices

- *Designing a deep learning network is difficult !*
  - Which architecture to chose?

- *How can we still do deep learning, especially with limited resources (computing, data, HR, ...)?*
  - Train well-known architectures, might work for your task
  - Pre-trained models are available
    - Tensorflow provides e.g. ResNet, AlexNet, ...
    - Can often be applied
      - Out of the box
        - » Have been trained on big datasets
        - » ➔ Generic enough for many application domains
      - After fine-tuning training on specific dataset
        - » *Transfer learning*

Transfer learning

- Assumption: Can not do deep learning unless you have a many labelled examples for your problem
  - Many models have millions of parameters to learn!

- However:
  - Can learn useful representations (“features”) from unlabelled data
  - Can train on a nearby surrogate objective for which it is easy to generate labels

  ➔ Can transfer learned representations from a related task

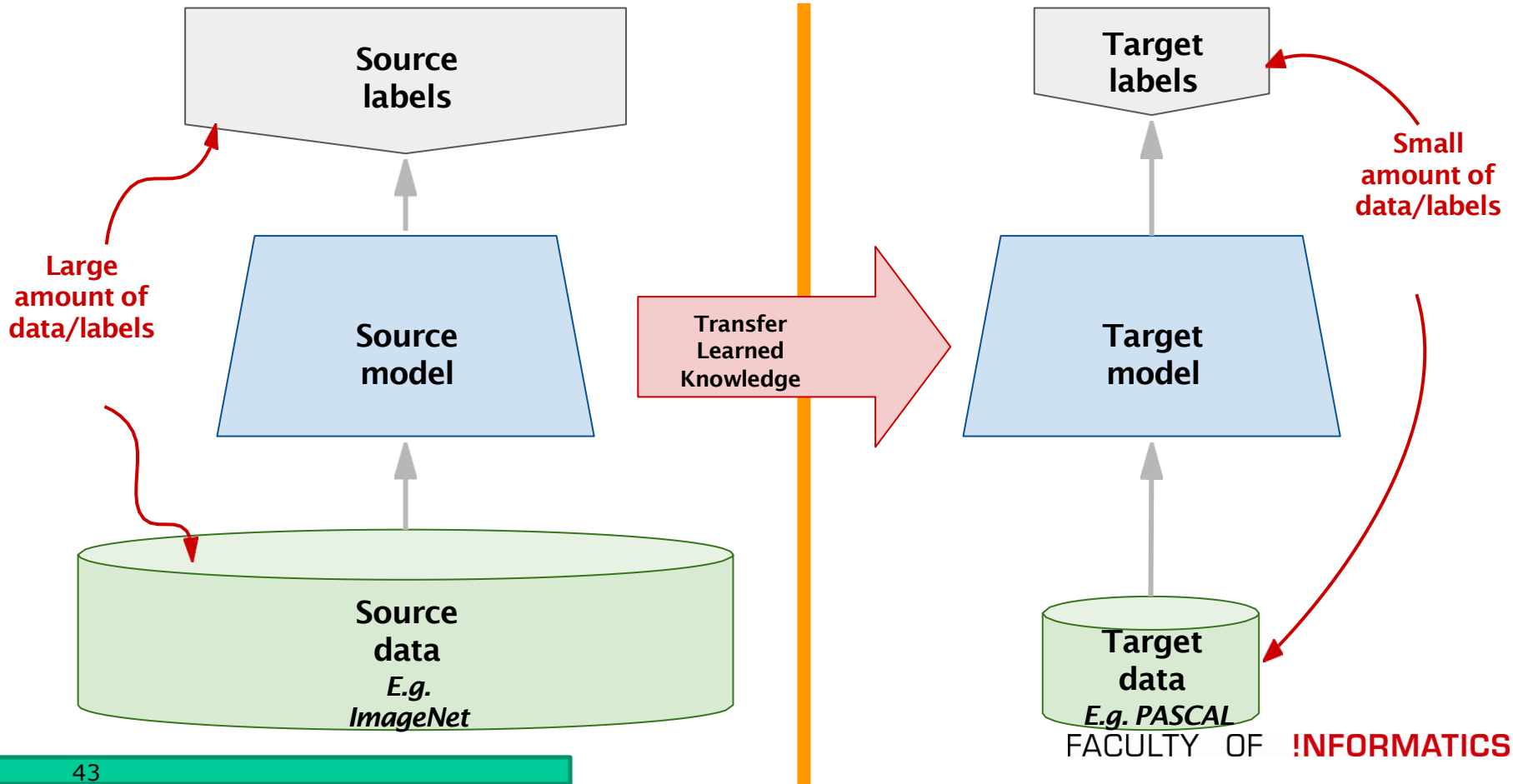
Transfer learning: idea

- Instead of training a deep network from scratch for a specific task
  - Take a network trained on a different **domain** for a different source **task**
  - Adapt it for your domain and your target task

- Variations
  - Same domain, different task
  - Different domain, same task

Transfer learning: idea

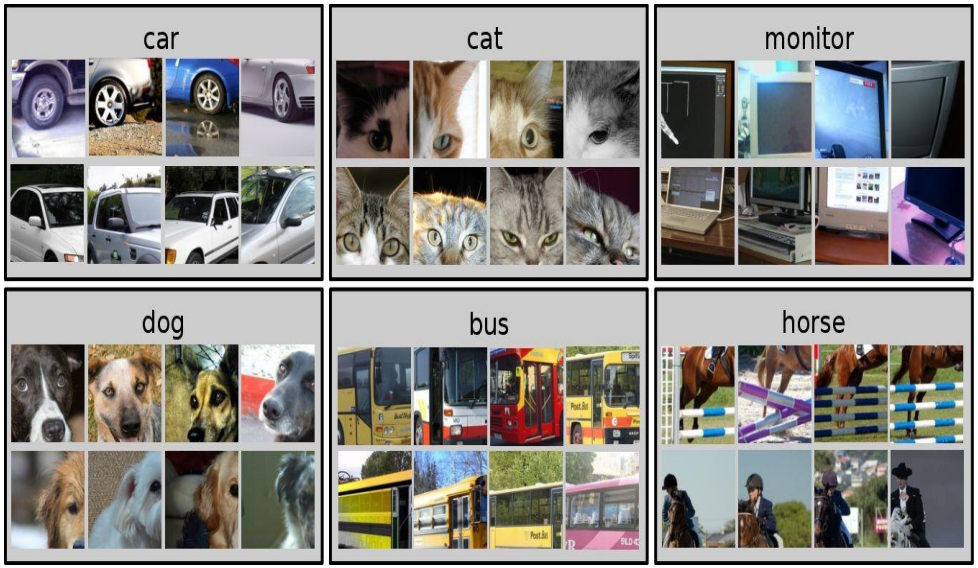
- Train network on similar task for which it is easy to get labels (e.g. ImageNet)
- Transfer to task with few data / labels



Example: PASCAL VOC 2007

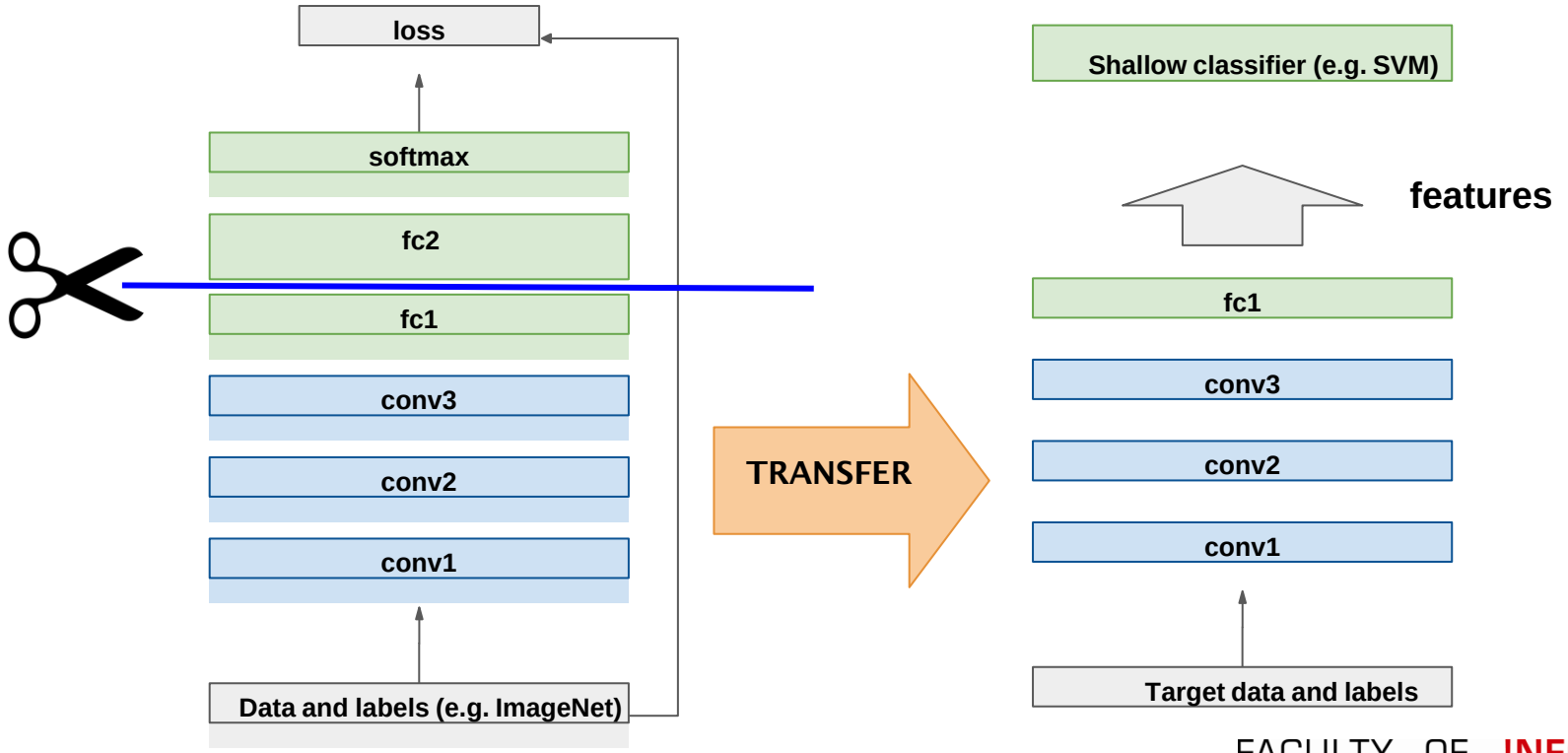
- Benchmark image classification dataset → but small!
  - 20 classes, ~**10K images**, 50% train, 50% test
- Deep networks can have many parameters (e.g. 60M in Alexnet)
- Direct training (from scratch) using only 5K training images problematic

  ➔ Model overfits too much



“Off-the-shelf“ Transfer Learning

- Network trained on different (but similar) task
- Use output of one/more layers as generic *feature detectors*
- Train *shallow* model on these features.



"Off-the-shelf" Transfer Learning

- Works surprisingly well in practice
  - Surpassed / on par with state-of-the-art in several tasks
  - Image classification
    - PASCAL VOC 2007
    - Oxford flowers
    - CUB Bird dataset
    - MIT indoors
  - Image retrieval
    - Paris 6k
    - Holidays
    - UKBench

➔ Can we do better?

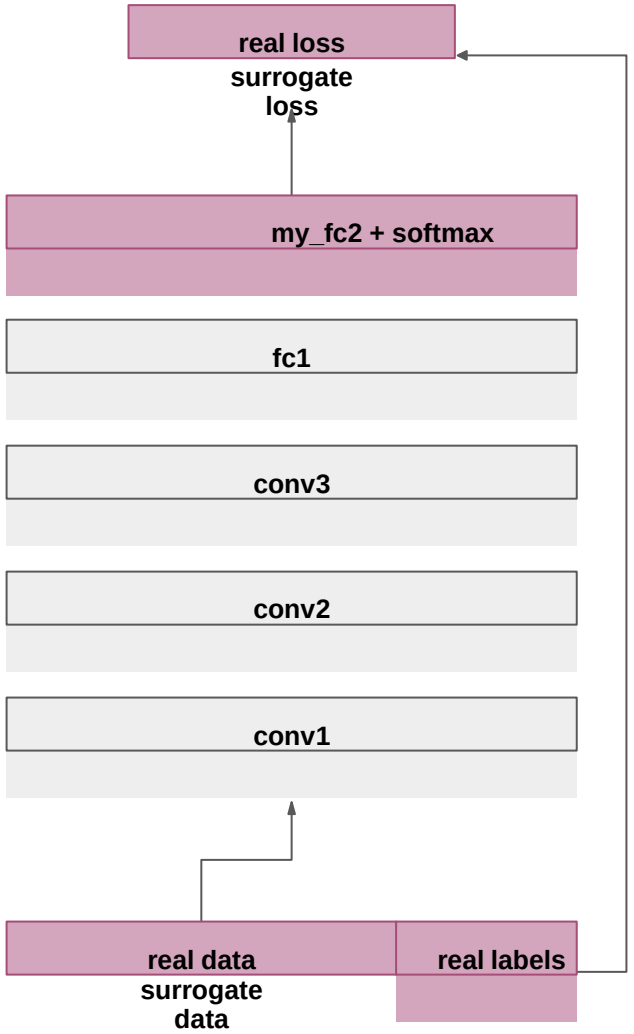
Method	mean Accuracy
HSV [27]	43.0
SIFT internal [27]	55.1
SIFT boundary [27]	32.0
HOG [27]	49.6
HSV+SIFTi+SIFTb+HOG(MKL) [27]	72.8
BOW(4000) [14]	65.5
SPM(4000) [14]	67.4
FLH(100) [14]	72.7
BiCos seg [7]	79.4
Dense HOG+Coding+Pooling[2] w/o seg	76.7
Seg+Dense HOG+Coding+Pooling[2]	80.7
CNN-SVM w/o seg	74.7
CNNaug-SVM w/o seg	**86.8**

Oxford 102 flowers dataset

Razavian et al, CNN Features off-the-shelf: an Astounding Baseline for Recognition, CVPRW 2014 http://arxiv.org/abs/1403.6382

Fine-tuning: supervised task adaptation

- Train network on similar task
  - For which it is easy to get labels
  - E.g. ImageNet classification
- Cut off top layer(s) of network
  - Replace with supervised objective for target domain
- Fine-tune network
  - With labels for target domain
  - Until validation loss starts to increase



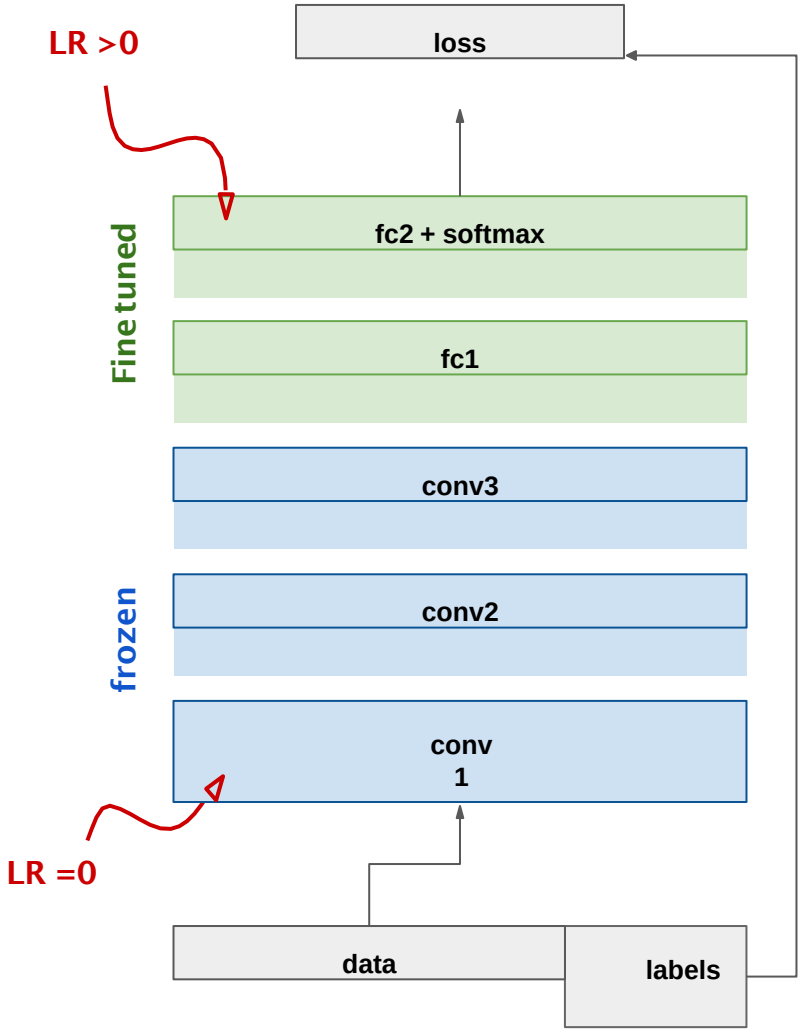
Freeze or fine-tune?

- First ***n*** layers can be frozen or fine tuned
  - Frozen: not updated during training
  - Fine-tuned: updated

➔ Depends on target task

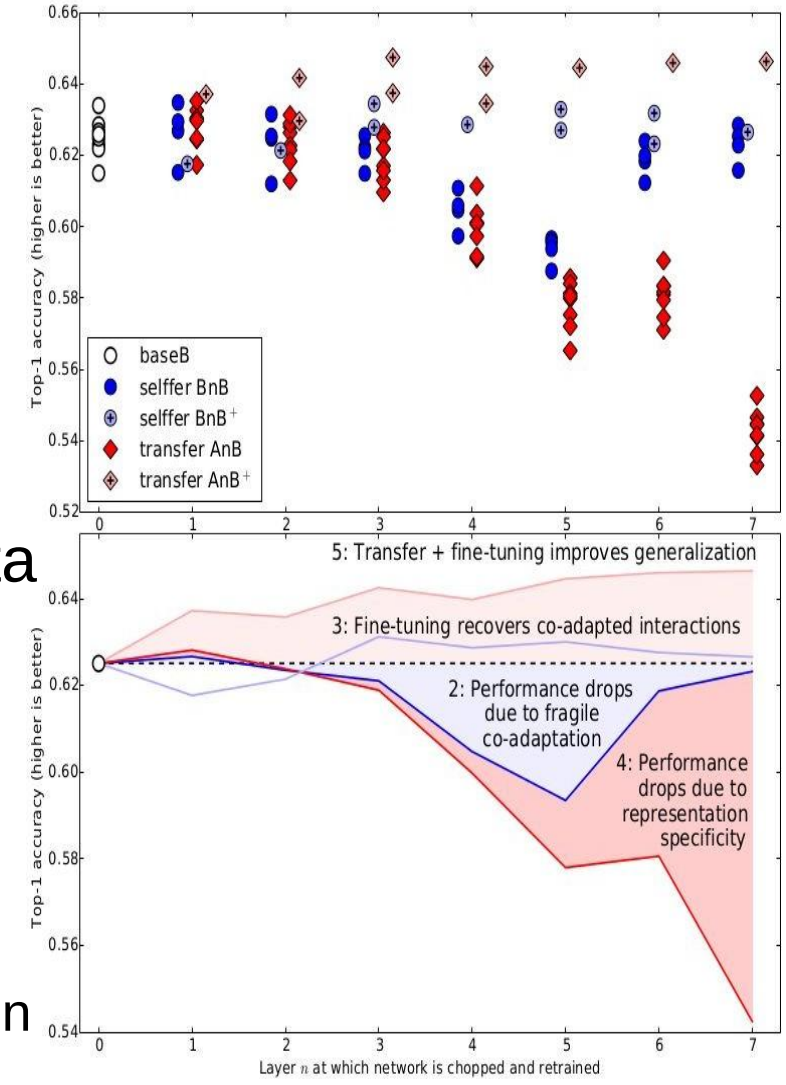
  - Freeze: target task labels are scarce, want to avoid overfitting
  - Fine-tune: target task labels are more plentiful
  - In general: can set learning rates (LR) differently for each layer

  ➔ E.g. less fine-tuning for earlier layers



How transferable are features?

- Lower layers
  - More general features
  - Transfer very well to other tasks
- Higher layers
  - More task specific; less transferable
- Fine-tuning improves generalization when sufficient data available
- Transfer learning + fine tuning often better than training from scratch on the target dataset
  - Even features from distant tasks often better than random initial weights

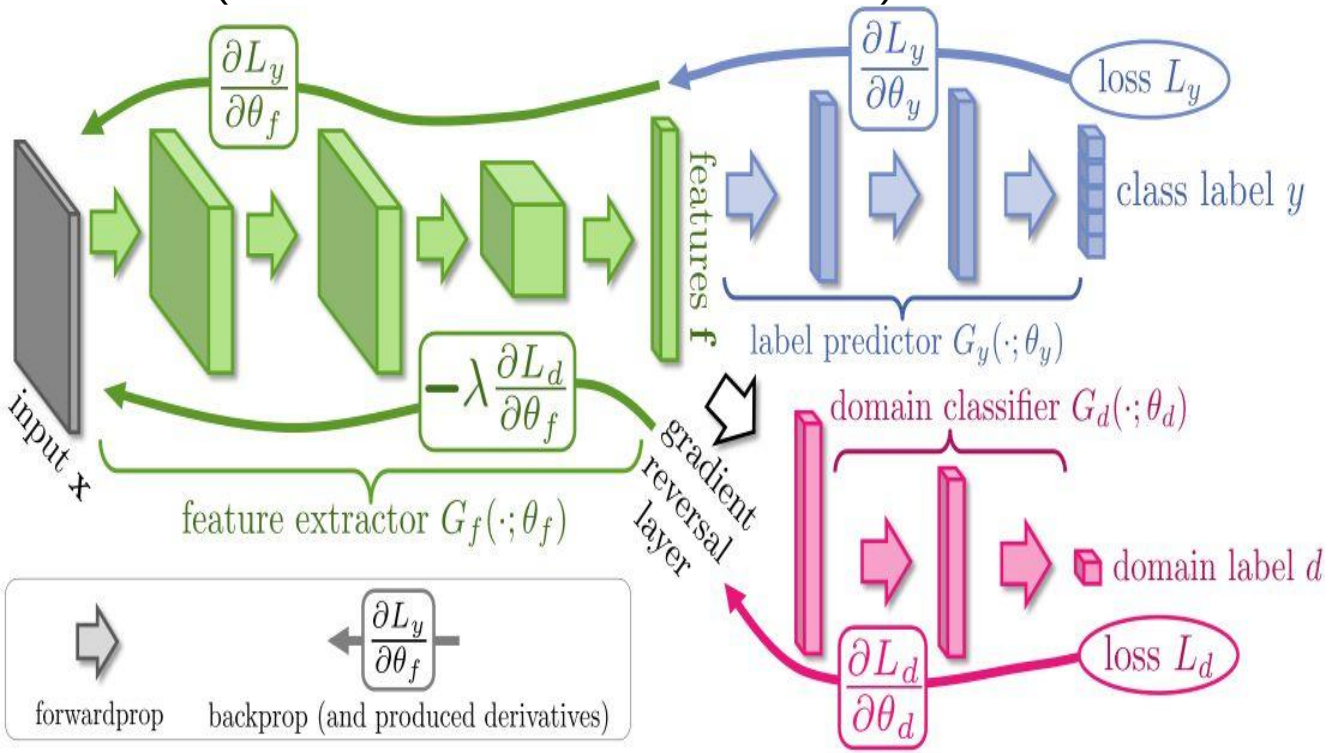


Yosinki et al. How transferable are features in deep neural networks. NIPS 2014

Unsupervised domain adaptation [Excursus]

- Domain adaptation possible also without labels in target set

Feature extractor (green) &
label predictor (blue)
(standard feed-forward architecture)



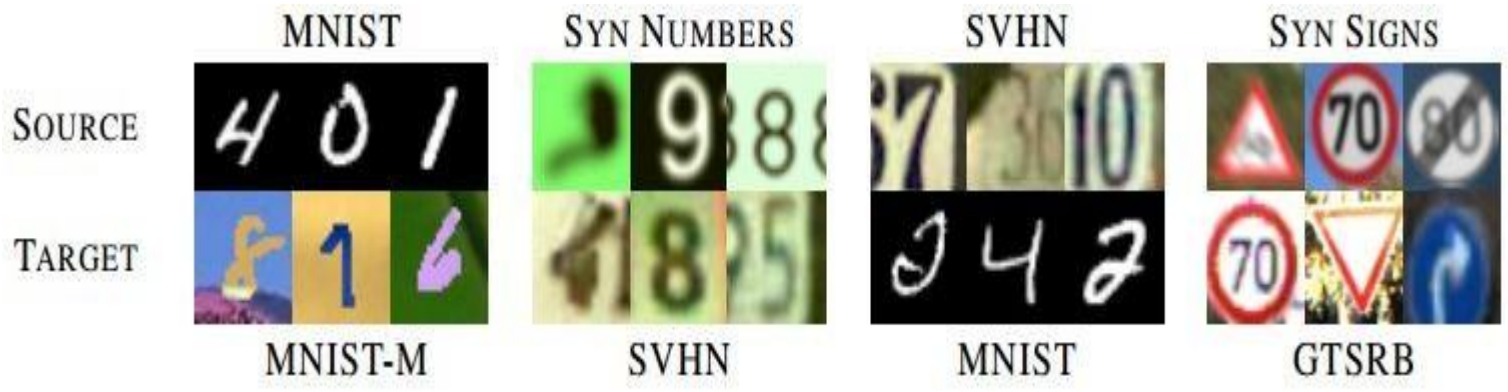
Domain classifier (pink) distinguishes between source/target

Training minimizes
- Label prediction loss (source examples) and
- Domain classification loss (for all samples)

- Gradient reversal ensures that **feature distributions over the two domains are made similar ➔ domain-invariant features.**

Y Ganin and V Lempitsky, Unsupervised Domain Adaptation by Backpropagation, ICML 2015

Unsupervised domain adaptation [Excursus]



Source only	57.49	86.65	59.19	74.00
SA (Fernando et al 2013)	60.78 (7.9%)	86.72 (1.3%)	61.57 (5.9%)	76.35 (9.1%)
Ganin et al (2015)	81.49 (57.9%)	90.48 (66.1%)	71.07 (29.3%)	88.66 (56.7%)
Train on target (goal)	98.91	92.44	99.51	99.87

Y Ganin and V Lempitsky, Unsupervised Domain Adaptation by Backpropagation, ICML 2015

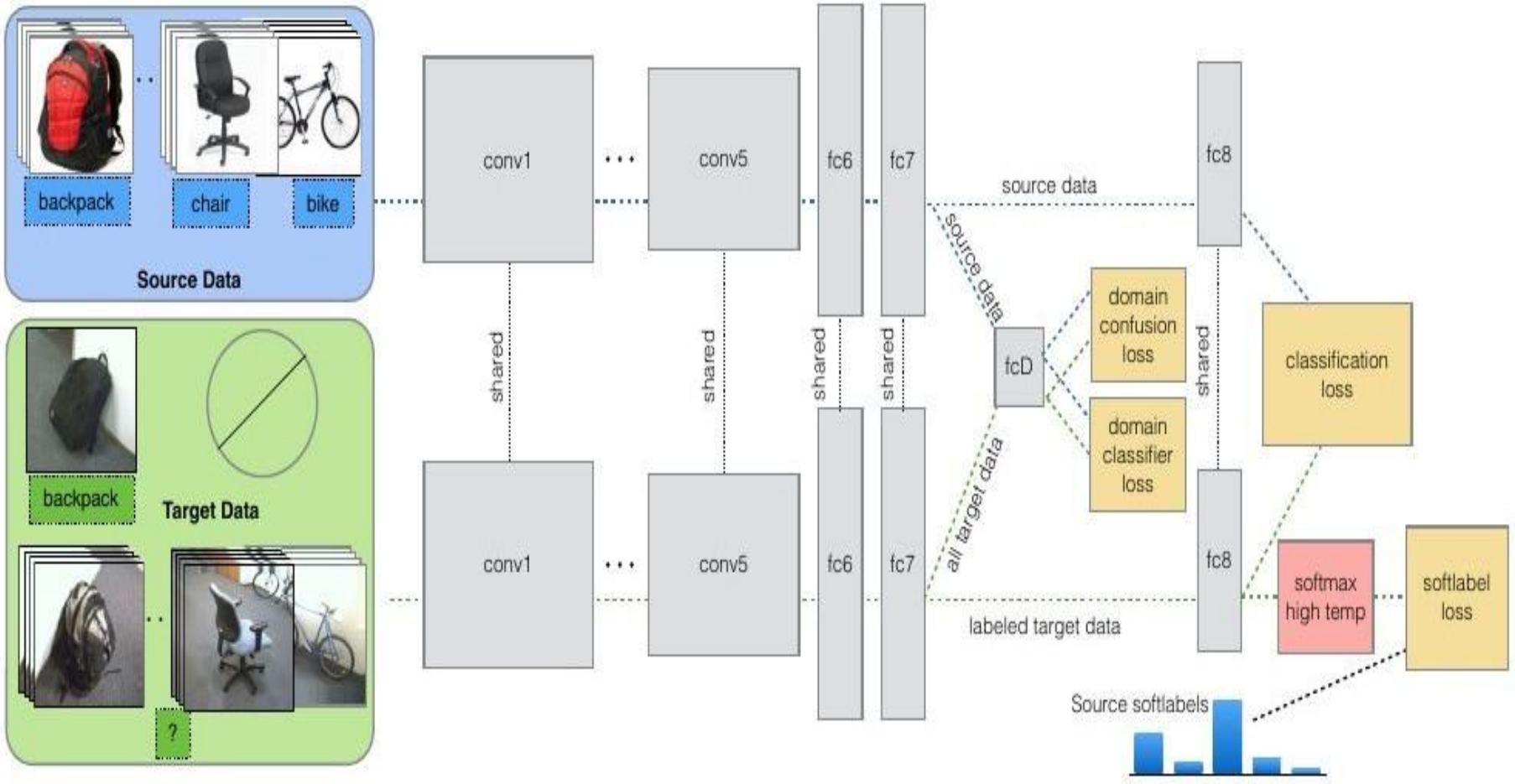
Semi-supervised domain adaptation [Excursus]

- When *few* labels are available in target domain
  - Use them for domain adaptation
  - I.e. combine fine tuning & unsupervised domain adaptation.

- Tzeng et al. try to simultaneously optimize a loss that maximizes
  - Classification accuracy on both source and target datasets
  - Domain confusion of a domain classifier
  - Agreement of classifier score distributions across domains

Tzeng, Eric, et al. Simultaneous deep transfer across domains and tasks. ICCV. 2015.

Semi-supervised domain adaptation [Excursus]



Tzeng, Eric, et al. Simultaneous deep transfer across domains and tasks. ICCV. 2015.

Transfer Learning – Conclusions

- Possible to train large models on small data
  - By using transfer learning and domain adaptation
- Off the shelf features work very well in various domains and tasks
  - Lower layers of network contain very generic features (higher layers more task specific features) that can be used as features for shallow classifiers

- Supervised domain adaptation via fine tuning *almost* always improves performance
- Possible to do un/semi-supervised domain adaptation by matching feature distributions

Deep Learning – still to come

- Recurrent Neural Networks
- Explainable AI
- Robustness / Attacks on Deep Learning
- ....

Outline

- Short Recap
- Deep Learning: Convolutional Neural Networks
- Transfer Learning
- Ensemble Learning

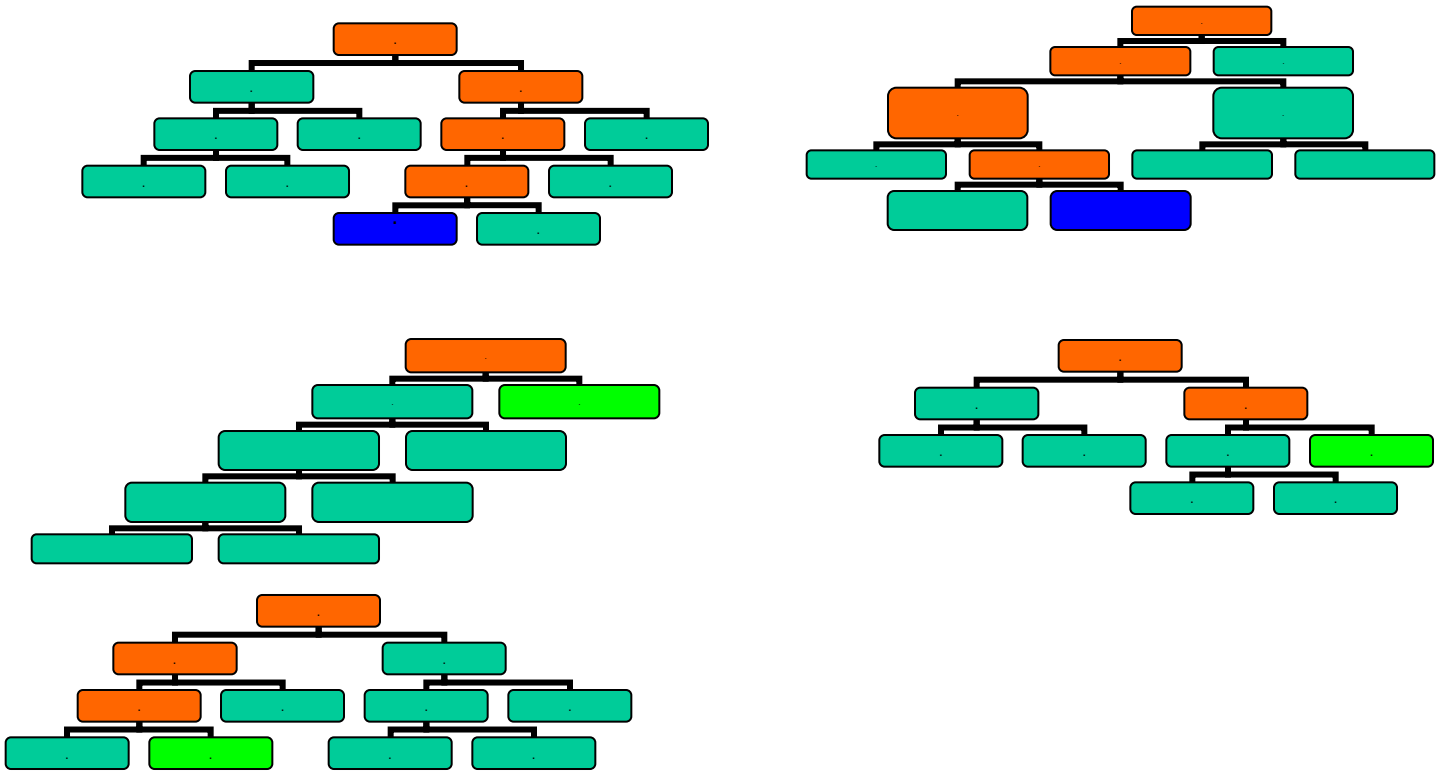
Ensemble learning

- Relatively new area in Machine Learning

- Ensembles combine several classifiers
  - Goal: improved performance
  - E.g. in terms of accuracy compared to (best) single classifier

- We already heard one ensemble classification technique..
  - Which one?

Ensemble learning



Ensemble learning – Random Forests

- Random Forests are one type of an ensemble technique
  - Train several (weak) models (decision trees)
  - Combine the trees by **majority voting** to get prediction

- RF use the general ensemble pattern
  - Train several classifiers (different algorithms, parameters, …)
  - Combine the individual predictions to get overall prediction according to some rule

Ensemble learning vs. model selection

- Model / algorithm selection
  - Train several classifiers (different algorithms, different parameters, …)
  - Finally, select ***one*** best classifier for prediction

- Ensemble classifier
  - Use ***all*** (or, at least, many) trained classifiers
  - Combine predictions

Majority voting

- Pick the class that is favoured by most
- Different approaches
  - Unanimity – only if all votes agree (otherwise: no result)
  - (Absolute) majority – if more than 50% of the votes agree (otherwise: no result)
  - Majority (*Plurality*): select best class, i.e. class w_k if

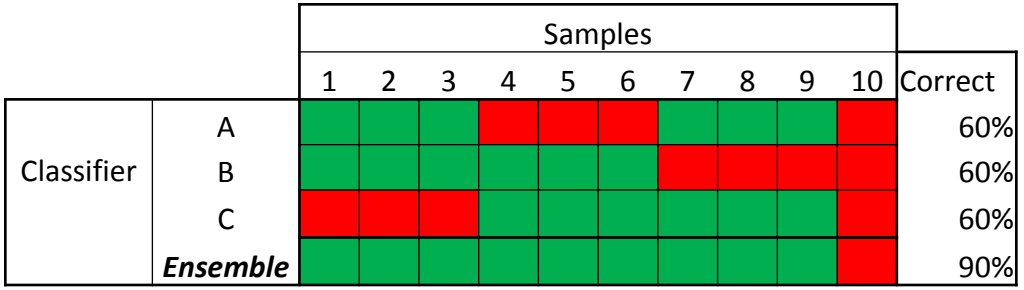
$$\sum_{i=1}^{L} d_{i.k} = \max_{j=1}^{c} \sum_{i=1}^{L} d_{i,j}$$

L = # classifiers; $d_{i,k}$ = 1 if classifier i predicts class k

- *When is the answer of the ensemble correct?*
  - If at least *([L/2] +1)* individual classifiers are correct

Extremes of majority voting accuracy

- 3 classifiers, each 60% correct
- **Ideal pattern (*pattern of success*)**
  - 3 x 30% cases where 2 classifiers correct & 1 wrong
  - 10% of the cases all classifiers wrong at the same time
  - i.e. for 10 items in the test set *(1 == correct, 0 == wrong)*
    - 3 x each of 011, 110, 101 patterns
    - 1 x 000 pattern



  - In this case, 90% accuracy in the ensemble
  - ***Nice improvement!***

Extremes of majority voting accuracy

- 3 classifiers, each 60% correct
- **Worst pattern (*pattern of failure*)**
  - 40% cases all three are correct
  - 3 x 20% cases: 1 classifier is correct, 2 are wrong
  - i.e. in a test set with 10 samples
    - 4 x 111
    - 2 x each of 001, 100, 010

		Samples										
		1	2	3	4	5	6	7	8	9	10	Correct
Classifier	A											60%
	B											60%
	C									1	1	60%
	Ensemble											40%

  - In this case, only 40% correctness in the ensemble!
  - ***Bad degradation !***

Potential of majority voting accuracy

- Estimated accuracy of ensemble can be computed (assuming independence)

$$P_{maj} = \sum_{m=\lfloor L/2 \rfloor + 1}^{L} \binom{L}{m} p^m (1-p)^{L-m}$$

L = # classifiers; p = accuracy

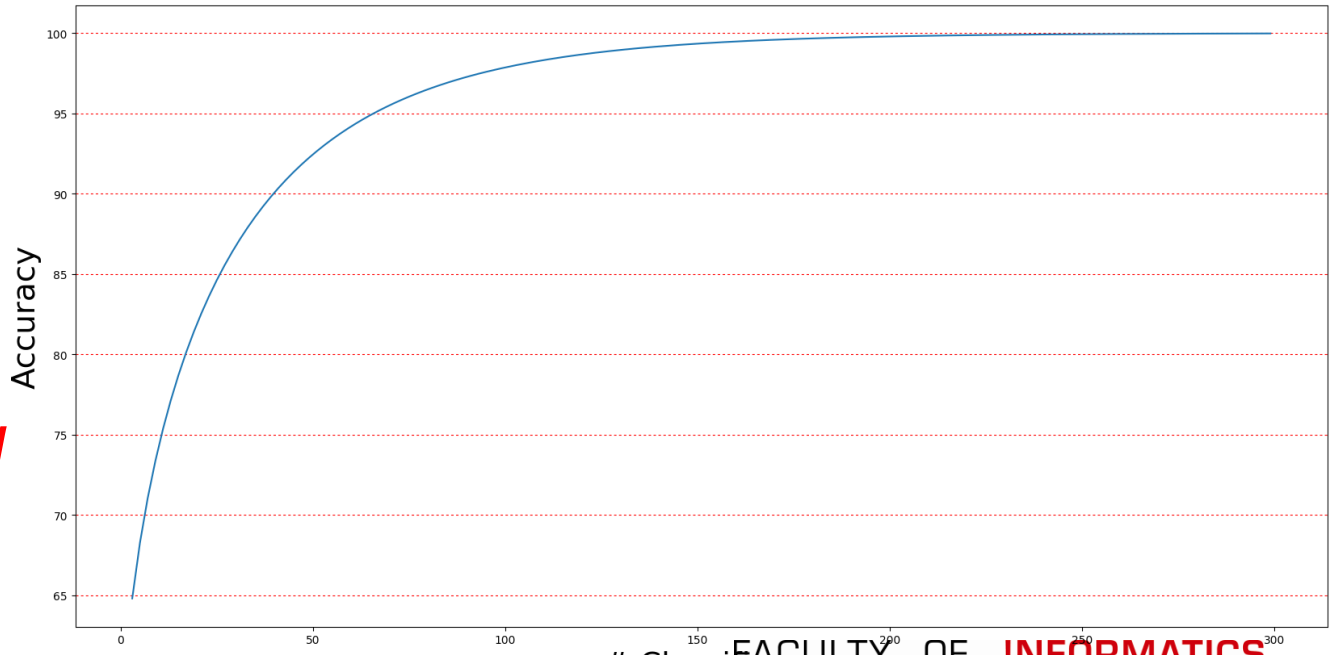
- Example: 3 classifiers, each 60% accuracy
  - Combined accuracy: 64.8%

    *(3 * 0.6^2 * 0.4 + 1 * 0.6^3 * 1 = 0.432 + 0.216 = 0.648)*
    - l=5: 68.26%
    - l=7: 71.02%
    - l=9: 73.34%

Potential of majority voting accuracy

- Example: l classifiers, each 60% accuracy
  - l=3: combined accuracy = 64.8%
  - l=5: combined accuracy = 68.26%
  - l=7: combined accuracy = 71.02%
  - l=9: combined accuracy = 73.34%
  - *Trend?*

- *Why is this not working in practice?*



Other Limits of Majority Voting

- Classifiers don't have the identical accuracies

- Example
  - 3 classifiers with 60%, 60% and 70% accuracies
  - Expected ensemble accuracy according to previous equation: ~69.6% (which is already an optimistic estimate!)
  - It would be more beneficial to just use the one with 70%!

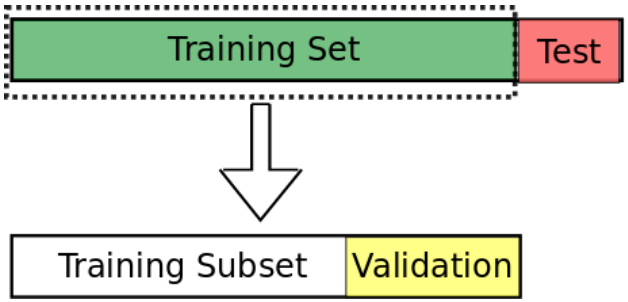
➔ How to make voting more "robust" to these settings?

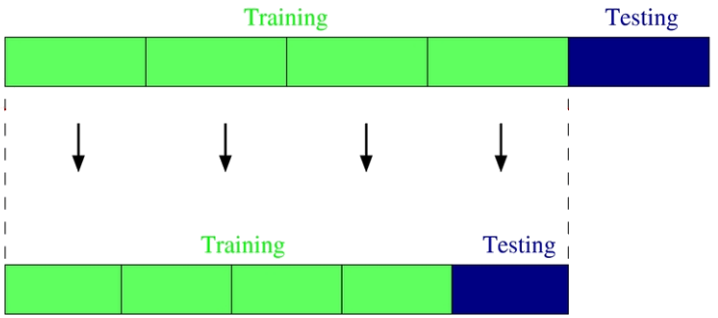
Weighted Majority Voting

- If the classifiers don't have the identical accuracies
  - More power/say to the "better" ones
- Add weights to plurality vote: select class w_k if

$$\sum_{i=1}^{L} b_i d_{i.k} = \max_{j=1}^{c} \sum_{i=1}^{L} b_i d_{i,j}$$

- *How to obtain weights?*
  - Weights can e.g. be tested accuracies
    - *Which set to obtain them from?*





Types of classifier outputs

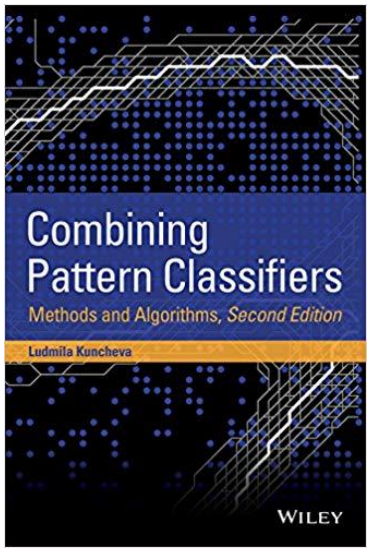
- Type 1 – Abstract level
  - Classifiers produce only class label/prediction
  - Most universal one
- Type 2 – Rank level
  - Classifiers produce ranked list of probable class labels
- Type 3 – Measurement level
  - Classifiers produce probability for each class

Types of classifier outputs

- Type 1 (class) ← 2 (rank) ← 3 (probability)
- *Why are classifier outputs important for ensemble learning?*
- Types of output influence combination possibilities
  - The more information on output, the more advanced combination possibilities
  - Type 1 suitable for majority voting

Ensemble Learning

- Many more combinational rules exist
  - E.g. using the probability of classes
- Methods to select only some models
  - Using Pareto optimality, only those models that are not dominated by others
- Further reading
  - e.g. *“Combining Pattern Classifiers”, Ludmila I. Kuncheva*



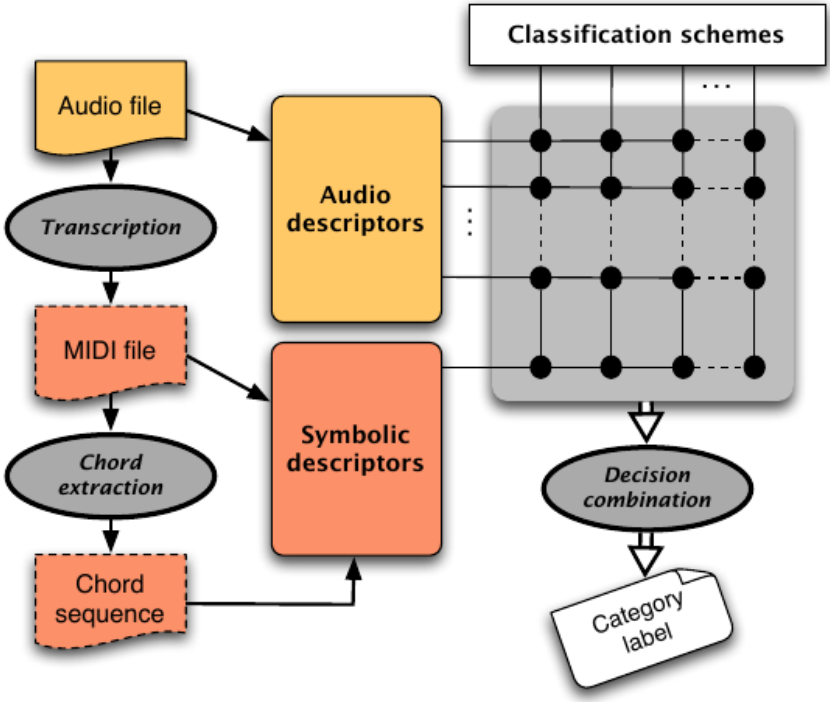
Heterogeneous vs. Homogenous

- Heterogeneous: different types of classifier models are trained
  - E.g. decision tree & k-NN & SVM
- Advantage: explore different *bias* of classifiers
  - combines e.g. hyperplanes and local neighbourhoods
  - However, rarely used
- Homogeneous ensemble. *Example?*
  - e.g. Random Forests

2D heterogeneous ensemble

- Different models, different feature sets

- Evaluation
  - ~100 different combinations
  - Different combination rules



Corpus	*Single best*	*Ensemble*	*Comb. rule*
9GDB	78.15 (SVM-Puk/TSSD)	**81.66**	AVG
GTZAN	72.60 (SVM-lin/SSD)	**77.50**	QBWWV
ISMIRgenre	81.28 (SVM-quad/TSSD)	**84.02**	QBWWV
ISMIRrhythm	87.97 (SVM-lin/RP)	**89.11**	BWWV

- Feature Selection in a Cartesian Ensemble of Feature Subspace Classifiers for Music Categorisation, ACM MML 2011

Bagging

- **B**ootstrap **AGG**regat**ING**
- Ensemble consists of classifiers built on bootstrap replicates of the training set

- Class decision by plurality vote
- Works best if classifiers unstable / high variance
  - i.e. small variations in the training set imply larger changes in the trained model & output
  - Otherwise, ensemble will consists of almost identical classifiers

Bagging

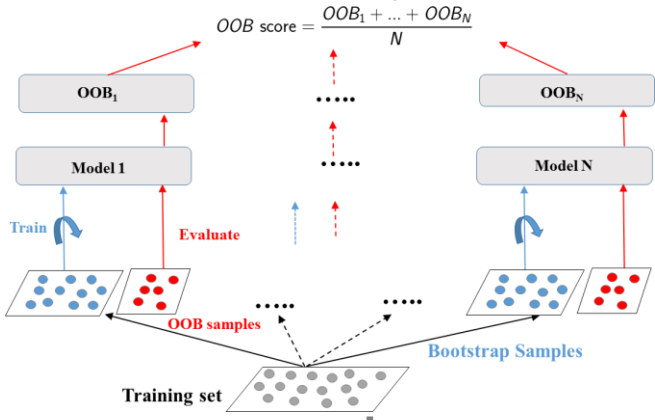
- If classifiers are independent, and have the same individual accuracies
  - Ensemble vote will be superior
- Bagging tries to achieve independent classifiers by varying the training set

- Example classifier?
  - *Random forests make use of bagging*

- Bagging classifiers are learned in parallel
- Bagging tries to decrease variance of ensemble

Recap: Bagging Evaluation

- *How to evaluate effectiveness?*
  - Holdout, cross-validation, etc..
- Alternatively: out-of-bag error / estimate (OOB)
  - General method for bootstrapped algorithms
  1. Compute mean error on each training sample x_i, using only trees that did not have x_i in their bootstrap sample
  2. Aggregate over each of the *n* sets



- Empirical evidence: OOB as accurate as using test set of the same size as training set

➔ Out-of-bag error removes need for a dedicated test set

Boosting

- Until now: classifiers were trained w/o knowing of each other (not building on top of each other)

- **Boosting**: classifier improves on predecessor(s)
  - Analyse errors from predecessor, decide on which “part of the data” to focus on
  - Focus on “hard” examples / mistakes
    - i.e. the ones that could not be classified by the previous model(s)

- Boosting tries to decrease bias of classifiers

Outline

- Short Recap
- Deep Learning: Convolutional Neural Networks
- Transfer Learning
- Ensemble Learning: AdaBoost

Ensemble learning – AdaBoost

- Boosting: Iterative/sequential ensemble technique
  - Next classifier tries to fix prediction mistakes so far

- Popular algorithm: AdaBoost:
  - **Ada**ptive**Boost**ing
  - Proposed 1995 (Yoav Freund & Robert Shapire)
  - Focus: assigning weights to the data samples
    - ➔ Rather implicit focus on “hard”/”difficult” examples

Ensemble Learning – AdaBoost

- Pool of classifiers *h*
  - **homogenous** or heterogeneous
- Assign weights to training samples
  - Initially equal weights 1/n
- Number of iterations T

- Final classifier *H* is a (linear) combination of the individual classifiers *h* (each weighted by α)
  - E.g. output +1 / -1 (binary):

$$H(x) = sign(f(x)) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Ensemble Learning – AdaBoost

$$H(x) = sign(f(x)) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

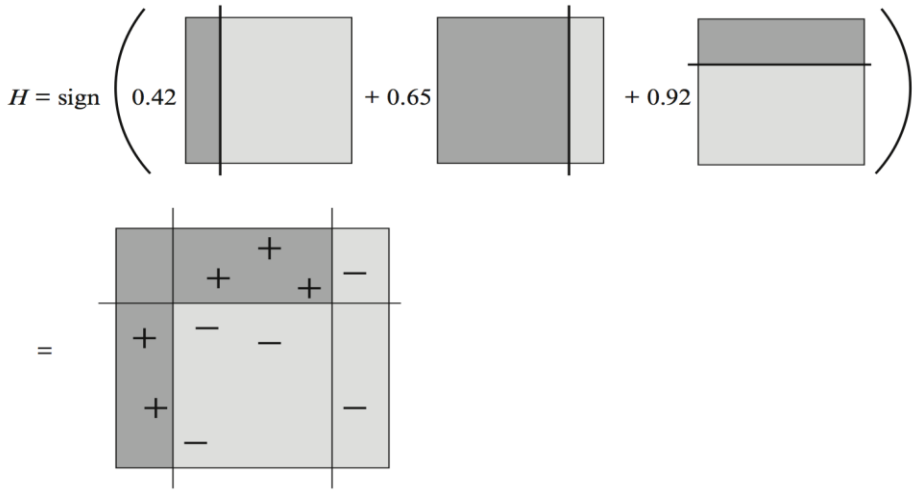
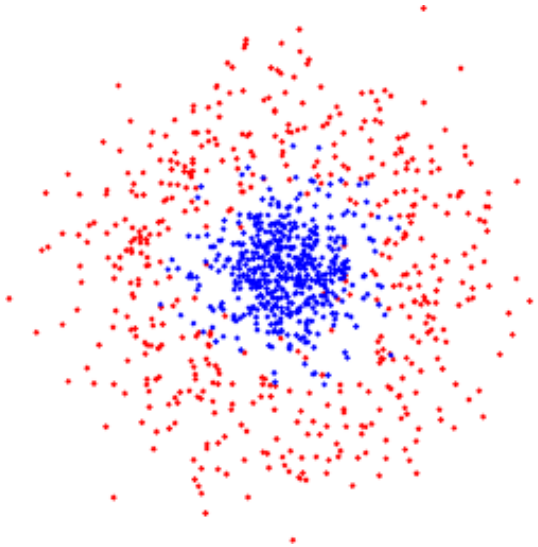


Figure: AdaBoost [Schapire and Freund, 2012]

Ensemble Learning – AdaBoost

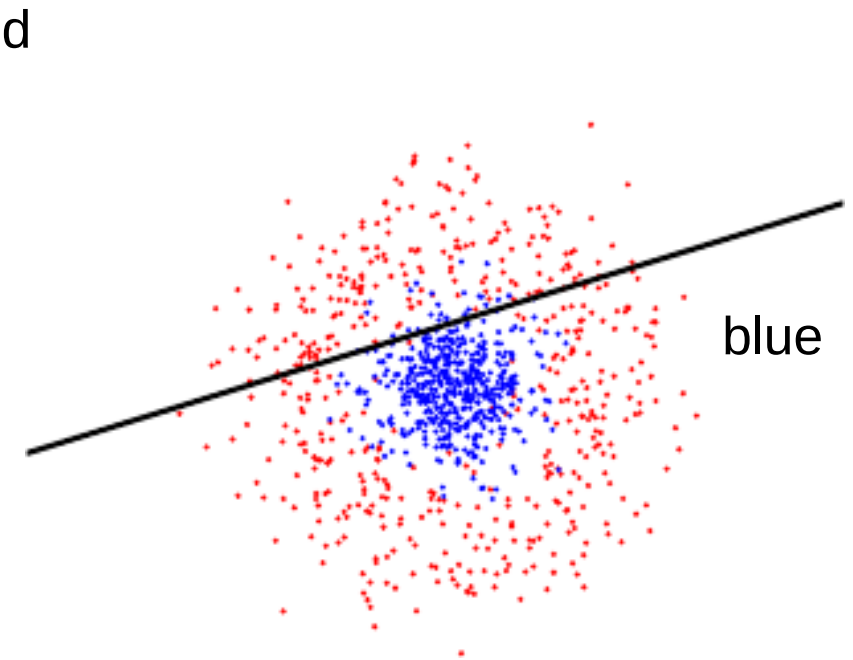
- Example with perceptron as (weak) learner

- Data set
  - Two Gaussians, overlapping



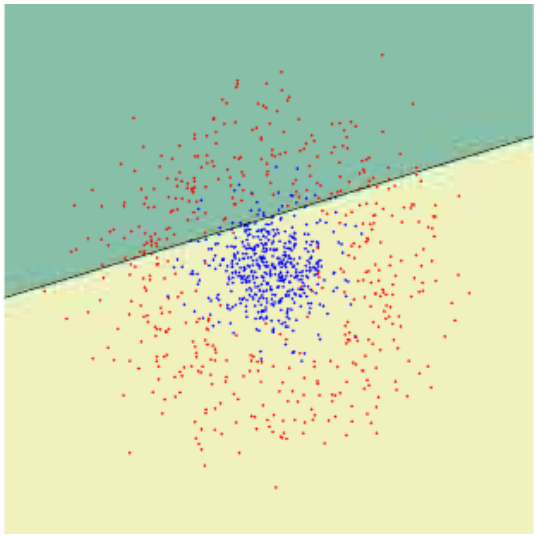
Ensemble Learning – AdaBoost

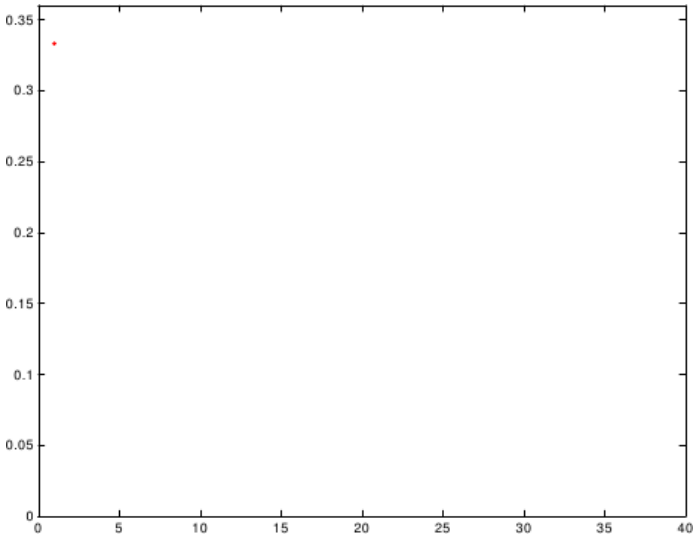
- Iteration 1:
  - Train many perceptrons on training set
  - Select the one with the lowest error (no weighting yet)



Ensemble Learning – AdaBoost

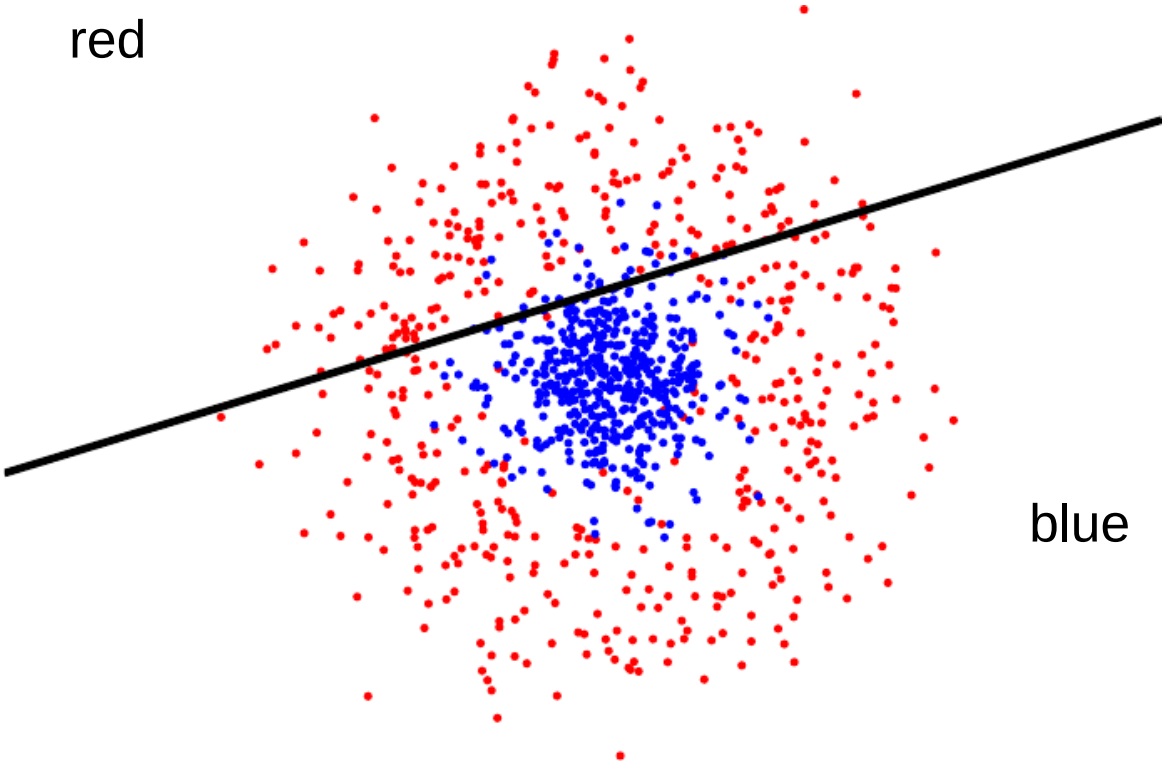
- Iteration 1:
  - Train a perceptron on training set





Ensemble Learning – AdaBoost

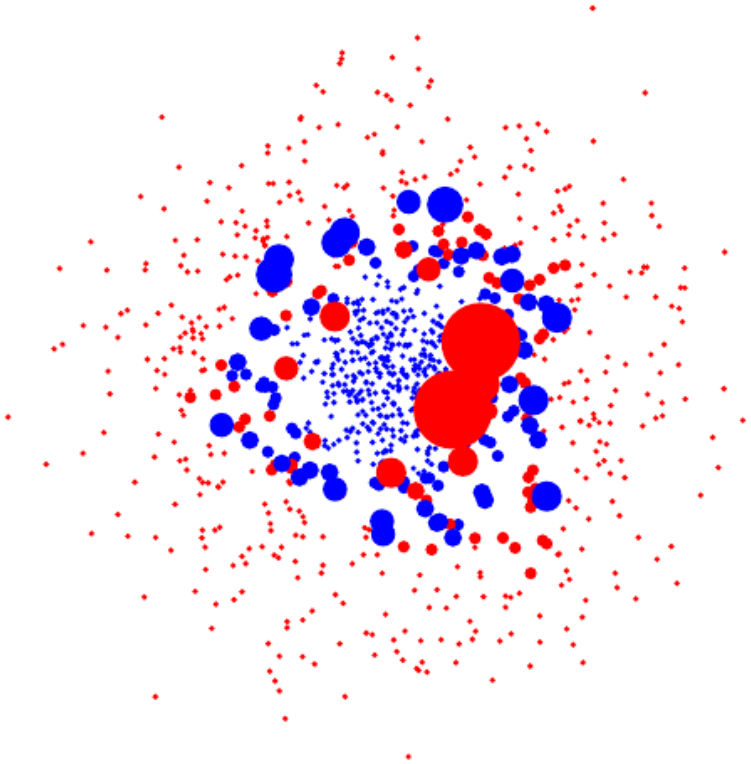
- Weights update



Ensemble Learning – AdaBoost

- Weights update

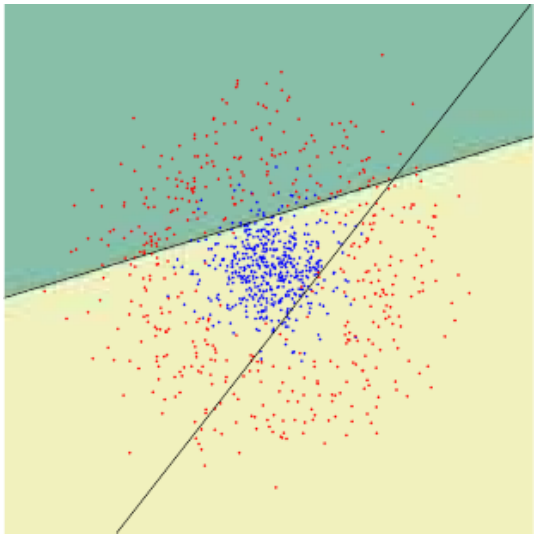
red

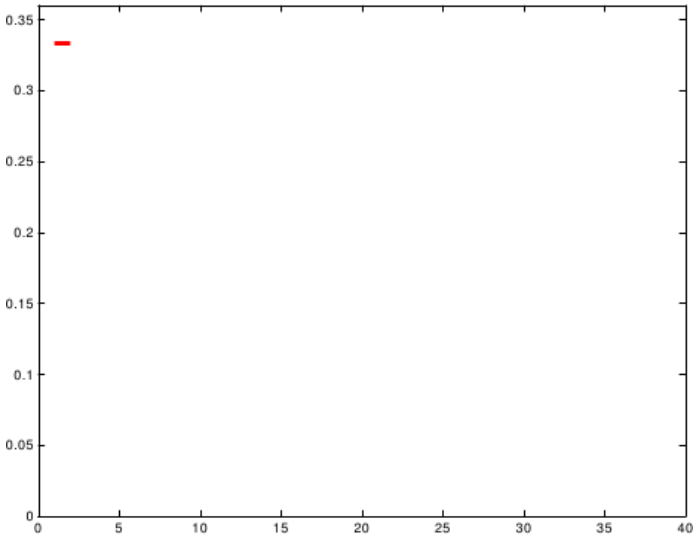


blue

Ensemble Learning – AdaBoost

- Iteration 2:

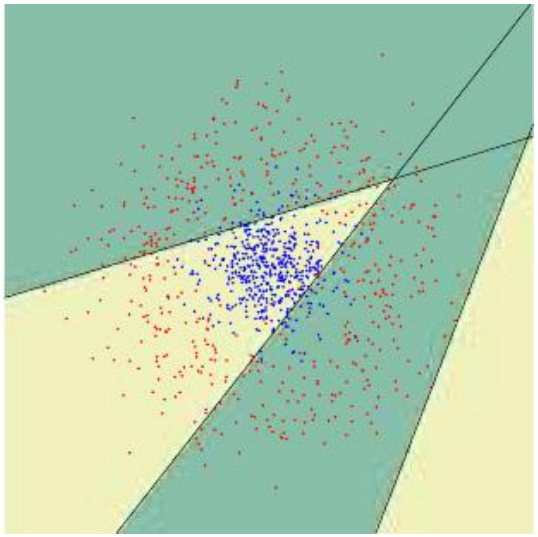


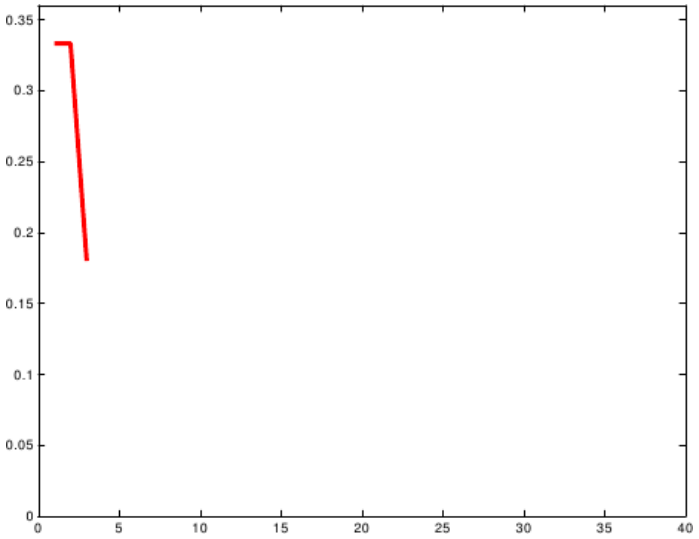


Error Rate

Ensemble Learning – AdaBoost

- Iteration 3:

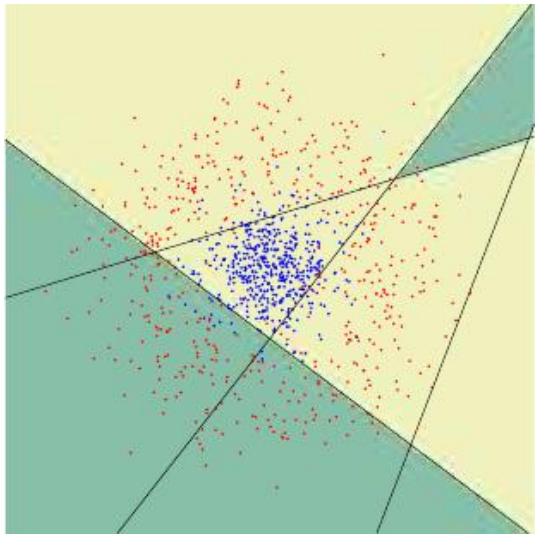


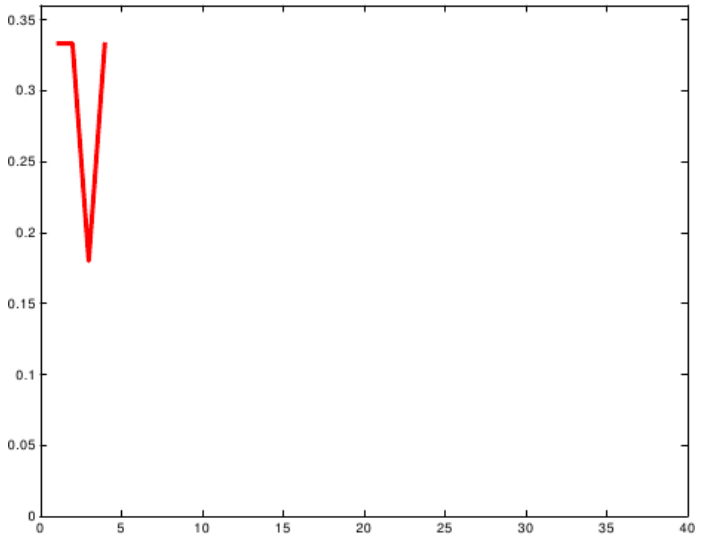


Error Rate

Ensemble Learning – AdaBoost

- Iteration 4:

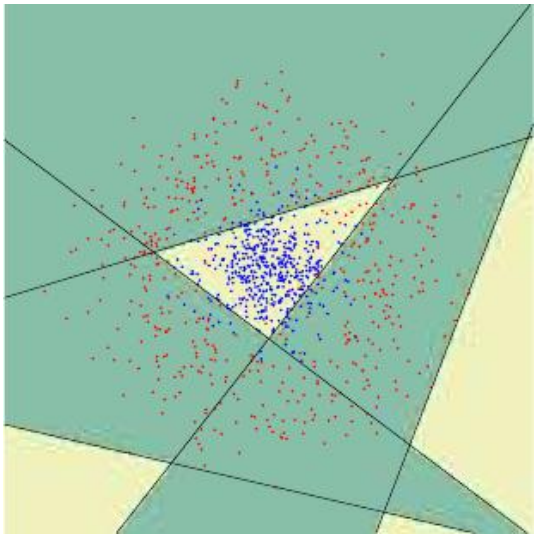


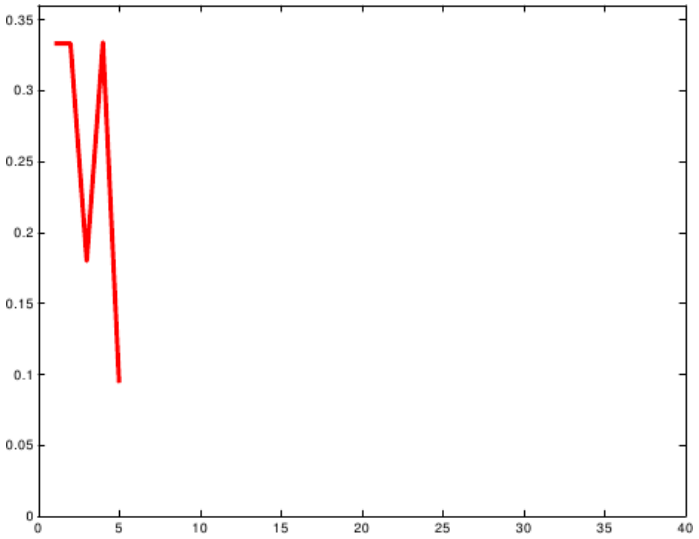


Error Rate

Ensemble Learning – AdaBoost

- Iteration 5:

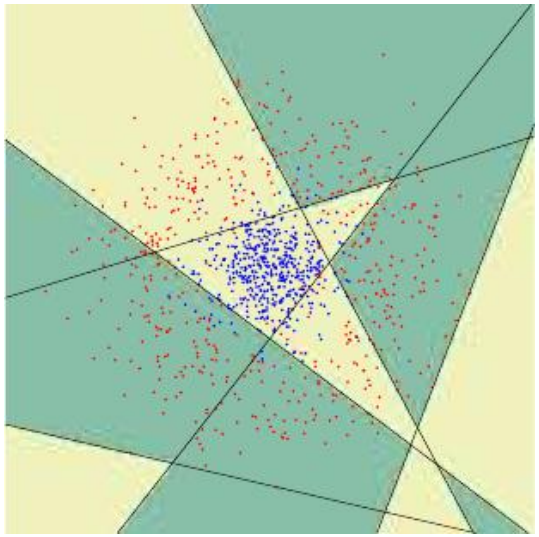


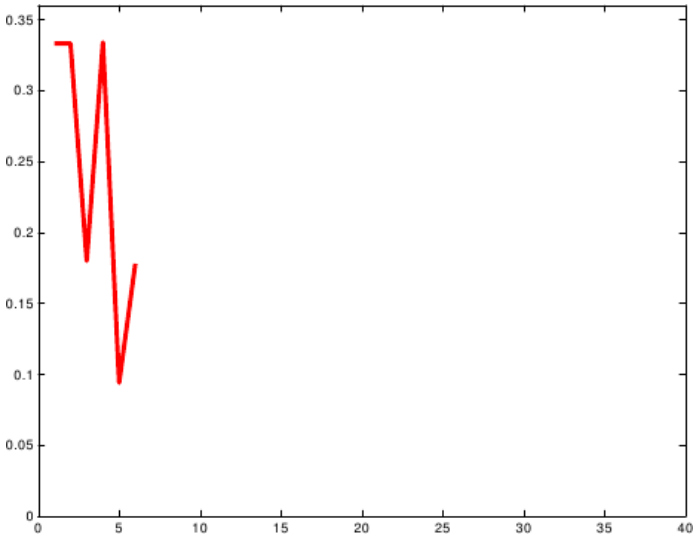


Error Rate

Ensemble Learning – AdaBoost

- Iteration 6:

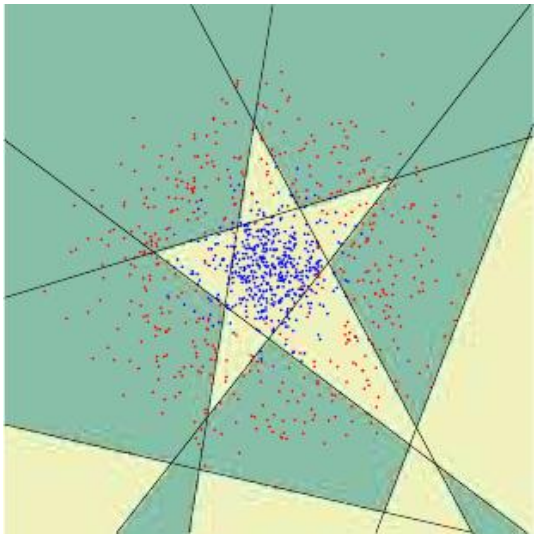


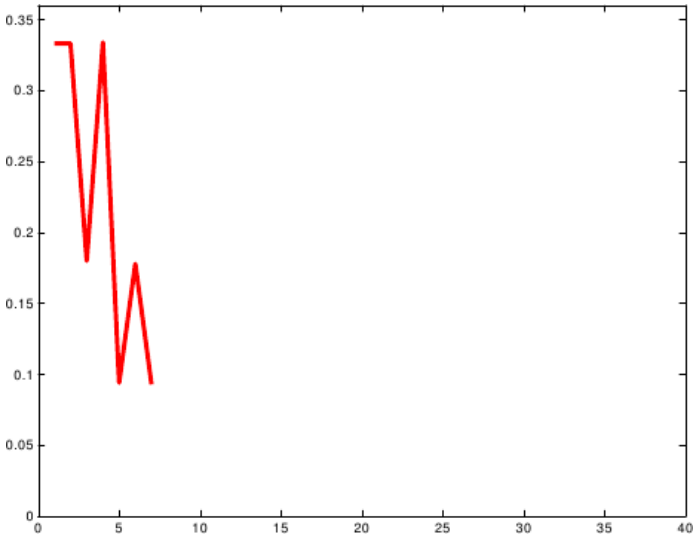


Error Rate

Ensemble Learning – AdaBoost

- Iteration 7:

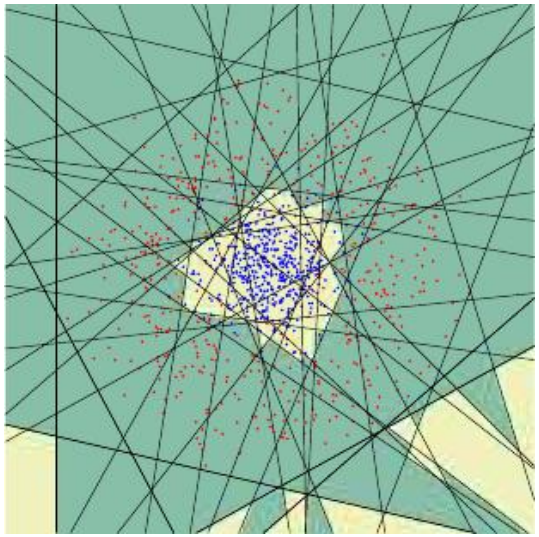


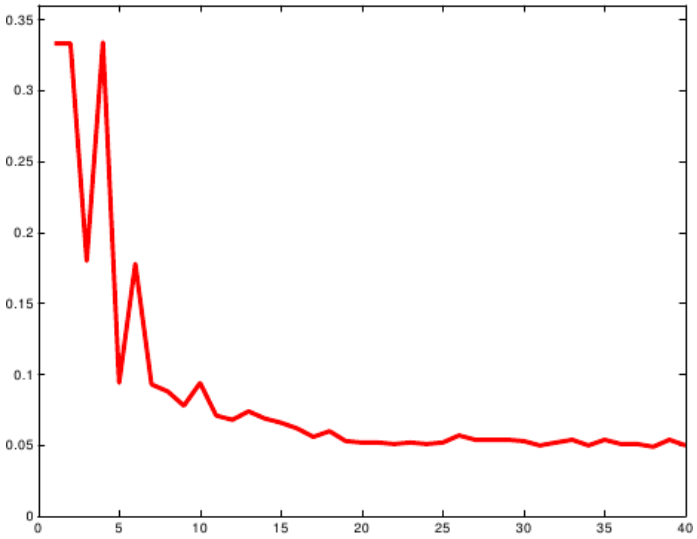


Error Rate

Ensemble Learning – AdaBoost

- Iteration 40:



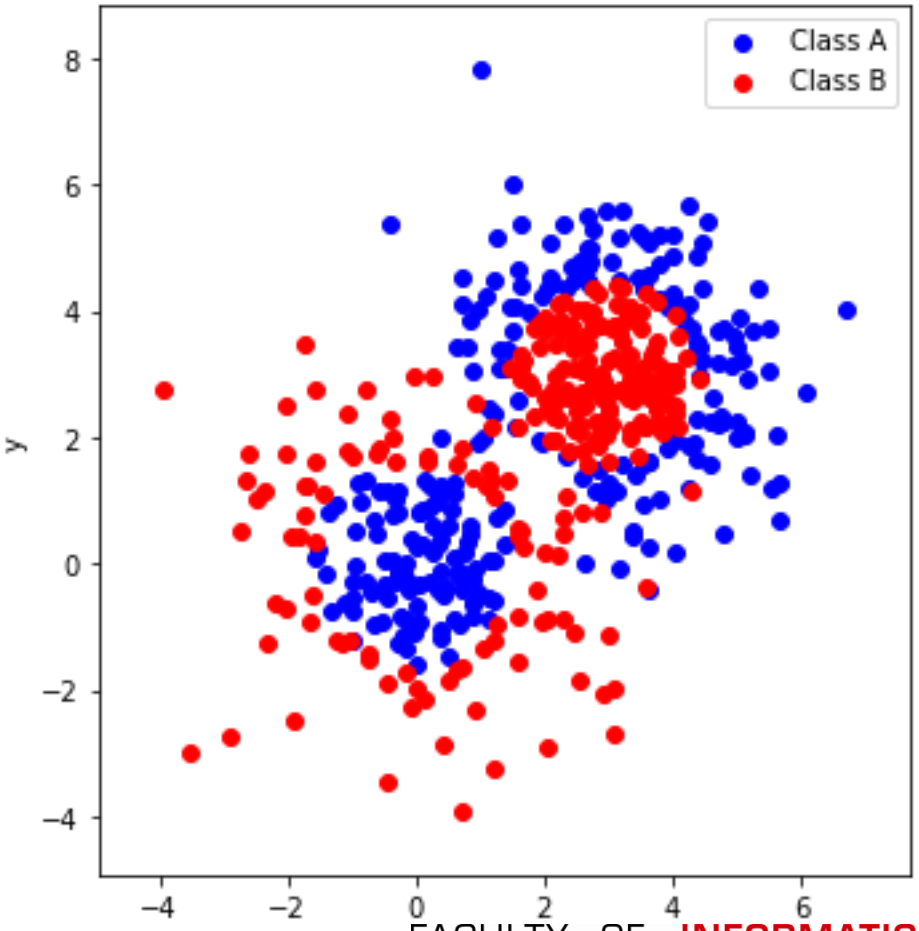


Error Rate

Ensemble Learning – AdaBoost

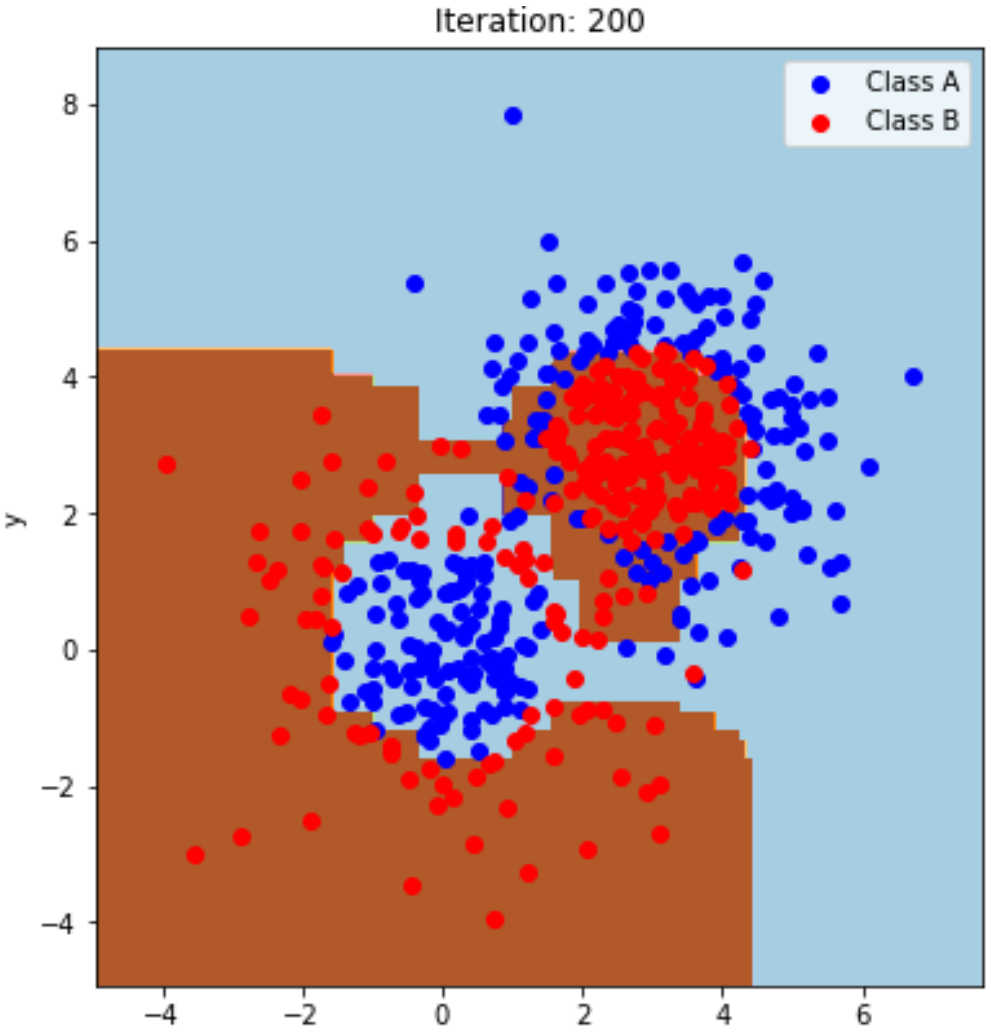
- Example with decision tree as (weak) learner

- Data set:
  - four Gaussians, overlapping



Ensemble Learning – AdaBoost

- Iterations:
  - Train a new decision tree on training set
  - Very simple model
    - maxheight=1
    - aka One-R (1R)
      aka Decision Stump
  - Combine all existing trees, weighted, to get a final vote



Ensemble Learning – AdaBoost

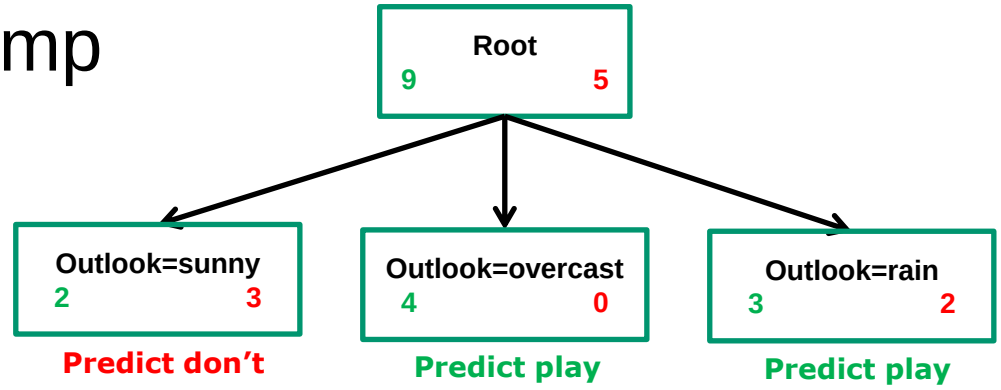
- Weights D_t on training data
  - Equal at the beginning, e.g. *1/N* (or 1)
  - Information about previously selected classifiers' performance is **implicitly** stored in weights

- Iterate
  - Train classifier *t*
  - Adjust weights to give more importance to misclassified samples
    - Subsequent classifiers thus should focus on those
  - Compute weight for current classifier *t*

AdaBoost: step by step

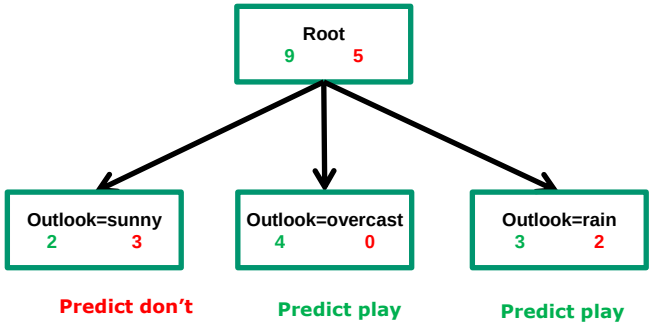
Outlook	Temp	Hum	Windy	Play?	Weight
sunny	85	85	false	Don't	1/14
sunny	80	90	true	Don't	1/14
overcast	83	78	false	Play	1/14
rain	70	96	false	Play	1/14
rain	68	80	false	Play	1/14
rain	65	70	true	Don't	1/14
overcast	64	65	true	Play	1/14
sunny	72	95	false	Don't	1/14
sunny	69	70	false	Play	1/14
rain	75	80	false	Play	1/14
sunny	75	70	true	Play	1/14
overcast	72	90	true	Play	1/14
overcast	81	75	false	Play	1/14
rain	71	80	true	Don't	1/14

- Weights in beginning: e.g. 1/n
- Train first decision stump



- Compute error
  - 5 errors, each weights 1/14
  - Total weighted error: 5/14 (1/14 + 1/14 + 1/14 + 1/14 + 1/14)
    - Base for classifier weight!

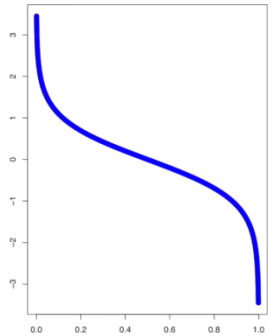
AdaBoost: step by step



- Weighted error:
  - 1/14 + 1/14 + 1/14 + 1/14 + 1/14 = 5 /14

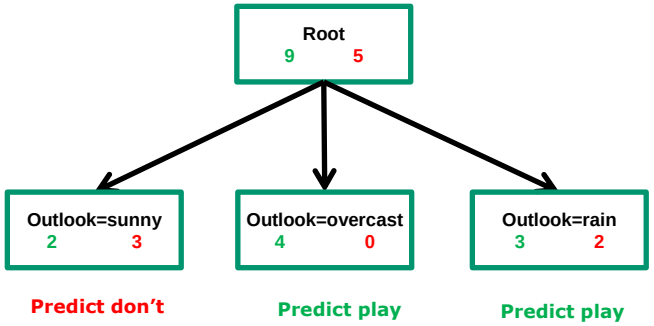
- Classifier weight: e.g. $\alpha(t) = \frac{1}{2} \times \log \frac{1 - TotalError}{TotalError}$

$$= \frac{1}{2} \times \log \frac{1 - 5/14}{5/14}$$



➔ 0.127636253

AdaBoost: step by step



- Adapt data weights D

$$D_{t+1}(i) = D_{t(i)} \times e^{\pm \alpha t}$$

Decreased if correct (-)

Increased if wrong (+)

- Normalise weights to sum = 1

Outlook	Temp	Hum	Windy	Play?	Weight	D(t+1)	D(t+1) (norm)
sunny	85	85	false	Don't	0,0714	0,0629	0,0647
sunny	80	90	true	Don't	0,0714	0,0629	0,0647
overcast	83	78	false	Play	0,0714	0,0629	0,0647
rain	70	96	false	Play	0,0714	0,0629	0,0647
rain	68	80	false	Play	0,0714	0,0629	0,0647
rain	65	70	true	Don't	0,0714	0,0812	0,0835
overcast	64	65	true	Play	0,0714	0,0629	0,0647
sunny	72	95	false	Don't	0,0714	0,0629	0,0647
sunny	69	70	false	Play	0,0714	0,0812	0,0835
rain	75	80	false	Play	0,0714	0,0812	0,0835
sunny	75	70	true	Play	0,0714	0,0812	0,0835
overcast	72	90	true	Play	0,0714	0,0629	0,0647
overcast	81	75	false	Play	0,0714	0,0629	0,0647
rain	71	80	true	Don't	0,0714	0,0812	0,0835

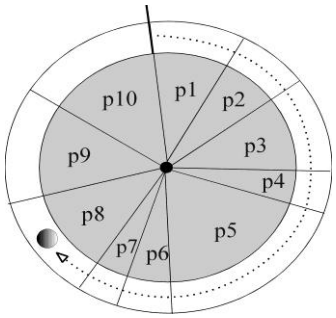
AdaBoost: step by step

- *How to use weights in next iteration?*

Outlook	Temp	Hum	Windy	Play?	Weight
sunny	85	85	false	Don‘t	0,0647
sunny	80	90	true	Don‘t	0,0647
overcast	83	78	false	Play	0,0647
rain	70	96	false	Play	0,0647
rain	68	80	false	Play	0,0647
rain	65	70	true	Don‘t	0,0835
overcast	64	65	true	Play	0,0647
sunny	72	95	false	Don‘t	0,0647
sunny	69	70	false	Play	0,0835
rain	75	80	false	Play	0,0835
sunny	75	70	true	Play	0,0835
overcast	72	90	true	Play	0,0647
overcast	81	75	false	Play	0,0647
rain	71	80	true	Don‘t	0,0835

- Option 1: When evaluating split, multiply criterion (IG / Gini / Error rate) by weights (i.e. computer weighted goodness)

- Option 2: Draw a new training sample, where weights are probabilities (with replacement)
  - Samples with higher weights will be over-represented
  - *Cf. Roulette Wheel Selection*



Outline

- Short Recap
- Deep Learning: Convolutional Neural Networks
- Transfer Learning
- Ensemble Learning: Gradient Boosting

Gradient Boosting vs. Adaboost

- Gradient Boosting = Gradient Descent + Boosting
- Recall Adaboost:

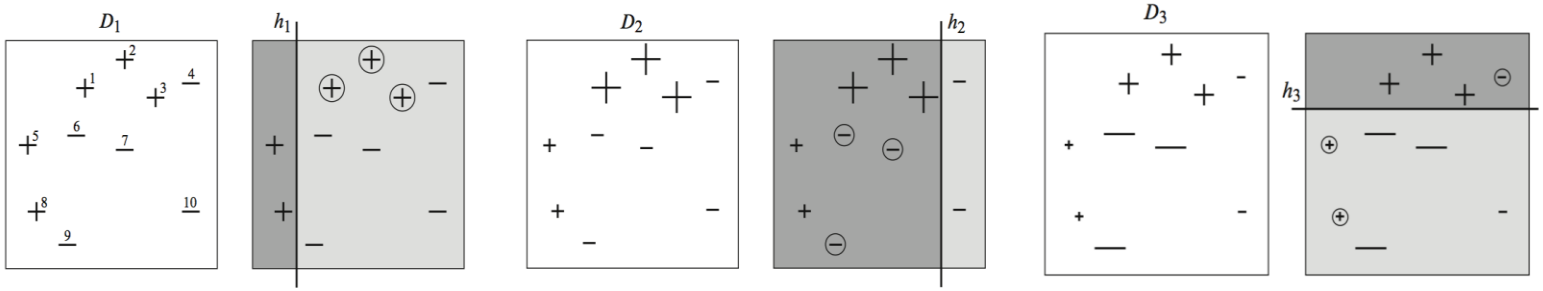


Figure: AdaBoost [Schapire and Freund, 2012]

  - Fit an additive model (ensemble): $\Sigma_t \rho_t h_t(x)$
    - In a forward stage-wise manner
  - In each stage: introduce weak learner to compensate shortcomings of existing weak learners.
  - "Shortcomings" are identified by high-weight data points

Gradient Boosting vs. Adaboost

- Gradient Boosting = Gradient Descent + Boosting
- Recall Adaboost: $H(x) = \Sigma_t \rho_t h_t(x)$

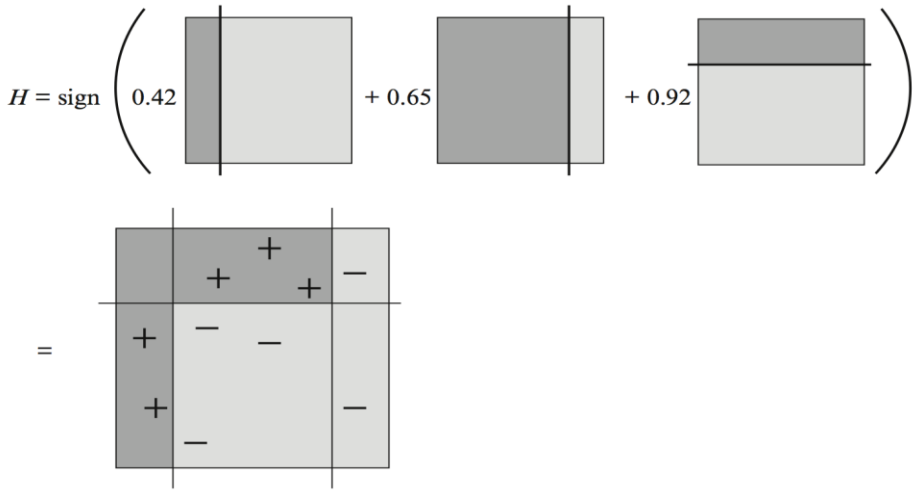


Figure: AdaBoost [Schapire and Freund, 2012]

Gradient Boosting vs. Adaboost

- Gradient Boosting = Gradient Descent + Boosting
  - Fit an additive model (ensemble): $\Sigma_t \rho_t h_t(x)$
    - In a forward stage-wise manner
  - In each stage: introduce weak learner to compensate shortcomings of existing weak learners
  - Gradient Boosting: “Shortcomings” identified by gradients
    - Adaboost: “shortcomings” identified by high-weight data points
      - Both tell how to improve model
  - Weight of model
    - Adaboost: depending on performance of individual model
    - Gradient Boosting: uniform, fixed “Learning Rate”
  - Models: Adabost: often decision stump (1R) only
    - Gradient Boosting: often limit the number of leaves; e.g. [8..32]
      - Exact number depends on the dataset!

Gradient Boosting: idea

- Given: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Task: fit a model F(x) to minimize square loss
- Obtain an initial model F
  - Assessed as **good**, but **not perfect:** there are some mistakes
    - $F(x_1) = 0.8$, while $y_1 = 0.9$, and
    - $F(x_2) = 1.4$ while $y_2 = 1.3$
  - *How can you improve this model?*

- Boosting
  - Can **add additional model h** to F
    - New prediction will be F(x) + h(x)

Gradient Boosting

- Simple solution: wish to improve existing model *F* by model *h* such that:
  - $F(x_1) + h(x_1) = y_1$
  - $F(x_2) + h(x_2) = y_2$
  - …
  - $F(x_n) + h(x_n) = y_n$

- Equivalently:
  - $h(x_1) = y_1 - F(x_1)$
  - $h(x_2) = y_2 - F(x_2)$
  - …
  - $h(x_n) = y_n - F(x_n)$
- $y_n - F(x_n)$: “residuals”

Gradient Boosting

- $y_i - F(x_i)$: (pseudo) residuals
  - Data that existing model *F* cannot predict well
  - Role of h: compensate shortcoming of existing model *F*

- If new model (*F+h)* not satisfactory:
  ➔ Add another model

- How is this related to gradient descent?
  - Gradient Descent: minimize a function by moving in the opposite direction of the gradient
  - *Residual ~ negative gradient*

Regression: Initial "Tree"

1. Build "zero-rule" model
   - Residuals:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1,\ldots,n$$

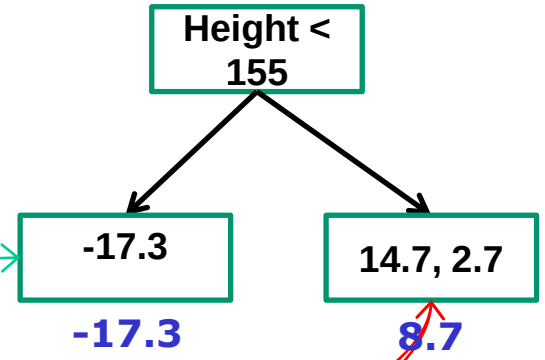
   - Regression loss function L: ½ (actual – predicted)^2
   - Take derivative, set zero ➔ Mean value

Height	Weight	Pred	Res
160	88	73.3	14.7
160	76	73.3	2.7
150	56	73.3	-17.3

2. Compute prediction: (88 + 76 + 56) / 3 = 73.3

3. Compute residuals
   - Prediction: 73.3 – <actual value>

GBR: Iterations



1. Build tree to ***predict residuals***
   - *Number of leaves: 2*
   - *Which split?*

Height	Weight	Pred	Res
160	88	73.3	14.7
160	76	73.3	2.7
150	56	73.3	-17.3

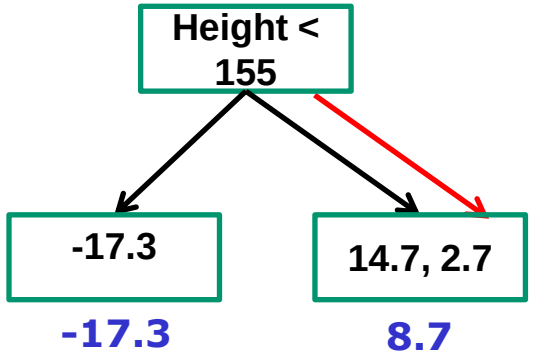
2. Compute outputs: $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$
   - ➔ Average of values

3. Now we can use this tree in our ensemble
   - *What does it consists of so far?*

GBR: Iterations

- Assume learning rate of 0.1

73.3 **+ 0.1 x**



Height	Weight	Pred	Res	Pred	Res
160	88	73.3	14.7	74.2	13.8
160	76	73.3	2.7	74.2	1.8
150	56	73.3	-17.3	71.57	-15.57

- *New prediction?*

  73.3 + 0.1 x 8.7 = 74.2

  73.3 + 0.1 x 8.7 = 74.2

  73.3 + 0.1 x -17.3 = 71.57

Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

- *Next step?*

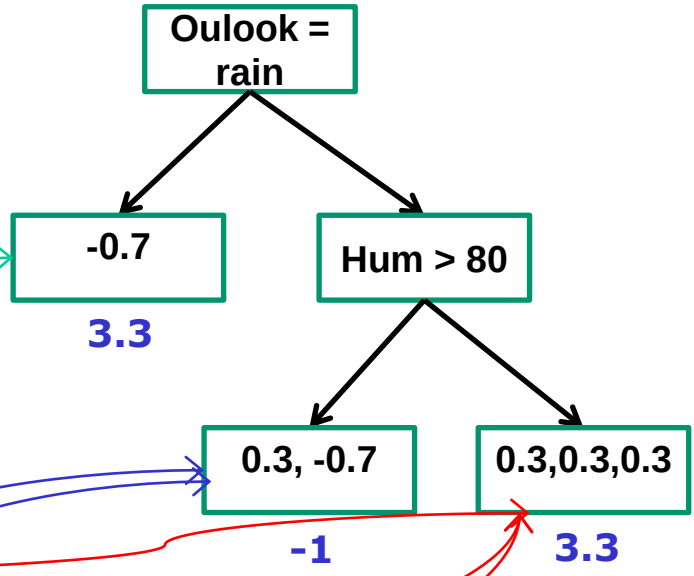
Classification

Windy	Hum	Outlook	Play?	Pred	Res
TRUE	70	sunny	Play	Play	0.3
TRUE	90	overcast	Play	Play	0.3
FALSE	85	sunny	Don‘t	Play	-0.7
TRUE	74	rain	Don‘t	Play	-0.7
FALSE	75	overcast	Play	Play	0.3
FALSE	71	sunny	Play	Play	0.3

1. Build “zero-rule“ model
   - Classification: log(odds) of majority class
     - Regression: mean value
2. Compute prediction $\log(\ odds\ ) = \ln(\frac{4}{2}) = 0.6931$
   - Probability: using logistic function $\frac{e^{\log(\ odds\ )}}{1 + e^{\log(\ odds\ )}}$
   - $\frac{e^{\ln(2)}}{1 + e^{\ln(2)}} = 0.6667 > 0.5$
3. Compute residuals
   - Prediction: 0.6667; Ground-truth: 0 / 1
   - *(to simplify computation here: round to 1 digit)*

GBC: Iterations

1. Build tree to ***predict residuals***
   - *Number of leaves: 3*



Windy	Hum	Outlook	Play?	Pred	Res
TRUE	70	sunny	Play	Play	0.3
TRUE	90	overcast	Play	Play	0.3
FALSE	85	sunny	Don't	Play	-0.7
TRUE	74	rain	Don't	Play	-0.7
FALSE	75	overcast	Play	Play	0.3
FALSE	71	sunny	Play	Play	0.3

2. Update: $\frac{\sum \text{Residual}_i}{\sum [\text{PreviousProbability}_i \times (1 - \text{PreviousProbability}_i)]}$

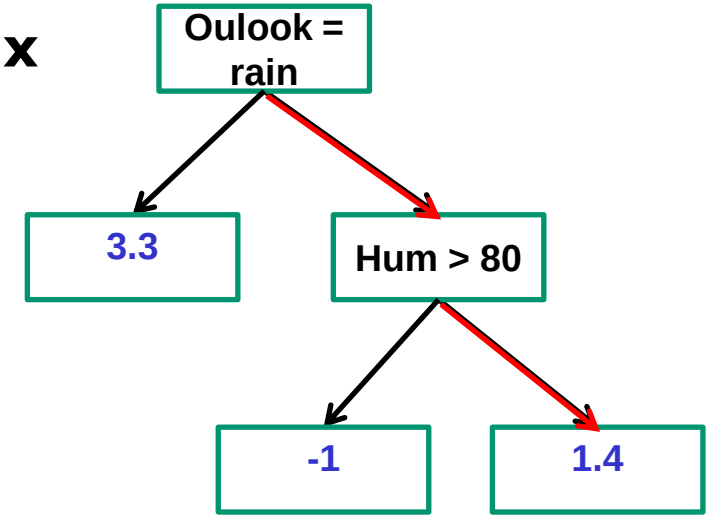
3. Now we can use this tree in our ensemble
   - *What does it consists of so far?*

GBC: Iterations

- Assume learning rate of 0.8 *(0.1 is a more common value)*

$$\log(\ odds\ ) = 0.7$$

+ 0.8 x



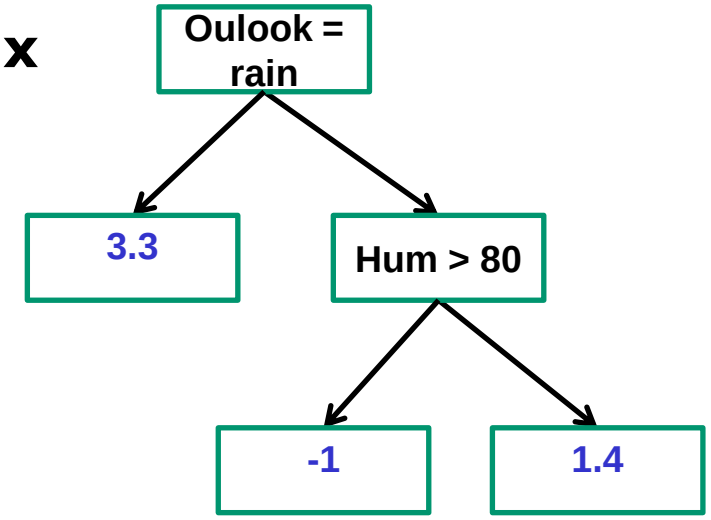
Windy	Hum	Outlook	Play?	Res
TRUE	70	sunny	Play	0.3
TRUE	90	overcast	Play	0.3
FALSE	85	sunny	Don't	-0.7
TRUE	74	rain	Don't	-0.7
FALSE	75	overcast	Play	0.3
FALSE	71	sunny	Play	0.3

Pred	Res
0.9	0.1
0.5	0.5
0.5	-0.5
0.1	-0.1
0.9	0.1
0.9	0.1

- *New prediction?*
  - log(odds) prediction = 0.7 + (0.8 x 1.4) = 1.8
  - Probability: $\frac{e^{\log(\ odds\ )}}{1 + e^{\log(\ odds\ )}} = \frac{e^{1.8}}{1 + e^{1.8}} = 0.9$
- *Next step?*

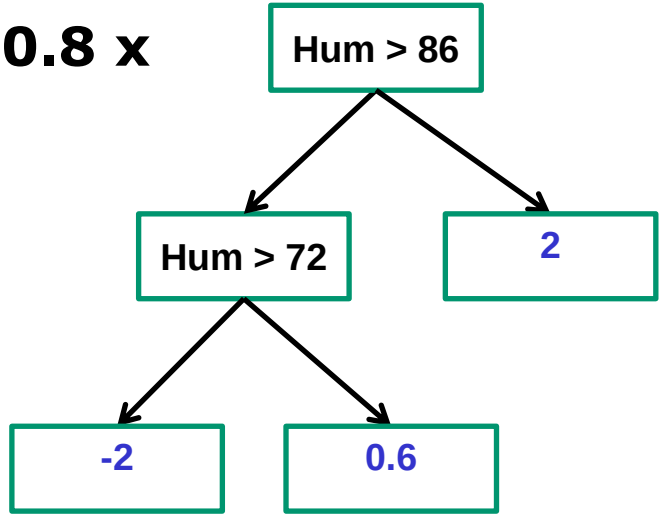
GBC: Iterations

$\log(\ odds\ ) = 0.7$ + 0.8 x



Windy	Hum	Outlook	Play?
TRUE	70	sunny	Play
TRUE	90	overcast	Play
FALSE	85	sunny	Don't
TRUE	74	rain	Don't
FALSE	75	overcast	Play
FALSE	71	sunny	Play

+ 0.8 x



GB: Classification vs. Regression

- *What changes between classification and regression?*

- Prediction
  - Regression: Average of outputs
  - Classification: log odds

- Computation of residuals
  - Simply difference (observation – prediction)

Questions ?