DLVC Questions

Abraham Hinteregger, Christof Schmidt

March 24, 2017

Acknowledgements

Questions are (more or less) along those released for the course *Deep Learning for VC* at TU Vienna, available here:

`https://github.com/cpra/dlvc2016/`

Images are "lifted" from `http://cs231n.github.io/`.

Most answers should be more or less correct and complete (enough). The sections about medical applications are mostly rather weak as the slide-set provided for this was pretty confusing.

Todo list

Improve rather weak capacity answers . . . 6

just a guess, not covered in lecture, can't find it in book . . . 12

How many pooling layers should CNN have? . . . 16

how to choose network depth for image classification problem . . . 17

dropout section is rather weak . . . 19

this is almost certainly wrong . . . 19

where to apply dropout? . . . 19

Medical appl.: How can we use the training data most efficiently? . . . 21

Image classification 3
Datasets 3
Case study 4
DL Motivation 4
Class. & Regression, (un)supervised learning 5
Testdata 5
Performance measures 5
Capacity, bias, variance 6
Hyperparameter 7
kNN 7
Image Classification 8
Parametric model 8
Linear model 9
Loss function 9
Optimization in ML 9
Local/global optimum 10
Gradient descent 10
Batch Gradient descent 11
Weight and bias 11
Preprocessing 12
Optimization vs ML 12
Regularization 13
Feedforward NN 13
MLP 14
Representation learning 14
Locally connected layers 15
Convolutional Layer 15
Receptive field 16
Pooling 16
CNN 16
CNN backends 17
ResNets 17
Back Propagation 18
Data Augmentation 18
Dropout 19
Batch normalization 19
Learning rate 19
Oversampling 20
Model ensembles 20
Medical tasks 21
2-3D deep CNN 21
Labelling 22
RNNs 22
LSTMs 22

1 Image classification

1.1 What is the task definition of image classification?

Predict the label (picked from a set of possible labels) of an image, i.e.: what is visible on the image. Output could also be a distribution over labels to indicate confidence in result.

1.2 Explain at least 5 challenges and give examples.

- Viewpoint variation: A single instance of an object can be oriented in many ways with respect to the camera.
- Scale variation: Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- Deformation: Many objects of interest are not rigid bodies and can be deformed in extreme ways, e.g. water
- Occlusion: Only a small portion of an object (as little as few pixels) could be visible.
- Illumination conditions: The effects of illumination are drastic on the pixel level.
- Background clutter: The objects of interest may blend into their environment, making them hard to identify.
- Intra-class variation: The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.
- Abstraction (human able to recognize some doodle as the correct object usually—difficult to recognize for machines as e.g. difference between cow and dog in a doodle may be the presence of grass
- Invariants (an object may be the same object if it's rotated by some degree. Other objects (e.g. "6" and "9") are not invariant under rotation

1.3 What is object detection and how does it differ from classification?

Object detection tries to find all objects present in an image whereas in image classification a label is assigned to the whole image. Object detection can be reduced to image classification (classify various parts of the image and combine results)

2 Datasets

2.1 Why do we need datasets?

Data required for training and testing of model.

2.2 What are the challenges of dataset collection and annotation in deep learning?

Datasets should be sampled from the same distribution as images which are then classified, i.e. it should span the whole space of possible features associated with each class. For example if dataset contains only cats run over by car a model trained with this set will have difficulties recognizing healthy cats.

2.3 Explain the purpose of the different subsets.

- Training set: Used to train the model
- Validation set: Used to train hyperparameters
- Test set: Final performance analysis.

Model is trained with training set and then evaluated with validation set. Then some other hyperparameters are chosen (e.g. lower learning rate, ...) and the step is repeated until there is no further improvement to the model. The datasets have to be disjunct as else it would be "learning to the test" (e.g. model is a hashmap containing the hash of the image and the correct class. Would work if testset is contained in training set but completely useless for new data)

2.4 What is the rule of thumb about how many images per classes are needed for CNNs to perform well?

At least 5000 per class—more is better though.

3 Case study

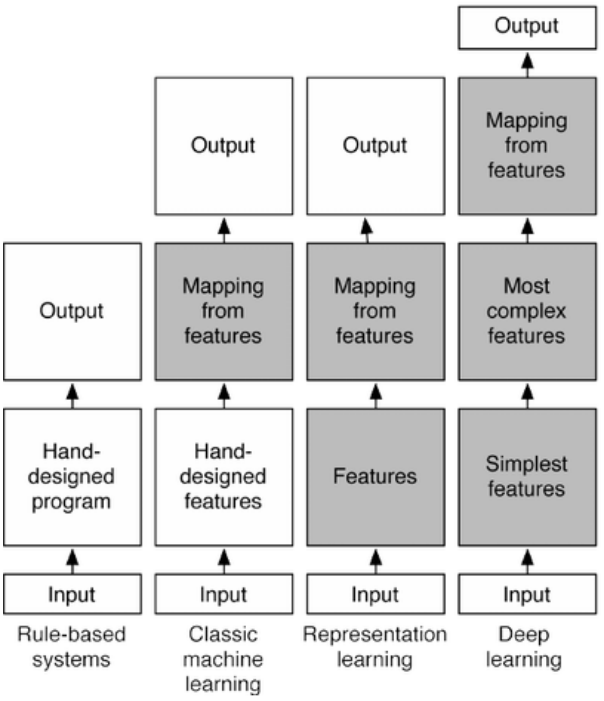
Assume a company asks you to develop an application that is able to predict which kind of bird is depicted in a given image. List and explain the individual steps you'd follow to solve this problem using deep learning.

1. (Set up some machine that allows to train the model with hardware acceleration)
2. Ask what kind of birds should be distinguished from each other (e.g. is "parrot" enough or should it be "macaw parrot" or even "Ara" or "Military macaw")
3. Only sitting birds or also while they're flying?
4. Look for dataset that fulfills the conditions resulting from the above questions
5. Proceed in the usual order to train a model:
   - Set up training process
   - Find suitable NN architecture
   - Augment dataset
   - Find hyperparameters and tune them until accuracy is good enough

4 DL Motivation

4.1 What is the motivation for solving vision tasks via machine learning?

- Rule based system: handwritten AI where an expert implements all rules (decision boundaries)
- Classic machine learning: The features are designed and the algorithm then maps the features to the boundaries
- Representation learning: The features are trained on the data and mapped to output.
- Deep learning: There are low and high level features and the algorithm learns how to train the features, how to combine them to higher level features and how to map the to the output



4.2 What is a machine learning algorithm and how are they used for solving image classification problems?

A machine learning algorithm is a method of statistical analysis that automates analytical model building. Their aim is to find hidden insights without being programmed where to look. This is applied for image classification by providing a dataset of labelled images which are then used to extract features which are used to infer the class of an image.

Applying rule based systems would be completely overwhelming (e.g. if there are 45% blue pixels and 55% white pixels it's a boat . . .)

5 Classification & regression, un-/supervised learning

5.1 Explain the differences between classification and regression and supervised vs. unsupervised learning

Classification is used to predict discrete outputs (e.g. an animal is a dog or a cat but not some 50:50 mixture of both) whereas regression is used to predict continuous values (e.g. IQ, height, time ...).

Supervised learning uses a training dataset where the desired prediction is already known and the algorithm tries to find a model that would come to the same conclusions.

Unsupervised learning only uses a dataset and the algorithm tries to find some structure in it (e.g. SOMs, EM-algorithm, PCA)

5.2 What do discriminative and generative models learn

Discriminative models learn the boundary between classes $P(y|x)$ (estimate response y when given input x whereas generative models learn the underlying distribution $P(x, y)$.

SVN and NN are discriminative, Naive Bayes and Hidden Markov models are generative.

5.3 How can generative models be used for classification?

It uses the distribution $P(x, y)$ to estimate which response y is most likely to cause the given input x ($\text{argmax}_y P(x|y)P(y)$). Though generative models can be used for classification, discriminative models usually perform better.

6 Testdata

6.1 Why are machine learning algorithms tested on data unseen during training?

The goal is to have a model that can predict new data. Therefore it's a good idea to measure exactly this as quality criterion. This is done by using unseen data for testing purposes. The same applies to the validation data—optimize the model to perform well on data not seen during training. If this would not be done, the model after training would be completely overfitted and would perform well on the training data (which is useless) but not necessarily well or even good on new data.

6.2 How can such algorithms perform well on unseen data?

If the model is not overfitted it should recognize general features ("wheels in the lower left corner") of the sample but not very specific features (e.g.: hash of the image).

6.3 What does dataset bias mean?

When a model was trained on a dataset the model will usually be applied to new problems (samples not yet found in the dataset). As the model will only take into consideration information from the training set this dataset should be sampled without bias from the same distribution as the samples that are then used in production. If this is not the case the model won't perform well. For example a model trained on whales and bees will have difficulties to predict whether a cow is an insect or a mammal.

7 Performance measures

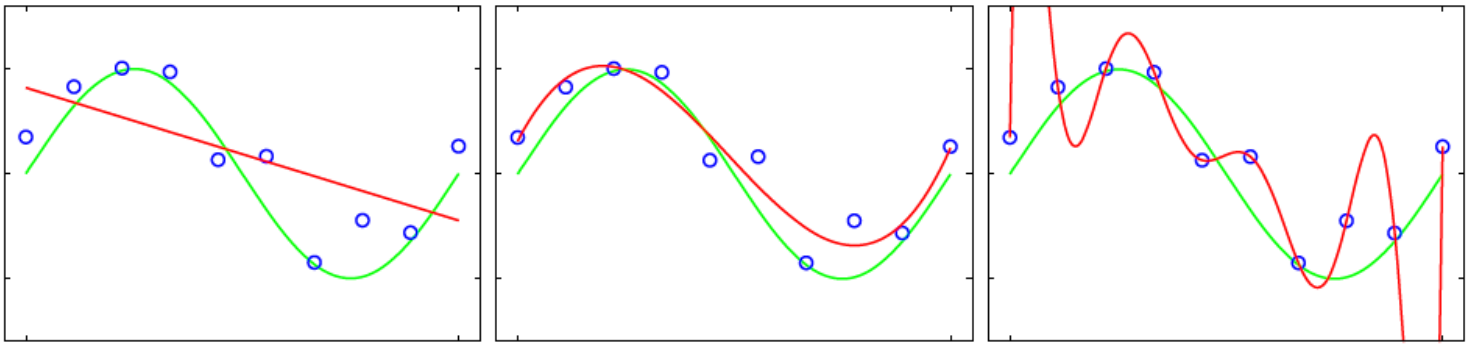
7.1 How is the performance of a classifier measured?

For classification the accuracy is used to measure the performance. The predictions are compared with the target labels and the accuracy is defined as the fraction of correctly classified samples (the error rate is the complement of this).

7.2 Which factors determine the performance, and how is underfitting and overfitting related to these factors?

The performance of an algorithm is determined by two factors

- Ability to minimize training error (if this fails: underfitting)
- Ability to minimize gap between training and test error (if the first succeeds but this fails: overfitting)

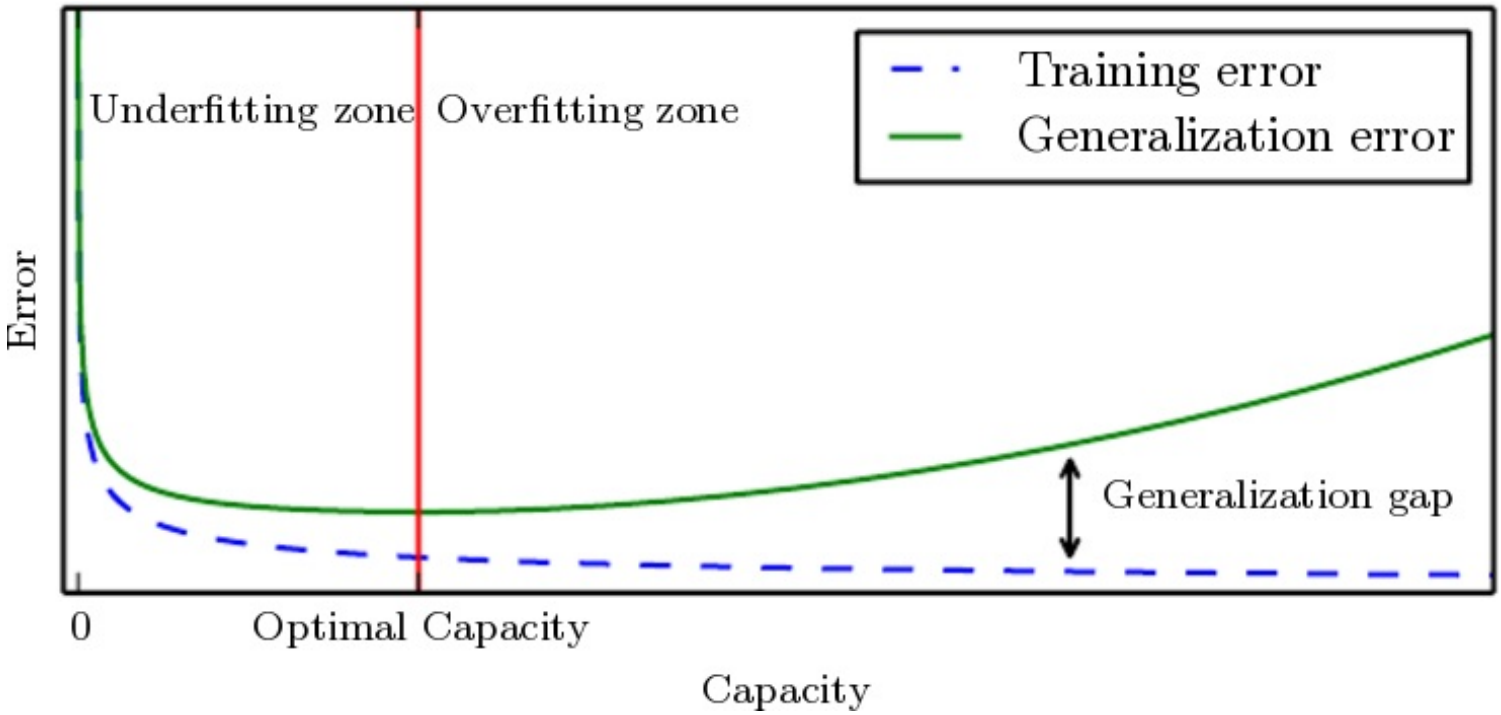


From left to right: underfit (low capacity), good model, overfit (high capacity)

8 Capacity, bias, variance

8.1 Explain the terms algorithm capacity, bias, and variance.

- Algorithm capacity: the capacity is the ability to compute complex decision boundaries. If two models have the same validation error the model with lower capacity is usually preferred ("most parsimonious model", AIC and BIC are formalizations of this notion in the context of linear regression)
- Bias: If the capacity is too low the model struggles to fit the training set. This results in a high bias.
- Variance: If the capacity is too high the model fits the data too well which results in high variance and bigger generalization error



When the capacity is too low neither the training error nor the test error can be reduced to the optimal point. The capacity is optimal when the test error is minimal. Starting from this point increasing the capacity may result in a lower training error due to overfitting but the test error increases again. The gap between training and test error is called generalization gap that results from tailoring the model too strongly to the training/validation data.

8.2 Why does the training set size affect the optimal capacity of a model?

When more training data is available the performance of the model will increase as the data reflects the problem better. This also results in a higher optimal performance

9 Hyperparameter

9.1 What is a hyperparameter?

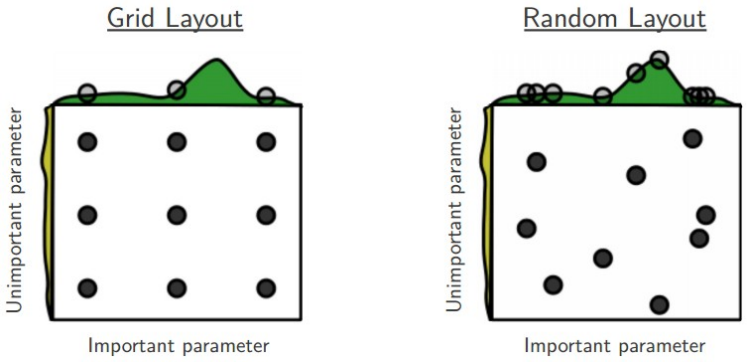
In ML we try to find parameters for a model from datasets. This learning process may have its own parameters, these are called hyperparameters. They govern behaviour of the model learning, .e.g. the capacity of the resulting model.

9.2 Name at least 3 hyperparameters in the context of deep learning using convolutional neural networks.

- Initial learning rate
- Learning rate decay
- Regularization strength
- Momentum settings

9.3 What is the purpose of hyperparameter selection, which search strategies exist, and how do they work?

Purpose is finding optimal settings to train the model. Grid-search and random search are two common approaches, random search is usually preferred as one parameter may be insignificant for the result and much time would be wasted optimizing this parameter whereas with random search all HP spaces are explored independently. See picture below.



10 kNN

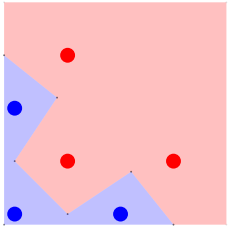
10.1 How does the k nearest neighbor classifier work?

Find the k samples in the training set that are most similar to the current sample according to some similarity measure

- Training
  - Preprocess data
  - Transform images to vectors ($32 \times 32 \times 3$ image $\rightarrow 32 \cdot 32 \cdot 3 = 3072$D vector) or use extracted feature vector
  - Split into training and validation data and optimize the hyperparameters (distance measure and k)
- Classification
  - Apply the same preprocessing transformations as before
  - Find the k images that are most similar and use the most common label as prediction (or output the distribution of labels)

10.2 Sketch of kNN

Create a sketch for illustration, assuming a two-dimensional feature space and two different classes, Draw at least three training samples per class (must not lie on a line) as well as (roughly) the resulting decision boundaries.
Image on right is kNN with two classes and $k = 1$.



10.3 What are the limitations of this classifier?

- Classification performance is rather bad as sample has to be compared to all/most of the samples in the training set (can be improved with KDTree from $O(n)$ to $O(log(n)^d)$ or similar)
- Does not recognize higher order features, e.g. "presence of wheels" but only uses similarity of features (pixels). Completely thrown off the track by e.g. different lighting or position of object in image (e.g. off center in sample and only centered pictures in training set)
- Very sensitive to background

11 Image Classification

11.1 Why do general machine learning algorithms (those expecting vector input) perform poorly on images?

Images have spatial information, i.e. a pixel is correlated with the pixels in its surrounding. This additional information is not leveraged with general ML algorithms and thus the resulting performance is subpar.

11.2 What is a feature, and what is the purpose of feature extraction?

Features are used to classify something. If it has four wheels and 5 people can sit in it it is most likely a car. If it has four legs it's most likely not a human etc.

Feature extraction is used to get the features of a sample, e.g. are there eyes on the image, is the object curved, ... These extracted features can then be used to classify an image.

11.3 Explain the terms low-level feature and high-level feature.

A low level feature would be the existence of edges in a certain direction or some color gradient. A high level feature is then built from a set of low level features, e.g. a face consists of a certain arrangment of strokes (the outline and nose and mouth) and round objects (eyes). An even higher level feature would then be a person consisting of various features such as "face", "body" and "limbs".

12 Parametric model

12.1 What is the definition of a parametric model?

A model is a function that maps from the input space to the output space: $f : \mathbb{R}^D \to \mathbb{R}^T$. A parametric model also takes a finite vector of parameters θ to get from input x to output y: $f(x, \theta) = y$.

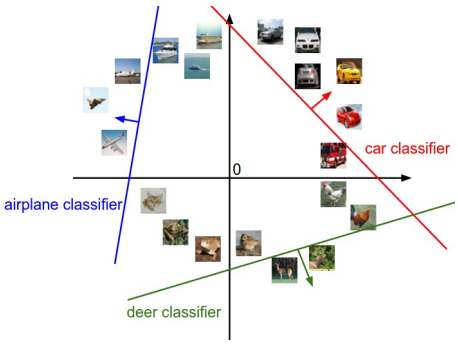
Whereas the model for a kNN classifier (a non-parametric model) gets more complex the bigger the training set is a parametric model only has parameters that are better chosen if the training set gets larger.

12.2 What do the parameters of such models control (what effect do they have?), and how are they set?

The parameters are determined by training the model with a trainingset which can be discarded afterwards. In a linear model the weight matrix and bias vector are the parameters and they govern the decision boundary $Wx + b$.

13 Linear model

13.1 What is a linear model, which types of parameters does it have, and what do they specify?



A linear model tries to seperate the samples with linear functions. The samples are in a D-dimensional space and there are T classes. The model then consists of T decision boundaries of the form $y_t = \sum_i^D W_{ti}x_i + b_t$ or in short $y = Wx + b$. The weight matrix W and the bias vector b are the parameters. The resulting model is a set of decision boundaries where each boundary provides a value for each class. Usually the class with the highest score is then chosen as prediction.
The output value of $f(x, \theta)$ has a score for each class where a positive number corresponds to the sample being on the side indicated with an arrow, meaning, that the sample is more likely than not a member of this class. The class with the highest score is then chosen as prediction.

14 Loss function

14.1 What is the purpose of a loss function?

When training a parametric model an optimization algorithm the minimizes a loss function is used. The loss function has the parameters as input and the global minimum of $L(\theta)$ corresponds to the parameter setting that is optimal for the task at hand (classifying images).

14.2 What does the cross-entropy loss on a dataset D measure?

For classification usually cross-entropy loss used. CEL measures the dissimilarity of the real distribution from the predicted distribution. $H(p, q) = -\sum_x p(x) \ln q(x)$. p, q are discrete probability distributions $p(x) \leq 0, \sum_x p(x) = 1$. The CEL of a dataset is then the sum of the CEL of all samples divided by the size of the dataset.

14.3 Which criteria must the ground-truth labels and predicted class-scores fulfill to support the cross-entropy loss, and how is this ensured?

Both must be probability distributions, therefore the class i is encoded as a vector $p_t = \delta_{it}$ (δ is the Kronecker symbol with $\delta_{ij} = 1$ iff $i = j$) and the predicted vector of class scores is normalized with the softmax function $w_k = \frac{\exp(w_k)}{\sum_t^T \exp(w_t)}$

15 Optimization in ML

15.1 What is the purpose of optimization in the context of machine learning?

When training a parametric model using machine learning, we need a loss function to measure the quality of the model. In this case, it is the cross-entropy loss called $L(\Theta)$ which measure the performance of a ML classifier on some dataset. The outcome changes if we change the parameter of Θ. To change the parameters in a way to minimize the loss, we need an optimization algorithm, and because $L(\Theta)$ is non-linear, the algorithm need to be non linear as well. For deep learning, gradient descent is the most popular approach.

15.2 How does the gradient descent algorithm work?

For better imagination, think of a rugged terrain, without sight. To get to the lowest/highest point, one will just follow the steepest slope.

In ML setting, the algorithm will compute the current gradient $\Theta' = \nabla L(\Theta)$ and then go a certain amount $\alpha\Theta'$ into that direction. The hyperparameter α is called the learning rate.

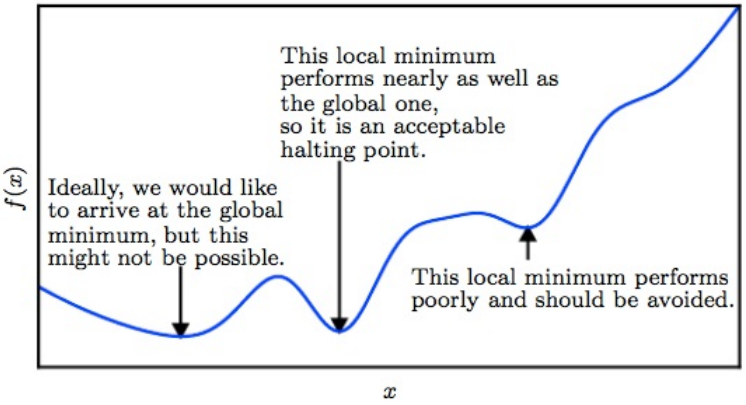
15.3 What is the gradient of a function?

A function f that represents a scalar field $f : \mathbb{R}^n \to \mathbb{R}$ that is differentiable, has partial derivatives for all input parameters so that $f_{x_i}(x)$ encodes how fast f changes with argument x_i at point x. The gradient is a vector that is obtained by applying the differential operator ∇ to f ($\nabla_i = \partial/\partial x_i$) to get a vector containing the partial derivatives of f. It points in the direction of greatest increase. If all entries of ∇f are 0 at some point L is flat there and we have to stop (local/global minimum or saddle point).

16 Local/global optimum

16.1 What is the difference between a local and a global optimum?

A function has only one global optimum but may have many local optima. Finding the global optimum is not trivial whereas finding a local optimum is usually easy (calculate derivation at current point and move in direction of steepest descent). When a local optimum is found it's nontrivial how to escape it again. Simulated annealing is an approach inspired from material physics.

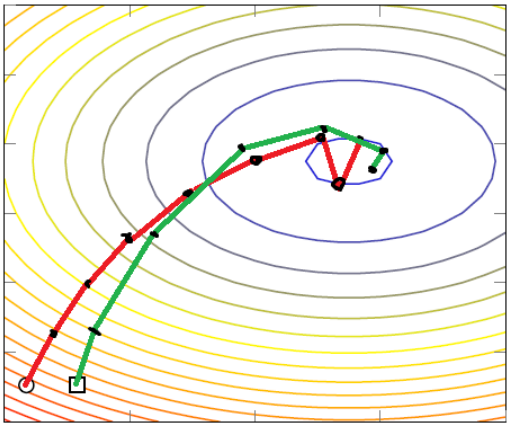


16.2 Are local minima a problem in deep learning? Why (not)?

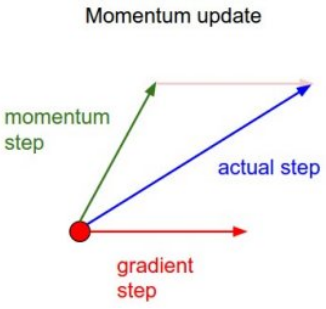
There is no definite consensus though it is believed, that the search landscape is similar to an egg carton with many almost identical local optima where the hit on the performance is not critical if the model finds any of them. It was also proposed that many local optima (as found by having a derivation of 0) in the search landscape are in reality saddle points where it should be possible to improve the results further. Calculating the full Hessian is too costly therefore an approximation may be enough. (see here: `https://blog.terminal.com/no-more-local-minima/`)

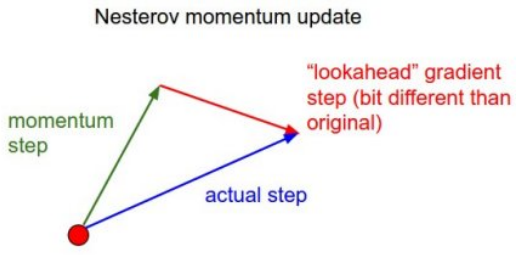
17 Gradient descent

17.1 How does gradient descent (red) /with momentum (green) work?



The normal gradient descent takes the gradient at the current position and then takes a step ($\propto$ learning rate) in this direction. This leads to unnecessary small steps when following a valley and too large steps at the end. GD with nesterov momentum first takes a step in direction of the momentum (approx. the same direction and length as the previous step) and then evaluates the gradient there and then takes a step with this "lookahead"-gradient.





18 Batch Gradient descent

18.1 Batch, minibatch, and stochastic gradient descent:

The gradient descent algorithm starts at some point, looks for the direction of steepest descent and makes a step proportional to the learning rate in this direction to get to a (at least local) minimum (or critical point). Calculating this gradient on the whole dataset is computationally expensive, therefore usually only a subset of the training samples are used for each step.

The following schemes exist[1]

- Stochastic gradient descent: Use only one sample — the resulting path in the search space may be rather erratic
- Batch gradient descent: Calculate gradient on the whole training data set. This is expensive and usually not necessary
- Minibatch gradient descent: Middle path between SGD and BGD—calculate gradient on a subset of medium size (32 to 128 samples are used usually). The resulting behaviour is smoother than SGD and faster as BGD.

Therefore MBGD is most commonly used.

18.2 What effects has the minibatch size?

The size of the minibatch isn't a critical parameter. A bigger minibatch results in a smoother descent of the objective function but makes leaving local minima more difficult and is also computationally more expensive.

18.3 Pseudo-code of minibatch-based training and validation with early stopping

```
for each of 200 epochs:
    for each minibatch in training set:
        train classifier, store loss and training accuracy
    for each minibatch in validation set:
        test classifier, store validation accuracy
    compute and report means over loss and accuracies
    if accuracy is better then best model so far store model
    if accuracy did not improve in the last n (5-20) epochs restore best model and stop
```

19 Weight and bias parameters

19.1 How do weight and bias parameters affect the input x?

The weight parameter is applied multiplicative to x, and the bias is added. $w = Wx + b$

19.2 What must be considered when initializing these parameters?

The bias is less critical and can be set to 0 for initializetion.

The weights are critical because of the multiplication and the gradient descent converges very different depending on the initialization. Extremely important is a difference in the weight matrix to break symmetry. A good initialization is $Norm(0, 1)\gamma$, the standard normal distribution times a hyper parameter γ that controls the magnitude of the weights.

19.3 What happens if the weights are set too large or too small?

The hyper parameter γ controls the magnitude of gradient and step size, so if set too large, the activation explodes, and if to small, the activation vanishes. In both cases, the algorithm most likely won't converge to a good model.

[1] Source: `http://sebastianruder.com/optimizing-gradient-descent/index.html#batchgradientdescent`

19.4 We discussed a heuristic for controlling the magnitude of weights. What is the intention behind this heuristic (no math required)?

The so called Xavier initialization tries to set the weights in way to preserve the variance of the input which is usually 1 with a mean of 0 (due to normalization). When each neuron is connected to D input neurons with weights sampled from $\gamma N(0,1)$ the resulting variance would be $\gamma^2 D \cdot 1$. Therefore γ is set to $\gamma = 1/\sqrt{D}$ s.t. the variance of the result is 1 again.

He initialization is an adaption of this scheme with $\gamma = \sqrt{2/D}$ that is optimized for ReLU activations.

20 Preprocessing

20.1 Explain the purpose of preprocessing.

The purpose is to cancel out variations that have a random cause (not related to class) thus should be neglected. For images, this are variations in brightness and contrast. Brightness can be normalized by mean subtraction. Contrast can be normalized by division with standard deviation or by histogram equalization. The final image will have zero mean and unit variance, which is the preferred input for ML settings.

20.2 How do per-sample normalization and per-trainingset normalization differ in terms of operation and purpose?

Per sample normalization takes statistic for mean and variance over one image, while per-trainingset normalization considers all images in the training set. While per sample normalization has the benefit of cancelling out under/overexposed pictures or the ones with different contrast.

Per sample normalization makes sure that each sample has zero mean and unit variance. For color images and per channel operation, this might yield unwanted results (e.g. underwater picture will have almost 0 red channel, and this IS important).

Thus, per sample normalization should only be applied for all channels or B/W pictures.

20.3 In the latter case, which (if any) preprocessing is applied during validation and testing?

For the statistics, only the training set is used, but the normalization operations must be applied to the test and validation set and also during inference.

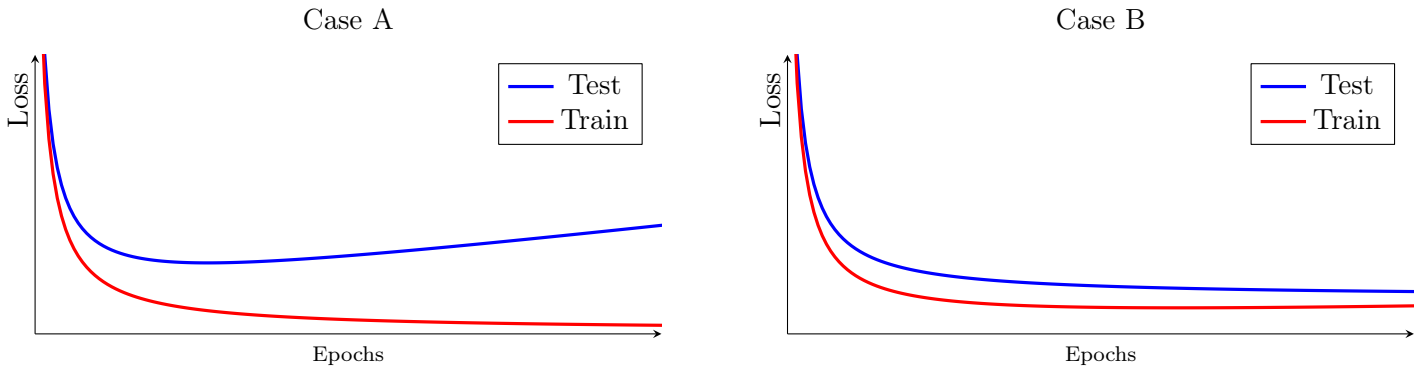
21 Optimization vs ML

21.1 What are the goals of optimization and machine learning?

In optimization, the goal is to find optimal parameters that minimize the loss on the training set. For machine learning, we need a general model that works well on unseen data.

21.2 Why do they differ?

A model with minimal loss on the training data is usually an overfitted model that will not generalize well and have worse results on unseen data. There are usually less specific parameters that have a higher loss on the training set, but work better on the test set.



Case A is the one with the focus on optimization, where the test set loss increases after a while, because the model overfits to the training data.

Case B is the desired outcome for ML, where we allow the training loss to stall or even increase, as long as the test set loss decreases. This will yield a more general model as in Case A.

22 Regularization

22.1 What is the purpose of regularization?

As stated above, in ML we are interested in a low test error, not necessarely a low training error. Regularization is a method to achieve a better test error, but with a possible higher training error, by getting a more general model that has decreased model variance (with increased biases) and is less prone to overfitting.

22.2 What is weight decay and what is its purpose?

Weight decay is a common regularization technique that penalizes too large weights (but leaves biases unaffected). This avoids that certain inputs have to much influence and forces the model to take all inputs into account.

Weight must be initialized with 0 mean, and in each step, the weights are adjusted in a regulization step:

$L_{reg}(\Theta) = \delta/2 \, \|w\|^2 + L(\Theta)$

where W is a vector of all weights and δ is the regularization-strength hyperparameter (if too small: no effect, if too big: dominates data loss). This way, all weights shrink in each update and decay to zero.

22.3 What is early stopping and how does it work?

For early stopping a validation set is needed. It works on the observation, that when training overfitting-prone models, the training error will always decrease, while at some point the validation error will start to increase.

So early stopping, a easy to implement and commonly used method, will store the model with the lowest validation error, and if the above situation occurs (no improvement or even higher error on validation set after some iterations), stop and return the stored model.

22.4 In deep learning, is it better to increase regularization or to decrease the model capacity by other means, and why?

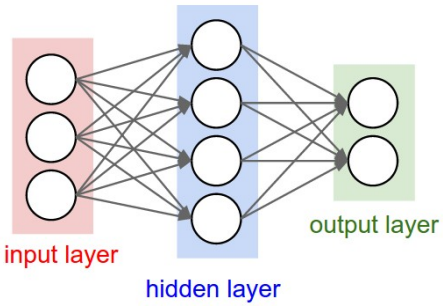
It's better to increase regularization, because in deep learning models work best if they have enough capacity to overfit, but are regularized properly.

23 Feedforward NN

23.1 What is the definition of a feedforward neural network?

A FF NN does not have cycles (as a RNN for example has). The samples get passed to the input layer and the values are then passed forward until they reach the output layer where the result is evaluated.

23.2 Which types of units do such networks have?



- Input units: Provide values, no computations, form the input layer
- Hidden units: Are connected to previous layer (either input or a hidden layer) and calculate a linear combination of the inputs (sum of value of connected units times weight of connection) which is then passed through an activation function (identity for regression, ReLU or tanh for classification)
- Output units: Last layer, may perform the same calculations as hidden layer.

24 MLP

24.1 What is the definition of a multilayer perceptron?

A MLP is a FF NN with input and output layer as well as some hidden layers. Neurons in layer l are connected to all neurons in layer $l-1$ (dense/fully connected layers).

24.2 What operation do (non-input) units perform?

Each neuron at layer l gets the output from the previous layer as input x_l and calculates the input x_{l+1} for the next layer by calculating $n_i(a_i^\top x_l + b_i)$ where b_i is the bias, n_i is the activation function and a_i is the vector of weights of neuron i.

24.3 What is an activation function and which functions are common?

After calculating the linear combination $a_i^\top x_l$ (scalar product of input and weight vectors) the resulting value is passed to the input function. For regression the identity function is used whereas for classification task a nonlinear function such as tanh or ReLU ($\max(0, x)$ is used to enable nonlinearities.

24.4 Why are multilayer perceptrons not suitable for deep learning for image analysis?

MLP usually don't improve their performance with 3 or more layers. This makes it impossible to learn a hierarchy of features as DL does. Also the number of parameters increases quickly with image dimensions.

25 Representation learning

25.1 What is the motivation and purpose for representation learning?

The idea behind representation learning is to extract features from images which are then used to train a model. Designing such features by hand is difficult though, therefore a two layer MLP approach is used: The first stage learns to extract features from the input and another stage then trains a linear classifier on these features.

25.2 How is deep learning related to representation learning, and what is its definition?

The problem with the outlined approach is that it must learn high level features in a single step. Deep learning is the idea to solve this problem in a divide and conquer way. Simple features that are easy to learn are combined to higher level features. The name DL is due to the many layers of this feature-hierarchy.

26 Locally connected layers

26.1 What is the motivation for using locally connected layers for image analysis? What is sparse connectivity

An image consists of several features, some are local and some are global. In most cases, the global features are a combination of several local features in a certain arrangement. The idea of locally connected layers is to train each layer only to extract features from some part of the image and then use additional layers to combine those.

This is called sparse connectivity. Instead of connecting the N input neurons with $N \times M$ connections (and corresponding weight) to the M output neurons only connect each of the M output neurons to some small subset of the input neurons.

26.2 Assume input data of dimension $W \times H \times D$. How many many weight and bias parameters does a locally connected (but not convolutional) layer have, assuming 3×3 connectivity and $W \times H \times 2$ output data dimension, and why?

Every output neuron is connected to 9 input neurons per channel, therefore $W \cdot H \cdot 2 \cdot 9 \cdot D$ weight parameters and $W \cdot H \cdot 2$ bias parameters (one per output unit).

27 Convolutional Layer

27.1 What is the purpose of a convolutional layer?

A convolutional layer is trained to extract features from the input. Early in the NN they aggregate information from input pixels in their receptive field (e.g. "a stroke in this direction"). Deeper in the network such low level features are then combined again to form higher level features ("a stroke in this and in that direction").... They use spatial structure of the input by applying their filter kernel to a pixel and its neighbours.

27.2 How do such layers differ from locally connected layers, and why are they almost always preferred?

Convolutional layers share their parameters, i.e. a kernel with a given size (often 3×3) is applied starting from the top left and then moved in stepsizes corresponding to the stride. This differs from locally connected layers that don't share the parameters and have a set of weights for every output pixel. As the features learned somewhere on the input are more often than not also useful on other parts of the input sharing the learned weights drastically reduces parameters while still maintaining accuracy.

27.3 How many parameters does a convolutional layer have?

A convolutional layer maps the $W \times H \times D$ input to $W' \times H' \times D'$ and has four hyperparameters:

- Number of filter K – this is then the output depth D_o
- Spatial extent F
- Stride S
- Amount of zero padding P

and must learn a $F \times F$ kernel for each of the K filter and each of the D input layers as well as a bias parameter for every layer in the output, therefore $F \cdot F \cdot D' \cdot D$ weight parameters and K bias parameters.

27.4 What are feature maps?

A feature map is obtained by applying a filter to the input. This is done by dragging the kernel in stepsizes corresponding to the stride over the input and calculate a new value for every step. The resulting filter map size depends on stride and zero padding as well as the size of the input: $W' = (W - F + 2P)/S + 1$ (same for height).

28 Receptive field

28.1 What is the receptive field of a neuron?

The receptive field is the region of the input that is seen by a certain neuron (directly or indirectly connected).

28.2 Assume a network consisting of two convolutional layers with 3×3 connectivity followed by a 2×2 max-pooling with stride 2 and again two convolutional layers with 3×3 connectivity. What is the receptive field of neurons in the final convolutional layer? How does the receptive field affect feature extraction?

Each 3×3 conv. Layer increases the receptive field by two neurons. A pooling layer with stride two increases the receptive field by a factor of 2. Starting from the final layer we have:

$$\text{Receptive field increase} = (1 + 2) \cdot 2 + 2 + 2 = 10$$

This is the increase in one dimension, therefore the receptive field of one final neuron is an area of 14×14 neurons (input pixels). A neuron can only extract a feature if it is in the receptive field. Therefore the higher level features in the final layers of the NN should have a receptive field that spans the whole (or at least almost the whole) input.

28.3 Why do convolutional layers usually use 3×3 connectivity?

As the number of parameters depends with $O(n^2)$ on the filter size a stack of 3 3×3 layer has approximately as much weights to learn as a single 5×5 layer. Therefore more smaller filters are usually preferred as additional layers improve the modelling of nonlinearity.

29 Pooling

29.1 What is the purpose of pooling layers?

Reduce spatial size to reduce amount of parameters and computation. Also helps against overfitting. Consolidate features learned in convolutional layers before the pooling layer.

29.2 Example: 2×2 max-pooling, stride 2

$$pool\left(\begin{bmatrix}1 & 1 & 2 & 4\\ 5 & 6 & 7 & 8\\ 3 & 2 & 1 & 0\\ 1 & 2 & 3 & 4\end{bmatrix}\right) = \begin{bmatrix}6 & 8\\ 3 & 4\end{bmatrix}$$

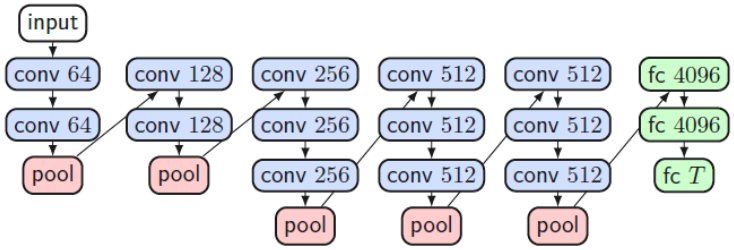
29.3 How many pooling layers with stride 2 should a CNN have assuming an input resolution of 64×64 and convolutional layers that do not change the resolution, and why?

The highest level features should have a receptive field that spans more or less the entire image. 4 to 5 pooling layers should therefore be a good choice. Exact number may vary based on the overall architecture.

30 CNN

30.1 Give a general overview of convolutional neural networks and their purpose

A convolutional neural network is a NN that consists of an input layer and several convolutional layers with many filters. Often convolutional layers are mixed with pooling layers to reduce the input size and the amount of parameters to learn.



30.2 What are the two stages of CNNs?

- Frontend: Consists of convolutional and pooling layers and is used to extract features
- Backend: Usually consists of one or two dense (fc) layers and is used to combine features learned in previous layers to make class predictions.

30.3 How is the depth of a CNN defined?

Depth is defined as the number of layers with parameters to learn and is thus mainly determined by the number of convolutional layers.

30.4 What effect does increasing the depth have?

A higher depth allows higher levels of abstraction. At least 6 conv layers should be used according to slides. Networks with higher depth may result in better performance, though with diminishing returns. In the case of ResNet the performance gains obtained by going from depth 50 to 100 is approx. twice as large as from going from 100 to 200. Without ResNet like architecture training more than 30 layers may be difficult though (increasing layers on VGGNet from 20 to 50 increases error rate as optimization algorithm doesn't find good parameters any more).

30.5 How does one choose a suitable network depth to solve a given image classification problem?

Current state of the art networks use many layers that learn features that achieve linear separability which allows simpler backends consisting of only one dense layer. When choosing higher depths additional measures must be used to still be able to efficiently train the model:

- Better initialization algorithms: Prevents signal from exploding or vanishing. Necessary that model with 30 ReLU layers even converges
- Batch normalization: Normalize output of layers to zero mean and std 1. Means and scaling factors are learned during training and should then be applied in testing mode.
- Residual learning: Propagte input to output before computing activation function. If optimal mapping is closer to identity small fluctuations are easier to find

31 CNN backends

31.1 What is the backend of a CNN?

In the backend the features extracted by the frontend are used to classify the samples. The last layer is therefore usually a dense (fc) layer with neuron count corresponding to the number of classes.

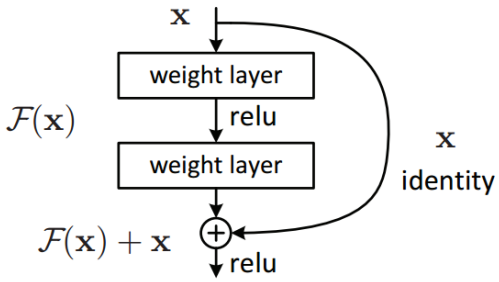
31.2 Discuss the backends of VGGNet and GoogLeNet/ResNet and their pros and cons.

- VGGNet: Uses two fc 4096 layers followed by a fc T (T = number of classes). Many parameters due to this MLP with two large hidden layers (later research revealed that these could be omitted with no performance downgrade)
- GoogLeNet: Average (instead of max as usual) pooling layer over the whole width and height of the previous layer, reducing the dimension from $W \times H \times D$ to D (number of filters of previous layer). These D features are then mapped to the classes with a single fc layer.
- ResNet: Same backend as GoogLeNet

32 ResNets

`http://kaiminghe.com/icml16tutorial/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf`

32.1 What are residual networks (ResNets) and which problem do they overcome?



ResNets are the current state of the art architecture. They consist of residual blocks which are composed of two 3×3 conv layers. An additional additive residual function $\mathcal{F}$ is learned that decides what should be added or removed from the input.

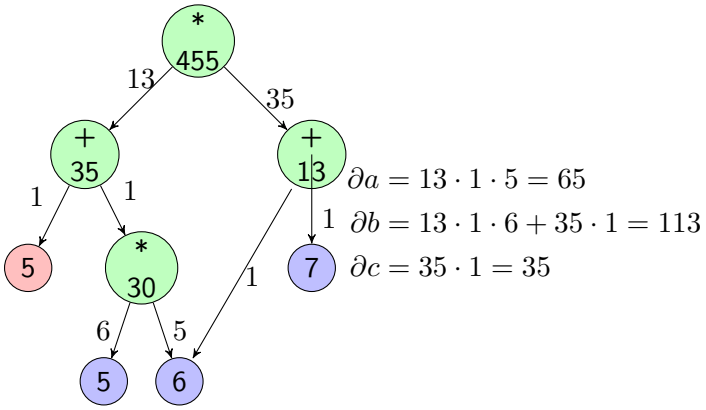
The output from a layer is added to all subsequent layers instead of propagated by multiplication with the weights from deeper layers. This makes the partial derivations also additive and prevents vanishing of the signal during backpropagation and enables efficient learning of nets with higher depth.

33 Back Propagation

33.1 What is the purpose of the backpropagation algorithm?

- Compute gradient of computation graph given by NN and current parameters
- Starting from input calculate derivatives of all nodes by their children. Sum up derivatives multiplied over all paths from output to input node to get derivation of output by input node.
- Gets inefficient really fast if multiple paths from input to output exist. Reverse accumulation strategies help in this case (reverse mode differentation)

33.2 Example



33.3 Explain algorithm at a given node.

- First calculate values at every node
- Then derive each node by input parameters to get partial derivatives
- Multiply every path from an input to output (chain rule $g(f)' = g'(f) \cdot f'$
- Sum up all paths from input to output (product rule: $(g \cdot f)' = g' \cdot f + f' \cdot g$

34 Data Augmentation

34.1 What is the purpose of data augmentation?

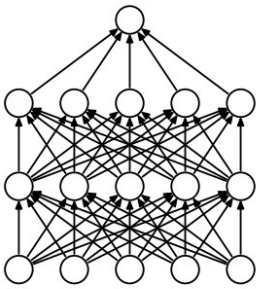
The bigger the training set the better the resulting NN can be. As obtaining huge datasets is not easy a good approach to circumvent this is extending the dataset at hand by various transformations such as mirroring, changing lighting, rotations, affine transformations, croppping, ...

34.2 What transformations work /don't work for a digit classifier

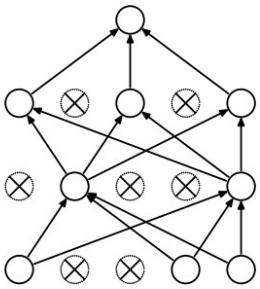
- Everything related to color/lighting does more or less not apply here as digits may be written by a pen of any color and therefore transforming the b/w should be the best approach
- Mirroring does not work on either axis
- Rotating the samples is a good idea as people may have a different skew in their writing. Though the rotation angle should be in a range between $\pm 10 - 20°$. Using arbitrary rotations is a bad idea as e.g. a "6" would then be mixed up with a "9".

35 Dropout

35.1 What is the purpose of dropout, how does it work, and why is it effective?



(a) Standard Neural Net



(b) After applying dropout.

Dropout sets the output of a neuron to 0 with probability p during training. This temporarily discards certain neurons which helps with regularization because the net tries to prevent depending too much on certain features.

35.2 Why do "dropped" neurons have no effect on the output of the next layer?

With very high probability, there will be enough paths left to allow backpropagation. There are ways for weights scaling. eg: The outputs are scaled with the dropout probability to prevent an effect from the dropout.

35.3 To which CNN layers is dropout commonly applied?

GoogLeNet applies dropout to the final pooling layer. ResNet does not use dropout. Generic CNNs can benefit from dropout on all layers.

36 Batch normalization

36.1 What is the purpose and aim of batch normalization?

The weights are initialized such that the signal progresses through the net with more or less the same strength. During training this changes due to updating of weights. The idea of batch normalization is to calculate mean and variance of every feature per minibatch and normalize the output.

It allows higher learning rates and increases robustness to bad initialization.

36.2 Why does it have a regularizing effect?

When using minibatch gradient descent (which is usually always the case), each sample is seen with several other (changing, due to shuffling every epoch) samples and therefore the normalization is nondeterministic for every sample which is advantageous for the generalization of the network. This may alleviate the need for dropout.

36.3 To which CNN layers is batch normalization applied and where?

CNN is applied to the activation (which is the argument of the activation function) in conv and fc layers, i.e. before the non-linearity. In convolutional layers the same normalization is applied to all neurons in one same feature map (the mean and variance per minibatch per feature map is used for this).

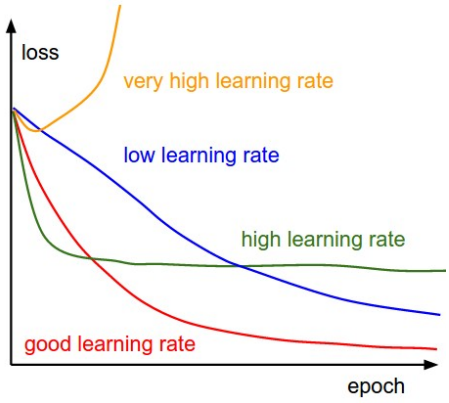
37 Learning rate

37.1 What does the learning rate hyperparameter specify?

The direction of steepest decrease can be calculated with the gradient, the size of the optimal stepsize is unknown and instead the stepsize given by the learning rate hyperparameter is used. The choice of this parameter is critical.

If learning rate is:

- very high: Gradient descent algorithm jumps over minima and is more or less a random walk in configuration space
- high: Finds a better configuration then the random starting configuration but gets stuck there as it "oscillates" around minimum (e.g. minimum is in a circle with radius 2 and algorithm makes steps with size 5
- proper: First missclassification rate quickly decreases and then slowly converges to some minimum
- low: Finds minimum as well but very slow (progress looks linear whereas in the beginning almost exponential decay of error should be possible)



37.2 Explain a heuristic for adapting the learning rate during training and why doing so can be beneficial.

Learning rate decay can be used to tune the learning rate to the optimal choice at each step. First start with a rather high learning rate and when there is no improvement for a few epochs scale the learning rate with a constant factor (e.g. $1/2$). Alternatively scale the learning rate every 5-20 epochs with a factor ($1/2$ to $1/10$). This ensures that big steps are taken by the algorithm when it is possible but the stepsize is reduced when further improvements can only be made by smaller steps.

38 Oversampling

38.1 What is the purpose of oversampling and how does it work?

A sampled passed to the neural net may depict something know but the viewpoint may be slightly off or the lighting may be unusual. Oversampling tries to overcome this challenge by applying various transformations to the image, classifying all transformed versions separately and average the class scores of the results.

38.2 What is ten-crop oversampling?

Ten-crop oversampling is a common technique which consists of making five crops of the image, one in the center and in all four corners and then using the 5 resulting images as well as their mirrored versions for classification.

38.3 What transformation could be applied for dog breed classification.

Ten-crop oversampling , Mirroring , Rotation , Skewing images , Adapting brightness

39 Model ensembles

39.1 What is the purpose and intuition behind using model ensembles?

As a CNN is trained starting from a random initialization different models may perform well on different samples. Model ensembles try to exploit this by taking a set of models (at most 10 models) that are trained independently and let each of the models classify the sample and then average over their predictions.

39.2 How might the individual models differ from each other?

- Various architectures
- Employing different data augmentation techniques
- Different initial weights

40 Medical tasks

40.1 What are the challenges in using medical imaging data?

- Small datasets/study populations
- Heterogenity of pathology
- Treatment response
- Subject specific variances
- Developmental differences
- Annotation needs experts

40.2 Can deep networks be used effectively for medical tasks (give 3 examples)?

- Pathology detection: Images of possible pathogens, 2D/2.5D/3D, Using CNNs. Input are e.g. images from MRT and network is used to assist diagnosis
- Segmentation:
- Shape modelling: Vessel detection in images with deep CNNs
- Classification:
- Action recognition: Use RNNs to recognize surgical activities from robot kinematics

40.3 How can we use the training data most efficiently?

Train traditional CNN with negative and positive samples. Use the false positives as extra instances for training the fine-tuned CNN.

Stacked autoencoders are another approach where first unlabeled data is used to pretrain the network layer by layer to get "a feeling" for the underlying distribution. Then use labelled data to fine tune the network for the desired purpose.

40.4 Give an explanation of the methodologies fine tuning and transfer learning and the benefit using them in medical imaging.

- Transfer learning: Train a CNN on huge dataset (ImageNet) and use the intermediate feature layers (up to the FC layer which connects to the 1000 classes) as feature extractor for the medical dataset.
- Fine-tuning: This method extends transfer learning. The idea is to retrain some of the deeper layers trained on ImageNet with the medical dataset as many features may be relevant for certain classes that are not present in the medical domain. It is usually a good idea to use different learning rates for the untrained and the pretrained layers.

41 2-3D deep CNN

41.1 What is the difference between 2D, 2.5D and 3D deep CNN networks?

- 2D: Takes image as input (X-Ray)
- 2.5D: Takes 3 images from orthogonal directions (3 X-Rays)
- 3D: Take many images, for example with MRT. Then either take Z 2 dimensional kernels or one three dimensional kernel.

42 Labelling

42.1 Explain the challenges, benefits and drawbacks of using expert vs. non-experts to obtain labels of medical image data?

Expert labelling:

- Expensive
- Time consuming
- Publicly availabe ground truth data may not be available

Non-expert labelling

- Annotations from single non-experts noisy
- Disagreements
- Crowd of non-experts can perform as well as medical experts

42.2 Give an example of an alternative method to expert-based manual labeling of medical data for deep learning approaches.

Crowdsourcing labels: Instead of using experts for annotation use a crowd of non experts. Add additional layer that aggregates the input from the crowd.

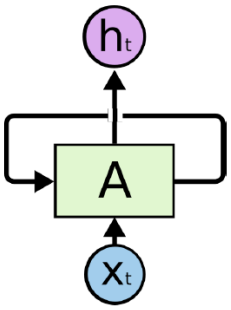
Example: Agg-Net (Deep Learning From Crowds for Mitosis Detection) had an aggregation layer, using CNN to weigh the votes of the different users with varying precision. The result are of the same quality as the ones from experts.

43 RNNs

43.1 Describe the main purpose of RNNs and give an example application.

RNNs are suited for temporal machine learning task, where not a single instance has to be taken into account, but a number of instances where the order is imporent. One example would be text generation, where the task is to create a single character or word, but the previous characters/words are very important for the next item. Video analysis, e.g. classification would be another example, where the isn't just a single image, but a whole video stream of several frames.

43.2 Draw and explain a sketch illustrating the overall architecture of such networks.



The main feature of RNNS are loops sucht that information can be passed from one temporal step to the next. Thus, the next state of A does not only depend on input x_t, for the step t, but also on the previous state of A_{t-1}. h_t is the output of the network at step t.

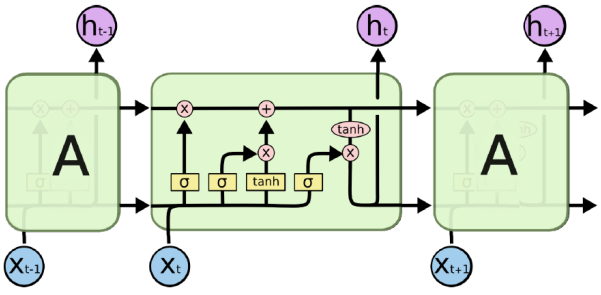
43.3 What is main limitation of traditional RNNs (opposed to LSTMs)?

A problem arises, when the important information is not only one or at least a few (defined) steps before t, but when it could be from an arbitrarily far away step. In the given example "I grew up in Italy ... I speak fluent *italian*" the important word to inference the language is Italy. The larger the potential gap, the more complicated the problem becomes.

44 LSTMs

LSTMs are designed for rememembering informations for long periods of time. So they solve above mentioned problem

44.1 List and explain the purpose of the different LSTM gates.



The Cell state C, the upper line, contains the core information. It is controlled by several sigmoid gates.
The left gate is the forget layer, which takes information from the previous output and the current input to get a multiplicative value between 0 and 1 for each information in C.
The middle gate is the input layer. It decides which values to update and create new candidate values for C.
The right gate is the output gate, which takes the updated and filtered state C, the previous output and the input to create a new output.