CNN2

Søren Olsen

Department of Computer Science
University of Copenhagen

Agenda

- A bit on Training, Annotation, Transfer learning
- Number of parameters and overfitting
- Regularization
- Convolutions
- Optimization: Gradient descent, SGD etc
- The zoo of network architectures (a start)

Training

To train the net (or estimate the value of the many parameters) these may be randomly initialized. Then an image is processed and the resulting error (difference between output and annotation) is recorded. Now a version of the backpropagation algorithm is used to change all parameter values.

This is repeated many times for a huge amount of annotated images spanning all the possible ways these may look.

Quite advanced numerical algorithms are used for training. Training may take from hours to weeks on a GPU. Without, it may (in many cases) not be realistic. Training may be complicated and involve a lot of tricks. If not done correctly training often fails.

Annotated data

A major obstacle in CNN usage is to establish large enough annotated data. Acquirering the images itself may be difficult. However human annotation (e.g. marking bounding boxes and writing class label) may take more time than the actual training.

Say that you can annotate an image with on average 10 items within one minute. Say that you for a 2M parameter net need 10 times as many training data. Then you have to work 2M minutes or 556 hours or about 14 weeks (40 efficient hours a week). Would you like that?

Annotation problems and solutions

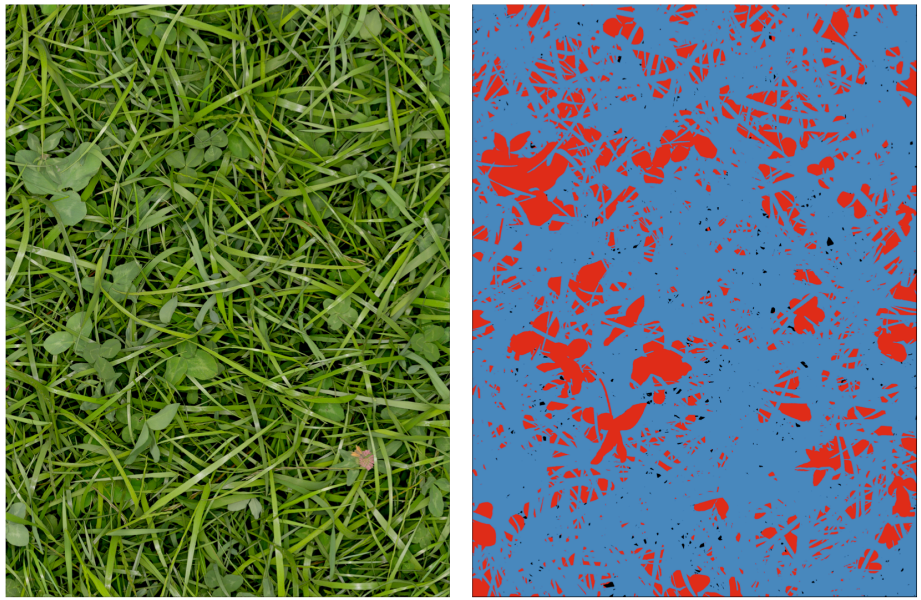
One solution is to use a Mechanical Turk, i.e. to pay other people to do the annotations. This may quickly give many annotations.

One problem in annotation is that this may be incorrect: Wrong label, badly placed bounding box or missing identification. Often people annotate differently. This will severely hurt the learning process. Correcting false annotations is a pain.

In some cases, use of (photo-realistic) synthetic images may solve the problem of getting enough annotated date. This approach has shown possible within semantic segmentation of street scenes, and classification of plants.

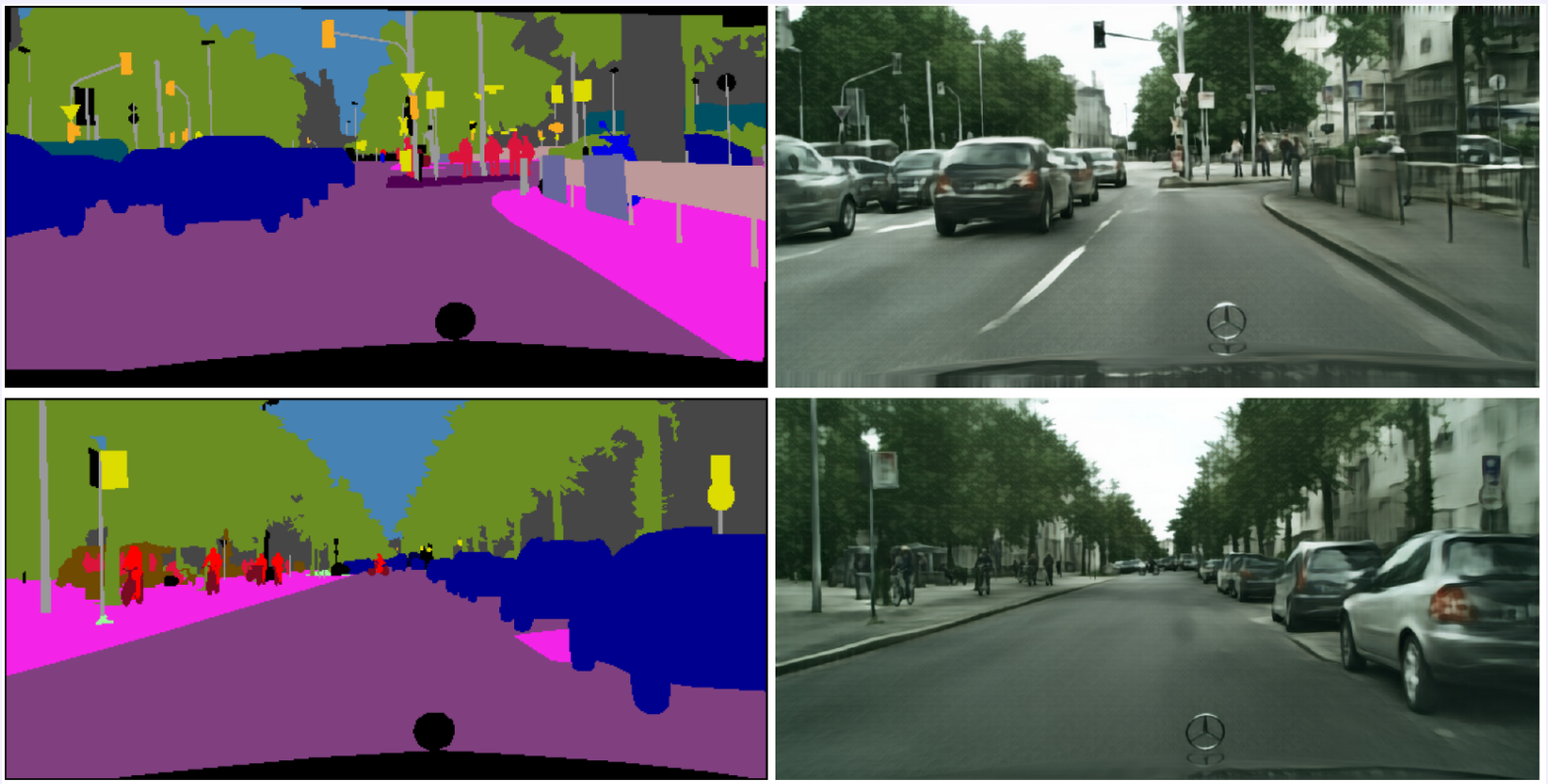
(**a**) (**b**) (**c**) (**d**)

Figure 5. Samples of single plants that are used for simulating images. (**a**) clover; (**b**) dandelion; (**c**) thistle; and (**d**) grasses.



(**a**) (**b**)

Figure 6. Simulated training data pair. (**a**) Simulated red, green, and blue (RGB) image; (**b**) Corresponding label image. Grass, clover and soil pixels in the RGB image are denoted by blue, red and black pixels in the label image, respectively.



Input semantic layouts | Synthesized images

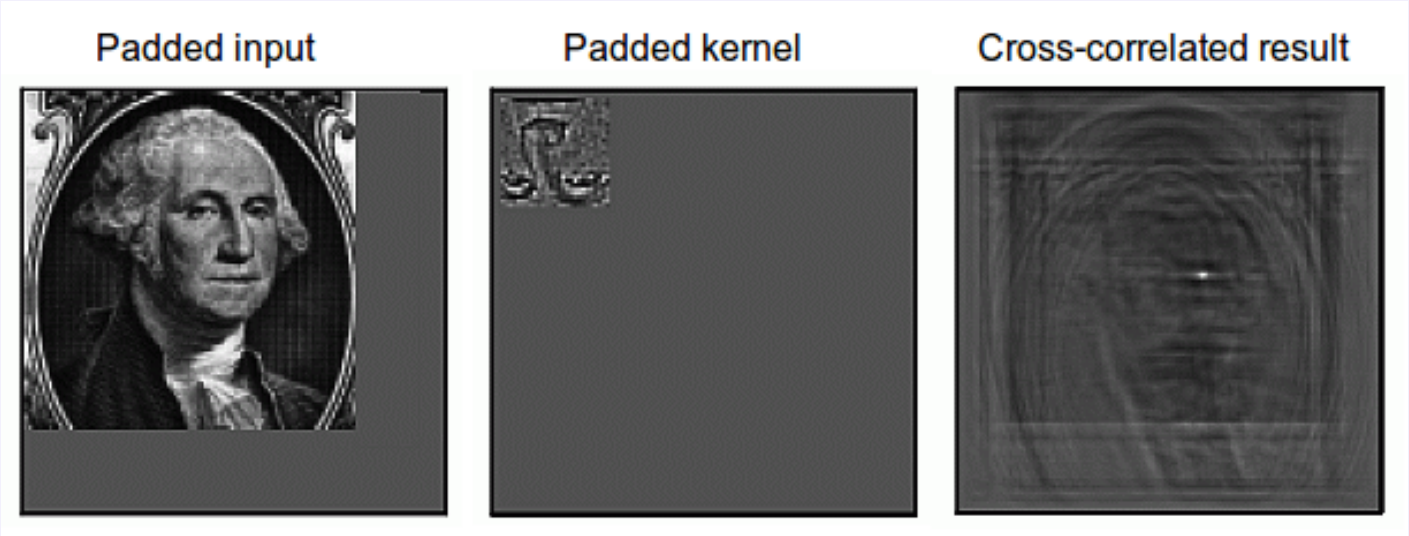
Transfer learning

A number of networks (say for recognition of different classes of objects) may share the backbone network. In this cases, if a good net exist, the trained parameters may be borrowed.

Many of the more famous nets, and their parameters, are publicly available. Then fine-tuning on a new (and much smaller) data set may result in fast and easy learning.

In some cases only the FC-layers may need redesign- and training. This limits the need for huge amounts of annotated data.

Correlations/convolutions are used in „matched filtering“ where we search for a maximum response to a fit to the filter.



Conceptually, this is just what we need.

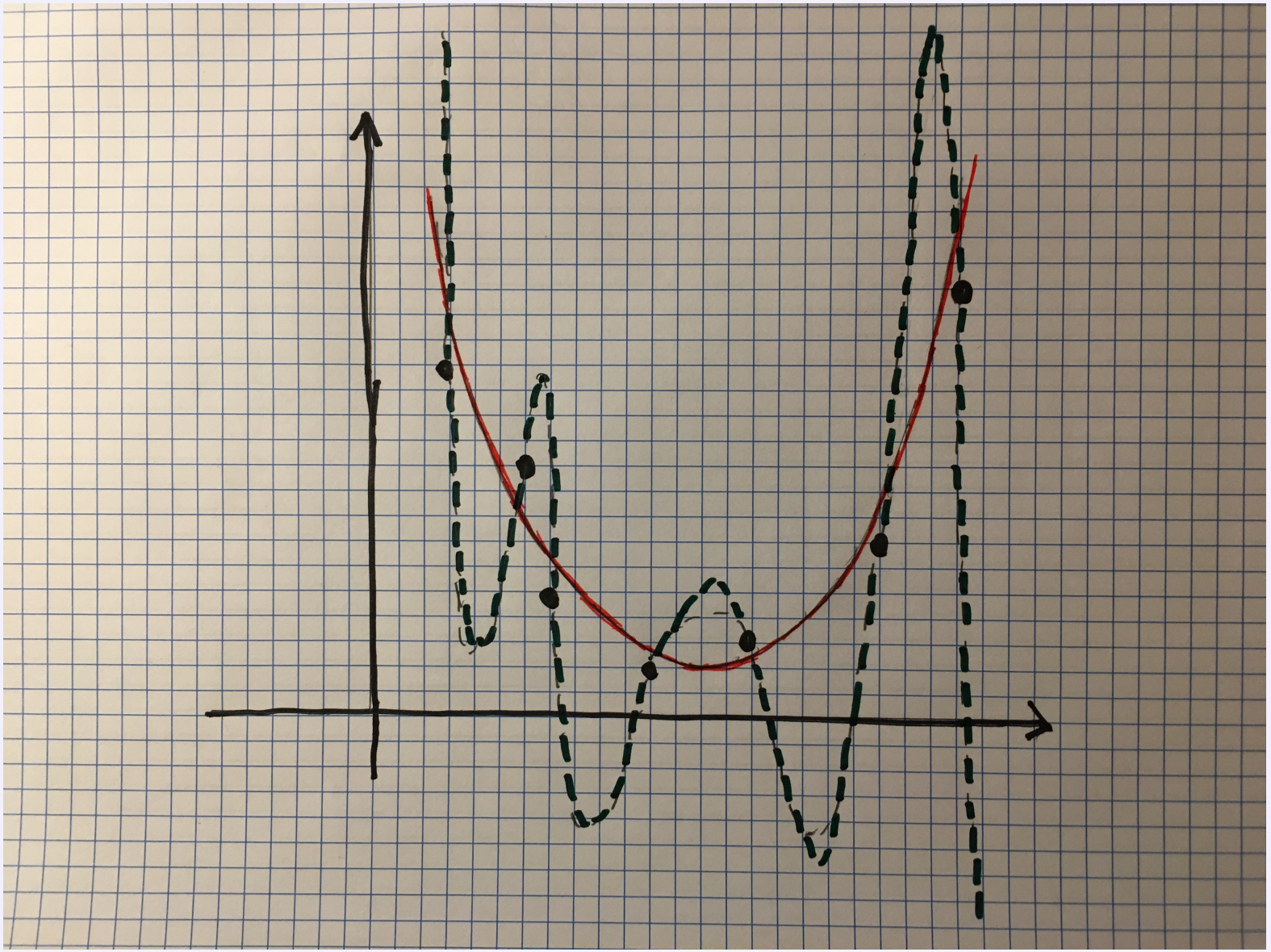
How many neurons do we need to detect specific objects and what happens if we choose too few or too many ?

Fitting and model selection

Assume that we want to fit a polynomium (in 1 variable x) to n data points. What order d of the polynomial should we use, i.e. how many parameters needed.

- If too few, we cannot model well and the fit will be bad
- Remember that all data er noisy
- If too high order, overfitting takes place (draw).
- How do we know if overfitting occurs ?

Overfitting - Exampel



Overfitting

Neural nets (of any kind) models by applying an exorbitant high number of parameters. Thus overfitting is likely to occur.

A large number of techniques and tricks are applied to avoid/reduce overfitting.

- Node pruning
- Weight sharing (CNN)
- Dropout
- Batch- (and other types of) normalization
- Early stopping
- Data augmentation
- Adversarial training
- etc

Regularization

A classical approach to avoid overfitting is to add a regularizing term to the objective function penalizing the number of parameters or the departure from smoothness (often measured by a sum of gradient magnitudes).

Such approaches are only partially applicable for CNN's, and often computationally costly.

We will later dive deeper into the different regularization methods.

Reducing the number of parameters

A fully connected network of $M \times M$ neurons working on an image of $M \times M$ pixels requires $M^2(M^2 + 1)$ parameters.

If only $N \times N$ pixels are contributing to each neuron we only need $(N^2 + 1)M^2$ parameters

If all neurons perform the same operation (i.e. a convolution) the the number of parameters reduce to $N^2 + 1$, a significant reduction and the motivation for CNNs.

If $N = 3$ then each convolution requires 10 parameters. In practice each input may be a vector with k channels increasing the number of parameters to $10k$.

Optimization

Many problems in Computer Vision may be formulated as an optimization problem. The idea is to formulate the wishes to the solution as a mathematical expression and then to apply optimization.

For CNN-based image classification we obviously want the network to produce an output that is consistent with the ground truth. Mathematically, this may be expressed in several ways, e.g. as the summed 2-norm differences or as the cross-entropy. (more about this later)

Many different techniques exist to perform optimization.

Optimization approaches

The loss function defines an energy landscape to which we search for the global minimum. Classical approaches include:

- Linear Algebra if the energy landscape is quadratic.
- Gradient descent if the energy landscape is convex.
- Simulated annealing if energy landscape is not simple.

Other approaches include:

- Exemplar/prototype coding (as in Bag of Words)
- Sparse coding
- Stochastic approaches if the data volume is high

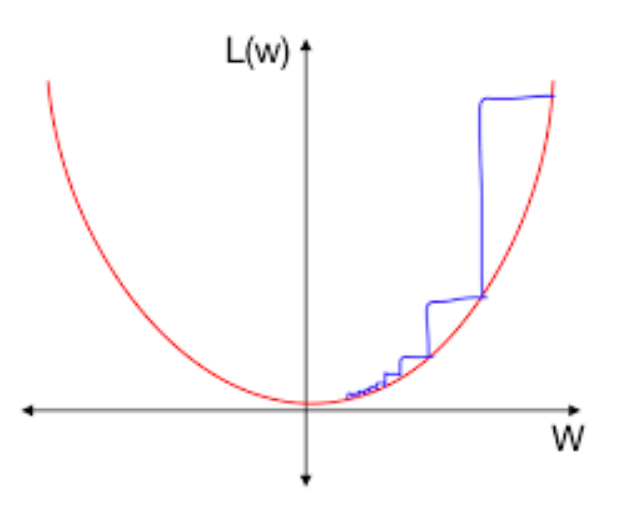
Optimization of CNNs are almost exclusively done using a variant of Gradient descent.

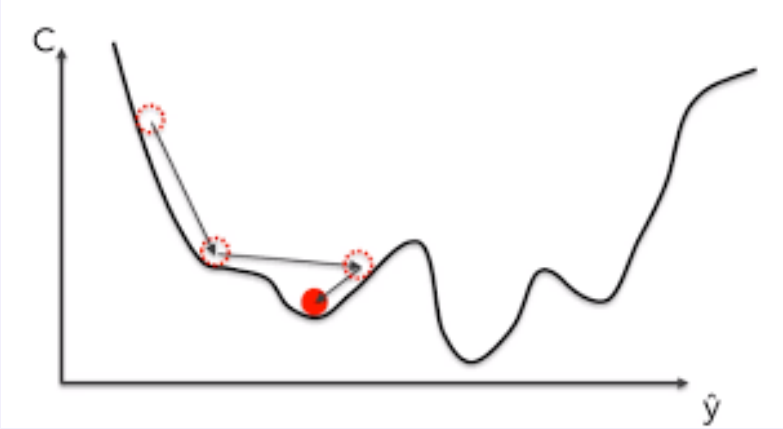
Gradient descent

Gradient descent works by iterative computation of the gradient and minimum approximation by moving downhill in the negative gradient direction.

$$x^{n+1} = x^n - \nu \nabla f(x^n)$$

where ν is the learning rate, i.e. the fraction of ∇f by which the change is made.

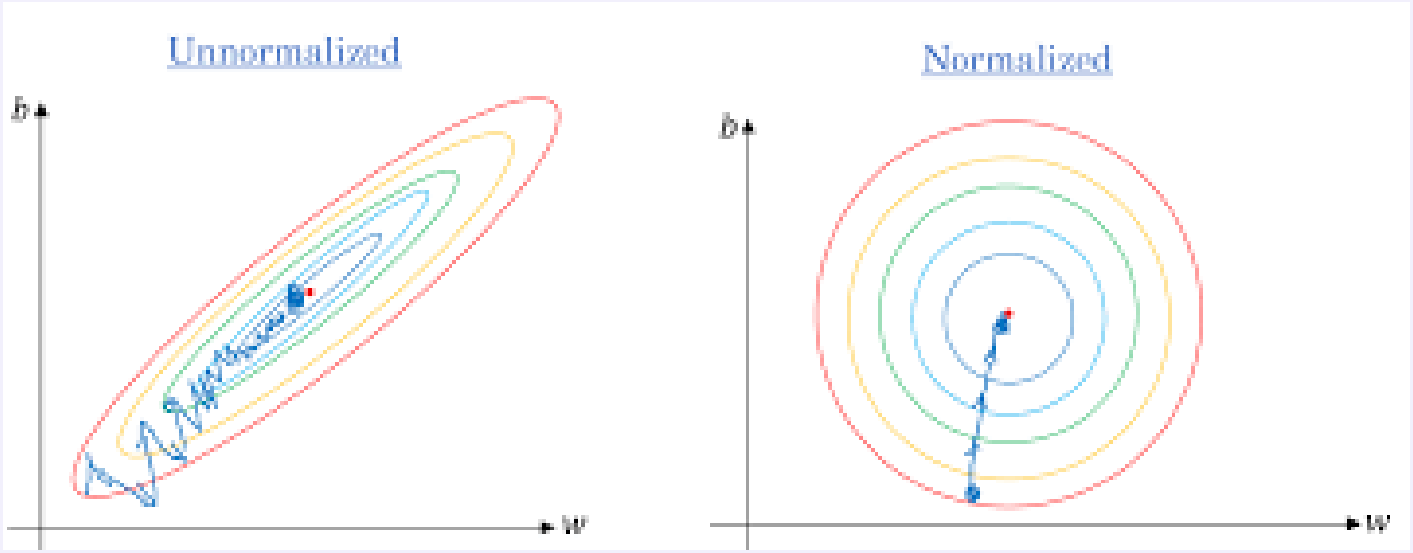




Gradient descent requires a good starting point and an energy landscape with no local minima.

To avoid oscilations (running zig-zag down through a narrow valley) often a momentum parameter is included:

$$x^{n+1} = x^n - \nu \nabla f(x^n) - \varepsilon \nabla f x^{n-1}$$



Momentum may accelerate convergence by smoothing out oscillations.

Autograd

The pytorch autograd package is a system for automatic differentiation (gradient computation) for all operations on tensors. A tensor may be seen as a generalization of a matrix to 3D. This is the tool for optimizing NNs, but have wider application areas.

To appreciate Autograd you first need to understand backpropagation. However, you may check out pytorch and the use of tensors (e.g. Google GettingStarted with PyTorch).

Stochastic optimization

The purpose of stochastic optimization is to avoid getting trapped in local minima. Depending on the framework, it may show up in many ways.

In NN-learning we could (in theory) compute the error (which we will back-propagate through the net) from all training data. It would be robust (i.e. identifying the right direction) but would be very slow.

As alternative, a single sample could form the basis for error propagation. This would be fast, but extremely noisy and risky.

In between, we may select a mini-batch including say 4-256 random samples (i.e. images).

Mini-batches and batch size

The larger the bach-size the better (more stable/robust) gradient for error propagation, but the more work per iteration.

It is not clear, given a net and a task, what batch size is optimal. What is clear is that smaller baches require more iterations for convergence.

If using small batch-sizes, a small larning rate, and use of momentum becomes essential.

Using large mini-batches may not be possible because of storage limitations in GPU's.

SGD, ADAM etc

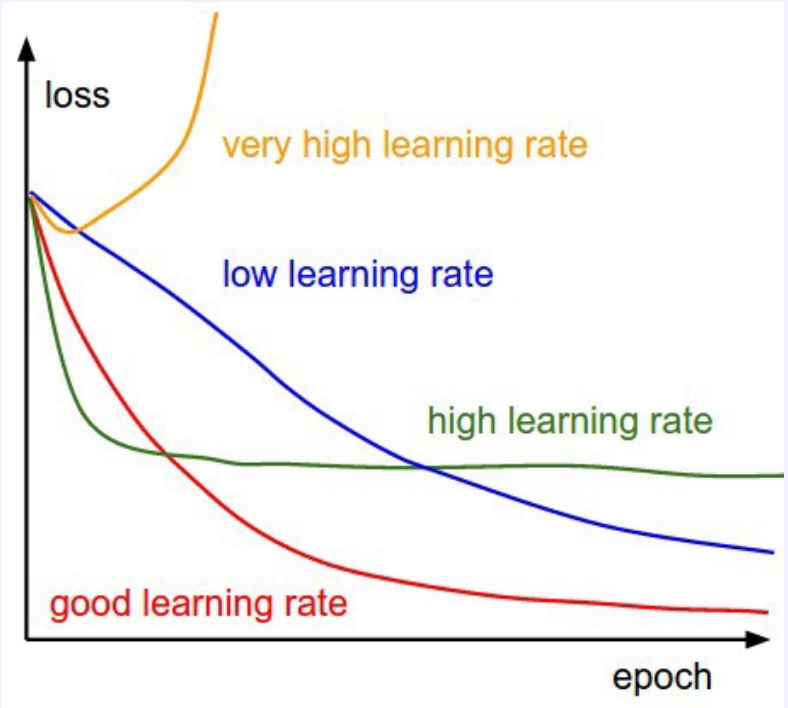
SGD is a standard NN-optimization framework where the user specifies the batch size, learning rate, momentum etc. Then, given training (and validation data) it optimizes the loss function during a number of epochs, each including as many iterations as necessary for covering the hole set of training data.

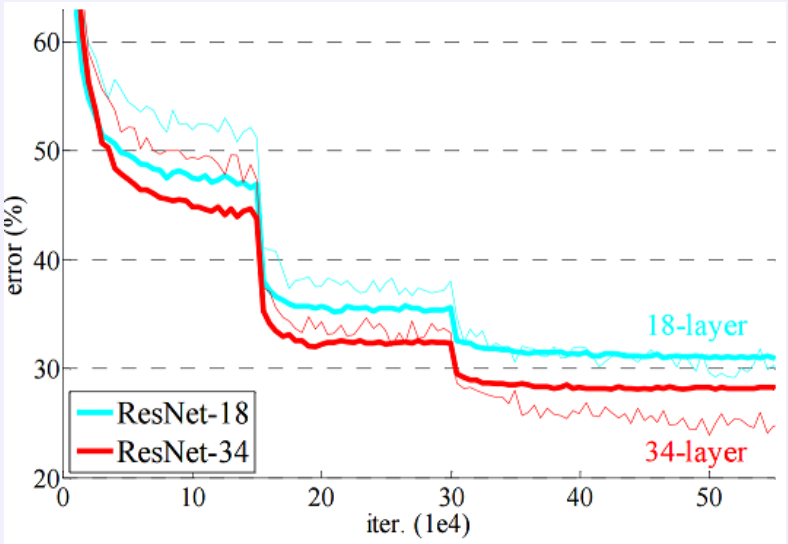
The user also specifies the maximum number of epochs and if the training rate should be lowered after each sequence of *n* epochs.

For an initial high learning rate, the movent in the energy landscape is large, probably bypassing the global minimum. Lowering the learning rate ensures that the locally deepest minimum is found.

Usually the choice of optimizer does not change the result (the minimum). Instead one optimizer may work/converge where another will diverge.

Chosing and lowering the learning rate





Class exercise (1 minute)

Overfitting happens when a too complex/expressive model is fitted to a too simple dataset.

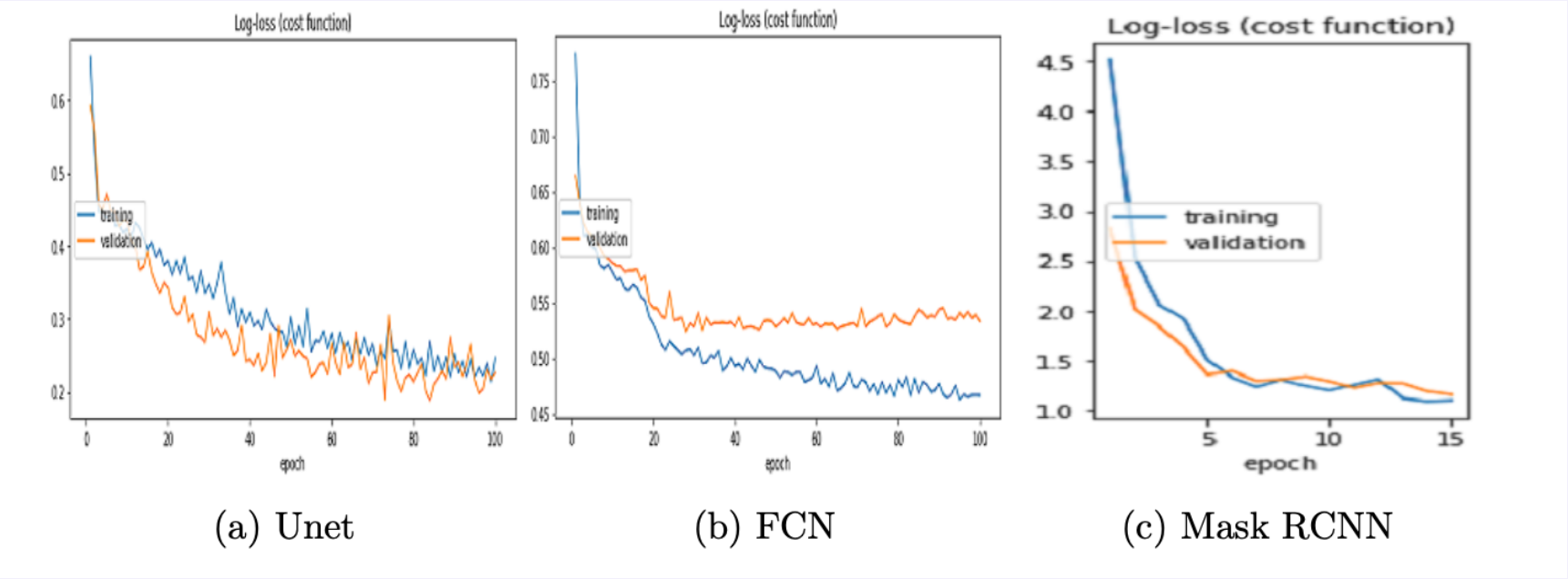
Figure out a method to detect when overfitting happens.

Training, validation and test data

When training a CNN, **never** use all data. Always keep back a test set, locked away and never used before all development is over. Often the test set includes say 10-15% of the data.

To train the network it is of vital impotance to avoid overfitting. To monitor that 10-15% is held back for validation. For each training epoch, the net is validated on this set. If the validation error is much larger than the training error, overfitting takes place and the training has failed.

If the validation error tend to increase early stopping may be used to avoid further overfitting.



(a) Unet (b) FCN (c) Mask RCNN

Overfitting takes place when the validation error flattens out while the training error keeps decreasing.

What to optimize

- The loss-function specifies what to be optimized
- The loss function should be differentiable
- Cross-Entropy is usually used for classification
- L2-norm usually used for regression
- The loss function may contain a number terms and may be designed to put weight on rare or difficult cases/classes.

More on loss-functions in a another set of slides.