Artificial Neural Networks 2

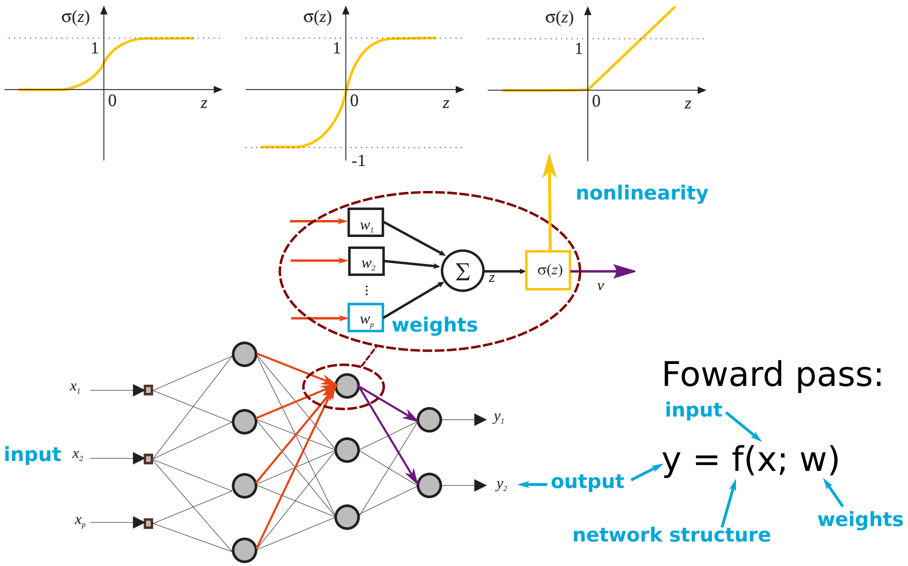
Tim de Bruin Robert Babuška
t.d.debruin@tudelft.nl

Knowledge-Based Control Systems (SC42050)
Cognitive Robotics

3mE, Delft University of Technology, The Netherlands

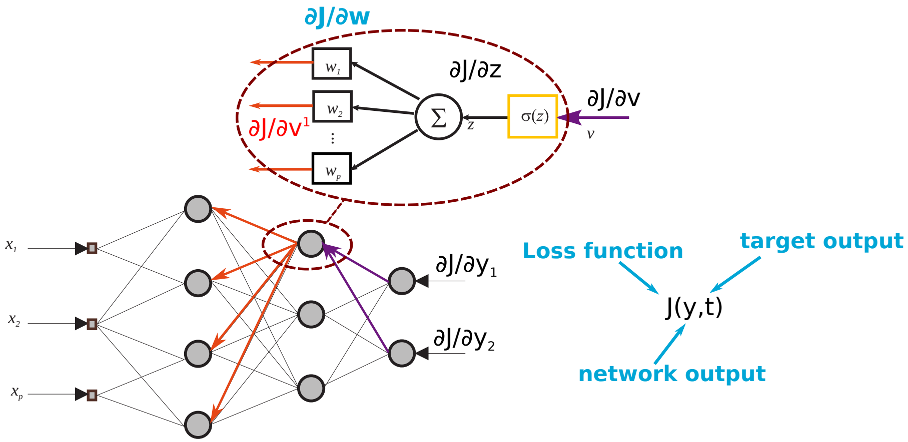
07-03-2018

Recap artificial neural networks part 1



Recap artificial neural networks part 1

Backward pass: calculate $\nabla_W J$ and use it in an optimization algorithm to iteratively update the weights of the network to minimize the loss J.



Outline

Last lecture:

1. Introduction to artificial neural networks
2. Simple networks & approximation properties
3. Deep Learning
4. Optimization

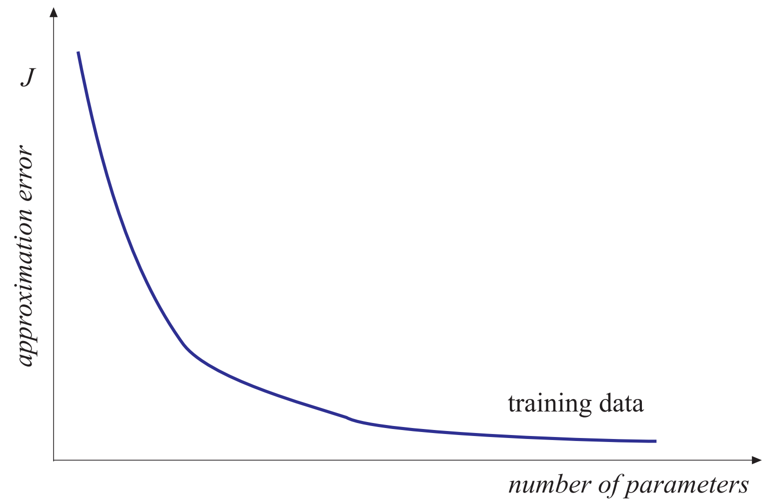
This lecture:

1. Regularization & Validation
2. Specialized network architectures
3. Beyond supervised learning
4. Examples

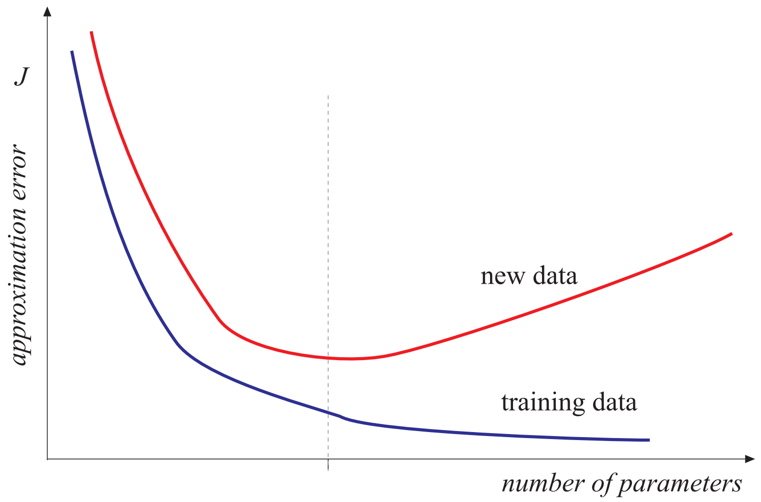
Outline

1. Regularization & Validation
2. Specialized structures
3. (Semi) Unsupervised Learning & Reinforcement Learning
4. Examples

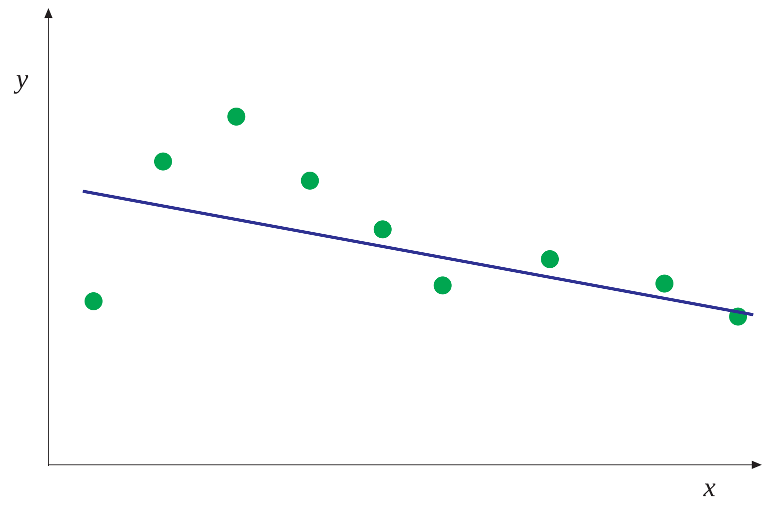
Approximation error vs. number of parameters



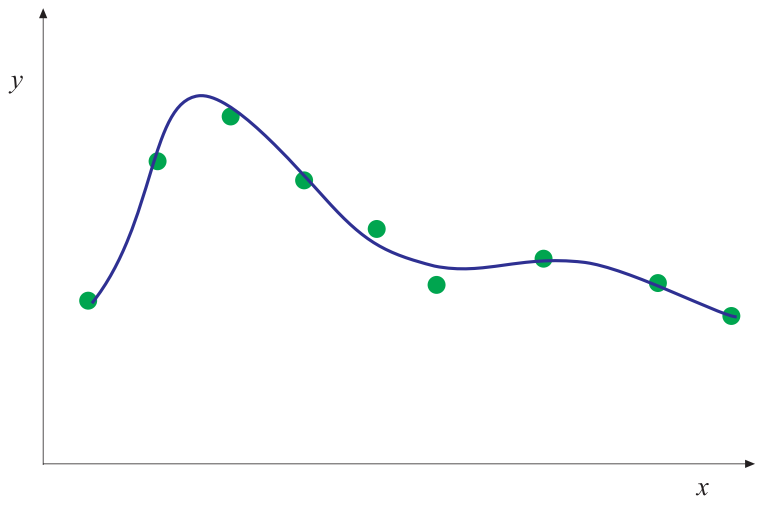
Approximation error vs. number of parameters



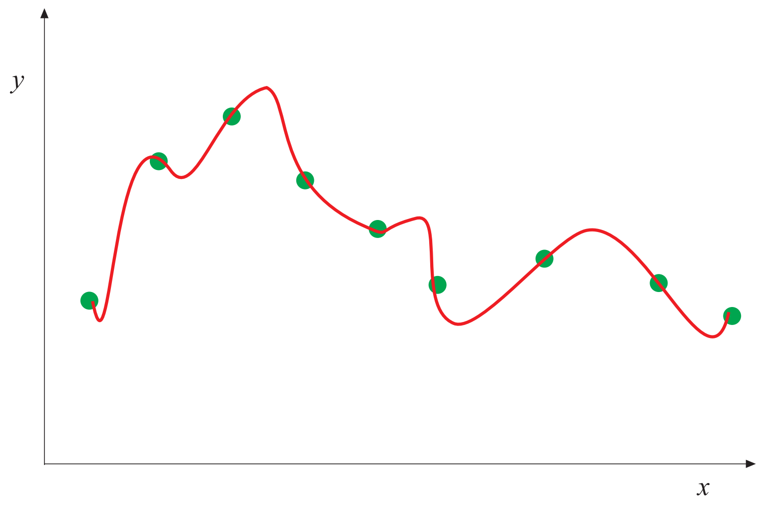
Underfitting



Good fit



Overfitting



Validation

System: $y = f(\mathbf{x})$ or $y(k+1) = f(\mathbf{x}(k), \mathbf{u}(k))$
Model: $\hat{y} = F(\mathbf{x}; \theta)$ or $\hat{y}(k+1) = F(\mathbf{x}(k), \mathbf{u}(k); \theta)$

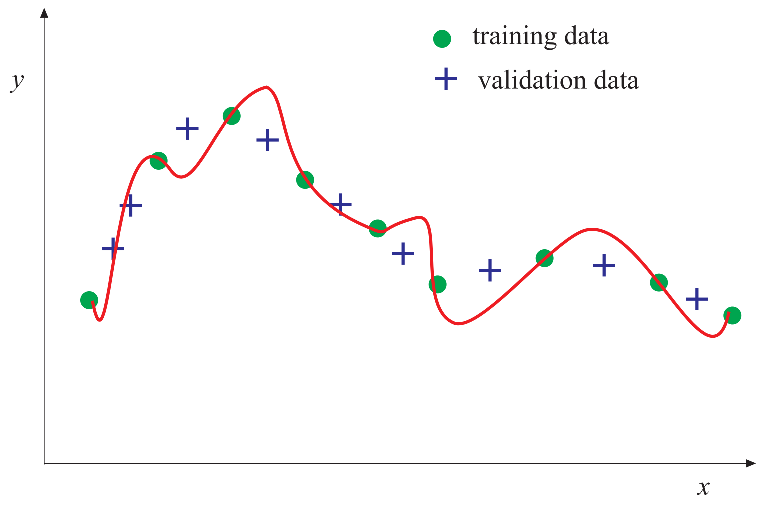
True criterion:

$$I = \int_X \| f(\mathbf{x}) - F(\mathbf{x}) \| \mathbf{dx} \qquad (1)$$

Usually cannot be computed as $f(\mathbf{x})$ is not available,
use available data to numerically approximate (1)

- use a validation set
- cross-validation (randomize)

Validation Data Set



Cross-Validation

- Regularity criterion (for two data sets):

$$RC = \frac{1}{2}\left[\frac{1}{N_A}\sum_{i=1}^{N_A}(y^A(i) - \hat{y}_B^A(i))^2 + \frac{1}{N_B}\sum_{i=1}^{N_B}(y^B(i) - \hat{y}_A^B(i))^2\right]$$

- v-fold cross-validation

Some Common Criteria

- Mean squared error (root mean square error):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y(i) - \hat{y}(i))^2$$

- Variance accounted for (VAF):

$$\text{VAF} = 100\% \cdot \left[1 - \frac{\text{var}(y - \hat{y})}{\text{var}(y)}\right]$$

- Check the correlation of the residual $y - \hat{y}$ to u, y and itself.

Test set

The *validation* set is used to select the right **hyper-parameters**.

- Structure of the network
- Cost function
- Optimization parameters
- ...

What might go wrong?

Use a separate *test* set to verify the hyper-parameters have not been over-fitted to the validation set.

Regularization

Regularization: Any strategy that attempts to improve the *test* performance, but not the *training* performance

- Limit model capacity (smaller network)
- Early stopping of the optimization algorithm
- Penalizing large weights (1 or 2 norm)
- Ensembles (dropout)
- ...

Weight penalties

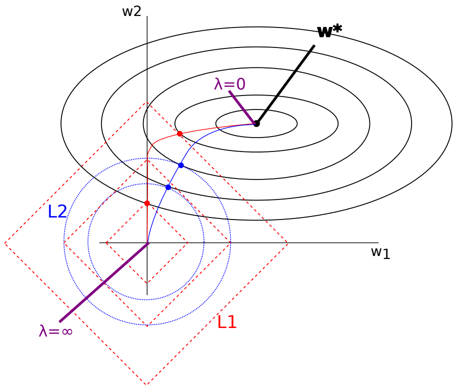
Cost function: $J_r(y, t, \mathbf{w}) = J^*(y, t) + \lambda ||\mathbf{w}||_p^p$

- $p = 1$: L^1 : Leads to 0-weights (sparsity, feature selection)
- $p = 2$: L^2 : Leads to small weights

Demo - Overfitting

Demo - L1 regularization

Demo - L2 regularization



Model ensembles

What if we train multiple models instead of one?

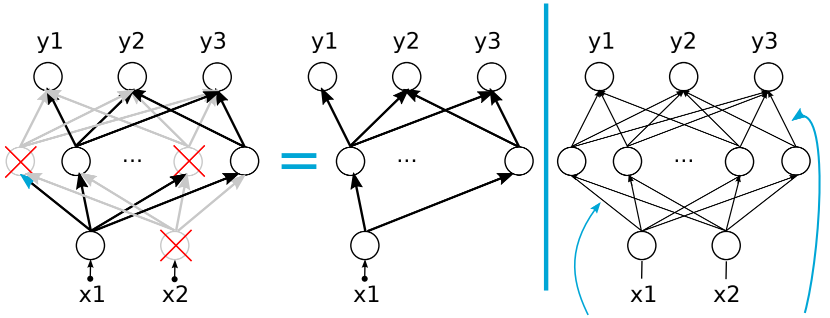
For k models, where the errors made are zero mean, normally distributed, with variance $v = \mathbb{E}[\epsilon_i^2]$, covariance $c = \mathbb{E}[\epsilon_i \epsilon_j]$. The variance of the ensemble is:

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j \neq i}\epsilon_i\epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

When the errors are not fully correlated ($c < v$), the variance will reduce.

Dropout

Practical approximation of an automatic ensemble method. During training, drop out units (neurons) with probability p. During testing use all units, multiply weights by $(1 - p)$.



randomly drop units during each training update, creating a new network (with shared parameters) every time.

To use the network, include all units but **scale weights**.

More data

The best regularization strategy is *more real data*

Spend time on getting a dataset and think about the biases it contains.

Data augmentation

Sometimes existing data can be transformed to get more data.
Noise can be added to inputs, weights, outputs (what do these do, respectively?) Make noise realistic.

Outline

Prior knowledge for simplification

Use prior knowledge to limit the model search space

Sacrifice some potential accuracy to gain a lot of simplicity

Example from control theory

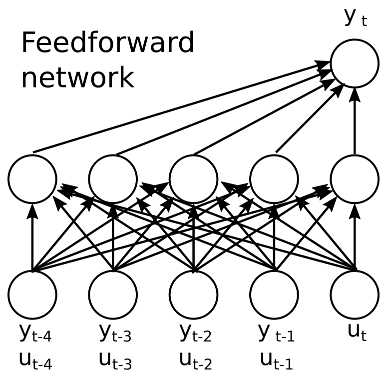
Reality: $y(t) = f(x, u, t), \quad \dot{x} = g(x, u, t)$

Usual LTI approximation: $y = Cx + Du, \quad \dot{x} = Ax + Bu$

Neural network analog

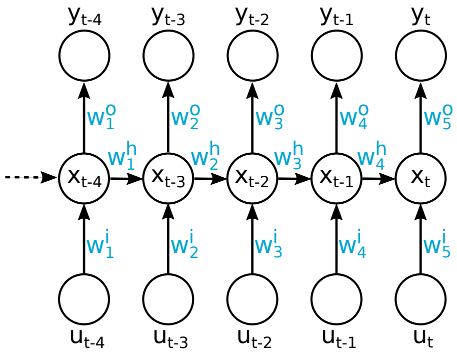
Predict y_t given $y_{t-n}, \ldots, y_{t-1}, u_{t-n}, \ldots, u_t$

Strategy so far:



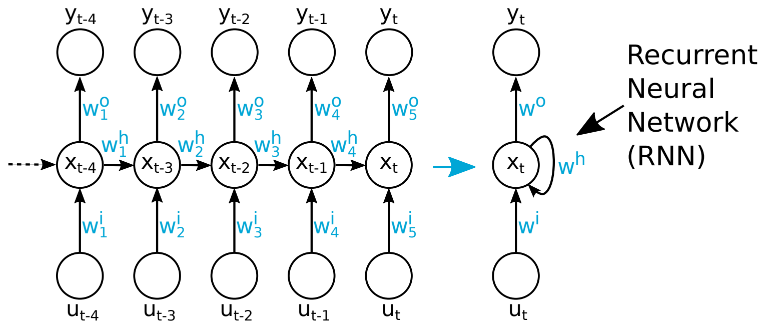
Neural network analog

Lets assume $y(t) = f(x(t), t)$ and $x(t) = g(x(t-1), u(t), t)$:



Weight sharing: temporal invariance

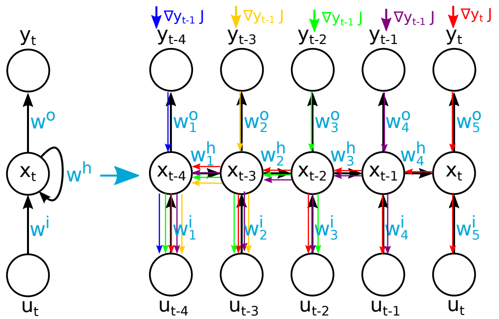
Lets add *temporal invariance*:
$y(t) = f(x(t))$ and $x(t) = g(x(t-1), u(t))$;
$\mathbf{w_1} = \mathbf{w_2} = \mathbf{w_3} = \mathbf{w_4} = \mathbf{w_5} = \mathbf{w}$



Significant reduction in the number of parameters $\mathbf{w}$

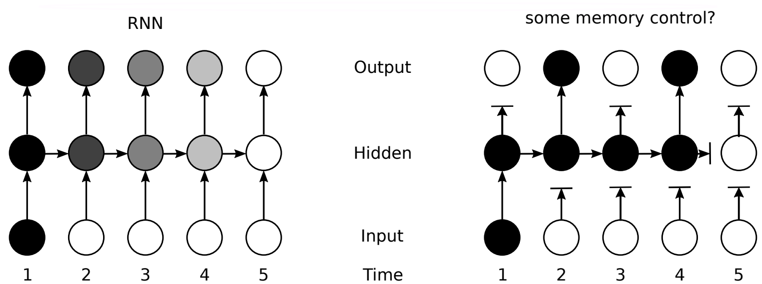
RNN training: Back Propagation Through Time (BPTT)

1. Make n copies of the network, calculate $y_1, \dots, y_n$
2. Start at time step n and propagate the loss backwards through the unrolled networks
3. Update the weights based on the average gradient of the network copies: $\nabla_w J = \frac{1}{n} \sum_{i=1}^{n} \nabla_{w_i} J$



The exploding / vanishing gradients problem

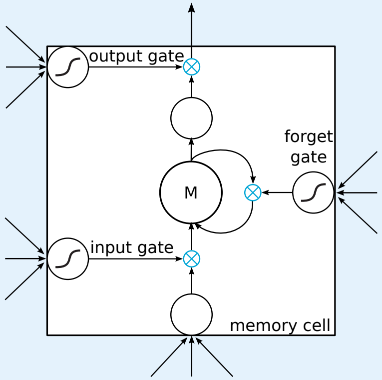
Scalar case with no input: $x_n = w^n \cdot x_0$
For $w < 1, x^n \to 0$, for $w > 1, x^n \to \infty$.
This makes it hard to learn long term dependencies.



Gating

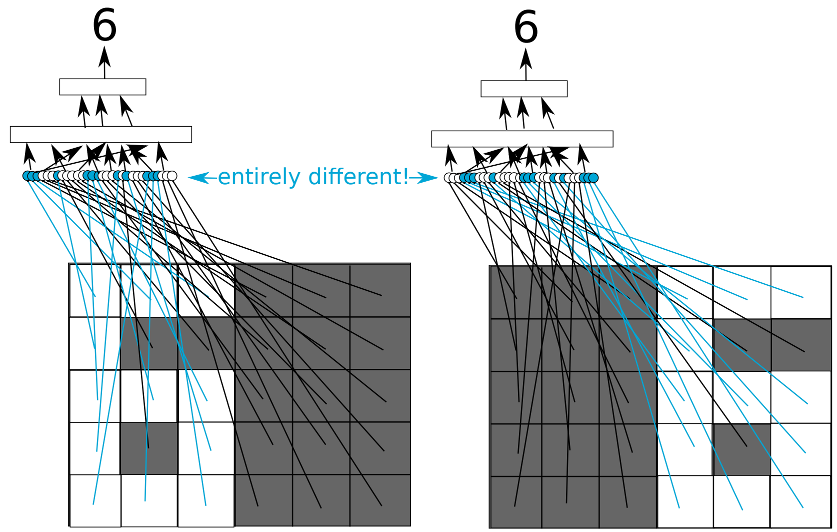
One more network component:
Element-wise multiplication of activations $\otimes$

Example: LSTM memory cell



Weight sharing: spatial equivariance

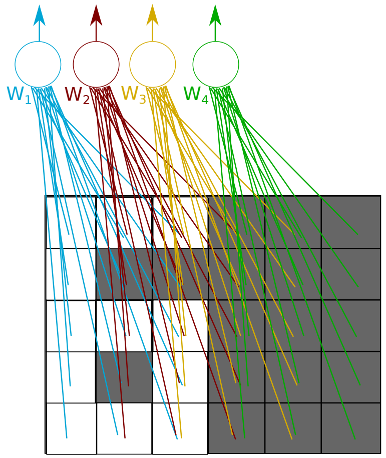
How to process grid like information (eg. images)? So far:



Weight sharing: spatial equivariance

We want *spatial invariance / equivariance*.

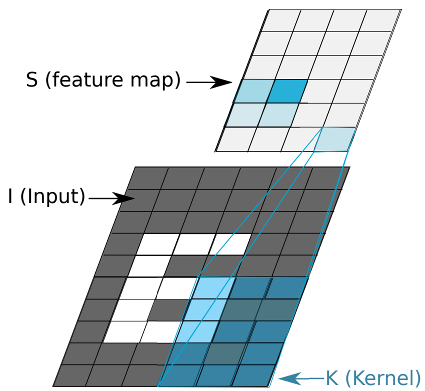
- Share pieces of network (eg our 6 feature detector).
- Copy the part of the network across the input space, enforce that the weights remain equal.



$\mathbf{w_1} = \mathbf{w_2} = \mathbf{w_3} = \mathbf{w_4} = \mathbf{w}$

Convolution

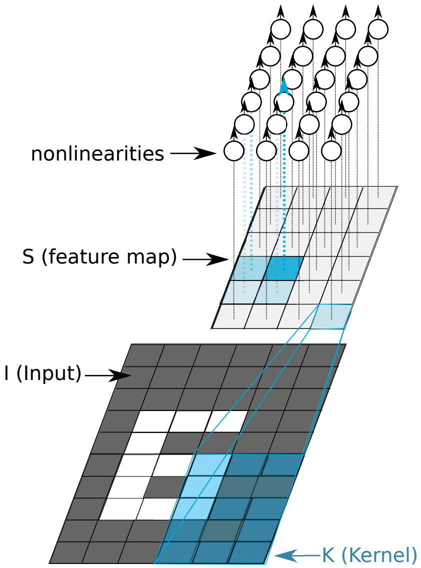
- Instead of thinking of copying parts of the network over the inputs, we can think of the same operation as sliding a network part over the input.
- Step 1: **Convolution**: $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n)$



Convolutional layer

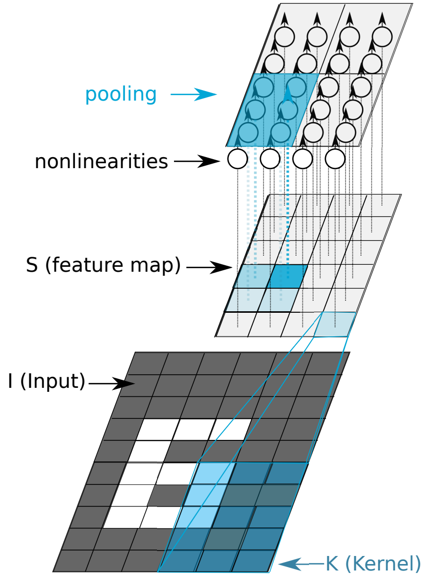
- Step 1: **Convolution**:
  $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$
- Step 2: **Detector stage:** nonlinearities on top of the feature map

What if we want *invariance*?



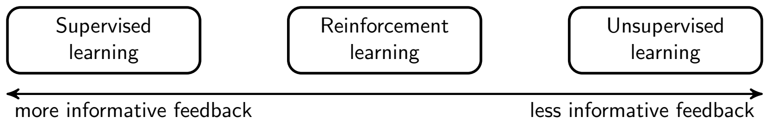
Pooling

- Step 1: **Convolution**:
  $S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$
- Step 2: **Detector stage:** nonlinearities on top of the feature map
- Step 3 (*optional*) **Pooling:** Take some function (eg max) of an area



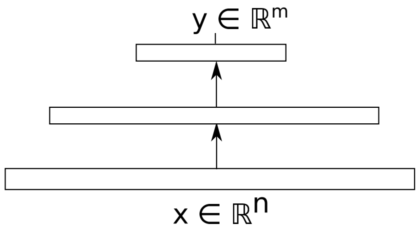
Outline

NN training: so far, we have seen supervised learning



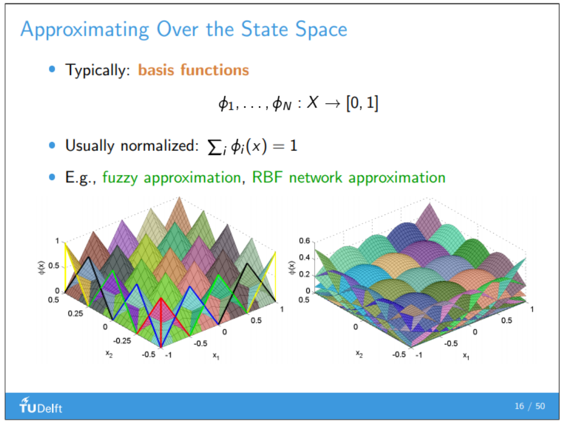
From SL to RL

So far: get a database of inputs x and target outputs t , minimize some loss between network predictions $y(x, \theta)$ and the targets t by adapting the network parameters θ:



RL with function approximation

Didn't we do this last week?


Approximating Over the State Space

- Typically: basis functions

$$\phi_1, \dots, \phi_N : X \to [0, 1]$$

- Usually normalized: $\sum_i \phi_i(x) = 1$
- E.g., fuzzy approximation, RBF network approximation

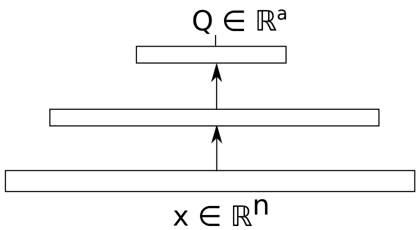
TUDelft 16 / 50

Global function approximation makes things trickier but potentially more useful, especially for high-dimensional state-spaces.

From SL to RL

DQN example: **get a database** of inputs x and **target outputs** t , minimize some loss between network predictions $Q(x, \theta)$ and the targets t by adapting the network parameters θ:

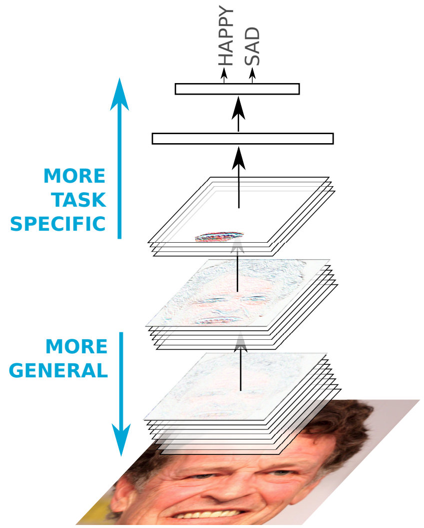
- Data {x, u, x', r} is collected on-line by following the exploration policy and stored in a buffer.
- $t(x, a) = r + \gamma \max_a Q(x', \theta^-)$: target network with parameters θ^- that slowly track θ for stability.



Additional training criteria

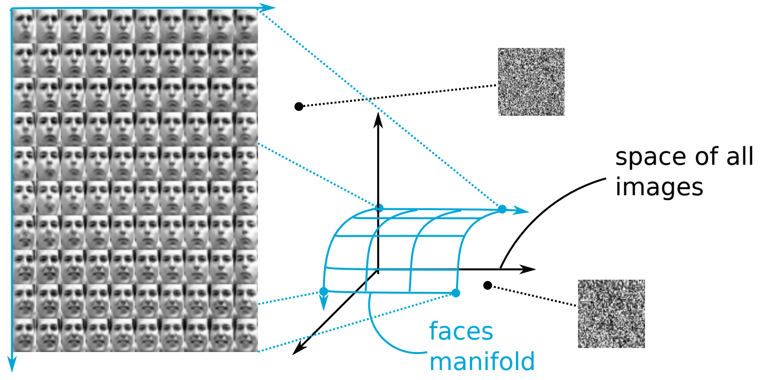
Inputs x are often much easier to obtain than targets t.

- For deep networks, many of the earlier layers perform very general functions (e.g. edge detection).
- These layers can be trained on different tasks for which there *is* data.



Additional training criteria

Previous lecture: data clustered around a (or some) low dimensional manifold(s) embedded in the high dimensional input space.



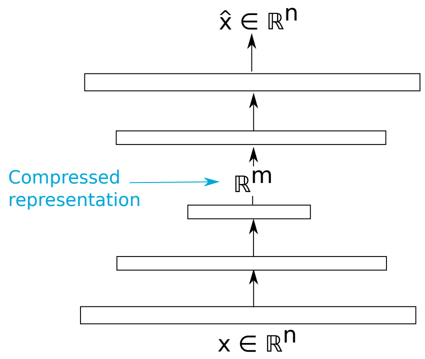
[1]

Can we learn a mapping to this manifold with only input data x?

[1] D. P. Kingma and M. Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*

Additional training criteria - auto encoders

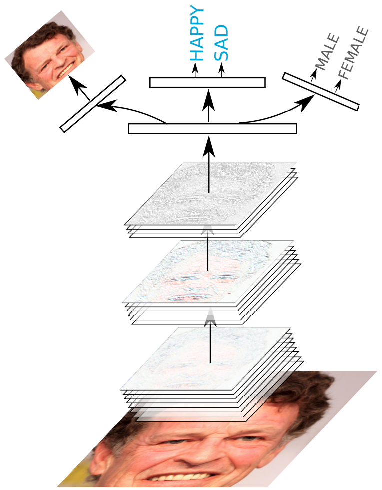
- Unsupervised Learning (UL): find some structure in input data without extra information(e.g. clustering).
- Auto Encoders (AE) do this by reconstructing their input ($t = x$).



Additional training criteria: regularization and optimization

Auxiliary training objectives can be added

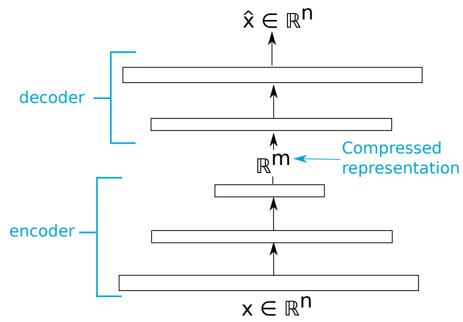
- Because they are easier and allow the optimization to make faster initial progress.
- To force the network to keep more generic features, as a regularization technique.



Generative models

Auto-Encoders consist of two parts:

- **Encoder:** compresses the input, useful feature hierarchy for later supervised tasks.
- **Decoder:** decompresses the input, can be used as a *generative model*.



Outline

1. Regularization & Validation
2. Specialized structures
3. (Semi) Unsupervised Learning & Reinforcement Learning
4. Examples

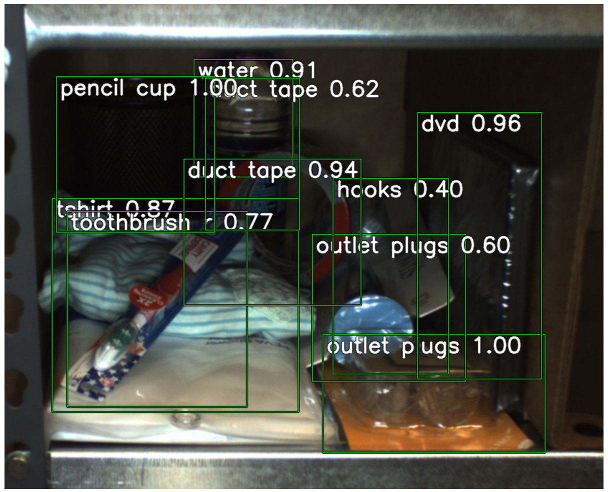
Applications of neural nets

- Black-box modeling of systems from input-output data.
- Reconstruction (estimation) – soft sensors.
- Classification.
- Neurocomputing.
- Neurocontrol.

Example: object recognition

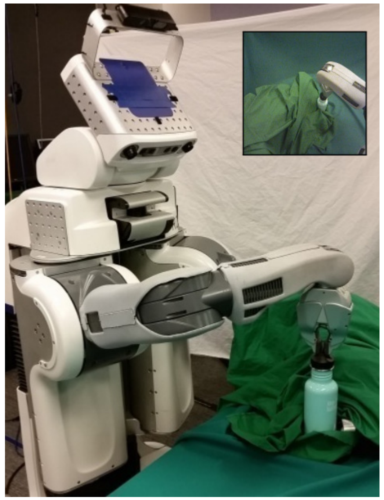
DELFTROBOTICS
intelligence at work
TU Delft
Robotics Institute

amazon picking challenge
winner 2016



Demo - movie

Example: control from images



2

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel (2016). "End-to-end training of deep visuomotor policies". In: *Journal of Machine Learning Research* 17.39, pp. 1–40

Summary

(Over-)fitting training data can be easy, we want to *generalize* to new data.

- Use separate **validation** and **test** data-sets to measure generalization performance.
- Use **regularization** strategies to prevent over-fitting.
- Use prior knowledge to make specific network structures that limit the model search space and the number of weights needed (e.g. RNN, CNN).
- Be aware of the biases and accidental regularities contained in the dataset.