Deep Reinforcement Learning

9 - Overview Deep Learning - Towards Deep RL

Prof. Dr. Malte Schilling

Autonomous Intelligent Systems Group

Overview Lecture

- Off-Policy Learning
  - Importance Sampling
- Deep Neural Networks

Recap – Off-Policy Learning

Evaluate target policy $\pi(a|s)$ to compute $q_\pi(s,a)$ while following behaviour policy $b(a|s)$:

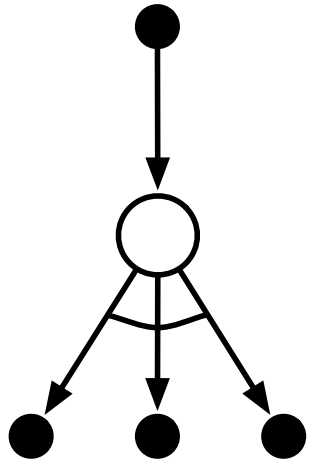
$$S_1, A_1, R_2, \ldots, S_T \sim b$$

Why is this important?

- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following a single behavioral policy

(Silver 2015)

Recap – Q-Learning – Off-Policy TD control



Q-learning

1. At time step t, we start from state S_t and pick action according to Q values, $A_t = \mathbf{arg\,max}_{a \in \mathcal{A}} Q(S_t, a)$; ε-greedy is commonly applied.
2. With action A_t, we observe reward R_{t+1} and get into the next state S_{t+1}.
3. Update the action-value function:
   $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \mathbf{max}_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$
4. $t = t + 1$ and repeat from step 1.

Difference to SARSA: Q-learning does not follow the current policy to pick the second action, but rather estimate q_* out of the best q values independently.

(Weng 2018; Sutton und Barto 2018)

Summary Q-Learning

Q-Learning is an off-policy approach for learning of action-values $q(s, a)$:

- Next action is chosen using behaviour policy $A_{t+1} \sim b(\cdot|S_t)$
- But we consider alternative successor action $a' \sim \pi(\cdot|S_t)$
- And update $q(S_t, A_t)$ towards value of alternative action

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \Big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t) \Big)$$

(Silver 2015)

Variance when using Importance Sampling

$$\mathbb{E}[f(x)] \approx \frac{1}{n} \sum_i f(x_i) \frac{p(x_i)}{q(x_i)}$$

When the importance sampling ratio is high, this will introduce large variance:

$$Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2, \text{ with } X = f(x) \frac{p(x)}{q(x)}$$

Therefore, we should aim for selecting $q(x)$ appropriately, i.e. in a way where $f(x)p(x)$ is already large.

Importance Sampling in RL

From a starting state S_t, the probability of the subsequent state–action trajectory occurring under a policy π is

$$
\begin{aligned}
& p(A_t, S_{t+1}, A_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi) \\
&= \pi(A_t, S_t) p(S_{t+1} | S_t, A_t) \pi(A_{t+1}, S_{t+1}) \cdots p(S_T | S_{T-1}, A_{T-1}) \\
&= \prod_{k=t}^{T-1} \pi(A_k | S_k) p(S_{k+1} | S_k, A_k)
\end{aligned}
$$

Importance Sampling in RL

In off-policy RL: we are optimizing policy π, but follow behavioral policy b.

Importance Sampling Ratio

The relative probability of the trajectory under the target and behavior policies is given as

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

Note: The ratio only depends on the probabilities of selecting an action wrt. to the two differing policies! Does not require knowledge on transition probabilities.

Importance Sampling as a Correction

Intuition:

- scale down rewards that are rare under π, but common under b
- scale up rewards that are common under π, but rare under b

Importance sampling can dramatically *increase variance*.

(Hasselt und Borsa 2021)

(Ordinary) Importance Sampling

Goal: estimate the expected returns for the target policy π

$$\mathbb{E}[G_t|S_t = s]$$

Available: only returns G_t due to the behavior policy which can give us

$$\mathbb{E}[G_t|S_t = s] = v_b(s)$$

The ratio $\rho_{t:T-1}$ transforms the collected returns to have the right expected value towards the target policy:

$$\mathbb{E}[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s)$$

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from b to evaluate π
- Weight return G_t according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/b} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{b(A_T|S_T)} G_t$$

- Update value towards corrected return

$$v(S_t) \leftarrow v(S_t) + \alpha\left(G_t^{\pi/b} - v(S_t)\right)$$

(Silver 2015)

Importance Sampling for Off-Policy TD

- Use TD targets generated from b to evaluate π
- Weight TD target $R + \gamma v(S_{t+1})$ according to similarity between policies
- Only need a single importance sampling correction

$$v(S_t) \leftarrow v(S_t) + \alpha\left(\frac{\pi(A_t|S_t)}{b(A_t|S_t)}(R_{t+1} + \gamma v(S_{t+1})) - v(S_t)\right)$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

(Silver 2015)

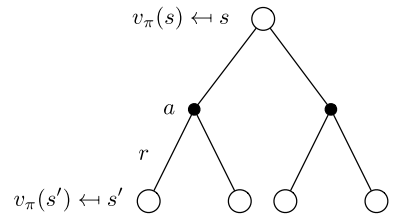
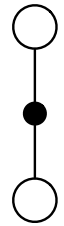
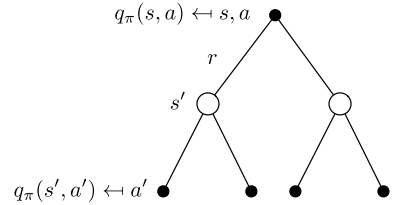
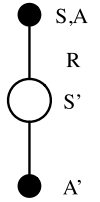
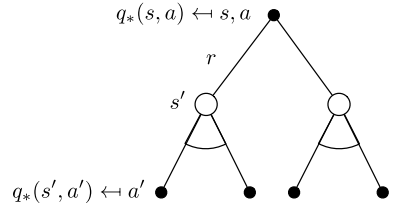
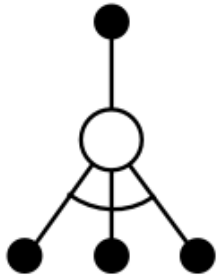
Q-Learning - No Importance Sampling needed

Q-Learning is off-policy. But we update the value functions:

$$q'(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha_t \Big(R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t) \Big)$$

- No importance sampling is required
- Next action is chosen using behaviour policy $A_{t+1} \sim b(\cdot|S_t)$
- But we consider alternative successor action $A' \sim (\cdot|S_t) and update$q(S_t,A_t) towards value of this alternative action

(Silver 2015)

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Relationship Between DP and TD 2

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

(Silver 2015)

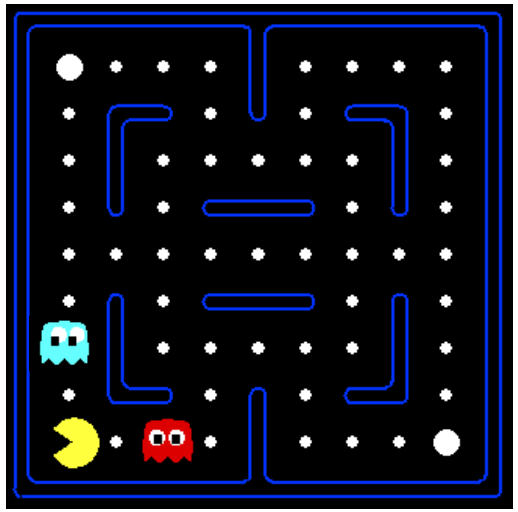
Towards Deep Reinforcement Learning

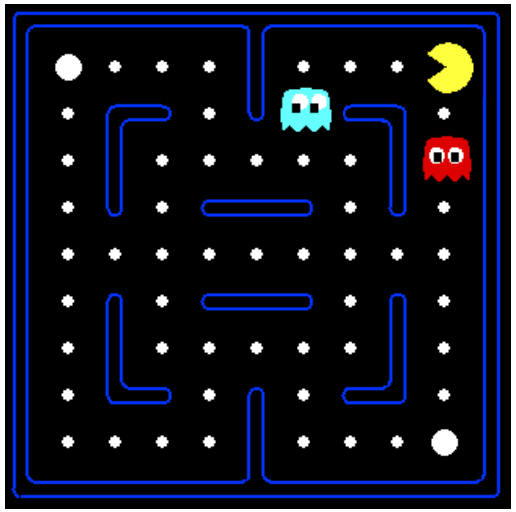
Drawbacks of Tabular methods

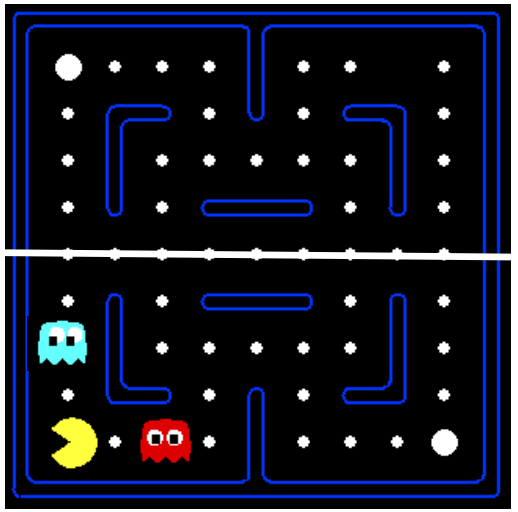
For tabular methods like basic Q-Learning: we keep a table of all Q-values.

In real world application: not possible to learn about every single state:

- too many states (or even continuous input spaces) to visit in training
- table would be too large for so many states







(Klein und Abbeel 2014)

Generalization over States

In order to deal with continuous or large state spaces, we want to generalize. For this, we use *Function Approximation*:

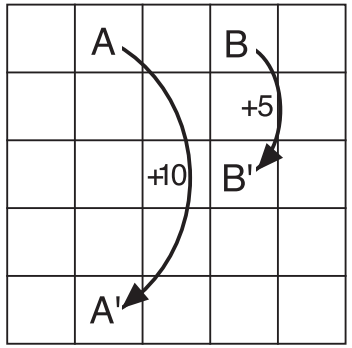
- Learn about a small number of training states from experiences.
- Generalize these experiences to new, similar situations.

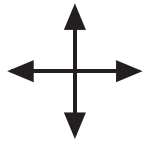
As a basic idea for *Deep Reinforcement Learning*: use Neural Networks for function approximation.

Value Function in Grid World Example

Towards a more realistic Grid

We considered this simple discrete grid environment.

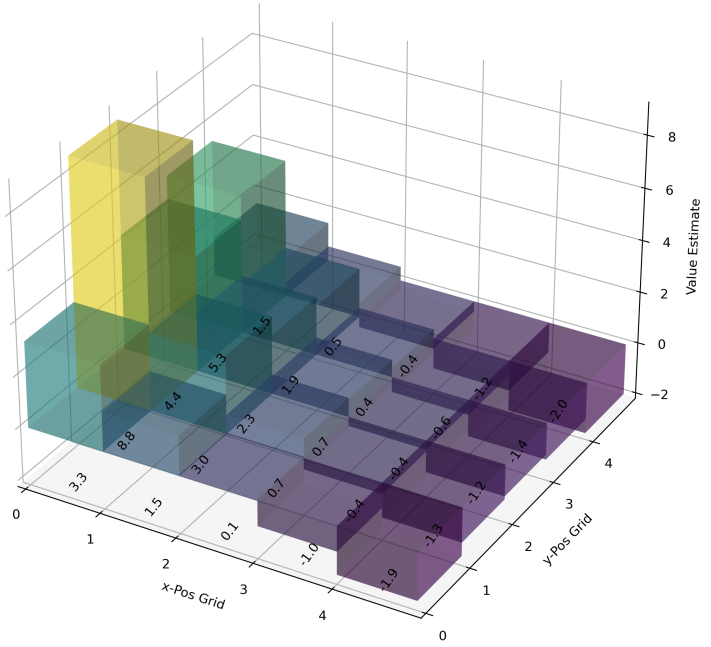




Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

For a real robot: We would consider a continuous state space (position).



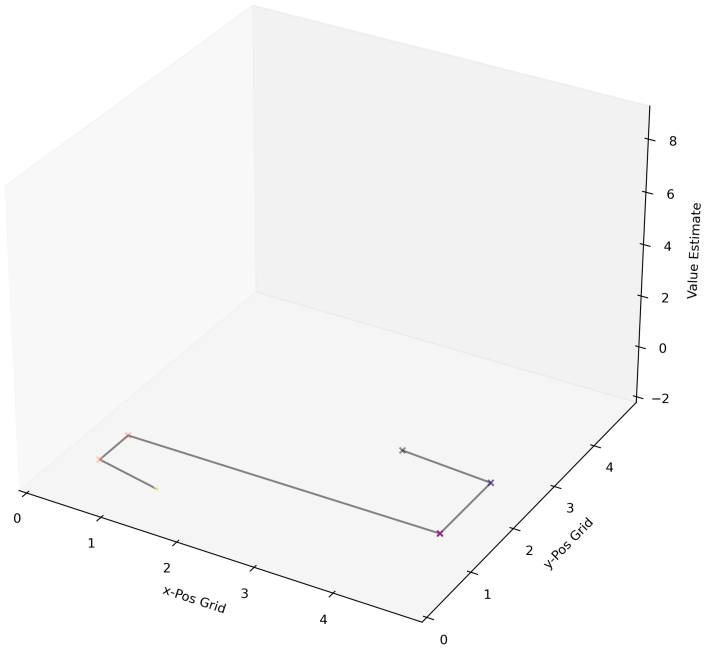
A robot sampling a random trajectory in the grid environment

Environment taken as continuous.

Observations: Continuous two dimensional position

Actions: Discrete (for now) directions and stepping

Position Update: Not completely deterministic



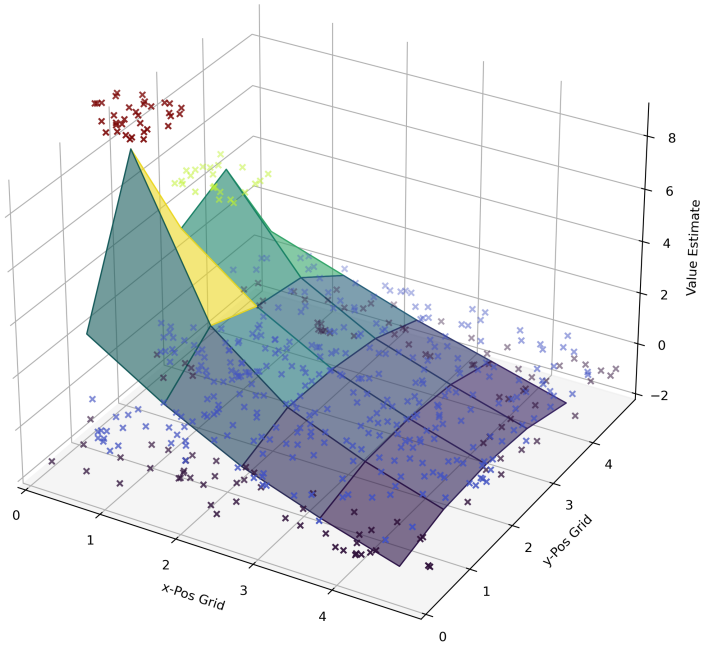
Introduction of a Value Function over the continuous Environment

Environment taken as continuous.

Observations: Continuous two dimensional position

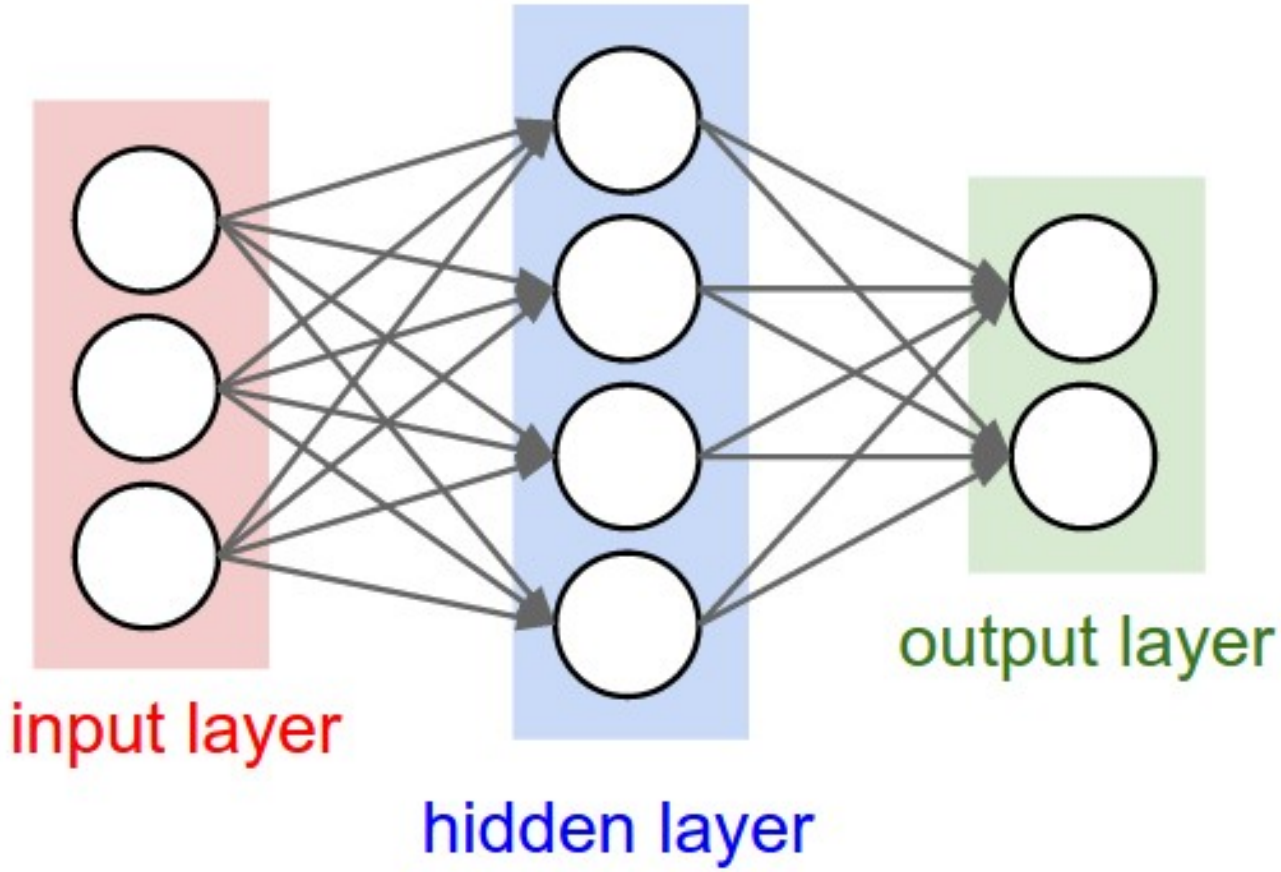
Actions: Discrete (for now) directions and stepping

Position Update: Not completely deterministic



Overview Deep Neural Networks

Neural Networks



(Karpathy 2015a)

Neural Networks

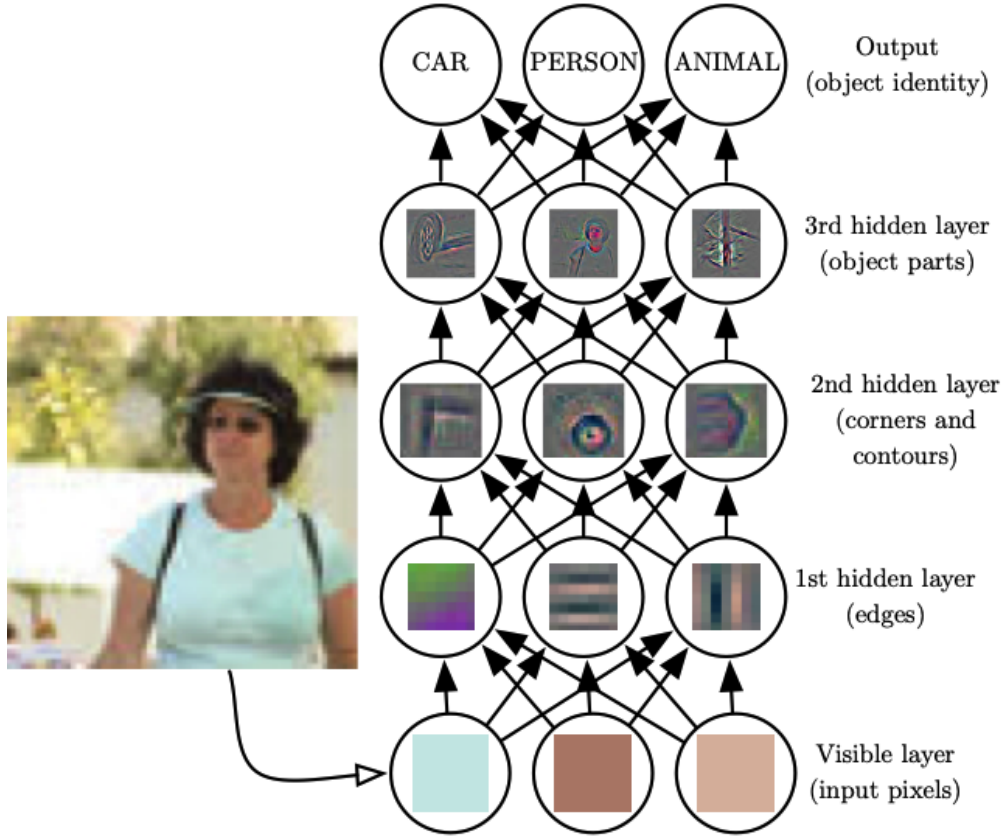
Approach is inspired by how a brain learns. It is characterized by:

- The human brain has approximately 10^{11} neurons.
- Each neuron has around 10^4 to 10^5 connections.
- The switching time of a neuron is around 0.001 seconds.
- Cognitive operations take above 0.1 s (e.g., face recognition). As a consequence: there can be only around 100 computation steps in sequence involved (100-step-rule).
- Massive parallel computation.
- Very robust (computation).

Overview of Deep Learning Architecture

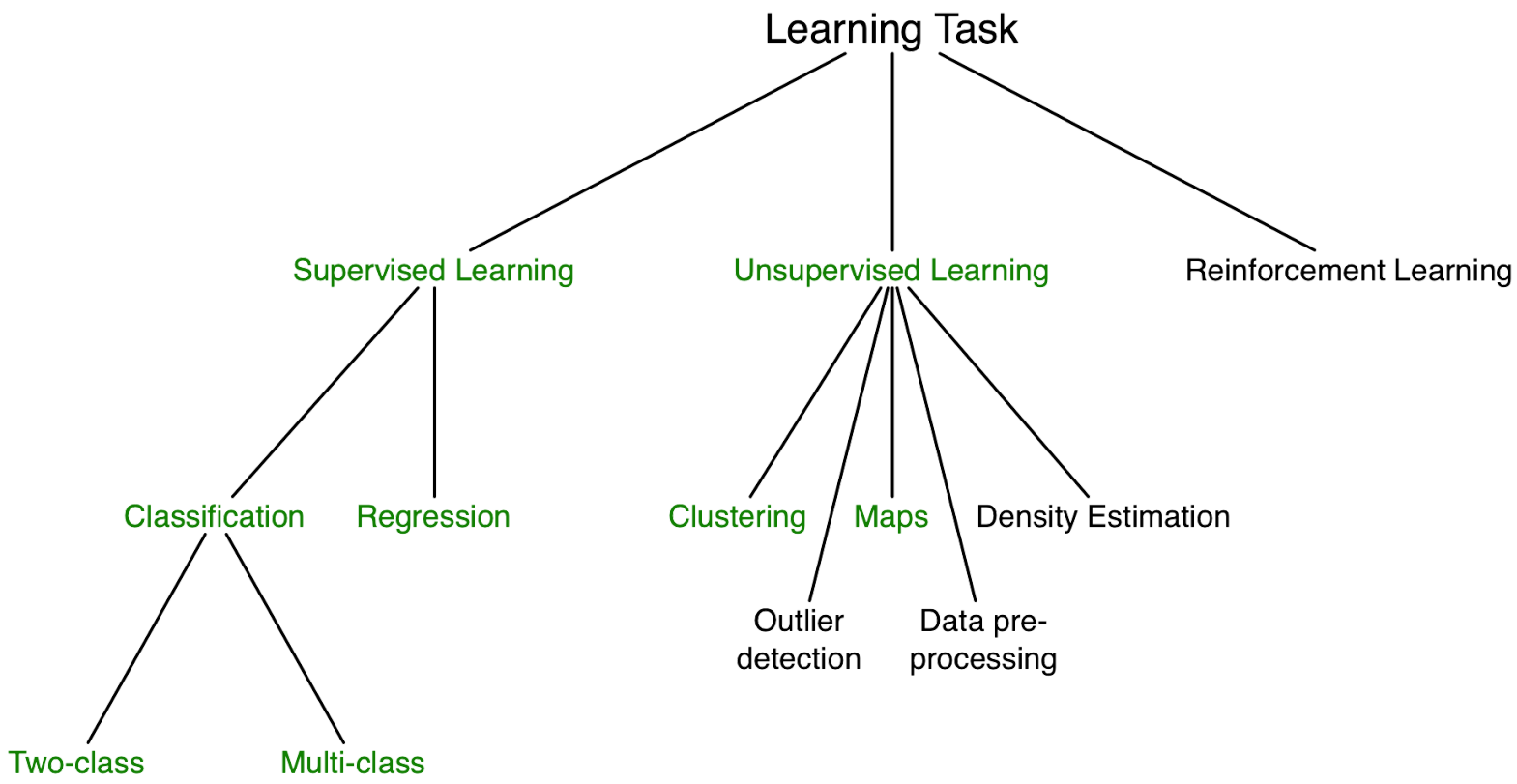
Basic approach of deep learning

- Raw, high dimensional data is processed in stages.
- Stages are realized in different layers of the network.
- Higher layer extract increasingly abstract features from lower levels



(Goodfellow, Bengio, und Courville 2016)

Forms of Learning



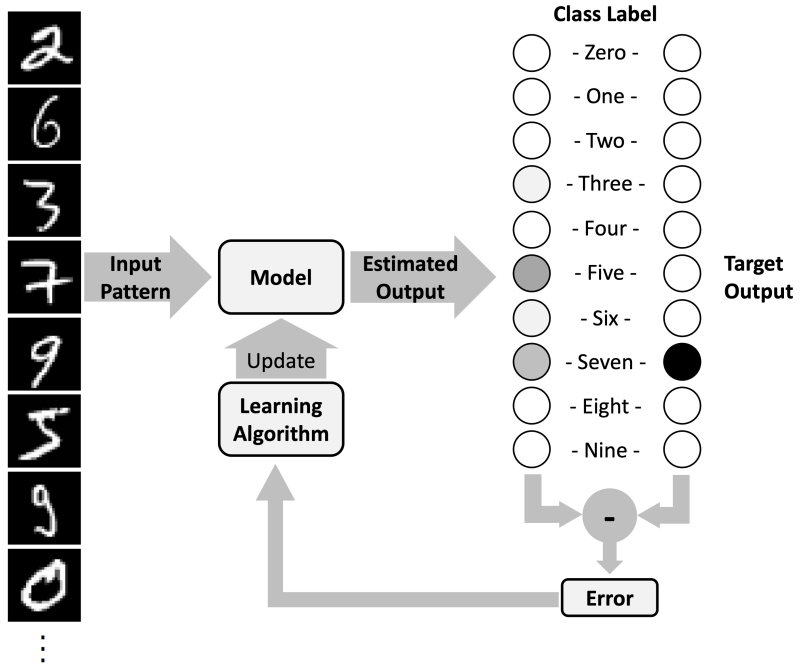
Supervised Learning

Teacher provides examples of a situation and a desired output.

The goal is to learn the functional relation between situations and the desired output.

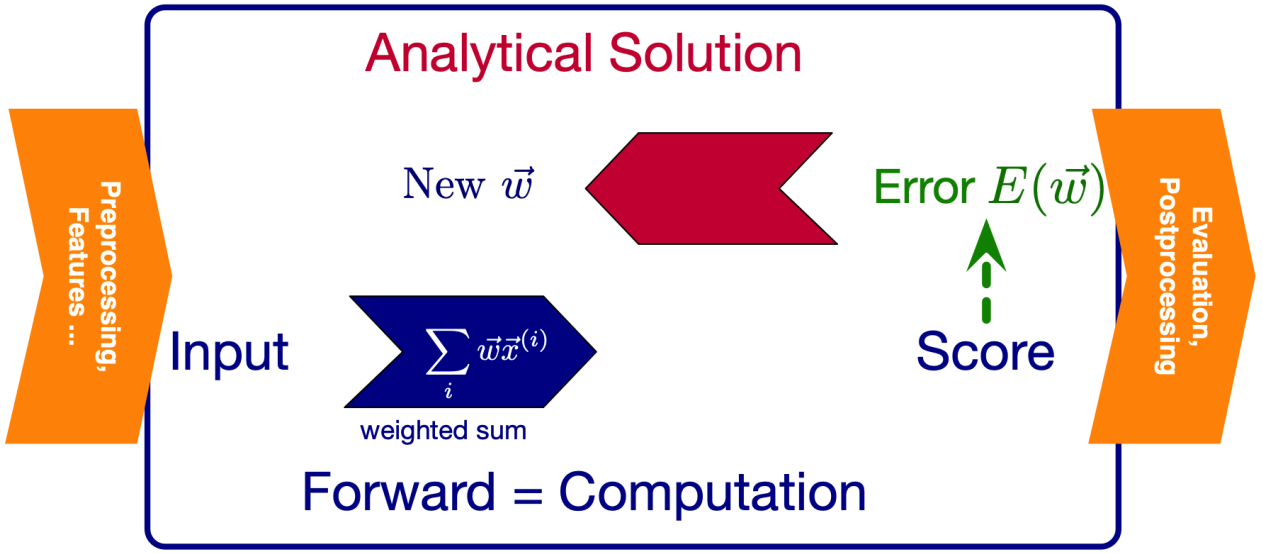
Classification

- desired output is taken from a small set of possibilities
- patterns are partitioned into classes as given by desired output
- example: digit recognition



(Bishop 2006)

Classification as an Example Task

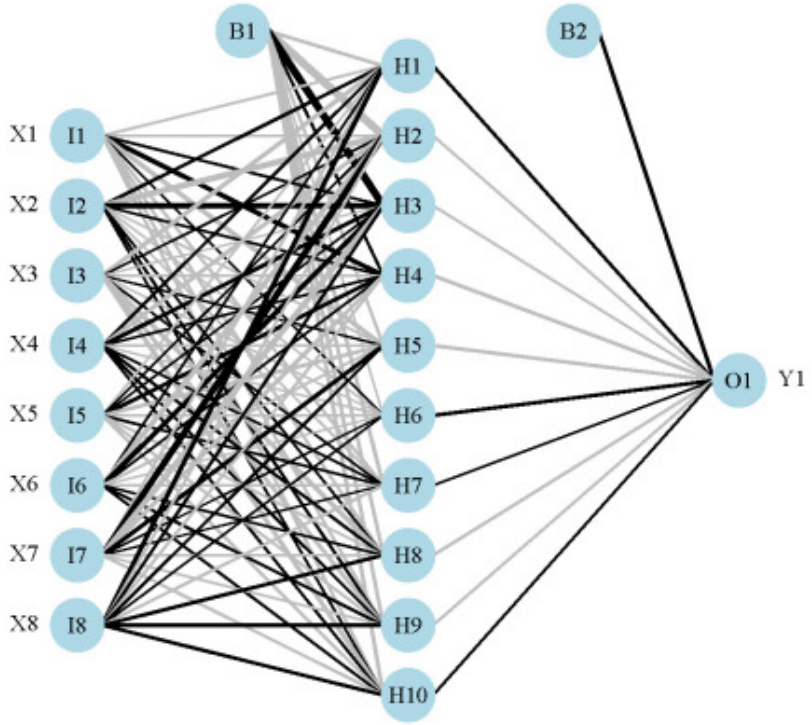


Neural Networks

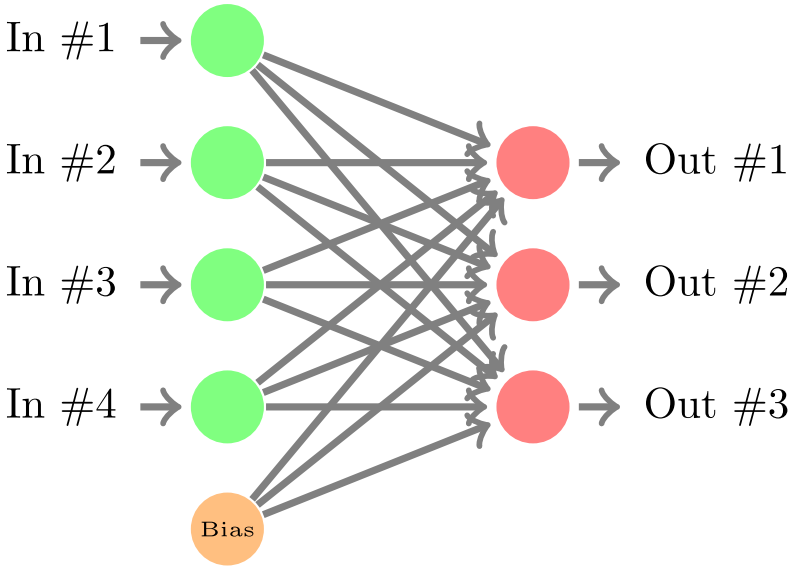
In general, they address the problem of (very) high dimensional input data and project it into lower dimensional spaces.

A neuronal network can therefore be understood as approximating a function in high dimensional spaces.

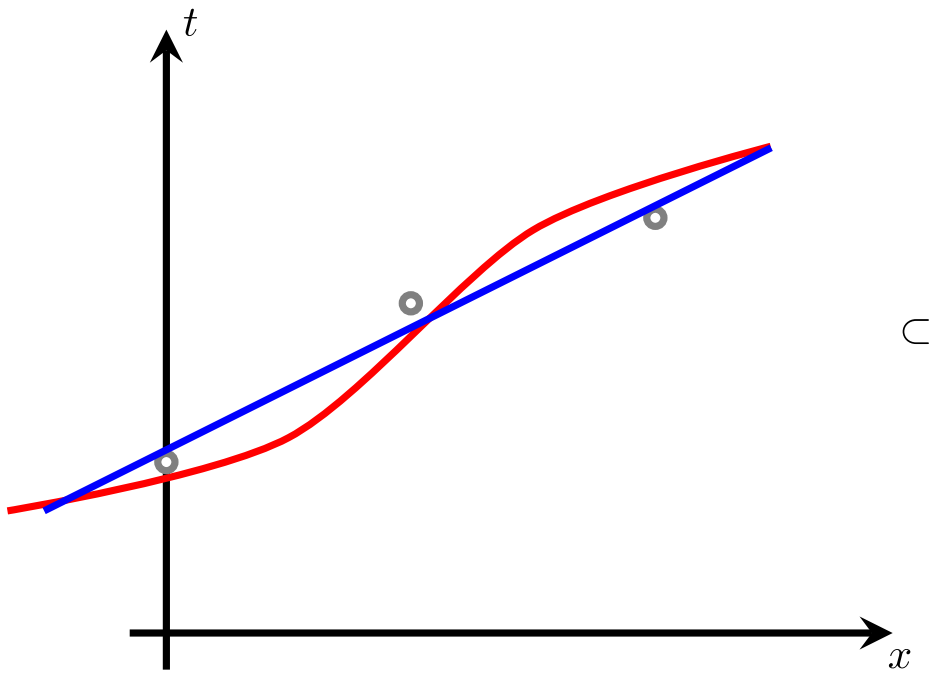
Computation in neural networks relies in such a model on matrix multiplication (multiply input vector with weight matrix describing the connections).



Simple Perspective: NNs as special case of Regression



$\simeq$



$\subset \sum_{i=0}^{M} w_i \Phi_i$

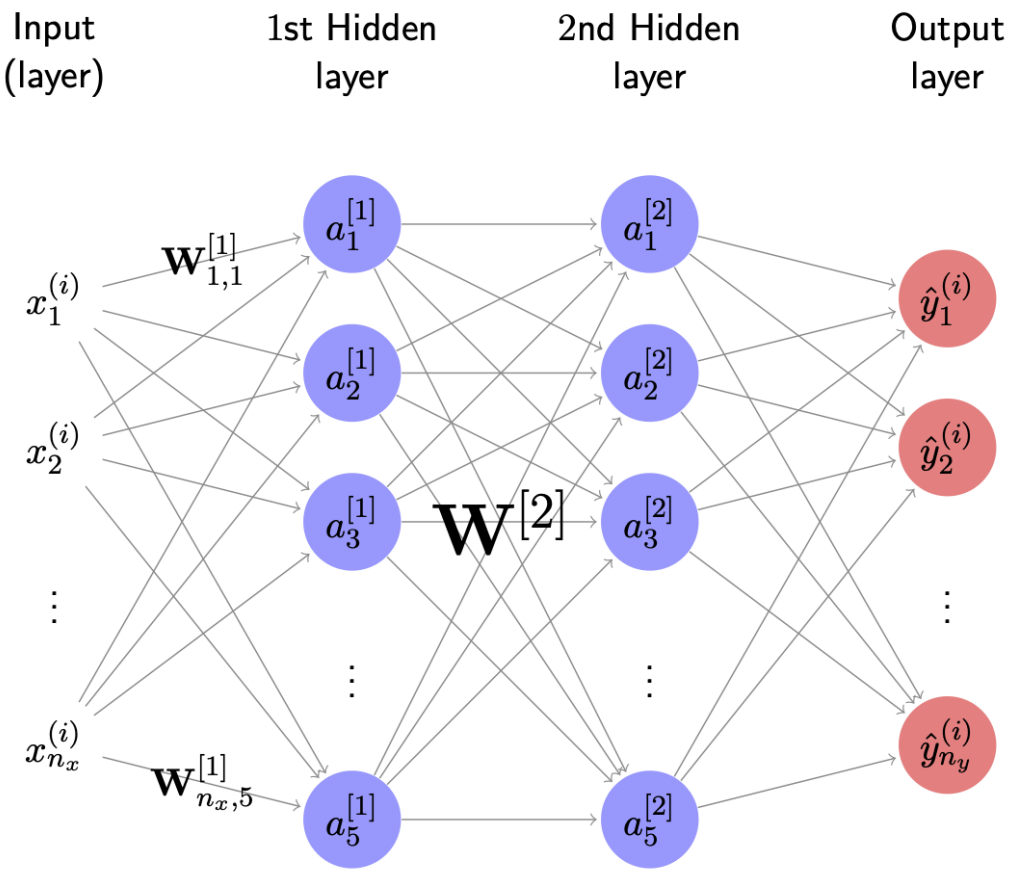
Notation for Neural Network 1

For neural networks the following notations are recommended (as used by Goodfellow (Goodfellow, Bengio, und Courville 2016)).

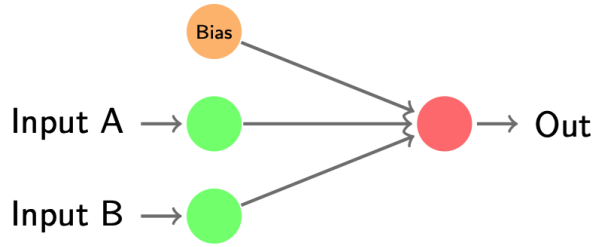
In general, superscript (i) denotes the i^{th} training example while superscript $[l]$ denotes the l^{th} layer.

A *vector* is denoted as $\overrightarrow{x}$ (or $\mathbf{x}$) and a *matrix* is denoted as $\mathbf{A}$.

Notation for Neural Network 2



Overview Learning Cycle



$$a(\vec{x}) = \vec{w}^T \vec{x} = \begin{pmatrix} w_0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} 1 \\ x_A \\ x_B \end{pmatrix} \Leftrightarrow$$

$$a(\vec{x}) = \vec{w}^T \vec{x} + b = \begin{pmatrix} w_1 & w_2 \end{pmatrix} \begin{pmatrix} x_A \\ x_B \end{pmatrix} + b$$ Error $E(\vec{w})$

Sample Input → $\sum_i (\vec{w}^1)^T \vec{x}^{(i)}$ → → *Predicted Output* − *Sample Label*

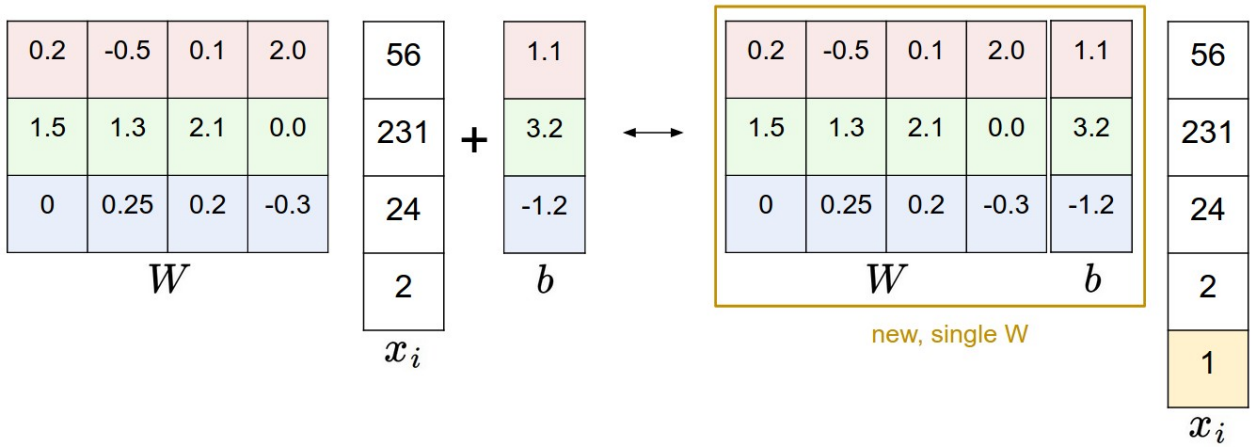
weighted sum | activation fct | **Score**

Forward = Computation

Forward computation for the single layer

For a single layer, we had as the basic equation for a neural network:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$$



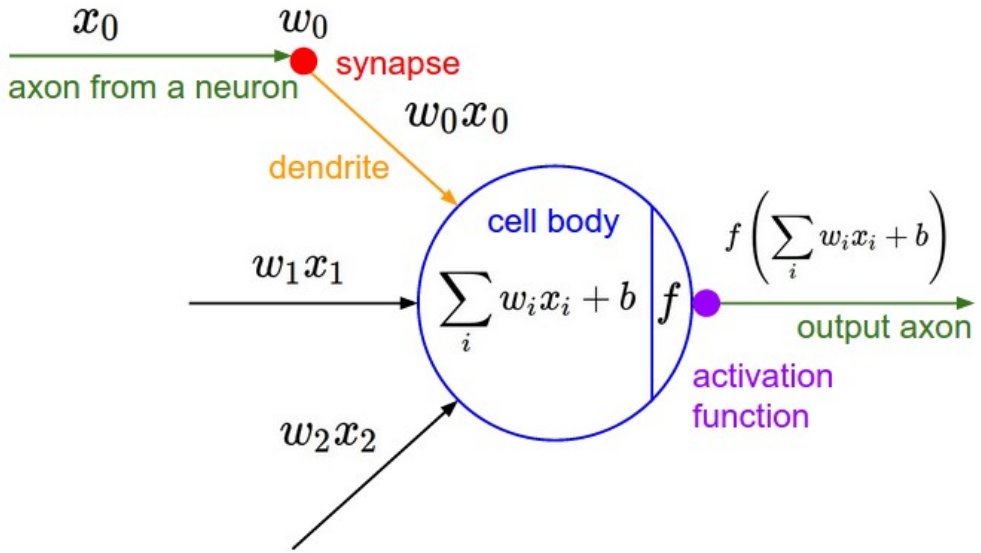
Or, written with the bias augmented as the first constant entry into the input vector:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$$

(Karpathy 2015a)

A Neuron Model

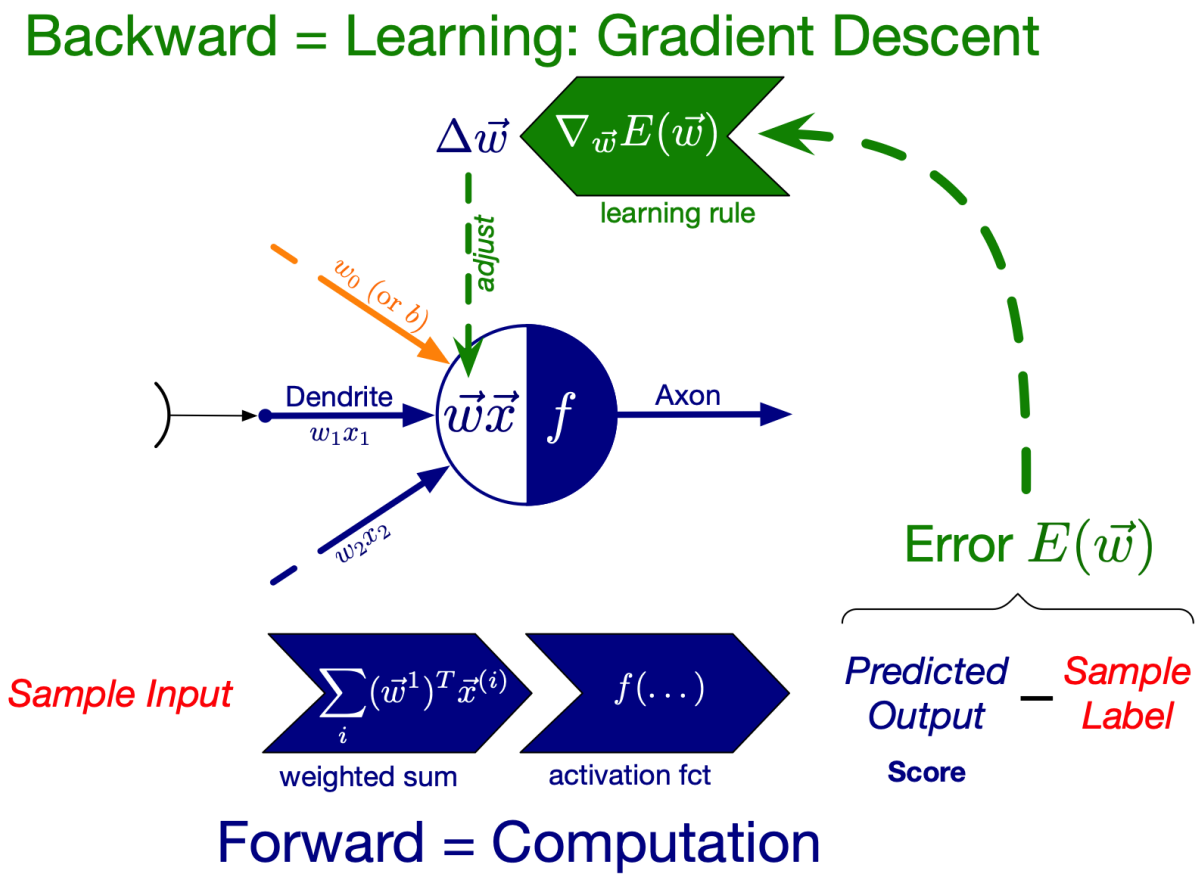
Biological neuron as the main processing unit of nervous systems. It integrates activity from a distributed network of neurons.



Model of a neuron: weighted input from different sources are summed up. An output function is applied on the integrated inputs.

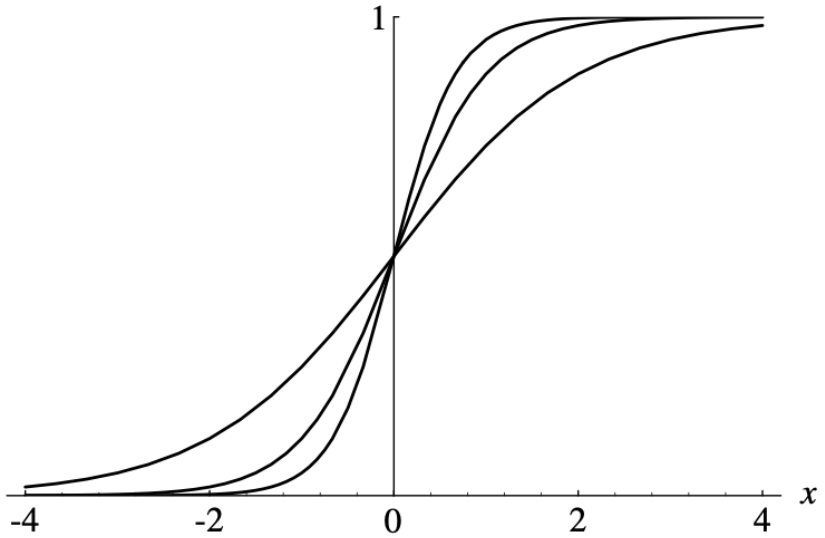
(Karpathy 2015a)

Computation in NN: Overview Learning Cycle



Neuron Model: Activation Function

An output function is applied on the integrated collected activities.



Shown is a sigmoid function for three different constants ($c = 1, 2, 3$).

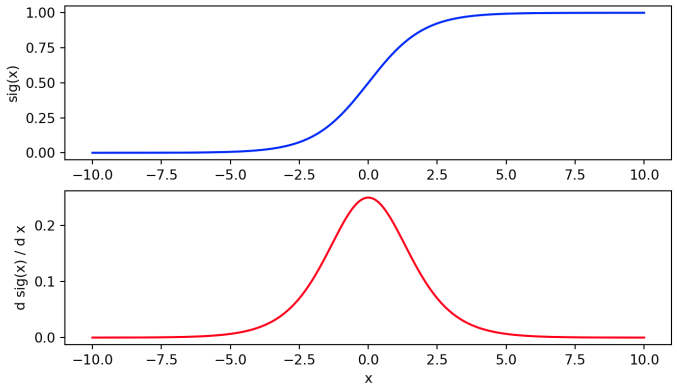
(Rojas 1996)

Activation Function: Sigmoid

Logistic sigmoid activation function:

$$f_{tanh}(a_j) = \tanh(a_j) = \frac{\exp(a_j) - \exp(-a_j)}{\exp(a_j) + \exp(-a_j)}$$

- saturate and squashes numbers to range $[0, 1]$
- but not zero centered
- direct probabilistic interpretation



Drawbacks

- when in saturation: kills gradient
- calculation of exp is expensive

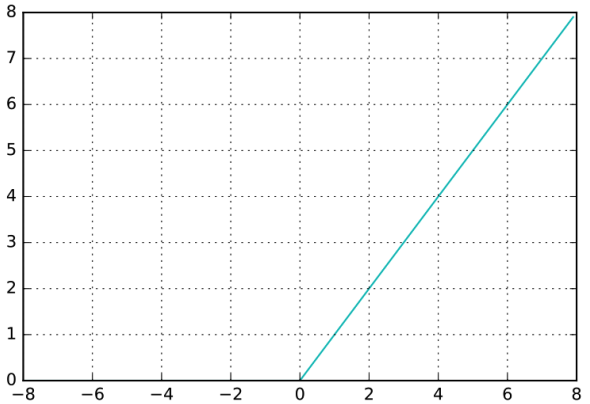
(Rojas 1996)

Activation Function: Rectified-Linear Unit

Rectified Linear (ReLU) activation function:

$$f_{relu}(a_j) = \max(0, a_j)$$

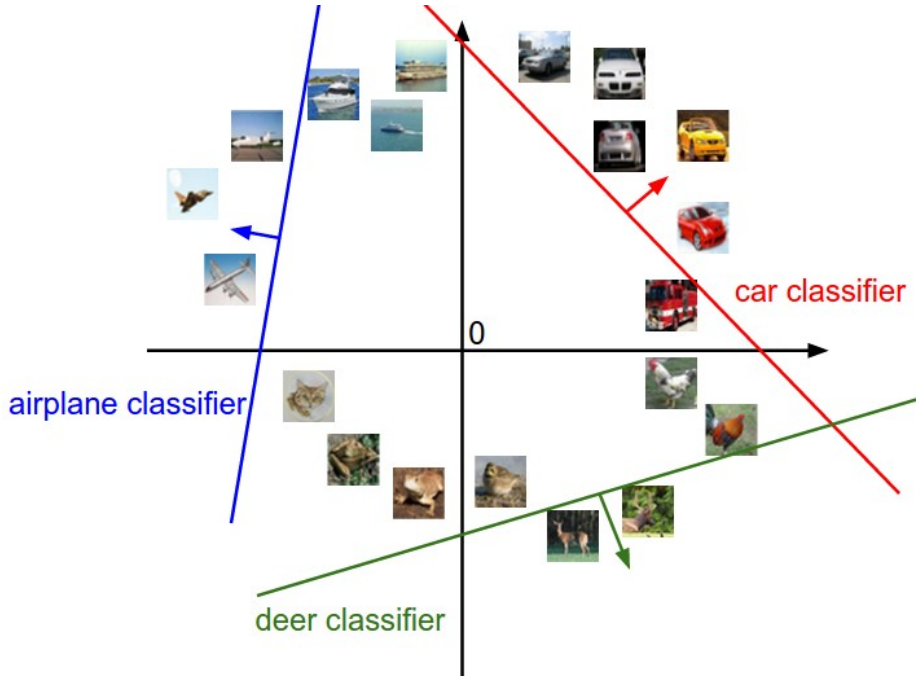
- does not saturate for positive values
- not zero centered
- computationally very efficient
- converges very fast (six times faster than sigmoid in practive)



(Rojas 1996)

Classification – Example Task

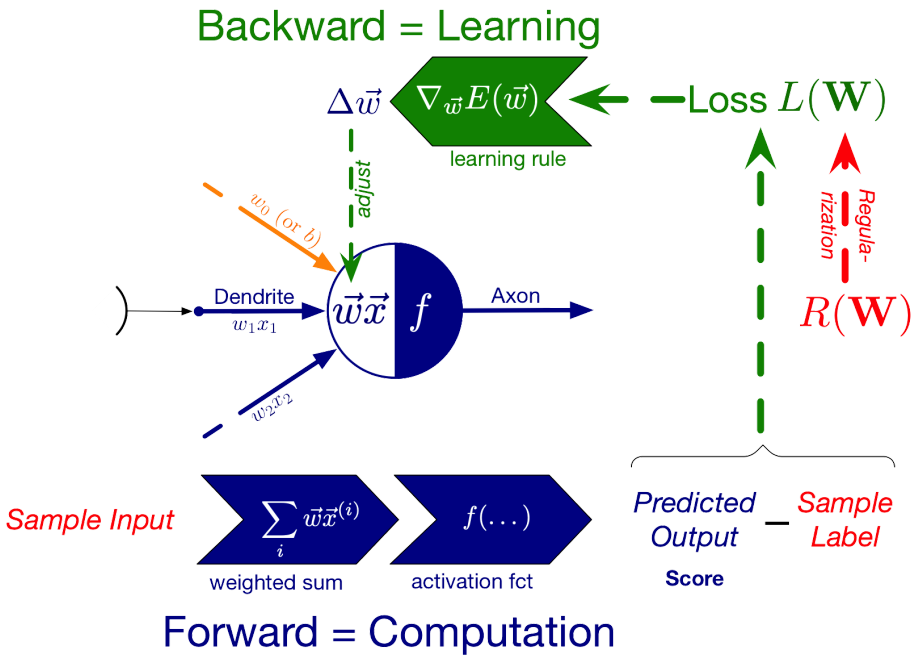
Goal: Linear Classifier



Visualization of the image space: each image is represented as a single point, three classifiers are shown.

(Karpathy 2015a)

Overview: Learning Cycle



Objective Function – Error, Loss, Cost

The function we want to minimize (or maximize) is called *objective* function (or *criterion*).

Gradient Descent

When analytical approach is not possible or feasible: A basic approach to obtain sequential learning rules is to apply *gradient descent*.

If the error function comprises a sum over the data points (cost),

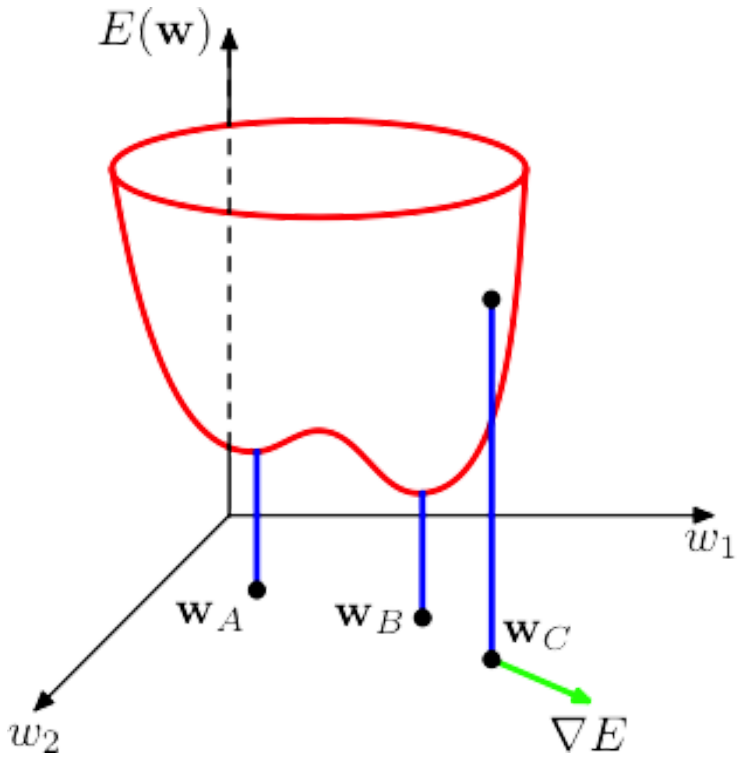
$$E(\mathbf{w}) = \sum_{i=1}^{m} E^{(i)}(\mathbf{w}) = \sum_{i=1}^{m} E(\mathbf{x}^{(i)}, \mathbf{w}),$$

then the gradient descent method

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} E(\mathbf{w}) = -\eta \sum_{i=1}^{m} \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w})$$

iteratively minimizes the overall error $E(\mathbf{w})$, where $\eta > 0$ is the learning rate.

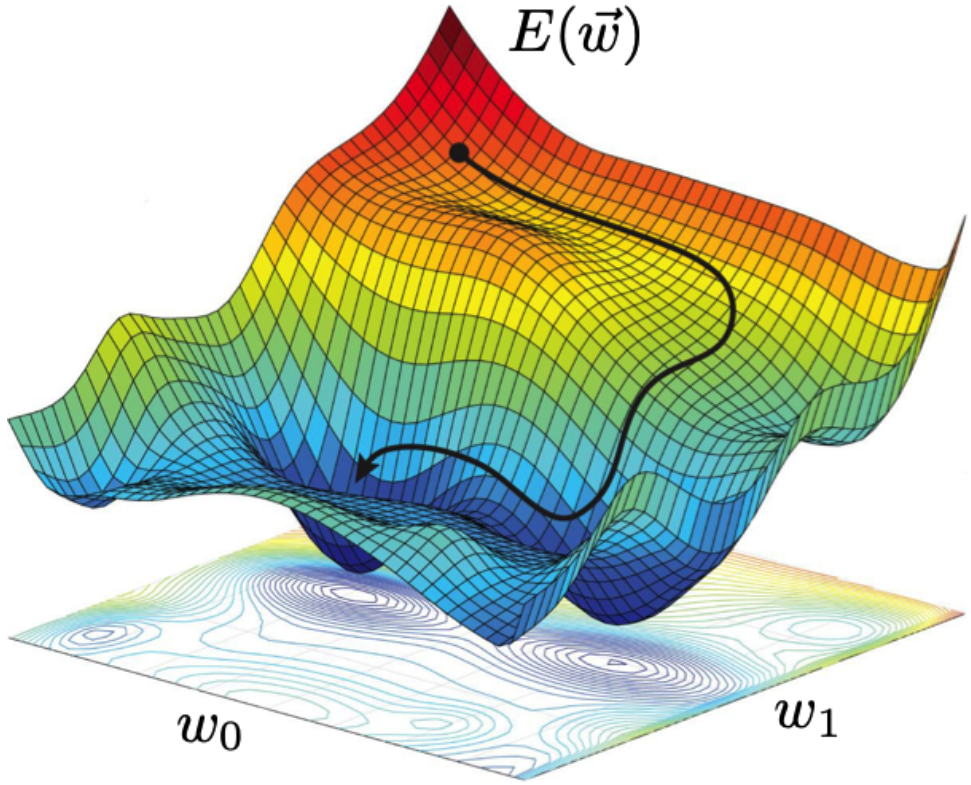
Visualization of the loss function



Gradient descent works on the surface of the loss function and iteratively tries to approach a minimum.

(Bishop 2006)

Gradient Descent: Iterative Search for Minimum



Iterative optimization algorithm

- start from an initial point $\mathbf{u}$ (initial guess) on the error function
- Iterate (k = iteration, η = learning rate):

Determine the gradient at that point and make a step: $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w})$

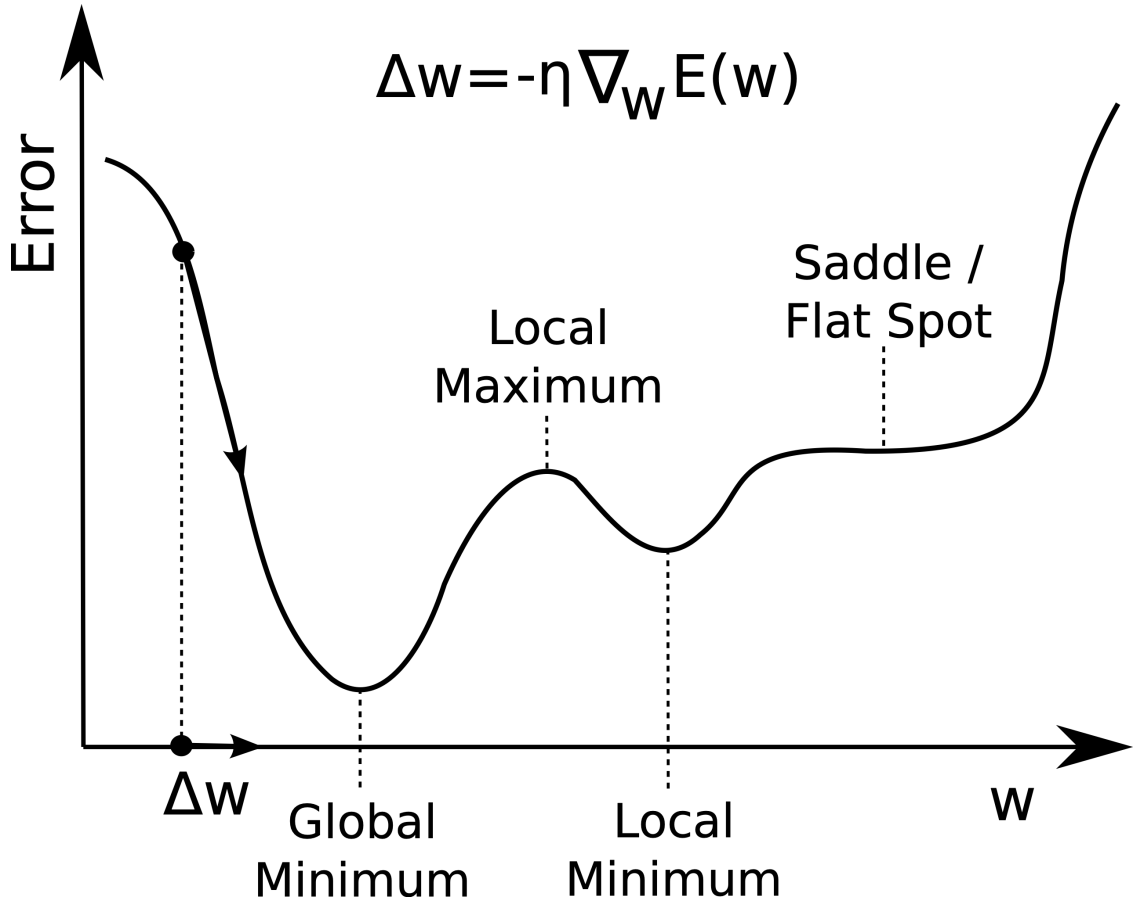
Until: $\nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}) \approx 0$

Then we found a minimum.

Possible Problems for Gradient Descents

- What kind of problems could we encounter in gradient descent?
- What might be solutions?

Gradient Descent: Possible Problems



Possible Problems

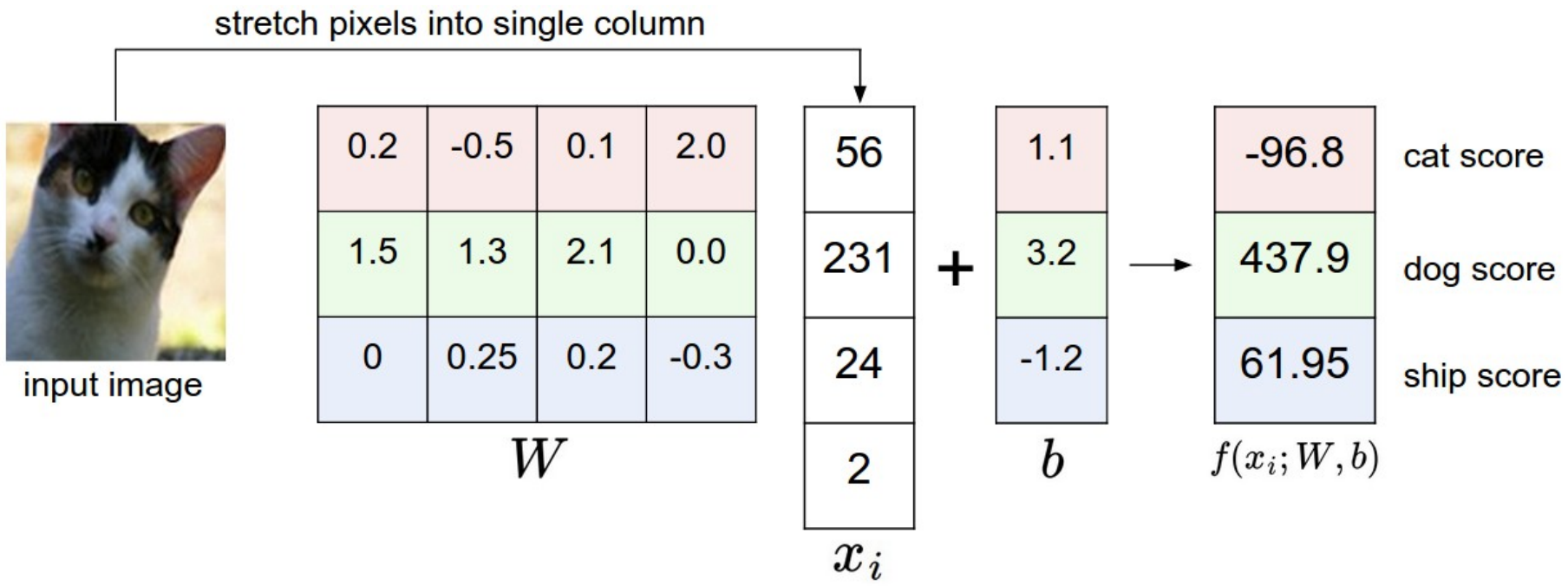
- which minimum is reached depends on the starting point
- too large steps can overshoot the minimum and oscillate
- too small steps can take too long
- flat regions in the error function lead to very small gradients that can not be distinguished from a minimum

Gradient Descent: Computation for the Linear Model

Application to neural networks (here shown without the output function) is straight forward:

$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \\ &= \frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{2} \sum_{i=1}^{m} (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)})^2 \right) \\ &= \sum_{i=1}^{m} (\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}\end{aligned}$$

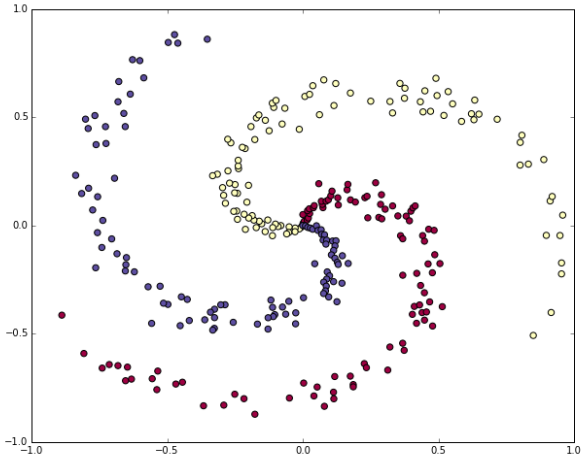
Computation in Linear Classification Example}



Classification of images – shown is the simplified mapping of an image to class scores for a linear classifier. Note that this particular set of weights $\mathbf{W}$ is not good at all: the weights assign our cat image a very low cat score.

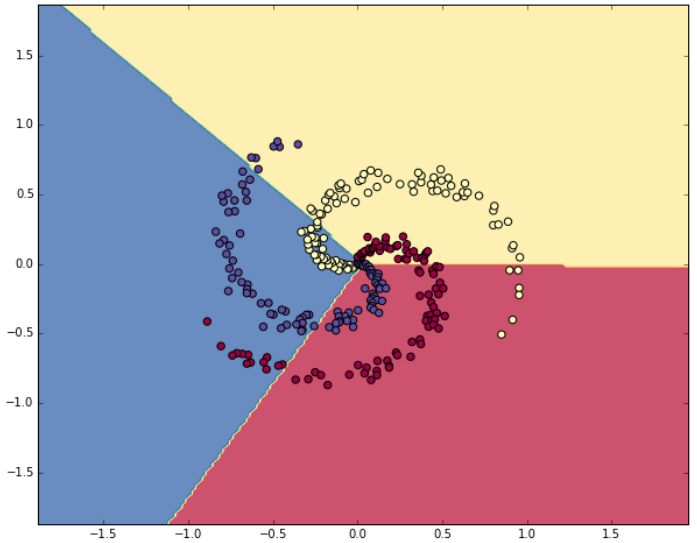
(Karpathy 2015a)

Non-Linear Classification Example: Spiral Data



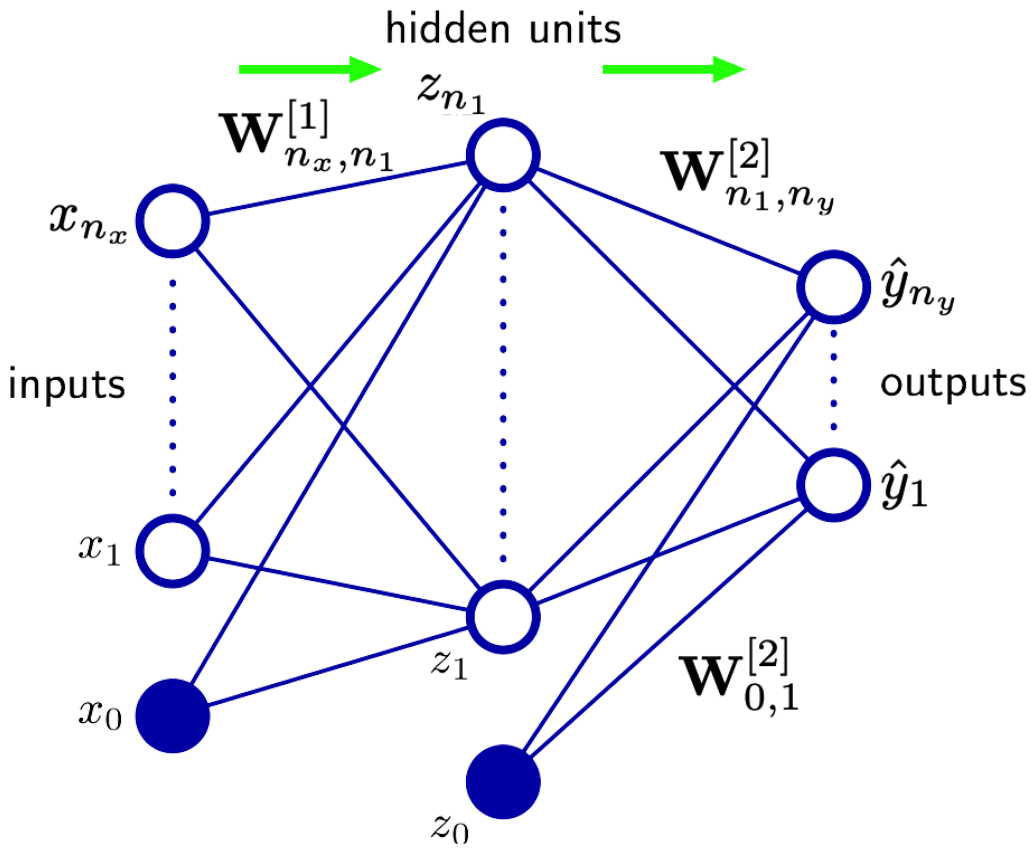
Three classes, constructed from sine and cosine functions.

Advantage: nicely centered and distributed – no preprocessing required.



The spiral dataset is not easily linearly separable.

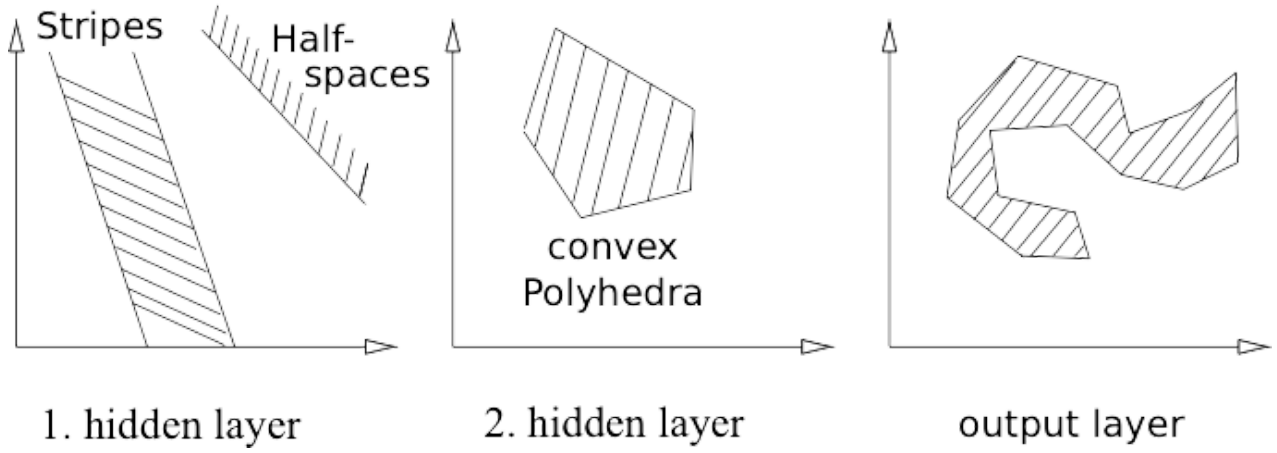
Introducing Hidden Layers: MLP



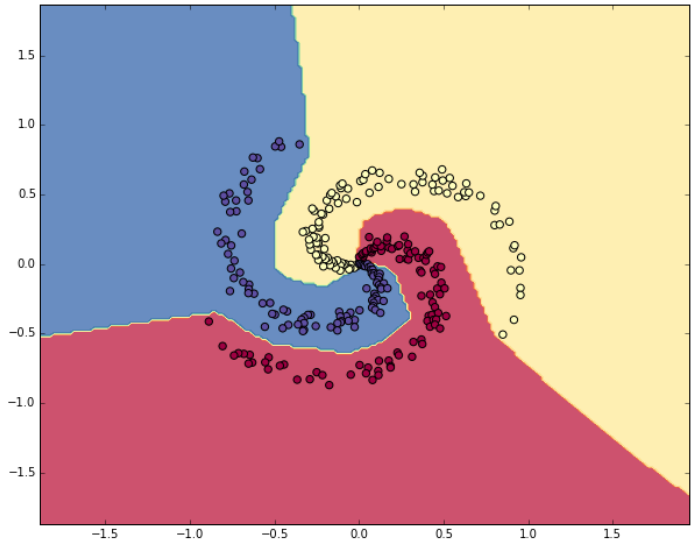
(Bishop 2006)

Geometric Interpretation of MLPs for Classification

- First hidden layer: positive responses in stripes and half-spaces of the input space
- Second hidden layer: the combination of stripes and half-spaces causes positive responses in convex polyhedra of the input space
- Output layer: Arbitrary combinations of the previous regions with high neural activity enables the formation of arbitrary decision boundaries



Non-Linear Classification Example: Spiral Data

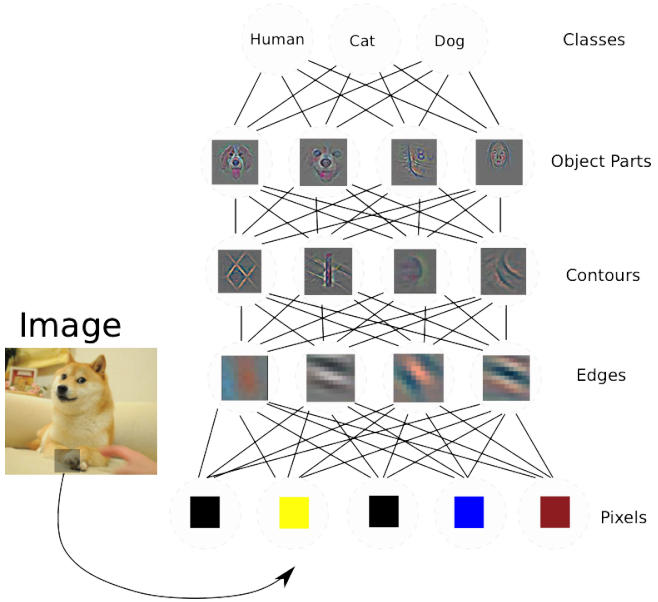


Successful classification (98%) using a neural network after introducing a hidden layer.

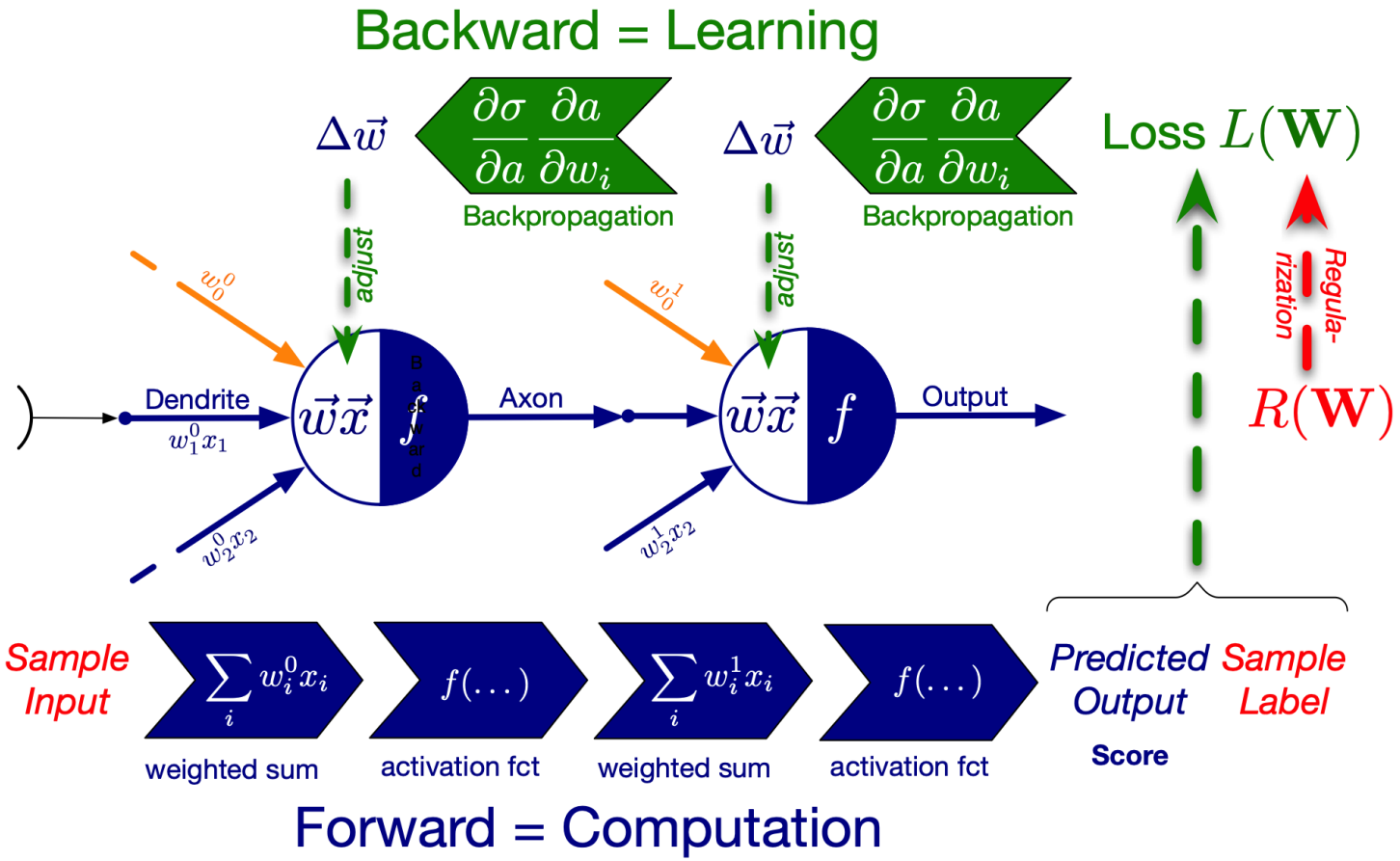
(Karpathy 2015a)

Deep Learning

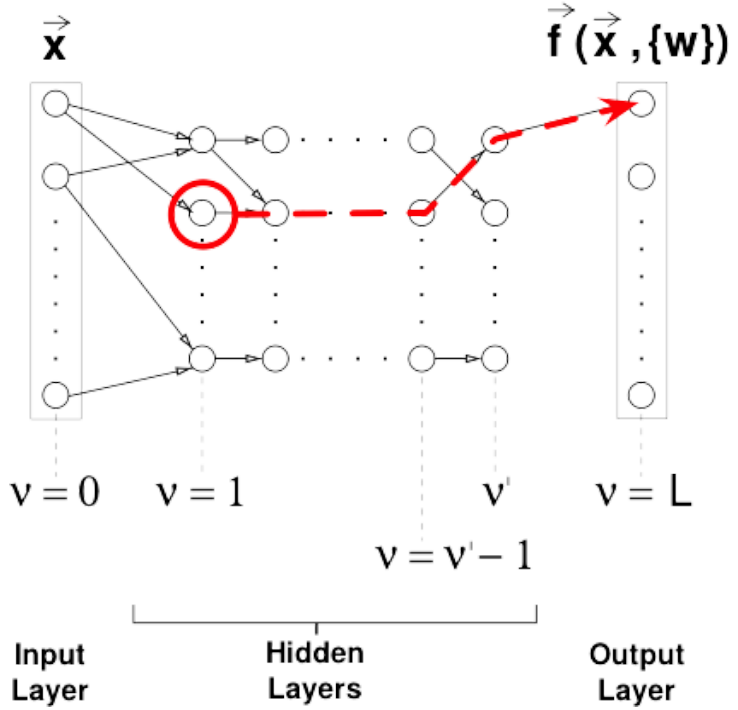
- Learning a hierarchy of representations, from simple to complex
- Quintessential deep learning model: Multilayer Perceptrons



Computation in NN: Overview Learning Cycle



Multilayer Perceptron: Adaptation of the Weights



Credit-assignment problem: how a weight contributes to the loss.

During backpropagation the pushed back error gets more and more diffuse.

Gradient Descent: Computation of the Gradient

Computation of the Gradient requires application of the chain rule:

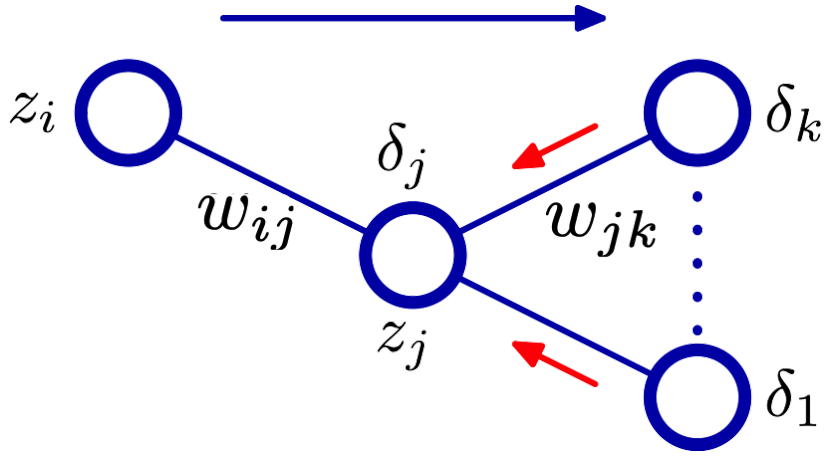
$$\begin{aligned}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \\ &= \frac{\partial}{\partial \mathbf{w}} \left(\tfrac{1}{2} \sum_{i=1}^{m} (f(\mathbf{x}^{(i)}, \mathbf{w}) - y^{(i)})^2 \right) \\ &= \left(\frac{\partial}{\partial w_1} E(\mathbf{w}), \ldots, \frac{\partial}{\partial w_n} E(\mathbf{w}) \right)\end{aligned}$$

$$\frac{\partial}{\partial \mathbf{w}_1} E(\mathbf{w}) = \sum_{i=1}^{m} (f(\mathbf{x}^{(i)}, \mathbf{w}) - y^{(i)}) \frac{\partial}{\partial \mathbf{w}_1} f(\mathbf{x}^{(i)}, \mathbf{w})$$

For complex models the chain rule has to be applied recursively.

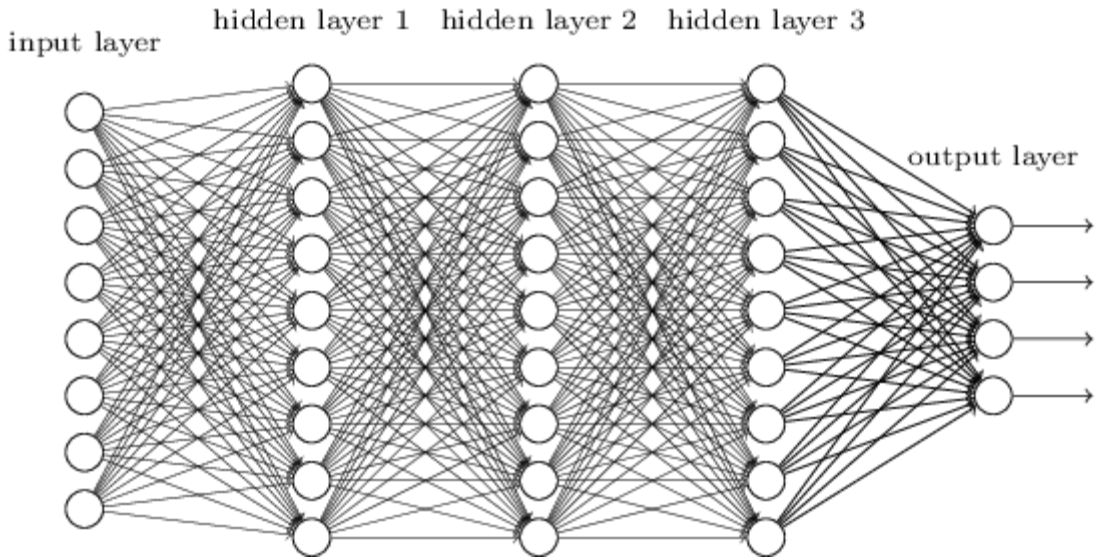
Calculating partial derivatives: For hidden units z_j

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j}$$
$$= \sum_k \delta_k w_{kj} h'(a_j) = h'(a_j)\sum_k \delta_k w_{kj}$$



(Bishop 2006)

Training Deep Neural Networks



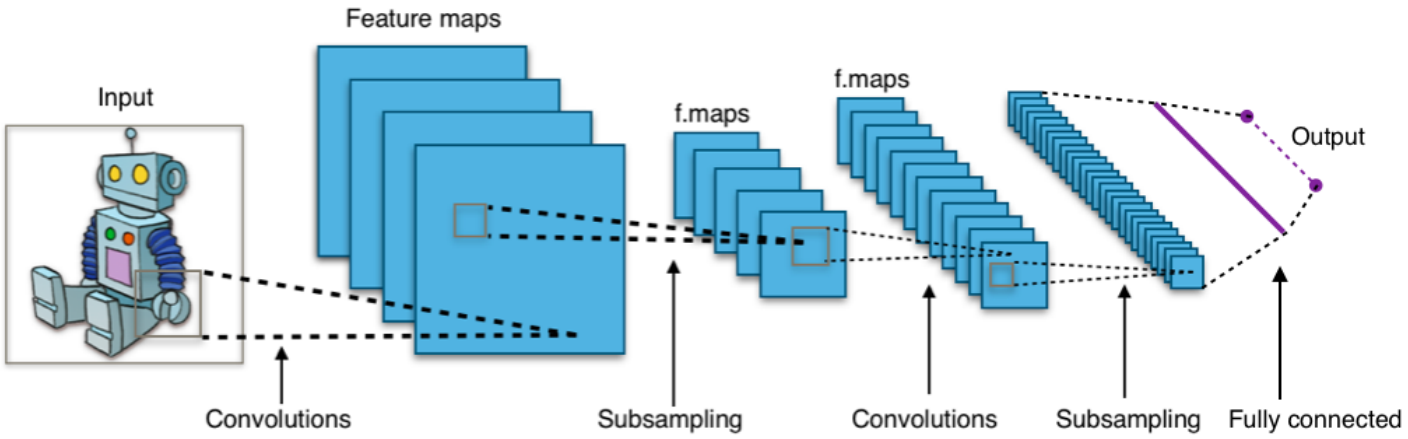
Why is it difficult to learn such Deep Neural Networks consisting of multiple layers?

There is a huge number of parameters in deep networks. Those have to be trained.

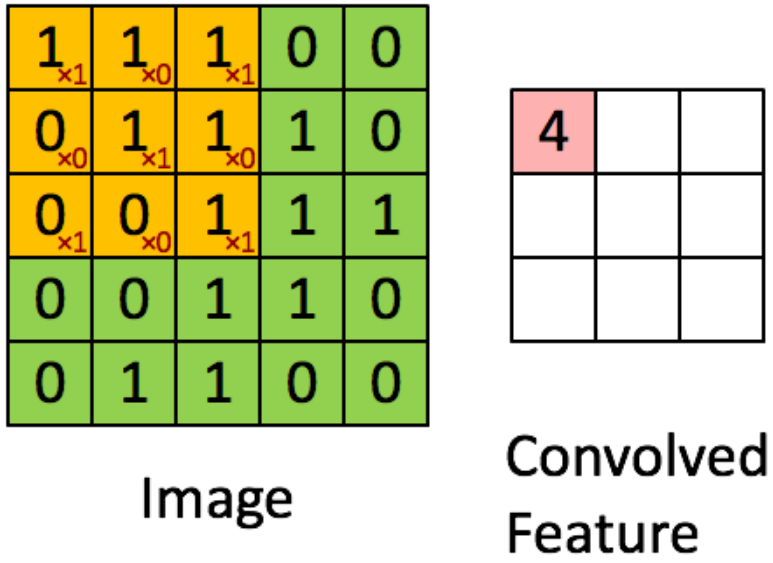
Convolutional Neural Networks

Key ideas of Convolutional Neural Networks

- Convolution
- Non-Linearity (ReLU)
- Pooling/ Sub-Sampling
- Classification – fully connected layer

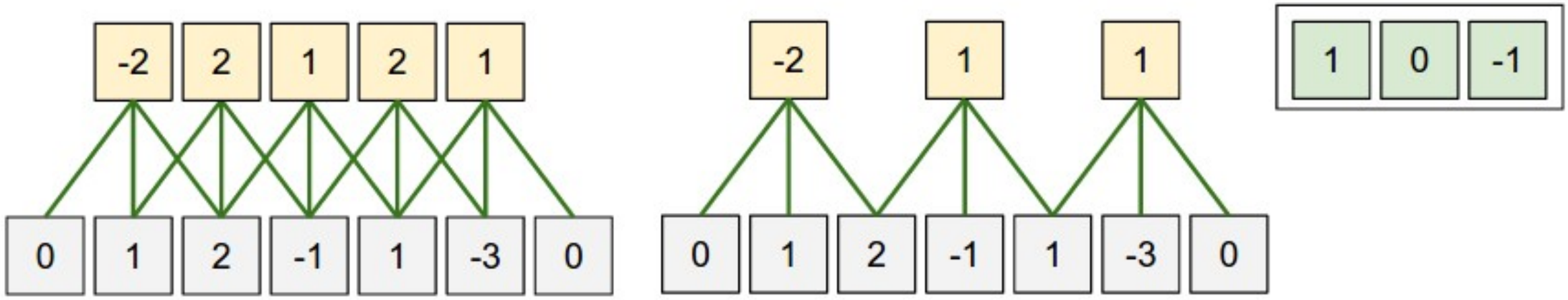


Convolution



Basic Idea: Convolve a filter on an image — filter slides over the image spatially computing the dot product

Convolution: Shared Weights



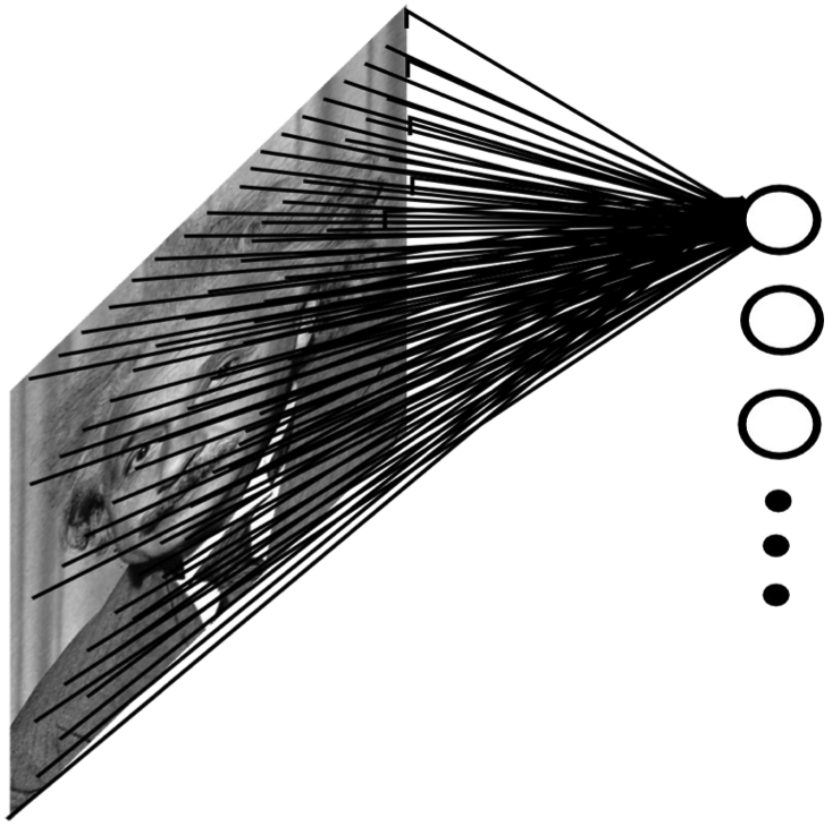
- Uses a sparse representation: only a local neighborhood of input values is integrated — signals have strong local correlations.
- Sharing of features: single parameter set (kernel/ mask) used on all possible locations — signals where features can appear anywhere, are invariant to translations.

(Karpathy 2015a)

Required Parameters: Fully-Connected Case

As an example: a 200×200 image.

For a fully-connected neural network with $400,000$ hidden units: this would require 16 billion parameters.

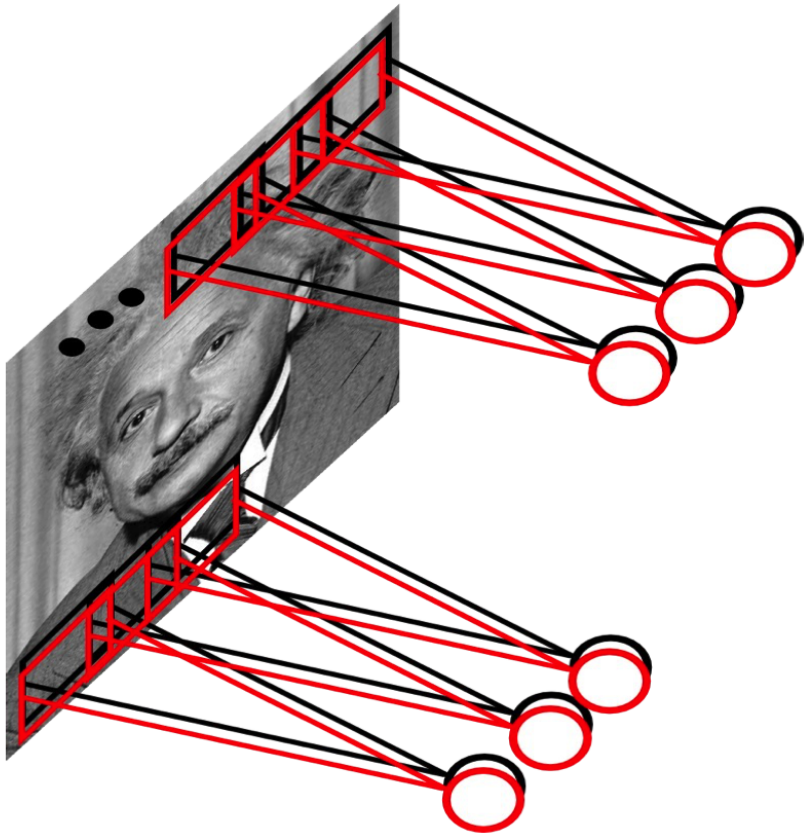


(lecun2013?)

Required Parameters: Convolutional Approach

As an example: a 200×200 image.

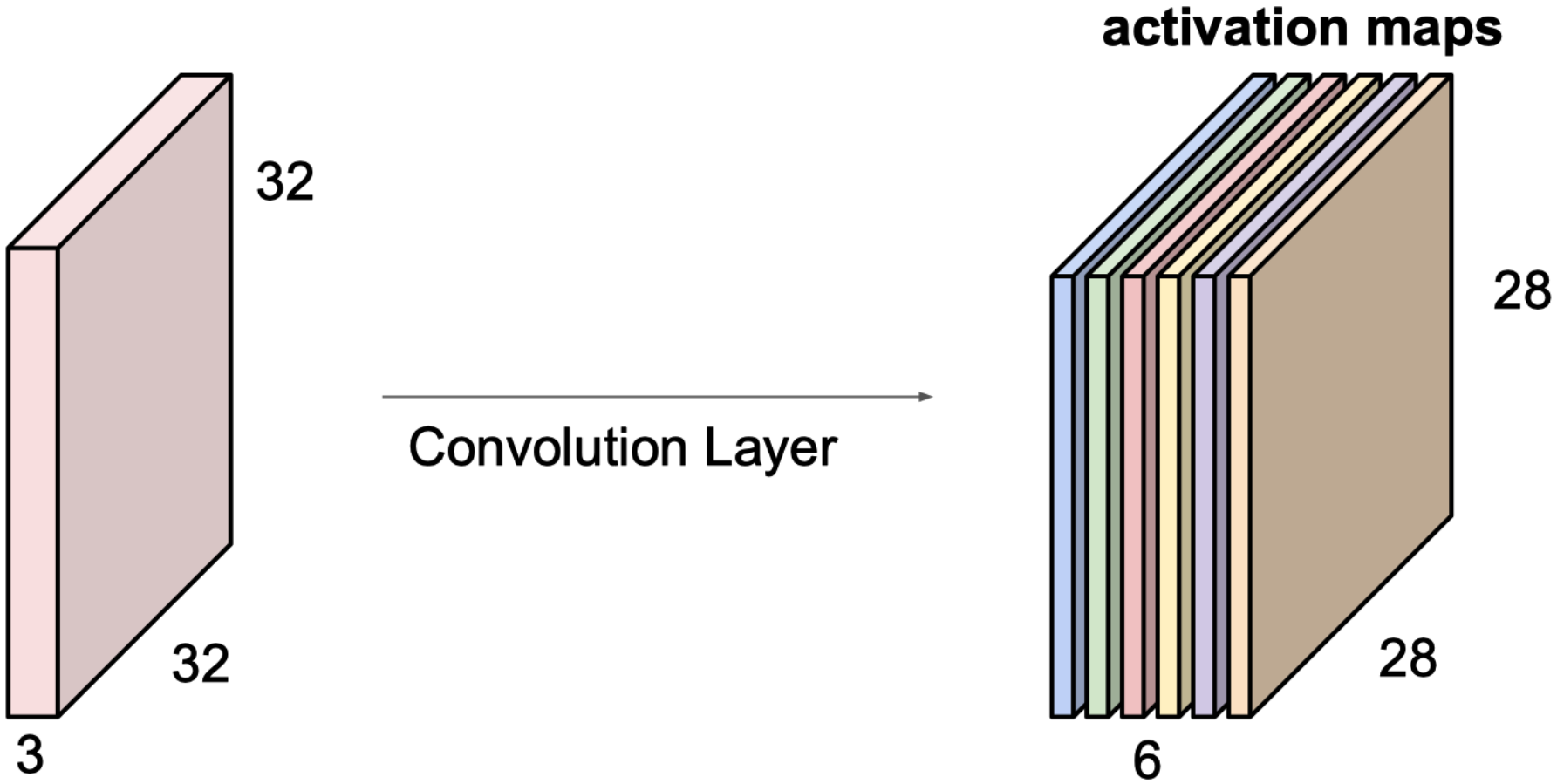
Convolutional: Shared parameters for a filter, using 10 feature maps of size 200×200, 10 filters of size 10×10: only 1000 parameters needed.



(lecun2013?)

Convolutions Illustrated

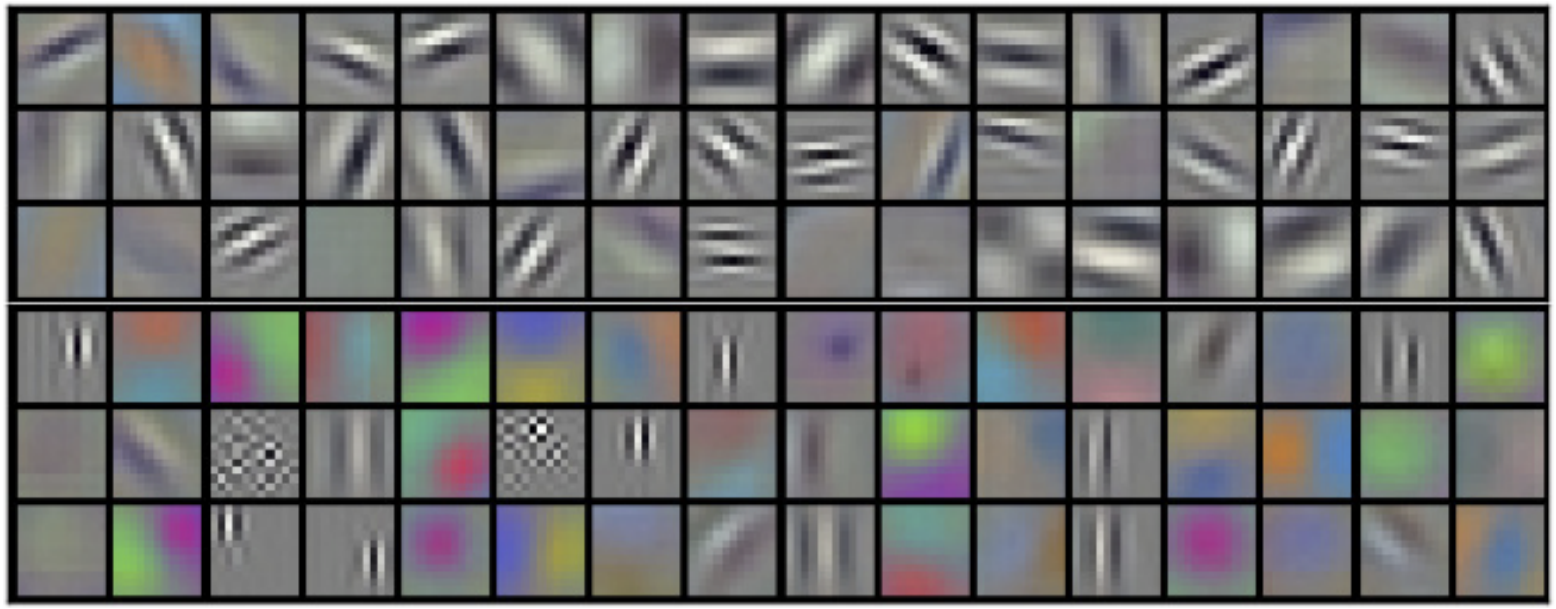
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

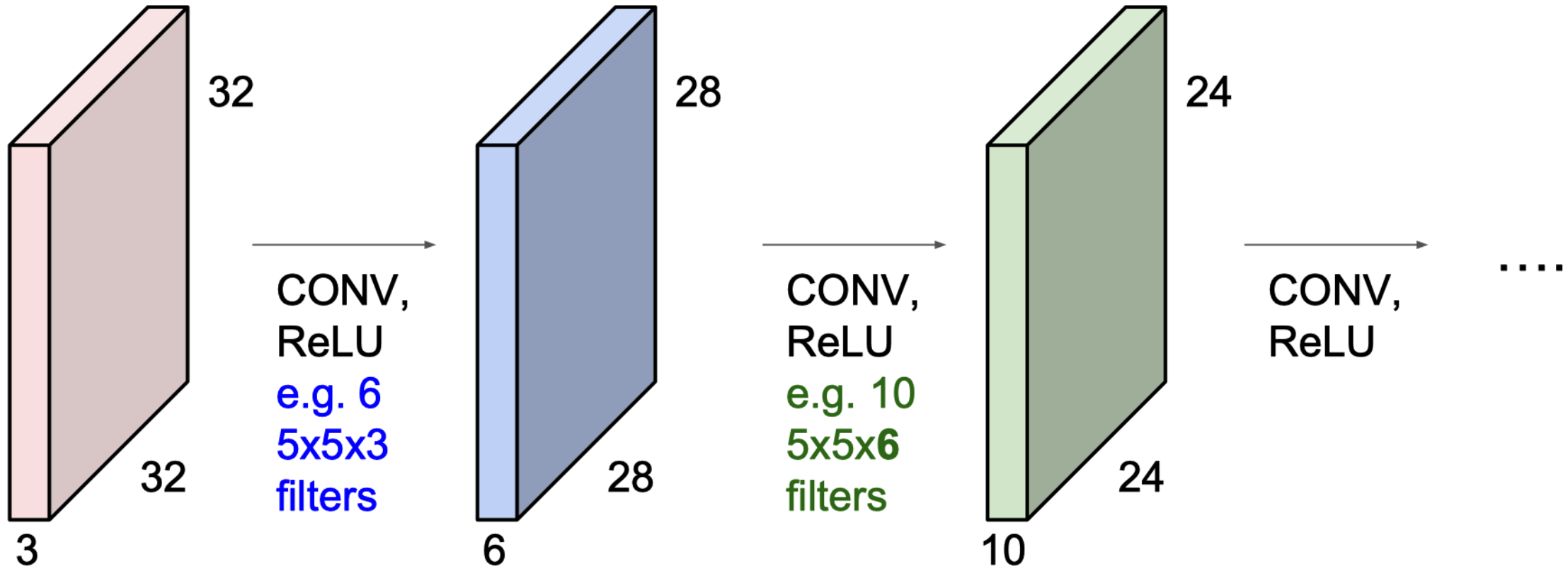
(Karpathy 2015a)

Example: AlexNet – Kernels (first convolutional layer)

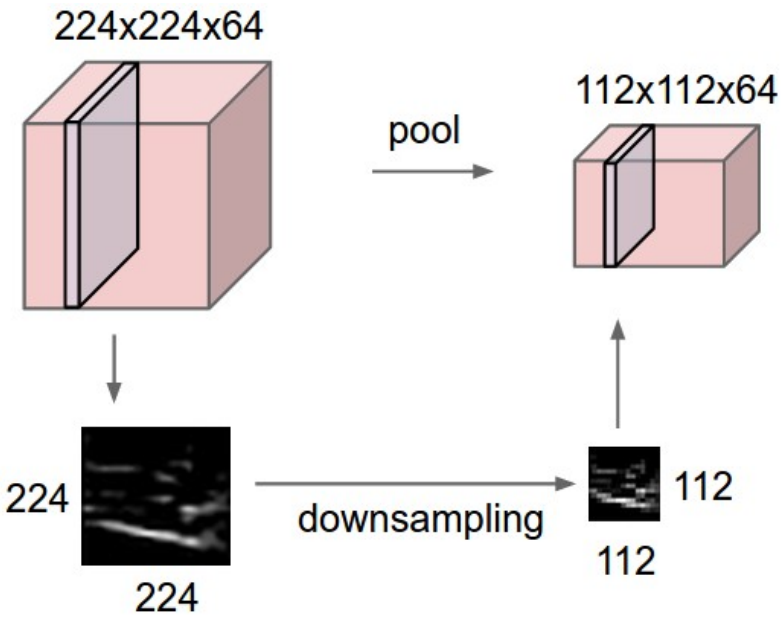


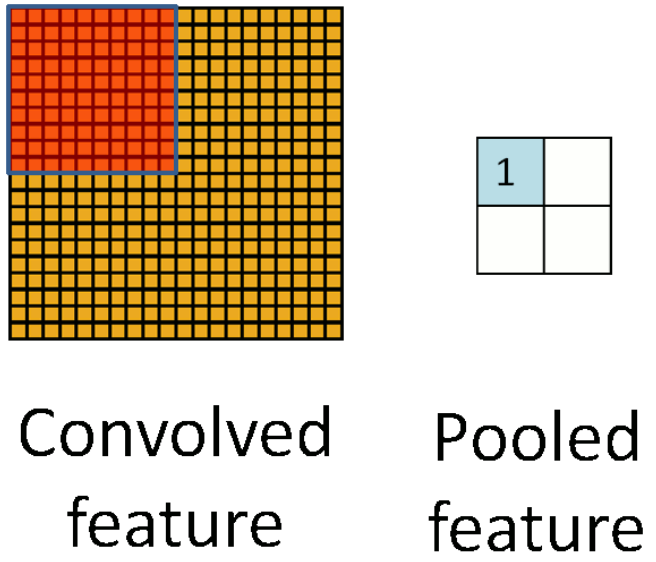
Convolutional kernels learned by AlexNet's first layers. The network learned a variety of frequency and orientation-selective kernels.

(Krizhevsky, Sutskever, und Hinton 2012)


32
32
3
CONV,
ReLU
e.g. 6
5x5x3
filters
28
28
6
CONV,
ReLU
e.g. 10
5x5x6
filters
24
24
10
CONV,
ReLU
....

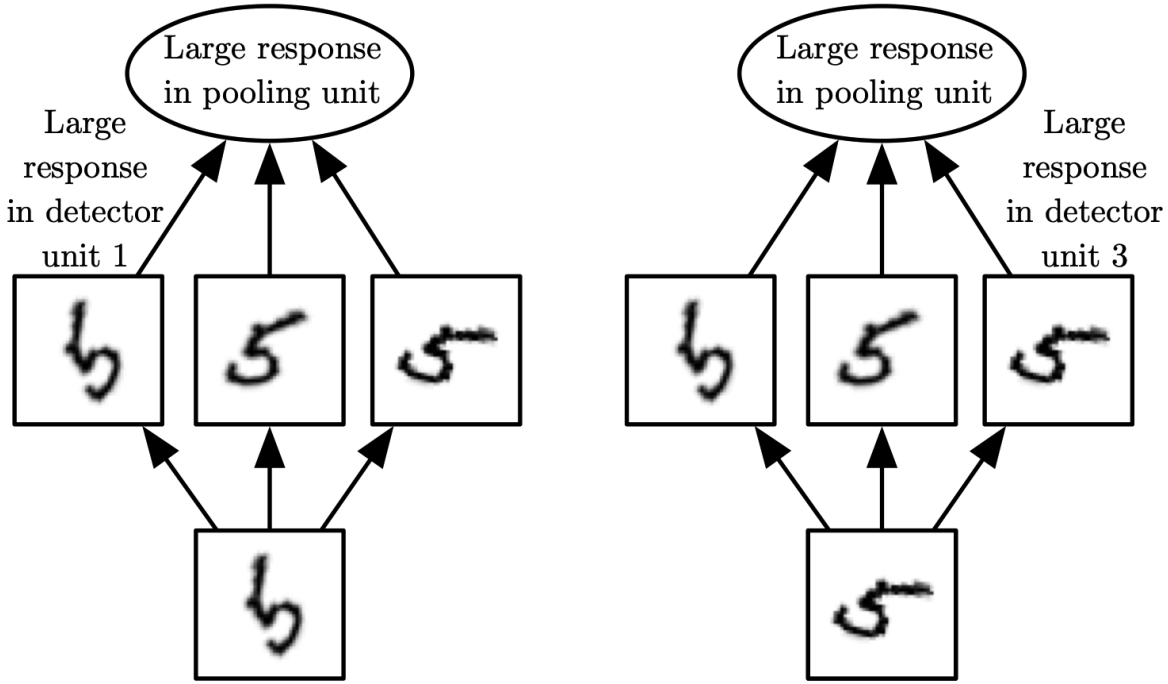
Pooling Layer





- Basic Idea: Progressively downsample spatial size — after convolutional layer.
- Advantage: becomes robust against variation of location (subsequently small translations after combining first convolutional layer).
- Different forms of pooling: max pooling, average pooling, …

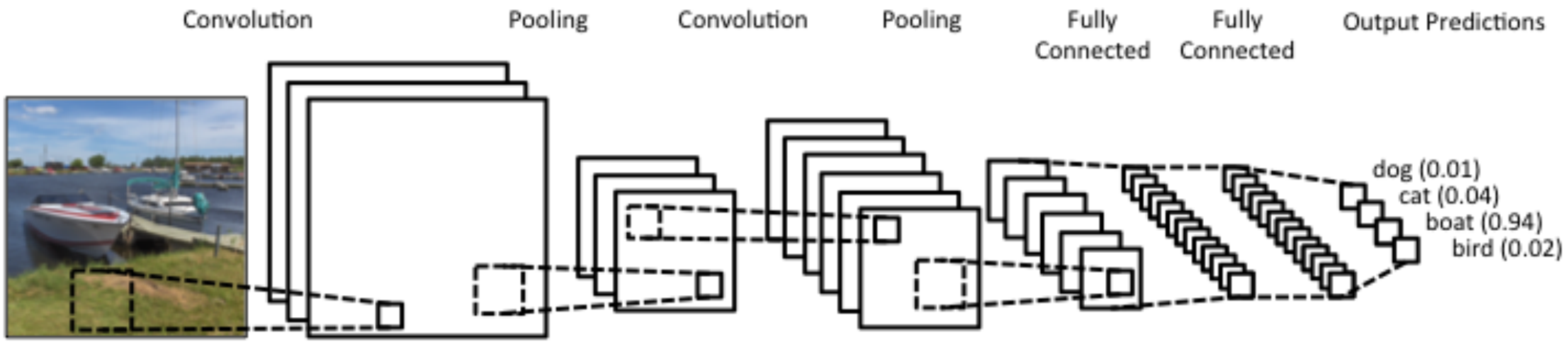
Combining Invariants on next level



Cross-Channel Pooling and Invariance to Learned Transformations.

(Goodfellow, Bengio, und Courville 2016)

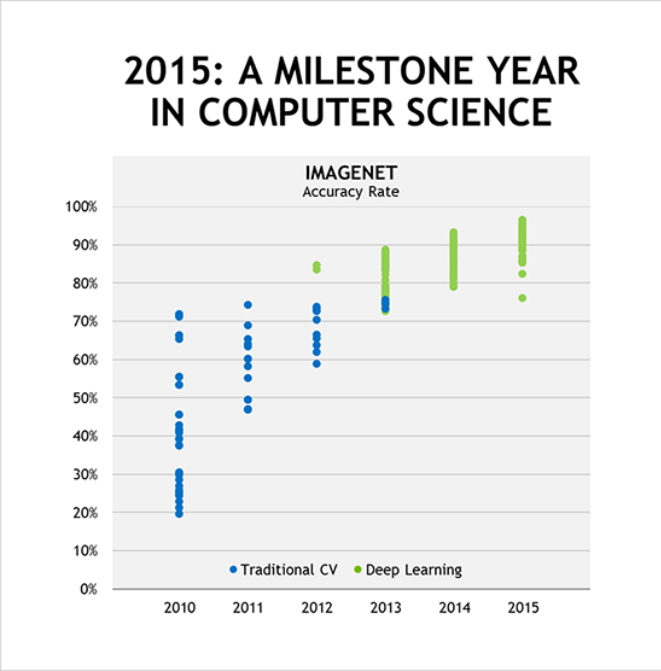
Fully Connected Layer – Classification



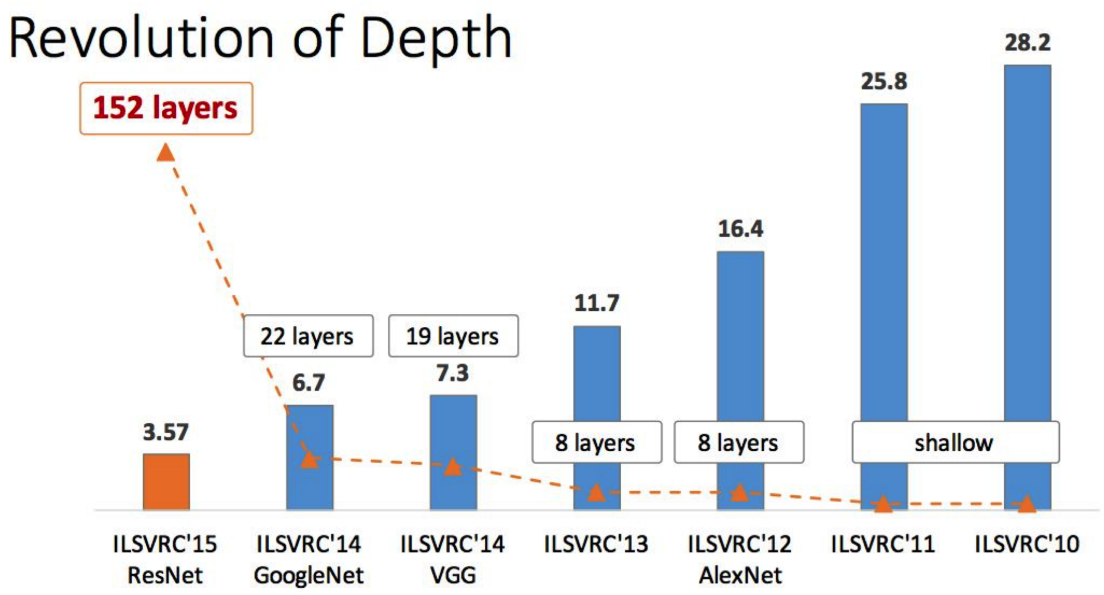
ImageNet Competition Development

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories, collected from the web.

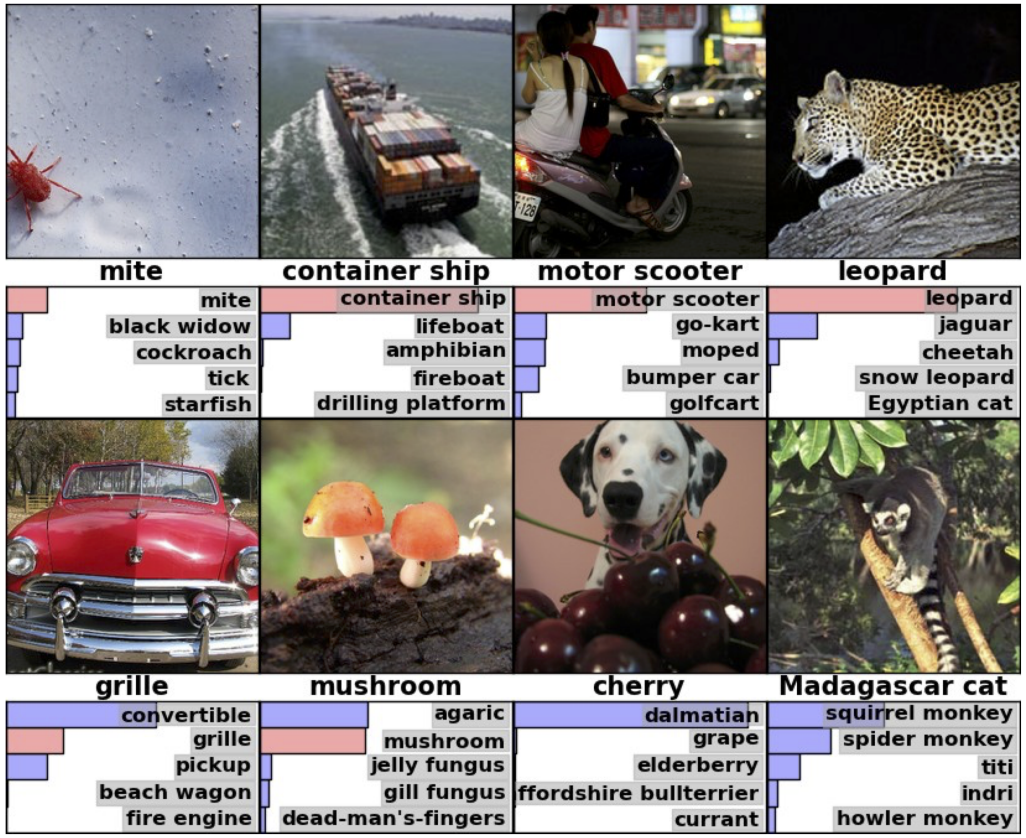
Accuracy over years



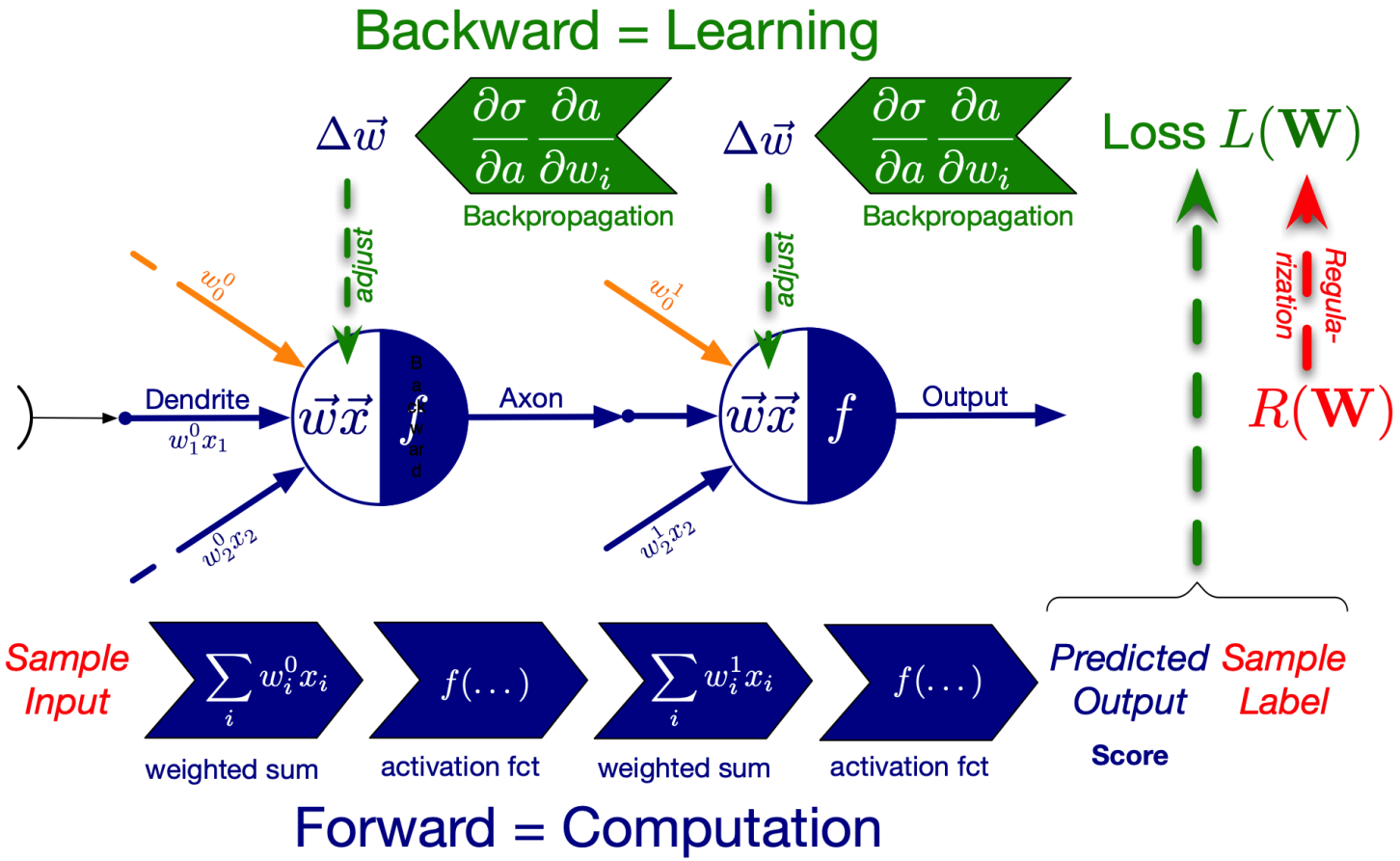
Error and Depth of NNs



AlexNet – Results



Overview Learning Cycle



Deep Reinforcement Learning

Possible Problems for Function Approximation

Goal: apply efficiency and flexibility of TD methods to realistic problems

Problem: Deadly Triad

Approach is …

- off-policy,
- employs non-linear function approximation,
- and uses bootstrapping.

Combined: can become unstable or does not converge!

(Weng 2018)

Deep Q-Networks

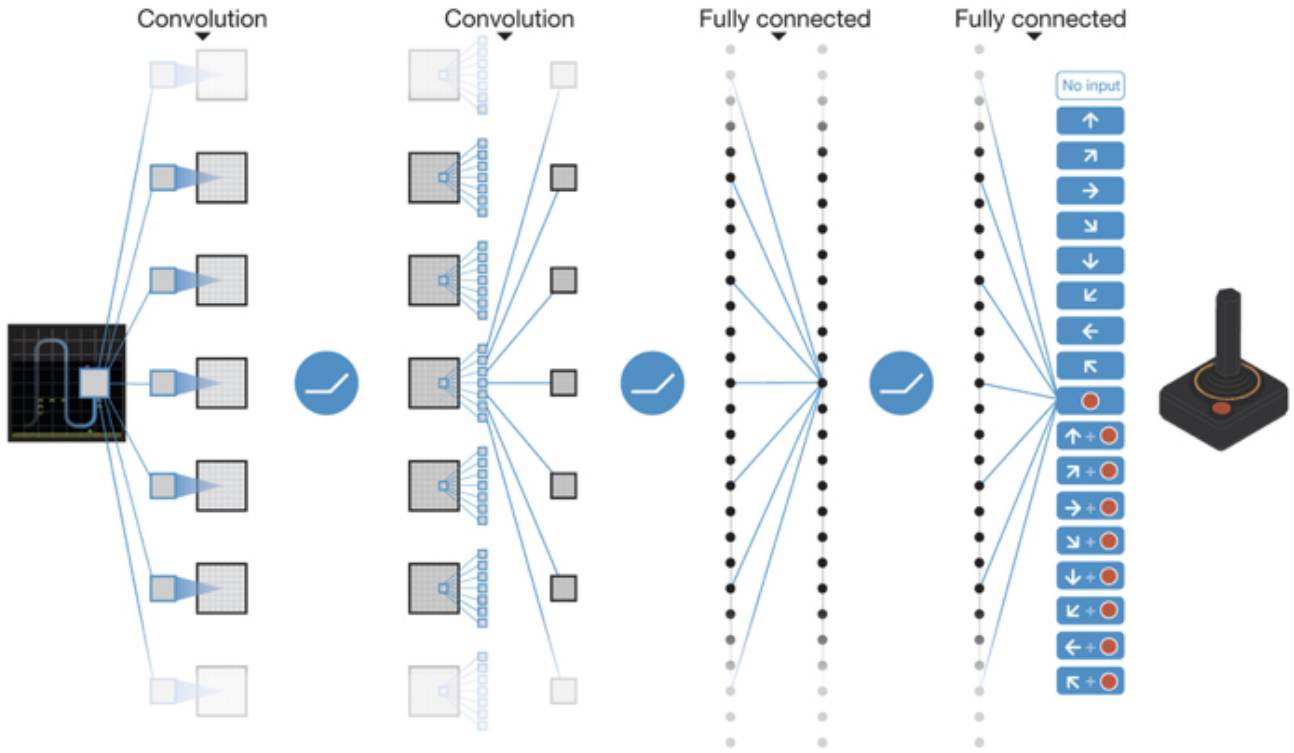
… improved and stabilized training of Q-learning when using a Deep Neural Network for function approximation.

Two innovative mechanisms:

- *Experience Replay:* use a replay buffer for storing experiences.
- Periodically Update *Target network* that are employed for bootstrapping.

(Weng 2018)

DQN Architecture Overview



“we developed a novel agent, a deep Q-network (DQN), which is able to combine reinforcement learning with a class of artificial neural network known as deep neural networks.”

(Mnih u. a. 2015)

Goal of DQN: Approximation of Q-Function

- Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP).
- It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter.
- One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment.
- Q-learning learns estimates of the optimal Q-values of an MDP, which means that behavior can be dictated by taking actions greedily with respect to the learned Q-values.

(Mnih u. a. 2015)

Problems for RL and Deep Neural Networks

Reinforcement learning is known to be unstable when a nonlinear function approximator such as a neural network is used to represent the action-value function.

This instability has several causes:

- the correlations present in the sequence of observations,
- the fact that small updates to Q may significantly change the policy and therefore change the data distribution,
- and the correlations between the action-values and the target values

(Mnih u. a. 2015)

References

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Goodfellow, Ian, Yoshua Bengio, und Aaron Courville. 2016. *Deep Learning*. MIT Press.

Hasselt, Hado van, und Diana Borsa. 2021. „Reinforcement Learning Lecture Series 2021“. https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021.

Karpathy, Andrej. 2015a. „Convolutional Neural Networks for Visual Recognition“. Course CS231, Stanford University, Lecture Notes.

———. 2015b. „REINFORCEjs“. https://github.com/karpathy/reinforcejs.

Klein, Dan, und Pieter Abbeel. 2014. „UC Berkeley CS188 Intro to AI“. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

Krizhevsky, Alex, Ilya Sutskever, und Geoffrey E. Hinton. 2012. „Imagenet classification with deep convolutional neural networks“. In *Advances in Neural Information Processing Systems*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, u. a. 2015. „Human-level control through deep reinforcement learning“. *Nature* 518 (7540): 529–33. http://dx.doi.org/10.1038/nature14236.

Rojas, Raul. 1996. *Neural Networks : A Systematic Introduction*. Springer.

Silver, David. 2015. „UCL Course on RL UCL Course on RL UCL Course on Reinforcement Learning“. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

Sutton, Richard S., und Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. Second. The MIT Press.

Weng, Lilian. 2018. „A (Long) Peek into Reinforcement Learning“. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html.