Course setup

- 9 ec course
- examination based on computer exercises
- weekly exercises discussed in tutorial class
- All course materials (slides, exercises) and schedule via `http://www.snn.ru.nl/~bertk/machinelearning/`

The Perceptron

Relevant in history of pattern recognition and neural networks.

- Perceptron learning rule + convergence, Rosenblatt (1962)
- Perceptron critique (Minsky and Papert, 1969) $\rightarrow$ "Dark ages of neural networks"
- Revival in the 80's: Backpropagation and Hopfield model. Statistical physics entered.
- 1995. Bayesian methods take over. Start of modern machine learning. NN out of fashion.
- 2006 Deep learning, big data.

The Perceptron

$$y(\mathbf{x}) = \text{sign}(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}))$$

where

$$\text{sign}(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases}$$

and $\boldsymbol{\phi}(x)$ is a feature vector (e.g. hard wired neural network).

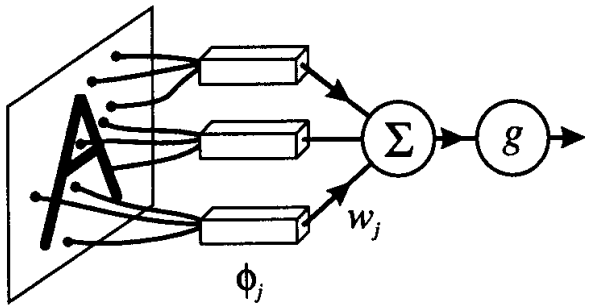
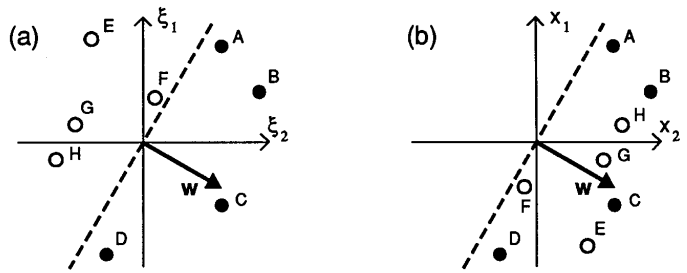


Figure 3.10. The perceptron network used a fixed set of processing elements, denoted ϕ_j, followed by a layer of adaptive weights w_j and a threshold activation function $g(\cdot)$. The processing elements ϕ_j typically also had threshold activation functions, and took inputs from a randomly chosen subset of the pixels of the input image.

The Perceptron

Ignore ϕ, ie. consider inputs x^μ and outputs $t^\mu = \pm 1$
Define $w^T x = \sum_{j=1}^{n} w_j x_j + w_0$. Then, the learning condition becomes

$$\text{sign}(w^T x^\mu) = t^\mu, \quad \mu = 1, \ldots, P$$



We have

$$\text{sign}(w^T x^\mu t^\mu) = 1 \quad \text{or} \quad w^T z^\mu > 0$$

with $z_j^\mu = x_j^\mu t^\mu$.

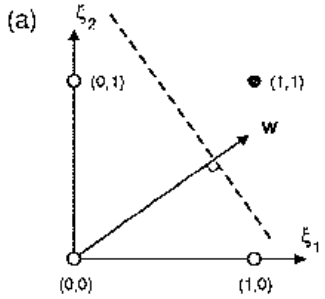
Linear separation

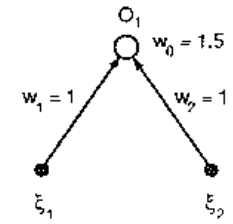
Classification depends on sign of $w^T x$. Thus, decision boundary is hyper plane:

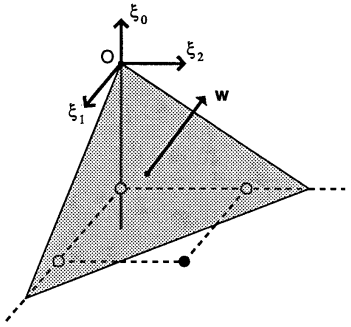
$$0 = w^T x = \sum_{j=1}^{n} w_j x_j + w_0$$

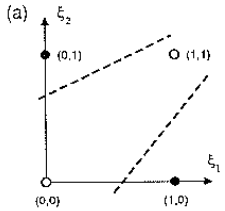
Perceptron can solve linearly separable problems.

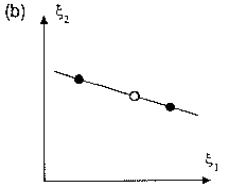
AND problem is linearly separable.











XOR problem and linearly dependent inputs not linearly separable.

Perceptron learning rule

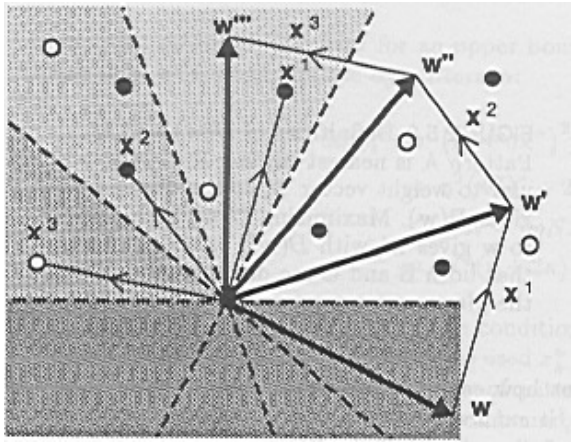
Learning succesful when

$$w^T z^\mu > 0, \quad \text{all patterns}$$

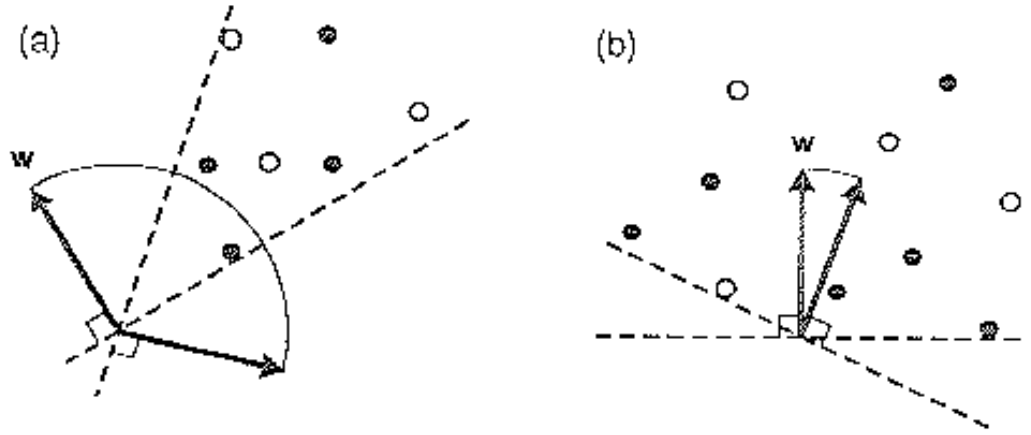
Learning rule is 'Hebbian':

$$\begin{aligned} w_j^{\text{new}} &= w_j^{\text{old}} + \Delta w_j \\ \Delta w_j &= \eta \Theta(-w^T z^\mu) x_j^\mu t^\mu = \eta \Theta(-w^T z^\mu) z_j^\mu \end{aligned}$$

η is the learning rate.



Depending on the data, there may be many or few solutions to the learning problem (or non at all)

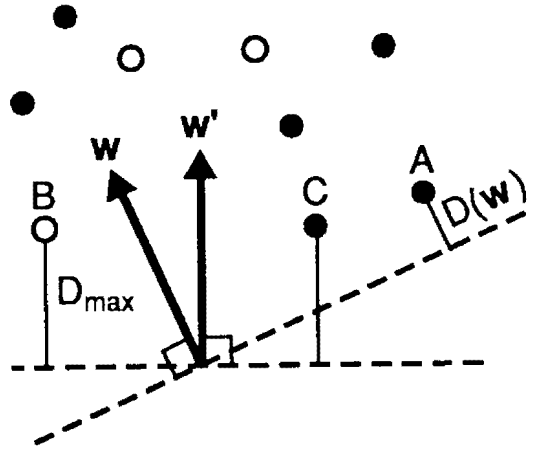


The quality of the solution is determined by the worst pattern. Since the solution does not depend on the size of w:

$$D(w) = \frac{1}{|w|} \min_{\mu} w^T z^{\mu}$$

Acceptable solutions have $D(w) > 0$.

The best solution is given by $D_{\max} = \max_w D(w)$.



$D_{\text{max}} > 0$ iff the problem is linearly separable.

Convergence of Perceptron rule

Assume that the problem is linearly separable, so that there is a solution w^* with $D(w^*) > 0$.

At each iteration, w is updated only if $w \cdot z^\mu < 0$. Let M^μ denote the number of times pattern μ has been used to update w. Thus,

$$w = \eta \sum_\mu M^\mu z^\mu$$

Consider the quanty

$$-1 < \frac{w \cdot w^*}{|w||w^*|} < 1$$

We will show that

$$\frac{w \cdot w^*}{|w||w^*|} \geq \mathcal{O}(\sqrt{M}),$$

with $M = \sum_\mu M^\mu$ the total number of iterations.

Therefore, M can not grow indefinitely. Thus, the perceptron learning rule converges in a finite number of steps when the problem is linearly separable.

Proof:

$$\begin{aligned} w \cdot w^* &= \eta \sum_\mu M^\mu z^\mu \cdot w^* \geq \eta M \min_\mu z^\mu \cdot w^* \\ &= \eta M D(w^*)|w^*| \\ \Delta|w|^2 &= |w + \eta z^\mu|^2 - |w|^2 = 2\eta w \cdot z^\mu + \eta^2 |z^\mu|^2 \\ &\leq \eta^2 |z^\mu|^2 = \eta^2 N \\ |w| &\leq \eta \sqrt{NM} \end{aligned}$$

Thus,

$$1 \geq \frac{w \cdot w^*}{|w||w^*|} \geq \sqrt{M} \frac{D(w^*)}{\sqrt{N}}$$

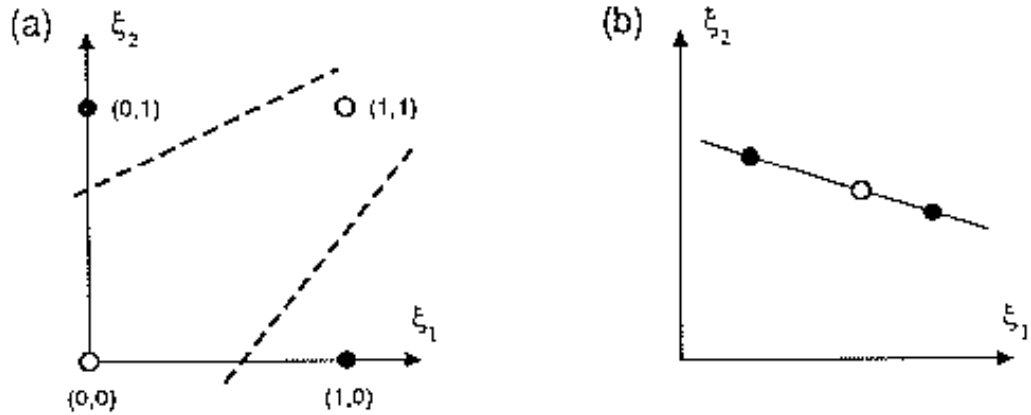
Number of weight updates:

$$M \leq \frac{N}{D^2(w^*)}$$

Capacity of the Perceptron

Consider P patterns in N dimensions in general position:

- no subset of size less than N is linearly dependent.
- general position is necessary for linear separability



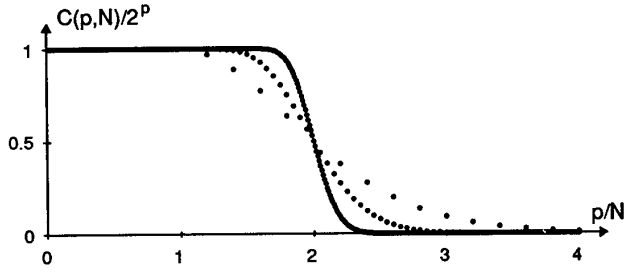
Question: What is the probability that a problem of P samples in N dimensions is linearly separable?

Define $C(P,N)$ the number of linearly separable colorings on P points in N dimensions, with separability plane through the origin. Then (Cover 1966):

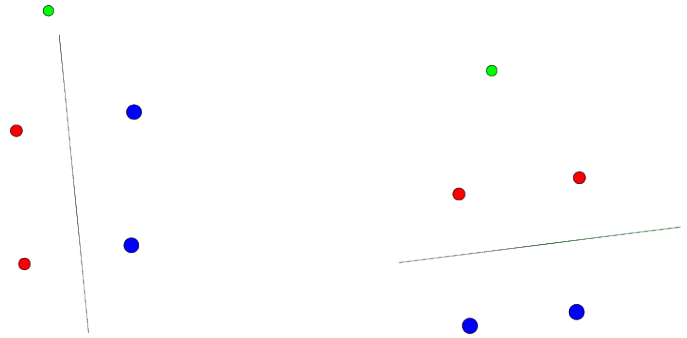
$$C(P,N) = 2\sum_{i=0}^{N-1}\left(\begin{array}{c} P-1 \\ i \end{array}\right)$$

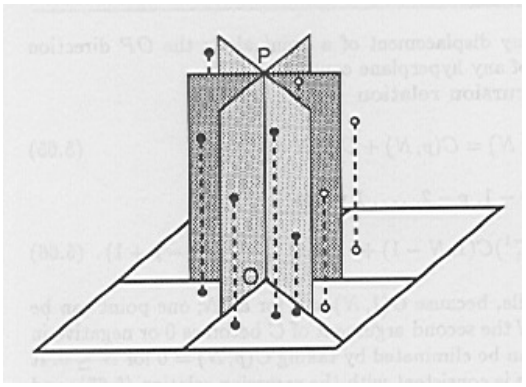
When $P \leq N$ small, then $C(P,N) = 2\sum_{i=0}^{P-1}\left(\begin{array}{c} P-1 \\ i \end{array}\right) = 2(1+1)^{P-1} = 2^P$

When $P = 2N$, then 50 % is linearly separable: $C(P,N) = 2\sum_{i=0}^{N-1}\left(\begin{array}{c} 2N-1 \\ i \end{array}\right) = \sum_{i=0}^{2N-1}\left(\begin{array}{c} 2N-1 \\ i \end{array}\right) = 2^{2N-1} = 2^{P-1}$



Proof by induction.





Add one point X. The set $C(P, N)$ consists of

- colorings with separator through X (A)
- rest (B)

Thus,

$$\begin{aligned} C(P+1, N) &= 2A + B = C(P, N) + A \\ &= C(P, N) + C(P, N-1) \end{aligned}$$

Yields

$$C(P, N) = 2 \sum_{i=0}^{N-1} \left(\begin{array}{c} P-1 \\ i \end{array} \right)$$

Network training

Regression: t_n continue valued, $h_2(x) = x$ and one usually minimizes the squared error (one output)

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}(y(\mathbf{x}_n, \mathbf{w}) - t_n)^2 \\ &= -\log\prod_{n=1}^{N}\mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}) + \ldots \end{aligned}$$

Classification: $t_n = 0, 1$, $h_2(x) = \sigma(x)$, $y(\mathbf{x}_n, \mathbf{w})$ is probability to belong to class 1.

$$\begin{aligned} E(\mathbf{w}) &= -\sum_{n=1}^{N}\{t_n \log y(\mathbf{x}_n, \mathbf{w}) + (1 - t_n)\log(1 - y(\mathbf{x}_n, \mathbf{w}))\} \\ &= -\log\prod_{n=1}^{N} y(\mathbf{x}_n, \mathbf{w})^{t_n}(1 - y(\mathbf{x}_n, \mathbf{w}))^{1-t_n} \end{aligned}$$

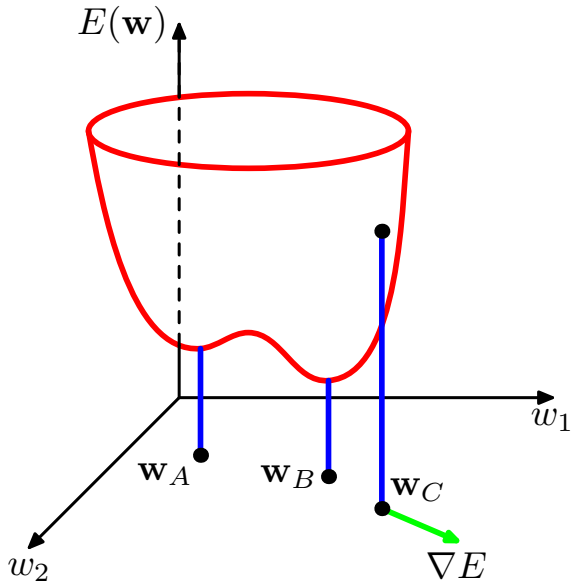
Network training

More than two classes: consider network with K outputs. $t_{nk} = 1$ if $\mathbf{x}_n$ belongs to class k and zero otherwise. $y_k(\mathbf{x}_n, \mathbf{w})$ is the network output

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log p_k(\mathbf{x}_n, \mathbf{w})$$

$$p_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(y_k(\mathbf{x}, \mathbf{w}))}{\sum_{k'=1}^{K} \exp(y_{k'}(\mathbf{x}, \mathbf{w}))}$$

Parameter optimization



E is minimal when $\nabla E(\mathbf{w}) = 0$, but not vice versa!

As a consequence, gradient based methods find a local minimum, not necessary the global minimum.

Gradient descent optimization

The simplest procedure to optimize E is to start with a random $\mathbf{w}$ and iterate

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E(\mathbf{w}^{\tau})$$

This is called batch learning, where all training data are included in the computation of ∇E.

Does this algorithm converge? Yes, if ϵ is "sufficiently small" and E bounded from below.

Proof: Denote $\Delta\mathbf{w} = -\eta \nabla E$.

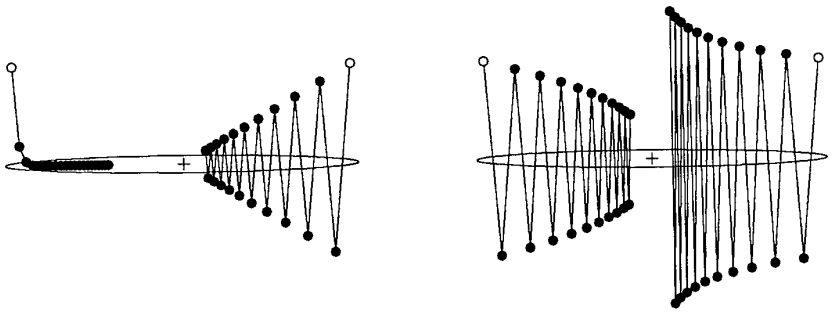
$$E(\mathbf{w} + \Delta\mathbf{w}) \approx E(\mathbf{w}) + (\Delta\mathbf{w})^T \nabla E = E(\mathbf{w}) - \eta \sum_i (\frac{\partial E}{\partial w_i})^2 \leq E(\mathbf{w})$$

In each gradient descent step the value of E is lowered. Since E bounded from below, the procedure must converge asymptotically.

Convergence of gradient descent in a quadratic well

$$
\begin{aligned}
E(w) &= \frac{1}{2}\sum_i \lambda_i w_i^2 \\
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} = -\eta \lambda_i w_i \\
w_i^{\text{new}} &= w_i^{\text{old}} + \Delta w_i = (1 - \eta\lambda_i) w_i
\end{aligned}
$$

Convergence when $|1 - \eta\lambda_i| < 1$. Oscillations when $1 - \eta\lambda_i < 0$.



Optimal learning parameter depends on curvature of each dimension.

Learning with momentum

One solution is adding momentum term:

$$\begin{aligned}\Delta w_{t+1} &= -\eta \nabla E(w_t) + \alpha \Delta w_t \\ &= -\eta \nabla E(w_t) + \alpha \left(-\eta \nabla E(w_{t-1}) + \alpha \left(-\eta \nabla E(w_{t-2}) + \ldots\right)\right) \\ &= -\eta \sum_{k=0}^{t} \alpha^k \nabla E(w_{t-k})\end{aligned}$$

Consider two extremes:

No oscillations all derivative are equal:

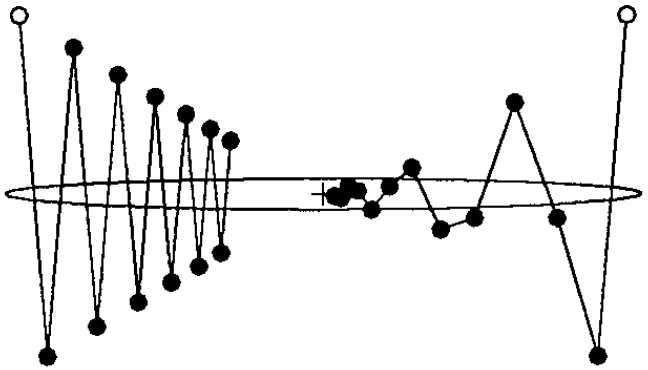
$$\Delta w_{t+1} \approx -\eta \nabla E \sum_{k=0}^{t} \alpha^k = -\frac{\eta}{1-\alpha} \frac{\partial E}{\partial w}$$

results in acceleration

Oscillations all derivatives are equal but have opposite sign:

$$\Delta w(t+1) \quad \approx \quad -\eta \nabla E \sum_{k=0}^{t} (-\alpha)^k = -\frac{\eta}{1+\alpha} \frac{\partial E}{\partial w}$$

results in decceleration



Newtons method

One can also use Hessian information for optimization. As an example, consider a quadratic approximation to E around $\mathbf{w}_0$:

$$\begin{aligned} E(\mathbf{w}) &= E(\mathbf{w}_0) + \mathbf{b}^T(\mathbf{w}-\mathbf{w}_0) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_0)\mathbf{H}(\mathbf{w}-\mathbf{w}_0) \\ b_i &= \frac{\partial E(\mathbf{w}_0)}{\partial w_i} \qquad H_{ij} = \frac{\partial^2 E(\mathbf{w}_0)}{\partial w_i \partial w_j} \\ \nabla E(\mathbf{w}) &= \mathbf{b} + \mathbf{H}(\mathbf{w}-\mathbf{w}_0) \end{aligned}$$

We can solve $\nabla E(\mathbf{w}) = 0$ and obtain

$$\mathbf{w} = \mathbf{w}_0 - \mathbf{H}^{-1}\nabla E(\mathbf{w}_0)$$

This is called Newtons method.

Quadratic approximation is exact when E is quadratic, so convergence in one step.

Quasi-Newton: Consider only diagonal of H.

Line search

Another solution is line optimisation:

$$w_1 = w_0 + \lambda_0 d_0, \qquad d_0 = -\nabla E(w_0)$$

$\lambda_0 > 0$ is found by a one dimensional optimisation

$$0 = \frac{\partial}{\partial \lambda_0} E(w_0 + \lambda_0 d_0) = d_0 \cdot \nabla E(w_1) = d_0 \cdot d_1$$

Therefore, subsequent search directions are orthogonal.

Conjugate gradient descent

We choose as new direction a combination of the gradient and the old direction

$$d_1' = -\nabla E(w_1) + \beta d_0$$

Line optimisation $w_2 = w_1 + \lambda_1 d_1'$ yields $\lambda_1 > 0$ such that $d_1' \cdot \nabla E(w_2) = 0$.

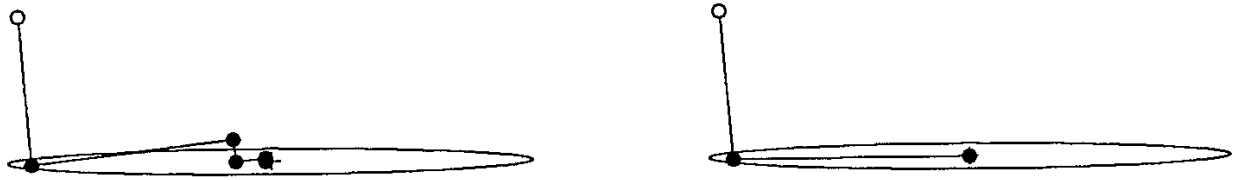
The direction d_1' is found by demanding that $\nabla E(w_2) \approx 0$ also in the 'old' direction d_0:

$$0 = d_0 \cdot \nabla E(w_2) \approx d_0 \cdot (\nabla E(w_1) + \lambda_1 H(w_1) d_1')$$

or

$$d_0 H(w_1) d_1' = 0$$

The subsequent search directions d_0, d_1' are said to be conjugate.



Polak-Ribiere rule

The conjugate directions can be computed without computing the Hessian matrix, for instance using the Polak-Ribiere rule:[1]

$$\beta = \frac{(\nabla E(w_1) - \nabla E(w_0)) \cdot \nabla E(w_1)}{\|\nabla E(w_0)\|^2}$$

For quadratic problems, it can be proven that this rule keeps the last n directions all mutually conjugate [Press et al., 1996]

$$d_i^T H d_j = 0 \qquad i, j = 1, \ldots, n$$

[1] We need $0 = d_0^T H(w_1) d_1'$. We use $\nabla E(w_0) \approx \nabla E(w_1) + (w_0 - w_1)^T H(w_1) = \nabla E(w_1) - \lambda_0 d_0^T H(w_1)$ and $d_1' = -\nabla E(w_1) + \beta d_0$. Then

$$0 = \lambda_0 d_0^T H(w_1) d_1' = (\nabla E(w_1) - \nabla E(w_0)) \cdot (-\nabla E(w_1) + \beta d_0) = -(\nabla E(w_1) - \nabla E(w_0)) \cdot \nabla E(w_1) + \beta \|\nabla E(w_0)\|^2$$

where in the last step we used that $d_0 \cdot \nabla E(w_1) = 0$.

Stochastic gradient descent

One can also consider on-line learing, where only one or a subset of training patterns is considered for computing ∇E.

$$E(\mathbf{w}) = \sum_n E_n(\mathbf{w}) \qquad \mathbf{w}_{t+1} = \mathbf{w}_t - \alpha_t \nabla E_n(\mathbf{w}^\tau)$$

May be efficient for large data sets. This results in a stochastic dynamics in $\mathbf{w}$ that can help to escape local minima.

Robbins Monro

Consider the problem to find x such that

$$M(x) = a, \qquad M(x) = \langle N(x,\xi)\rangle = \int d\xi p(\xi) N_i(x,\xi)$$

x, a, M, N are vectors. $N_i(x,\xi)$ some non-linear function, $p(\xi$ is a probability distribution and a_i a constant.

Method of *stochastic approximation* originally due to Robbins and Monro 1951:
- Initialize x_0 random
- For $t = 0, \ldots,$ Choose $\xi_t \sim p(\xi)$; Update $x_{t+1} = x_t + \alpha_t(a - N(x_t, \xi_t))$

If $M_i(x) = \nabla_i E$ and E is convex and x^* the unique solution, then one can prove that $||x_t - x^*||^2 \to 0$, provided that

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

For instance $\alpha_t = 1/t$.

Stochastic gradient descent

Denote training error

$$E(w) = \frac{1}{P} \sum_{\mu} E_{\mu}(w)$$

we wish to find solution of

$$\nabla E(w) = \frac{1}{P} \sum_{\mu} \nabla E_{\mu}(w) = 0$$

This is an instance of the Robbins-Monro problem with $\xi = \mu = 1, \ldots, P$ and

$$p(\mu) = \frac{1}{P} \qquad a_i = 0 \qquad N(w, \mu) = E_{\mu}(w)$$

The SGD method is
- Choose random a pattern $\mu \in [1, \ldots, P]$
- Update $w_{t+1} = w_t - \eta_t \nabla E_{\mu}(w)$

Extensions of SGD and comparisons see [Sohl-Dickstein et al., 2013].

Feed-forward Network functions

We extend the previous regression model with fixed basis functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

to a model where ϕ_j is adaptive:

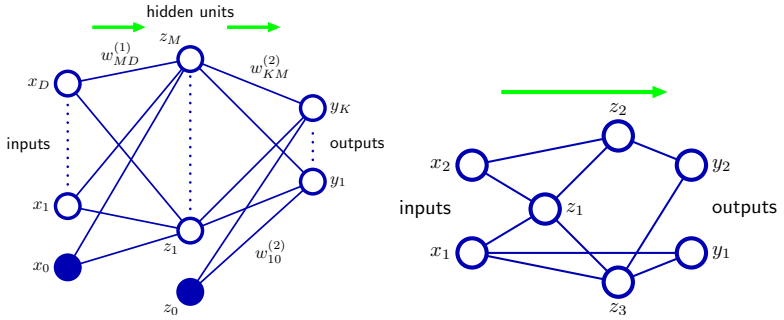
$$\phi_j(\mathbf{x}) = h(\sum_{i=0}^{D} w_{ji}^{(1)} x_i)$$

Feed-forward Network functions

In the case of K outputs

$$y_k(\mathbf{x}, \mathbf{w}) = h_2\left(\sum_{j=1}^{M} w_{kj}^{(2)} h_1\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

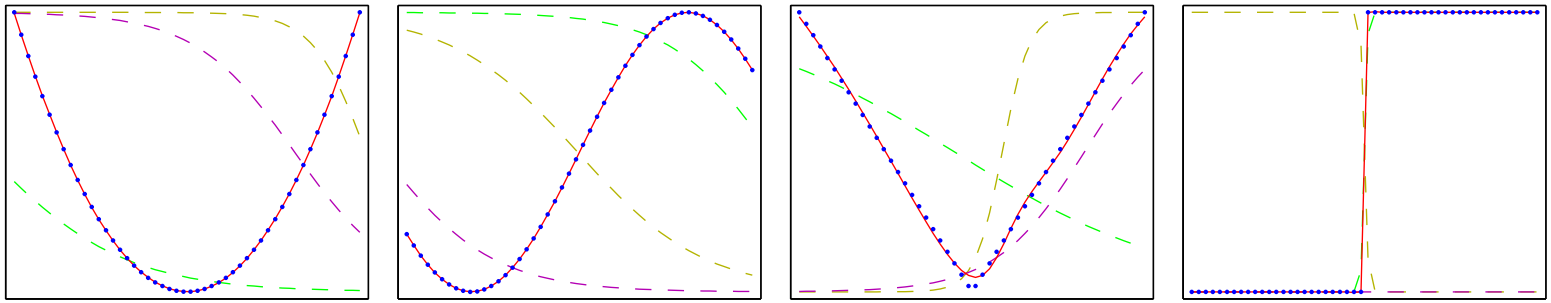
$h_2(x)$ is $\sigma(x)$ or x depending on the problem. $h_1(x)$ is $\sigma(x)$ or $\tanh(x)$.



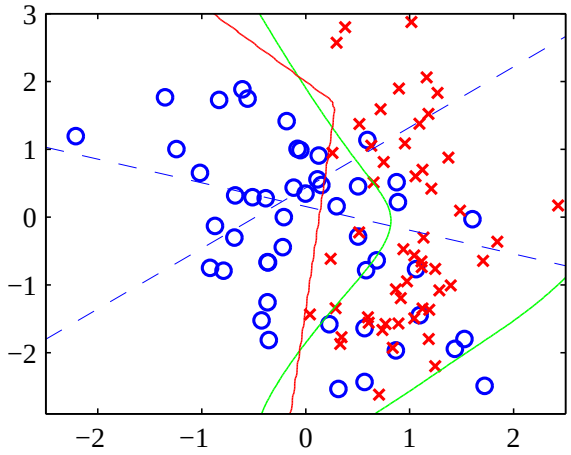
Left) Two layer architecture. Right) general feed-forward network with skip-layer connections.

If h_1, h_2 linear, the model is linear. If $M < D, K$ it computes principle components (Bishop section 12.4.2).

Feed-forward Network functions



Two layer NN with 3 'tanh' hidden units and linear output can approximate many functions. $x \in [-1, 1]$, 50 equally spaced points. From left to right: $f(x) = x^2, \sin(x), |x|, \Theta(x)$. Dashed lines are outputs of the 3 hidden units.



Two layer NN with two inputs and 2 'tanh' hidden units and sigmoid output for classification. Dashed lines are hidden unit activities.

Feed-forward neural networks have good approximation properties.

Weight space symmetries

For any solutions of the weights, there are many equivalent solutions due to symmetry:
- for any hidden unit j with $\tanh$ activation function, change $w_{ji} \rightarrow -w_{ji}$ and $w_{kj} \rightarrow -w_{kj}$: 2^M solutions
- rename the hidden unit labels: $M!$ solutions

Thus a total of $M!2^M$ equivalent solutions, not only for $\tanh$ activation functions.

Error backpropagation

Error is sum of error per pattern

$$
\begin{aligned}
E(\mathbf{w}) &= \sum_n E^n(\mathbf{w}) \qquad E^n(\mathbf{w}) = \frac{1}{2}\|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2 \\
y_k(\mathbf{x}, \mathbf{w}) &= h_2\left(w_{k0} + \sum_{j=1}^{M} w_{kj} h_1\left(w_{j0} + \sum_{i=1}^{D} w_{ji} x_i\right)\right) \\
&= h_2(a_k) \\
a_k &= w_{k0} + \sum_{j=1}^{M} w_{kj} h_1(a_j) = \sum_{j=0}^{M} w_{kj} h_1(a_j) \qquad h_1(a_0) = 1 \\
a_j &= w_{j0} + \sum_{i=1}^{D} w_{ji} x_i = \sum_{i=0}^{D} w_{ji} x_i \qquad x_0 = 1
\end{aligned}
$$

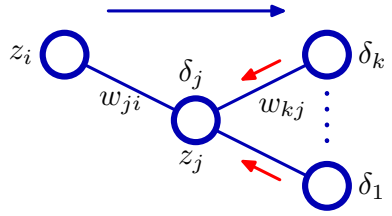
i labels inputs, j labels hiddens, k labels outputs.

Error backpropagation

We do each pattern separately, so we consider E^n

$$
\begin{aligned}
y_k(\mathbf{x}^n, \mathbf{w}) &= h_2(a_k^n) = h_2\left(\sum_{j=0}^{M} w_{kj} h_1(a_j^n)\right) = h_2\left(\sum_{j=0}^{M} w_{kj} h_1\left(\sum_{i=0}^{D} w_{ji} x_i^n\right)\right) \\
\frac{\partial E^n}{\partial w_{kj}} &= (y_k^n - t_k^n)\frac{\partial y_k^n}{\partial w_{kj}} = (y_k^n - t_k^n) h_2'(a_k^n)\frac{\partial a_k^n}{\partial w_{kj}} = (y_k^n - t_k^n) h_2'(a_k^n) h_1(a_j^n) \\
&= \delta_k^n h_1(a_j^n) \qquad \delta_k^n = (y_k^n - t_k^n) h_2'(a_k^n) \\
\frac{\partial E^n}{\partial w_{ji}} &= \sum_{k=1}^{K} (y_k^n - t_k^n)\frac{\partial y_k^n}{\partial w_{ji}} = \sum_{k=1}^{K} (y_k^n - t_k^n) h_2'(a_k^n)\frac{\partial a_k^n}{\partial w_{ji}} \\
&= \sum_{k=1}^{K} \delta_k^n w_{kj} h_1'(a_j^n)\frac{\partial a_j^n}{\partial w_{ji}} = \sum_{k=1}^{K} \delta_k^n w_{kj} h_1'(a_j^n) x_i^n = \delta_j^n x_i^n \\
\delta_j^n &= h_1'(a_j^n) \sum_{k=1}^{K} \delta_k^n w_{kj}
\end{aligned}
$$

Error backpropagation



The back propagation extends to arbitrary layers:

1. $z_i^n = x_i^n$ *forward propagation* all activations $z_j^n = h_1(a_j^n)$ and $z_k^n = h_2(a_k^n)$, etc.

2. Compute the δ_k^n for the output units, and *back-propagate* the δ to obtain δ_j^n each hidden unit j

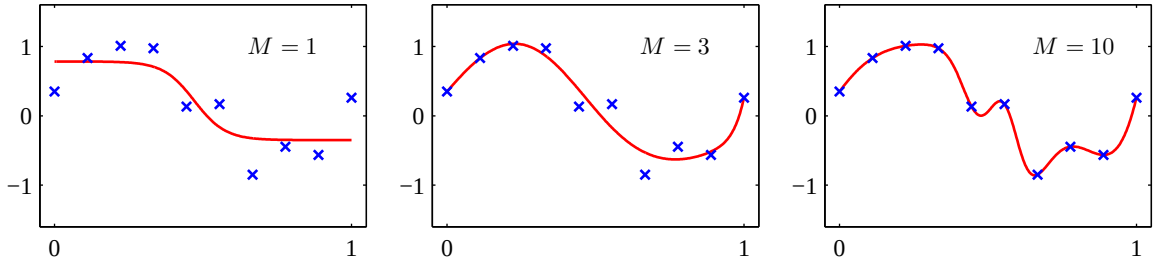
3. $\partial E^n / \partial w_{kj} = \delta_k^n z_j^n$ and $\partial E^n / \partial w_{ji} = \delta_j^n z_i^n$

4. for batch mode, $\partial E / \partial w_{ji} = \sum_n \partial E^n / \partial w_{ji}$

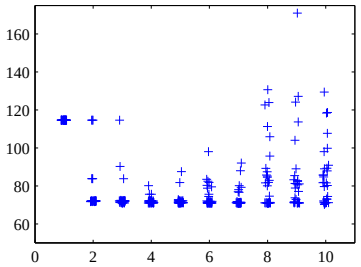
E is a function of $O(|\mathbf{w}|)$ variables. In general, the computation of E requires $O(|\mathbf{w}|)$ operations. The computation of ∇E would thus require $O(|\mathbf{w}|^2)$ operations.

The backpropagation method allows to compute ∇E efficiently, in $O(|\mathbf{w}|)$ operations.

Regularization



Complexity of neural network solution is controlled by number of hidden units



sum squared test error for different number of hidden units and different weight initializations. Error is also affected by local minima.

Part of the cause of local minima is the saturation of the sigmoid functions $\tanh(\sum w_{ij}x_j)$. When w_{ij} becomes large, any change in its value hardly affects the output, implying $\nabla_{ij}E = 0$.

One can partly prevent this from happening by

- chosing $\tanh$ instead of σ transfer functions and scaling of inputs and outputs with mean zero and standard deviation one
- proper initialisation of w_{ij} with mean zero and standard deviation of order $1/\sqrt{n_1}$, with n_1 the number of inputs to neuron i.
- add regularizer such as $\sum_i w_i^2$ to cost keeps weights small
- dropouts, other transfer functions, adding noise, ...

MLPs are universal approximators

Consider 2^n binary patterns in n dimensions and two classes:

$$x^\mu \to c^\mu = \pm 1, \qquad x_i^\mu = \pm 1$$

Use 2^n hidden units, labeled $j = 0, \ldots, 2^n - 1$, i labels input. Set

$$\begin{aligned} w_{ji} &= b \quad \text{if } i\text{th digit in binary repr. of } j \text{ is 1} \\ w_{ji} &= -b \quad \text{else} \end{aligned}$$

j	binary	w_{j1}	w_{j2}
0	00	-b	-b
1	01	-b	b
2	10	b	-b
3	11	b	b

x_1	x_2	$\sum_i w_{0i}x_i$	$w_{1i}x_i$	$w_{2i}x_i$	$w_{3i}x_i$
-1	-1	**2b**	0	0	-2b
-1	1	0	**2b**	-2b	0
1	-1	0	-2b	**2b**	0
1	1	-2b	0	0	**2b**

MLPs are universal approximators

Use threshold of $(n-1)b$ at each hidden unit. $z_j = \Theta[\sum_i w_{ji} x_i - (n-1)b)]$. The remaining problem has $p = 2^n$ patterns in 2^n dimensions and is linearly separable.

Define $c = \text{sign}[\sum_{j=0}^{3} w_j z_j]$.

x_1	x_2	z_0	z_1	z_2	z_3	c
-1	-1	1	0	0	0	sign$[w_0]$
-1	1	0	1	0	0	sign$[w_1]$
1	-1	0	0	1	0	sign$[w_2]$
1	1	0	0	0	1	sign$[w_3]$

The combination of linear summation and non-linear functions can create many different functions.

- The MLP with a single hidden layer can map any continuous function [Cybenko, 1989, Hornik et al., 1989]

Convolution networks

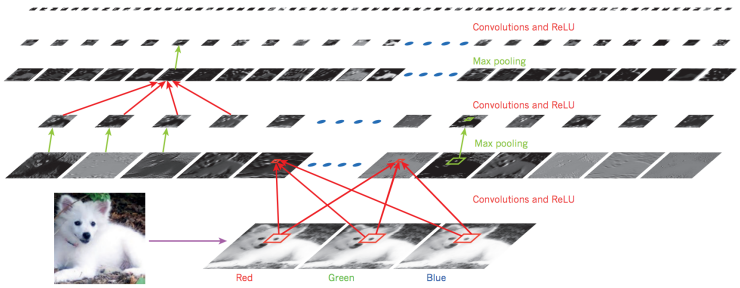
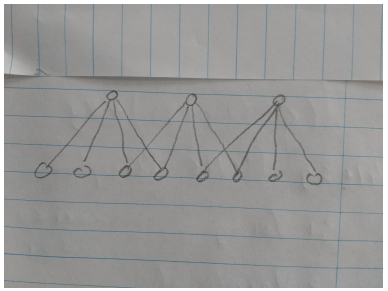
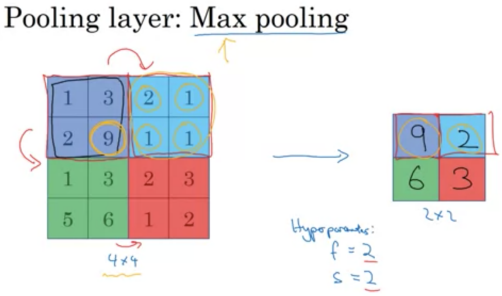


Figure 2 | Inside a convolutional network. The outputs (not the filters) of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left; and RGB (red, green, blue) inputs, bottom right). Each rectangular image is a feature map corresponding to the output for one of the learned features, detected at each of the image positions. Information flows bottom up, with lower-level features acting as oriented edge detectors, and a score is computed for each image class in output. ReLU, rectified linear unit.





A special architecture for images using 4 ideas:

local connections: Connect not fully the bipartite graph between two layers.

weight sharing: use the same parameters to different neurons that detect same feature (feature layer)

(max) pooling: down sample each feature map (not adaptive) many layers: repeat many times.

Fukushima 1982, LeCunn 1990

Imagenet 2012 competition

Imagenet is an annual computer vision competition. Classify images in 1000 classes. 1.2 million training images, 50,000 validation images, and 150,000 testing images.

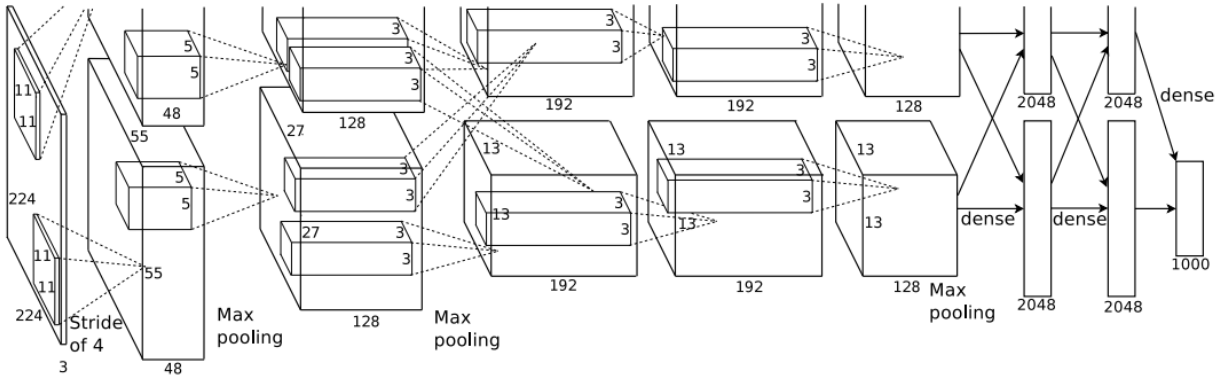
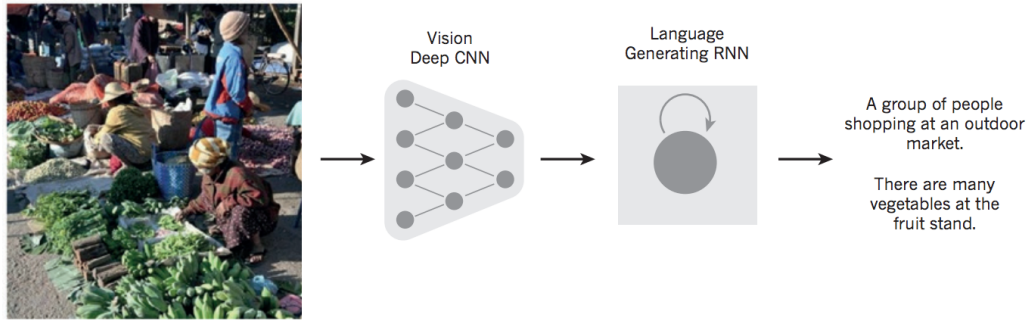


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

Interpreting images




A woman is throwing a **frisbee** in a park.


A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background


A little **girl** sitting on a bed with a teddy bear.


A group of **people** sitting on a boat in the water.


A giraffe standing in a forest with **trees** in the background.

Vinyals et al. 2014

Deep learning papers

Fukushima Neocognitron 1982:
https://www.sciencedirect.com/science/article/pii/0031320382900243

LeCunn Convolutional networks 1990:
http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propag
pdf

Krizhevsky Imagenet 2012:
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutiona
pdf

Vinyals Interpreting images 2014:http://arxiv.org/abs/1502.03044

LeCunn, Bengio, Hinton. Deep learning 2015:
https://www.nature.com/articles/nature14539.pdf