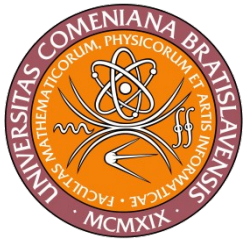
Faculty of Mathematics, Physics and Informatics
Comenius University Bratislava




# Neural Networks

## Lecture 3

## Supervised single-layer models

Igor Farkaš

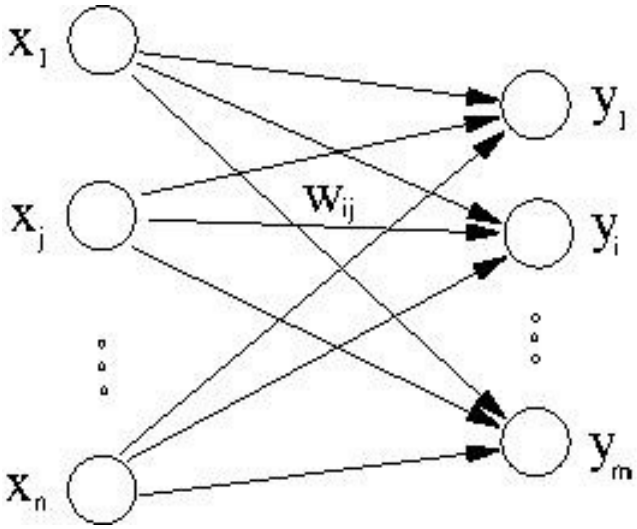
2024

# Linear NN models

Input vector: $\boldsymbol{x} = [x_1, x_{2,\ldots}, x_n]^T$

Output vector: $\boldsymbol{y} = [y_1, y_{2,}.., y_m]^T$

Weight matrix: $\mathbf{W}$ ~ type $[m \times n]$




Linear transformation $\varphi : \Re^n \to \Re^m$, $\boldsymbol{y} = \mathbf{W}\boldsymbol{x}$

☹ ignores saturation property of neurons
☺ allows to find analytic solutions using linear algebra.

(Kohonen, 1970; Anderson, 1972; Cooper, 1973)

- Adding layers in a linear NN does not appear reasonable (since no complexity is added).
- But: It allows nonlinear learning dynamics in linear deep nets (Saxe, 2015).

# Closed-form solution

Let's consider the data set: $A_{train} = \{(\boldsymbol{x}^{(p)}, \boldsymbol{y}^{(p)}), p = 1,2,...,N\}$.

We look for a matrix $\boldsymbol{W}$ that satisfies $\boldsymbol{y}^{(p)} = \boldsymbol{W}\boldsymbol{x}^{(p)}$, for each $p$.

In matrix notation: $\boldsymbol{Y} = \boldsymbol{W}\,\boldsymbol{X}$

$$\underset{(m\times N)}{[\boldsymbol{y}^{(1)}\ \boldsymbol{y}^{(2)}\ \ldots\ \boldsymbol{y}^{(N)}]} = \underset{(m\times n)}{\boldsymbol{W}} \times \underset{(n\times N)}{[\boldsymbol{x}^{(1)}\ \boldsymbol{x}^{(2)}\ \ldots\ \boldsymbol{x}^{(N)}]}$$

If $\boldsymbol{X}$ was regular (i.e., square matrix with $N = n$, with linearly independent rows) then $\boldsymbol{X}^{-1}$ would exist and $\boldsymbol{W} = \boldsymbol{Y}\boldsymbol{X}^{-1}$.

However, in general we cannot assume $N = n$, nor linear independence of input vectors ($\Rightarrow \boldsymbol{X}^{-1}$ does not exist). Only generalized solutions exists:

$$\boldsymbol{W} = \boldsymbol{Y}\boldsymbol{X}^{+}$$

$\boldsymbol{X}^{+}$ is called (Moore-Penrose) pseudoinverse matrix of $\boldsymbol{X}$. Theorem: $\forall\ \boldsymbol{X}$, $\exists\ \boldsymbol{X}^{+}$ with properties: 1) $\boldsymbol{X}\boldsymbol{X}^{+}\boldsymbol{X} = \boldsymbol{X}$, 2) $\boldsymbol{X}^{+}\boldsymbol{X}\boldsymbol{X}^{+} = \boldsymbol{X}^{+}$, 3) symmetric $\boldsymbol{X}\boldsymbol{X}^{+}$ and $\boldsymbol{X}^{+}\boldsymbol{X}$.

a) $\boldsymbol{X}^{+} = \boldsymbol{X}^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}})^{-1}$, if $n < N$ and $rank(\boldsymbol{X}) = n$.

b) $\boldsymbol{X}^{+} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}$, if $n > N$ and $rank(\boldsymbol{X}) = N$.

# Auto-association in a linear network

Let us now look at autoassociation: $\boldsymbol{y}^{(p)} \equiv \boldsymbol{x}^{(p)}$, $\dim(\boldsymbol{y}) = \dim(\boldsymbol{x}) = n$

This can be useful for a memory, if $n > N$ (i.e. few examples of large dimension).

*Consider the task:* to remember $N$ prototypes $[\boldsymbol{x}^{(1)}\ \boldsymbol{x}^{(2)} \ldots \boldsymbol{x}^{(N)}]$.

*Goal:* train a linear model on prototypes and then submit a corrupted version of a prototype. Model should be able to reconstruct it.

Since $\boldsymbol{Y} \equiv \boldsymbol{X}$, then $\boldsymbol{W} = \boldsymbol{X}\boldsymbol{X}^{+}$. How to interpret $\boldsymbol{W}$?

In a special, restrictive case ($N = n$, linearly independent inputs), we would have a trivial solution $\boldsymbol{W} = \boldsymbol{I}$ (identity).

How about a general (non-trivial) case?

# Basics of linear vector spaces

Let's have a linear space $\mathfrak{R}^n$.

Linear manifold $\mathscr{L} = \{\boldsymbol{x} \in \mathfrak{R}^n \,|\, \boldsymbol{x} = a_1 \boldsymbol{x}^{(1)} + a_2 \boldsymbol{x}^{(2)} + \ldots + a_N \boldsymbol{x}^{(N)},\ a_p \neq 0\}$

$$\mathscr{L} \subset \mathfrak{R}^n$$

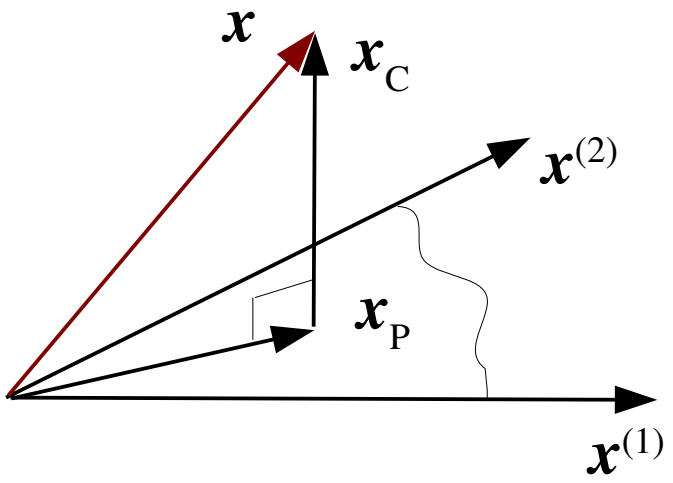
Orthogonal complement $\mathscr{L}^{\perp} = \{\boldsymbol{x} \in \mathfrak{R}^n \,|\, \boldsymbol{x} \perp \mathscr{L}\}$

Hence, $\mathscr{L} \oplus \mathscr{L}^{\perp} = \mathfrak{R}^n$

Each vector $\boldsymbol{x} \in \mathfrak{R}^n$ can be uniquely decomposed:

$$\boldsymbol{x} = \boldsymbol{x}_{\mathrm{P}} + \boldsymbol{x}_{\mathrm{C}}$$

where $\boldsymbol{x}_{\mathrm{P}} \in \mathscr{L}$ and $\boldsymbol{x}_{\mathrm{C}} \in \mathscr{L}^{\perp}$.



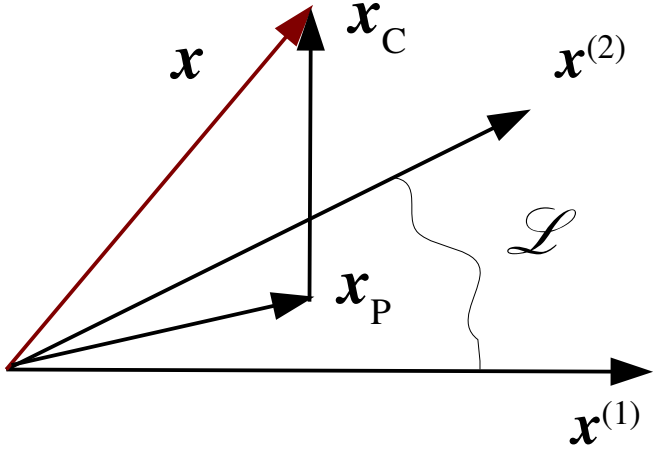
# What does an autoassociative NN do?

Training set $\mathrm{A}_{\mathrm{train}} = \{\boldsymbol{x}^{(p)}, p = 1,2,...,N\}$ forms the linear manifold $\mathscr{L}$.

NN considers every departure $\boldsymbol{x}$ from $\mathscr{L}$ as added noise that needs to be filtered out by projecting $\boldsymbol{x}$ to $\mathscr{L}$:

We need to show that output $\boldsymbol{W}\boldsymbol{x} = \boldsymbol{X}\boldsymbol{X}^{+}\boldsymbol{x} = \boldsymbol{x}_{\mathrm{P}}$ (filtered version of $\boldsymbol{x}$), i.e. that operator $\boldsymbol{W} = \boldsymbol{X}\boldsymbol{X}^{+}$ makes an orthogonal projection to $\mathscr{L}$.

Alternatively, the NN model with operator $\boldsymbol{W} = \boldsymbol{I} - \boldsymbol{X}\boldsymbol{X}^{+}$ is called novelty detector, where $\boldsymbol{W}\boldsymbol{x} = \boldsymbol{x}_{\mathrm{C}} \in \mathscr{L}^{\perp}$.




Now assume: you learned $N$ patterns, and you want to add $(N+1)$-st pattern.

How to change $\boldsymbol{W}$ efficiently?

# Gram-Schmidt orthogonalization process

Let's have a base $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(k)} \in \mathscr{L}$, for which we want to create an orthogonal base $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(k)} \in \mathscr{L}$.

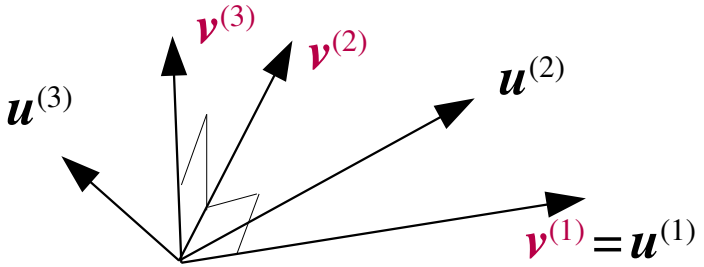
Procedure:

Let $\boldsymbol{v}^{(1)} \equiv \boldsymbol{u}^{(1)}$. In space with base formed by $\boldsymbol{v}^{(1)}$, $\boldsymbol{u}^{(2)}$ let's find vector $\boldsymbol{v}^{(2)}$ such that $\boldsymbol{v}^{(2)} \perp \boldsymbol{v}^{(1)}$. Hence, $\boldsymbol{v}^{(2)} = a_1 \boldsymbol{v}^{(1)} + a_2 \boldsymbol{u}^{(2)}$.

$$\boldsymbol{v}^{(2)} = \boldsymbol{u}^{(2)} - \frac{\boldsymbol{v}^{(1)T} \boldsymbol{u}^{(2)}}{\left|\boldsymbol{v}^{(1)}\right|^2} \boldsymbol{v}^{(1)}$$

Recursive formula: we have $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(k-1)}$ and compute $\boldsymbol{v}^{(k)}$ such that $\boldsymbol{v}^{(k)} \perp \boldsymbol{v}^{(i)}$, $i = 1,2, \ldots, k\text{-}1$

$$\boldsymbol{v}^{(k)} = \boldsymbol{u}^{(k)} - \sum_{i=1}^{k-1} \frac{\boldsymbol{v}^{(i)T} \boldsymbol{u}^{(k)}}{\left|\boldsymbol{v}^{(i)}\right|^2} \boldsymbol{v}^{(i)}$$




How to use this recursion for a GI model?

# General Inverse model

If we have patterns $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}$, we can use Gram-Schmidt process to calculate orthogonal base $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, ..., \boldsymbol{z}^{(N)}$

Memory model $\boldsymbol{W}$ is computed recursively:

1. Initialize $\boldsymbol{W}^{(0)} = \boldsymbol{0}$, $i = 1$
2. Repeat

$$\boldsymbol{z}^{(i)} = \boldsymbol{x}^{(i)} - \boldsymbol{W}^{(i-1)} \boldsymbol{x}^{(i)}$$

$$\boldsymbol{W}^{(i)} = \boldsymbol{W}^{(i-1)} + \frac{\boldsymbol{z}^{(i)} \boldsymbol{z}^{(i)T}}{\left|\boldsymbol{z}^{(i)}\right|^2}$$

$$i = i+1$$

$$\textit{Hint:} \sum_{i=1}^{k-1} \frac{\boldsymbol{v}^{(i)T} \boldsymbol{u}^{(k)}}{\left|\boldsymbol{v}^{(i)}\right|^2} \boldsymbol{v}^{(i)} = \sum_{i=1}^{k-1} \frac{\boldsymbol{v}^{(i)} \boldsymbol{v}^{(i)T}}{\left|\boldsymbol{v}^{(i)}\right|^2} \boldsymbol{u}^{(k)}$$

# Example: 8 faces from CMU image data repository

face 1

face 2

face 3

face 4

face 5

face 6

face 7

face 8

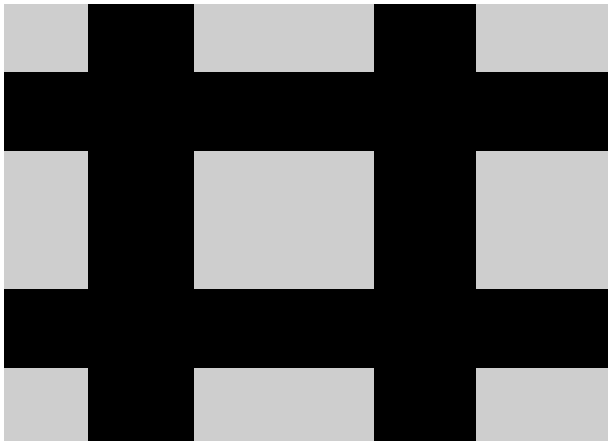
face 1 – corrupt

# Recall by GI and novelty detection

GI

Novelty detection

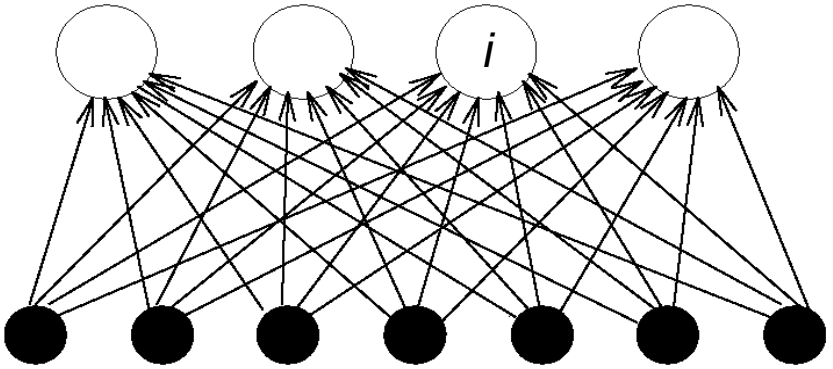
New input



Restoration – GI

# Supervised single-layer perceptrons

- $i$ = 1,2,..., $m$ neurons, that can work independently
- then each neuron independently splits input space into two subspaces




- For sigmoidal neurons we get learning rule: $(net_i \equiv o_i)$

$$w_{ij}(t+1) = w_{ij}(t) + \alpha.(d_i - y_i).f'(o_i).x$$

- for linear neurons we get the least-means-square (LMS) learning rule

$$w_{ij}(t+1) = w_{ij}(t) + \alpha.(d_i - y_i).x_j$$

- closed-form solution may exist via pseudoinverse: for a linear model

$\boldsymbol{W} = \boldsymbol{D}.\boldsymbol{X}^{+}$ , or $\boldsymbol{W} = f^{-1}(\boldsymbol{D}).\boldsymbol{X}^{+}$, if a nonlinearity is used [$f^{-1}(d_i) = o_i$]

# Softmax regression

- In classification tasks, where #output_neurons = #classes, it is convenient to interpret outputs as (conditional) probabilities
- hard assignment vs soft assignment (of classes)
- Softmax activ. function (Luce, 1959) enables that (by normalization): all nonnegative values sum up to one

$$y_i = \frac{\exp(o_i)}{\sum_k \exp(o_k)} \qquad \operatorname{argmax}_i y_i = \operatorname{argmax}_i o_i \qquad o_i = \sum_j w_{ij} x_j + b_i$$

logit

- Important: the model remains differentiable
- Despite nonlinearity, the outputs of softmax regression are still determined by an affine transformation of input features (→ logits)
- Hence, softmax regression is a linear model.

(Zhang et al, 2020)

# Log-likelihood estimation

- Assume a training set $\{\boldsymbol{X},\boldsymbol{D}\}$, consisting of $N$ pairs of input vectors $\boldsymbol{x}^{(p)}$ and one-hot label vectors $\boldsymbol{d}^{(p)}$. Let NN model predictions be $\boldsymbol{y}^{(p)}$.
- We compare the predictions with ground truth by checking how probable the actual classes are according to our model, given the features:

$$P(\boldsymbol{D}|\boldsymbol{X}) = \prod_{p=1}^{N} P(\boldsymbol{d}^{(p)}|\boldsymbol{x}^{(p)})$$

- According to maximum likelihood estimation, we maximize $P(\boldsymbol{D}|\boldsymbol{X})$, which is equivalent to minimizing the negative log-likelihood, i.e. $-\log P(\boldsymbol{D}|\boldsymbol{X})$

$$-\log P(\boldsymbol{D}|\boldsymbol{X}) = -\sum_{p=1}^{N} \log P(\boldsymbol{d}^{(p)}|\boldsymbol{x}^{(p)}) = \sum_{p=1}^{N} loss(\boldsymbol{d}^{(p)}, \boldsymbol{y}^{(p)})$$

- For each pattern *p* (with *m* classes assumed)

$$loss_{CE}^{(p)} = CE^{(p)} = -\sum_{i=1}^{m} d_i^{(p)} \log y_i^{(p)}$$

# Cross-entropy loss calculation

- $CE = -\log P(\boldsymbol{d}|\boldsymbol{x}) = -\Sigma_i d_i \log y_i,$ $\quad y_i = \dfrac{\exp(o_i)}{\sum_k \exp(o_k)}$
- $i, k = 1,2,\ldots,m$

  $CE = \Sigma_i d_i \log \Sigma_k \exp(o_k) - \Sigma_i d_i o_i = \log \Sigma_k \exp(o_k) - \Sigma_i d_i o_i$
- The derivative:

$$\frac{d e_{CE}}{d o_i} = \frac{\exp(o_i)}{\sum_k \exp(o_k)} - d_i = P(y=i|\boldsymbol{x}) - d_i$$

- has the same effect as in case of the squared error
- allows easy gradient computation
- provides a useful link to information theory

# Basics of information theory

- assume a random discrete event that can take values $j$
- Entropy: $H(p) = -\Sigma_j\, p_j \log p_j$, reaches max. for uniform distribution
- In order to encode data drawn randomly from the distribution $p$, we need at least $H(p)$ “nats” to encode it (analogy to a “bit”).
- $-\log q_j$ quantifies surprisal observing an event $j$ having assigned it a (subjective) probability $q_j$
- Cross entropy $H(p, q)$ is the expected surprisal of an observer with subjective probabilities $q$ upon seeing data that was actually generated according to probabilities $p$.
- Kullback-Leibler divergence measures the distance b/w two distributions: $D_{KL}(p,q) = H(p,q) - H(p) = -\Sigma_j\, p_j \log (p_j / q_j)$
- hence, minimizing $D_{KL}(p,q)$ corresponds to minimizing $H(p,q)$

(Zhang et al, 2020)

# Model learning: Sequential or batch mode

## Sequential mode

- on line (example-by-example), stochastic
- able to track small changes in training data
- easier to implement, requires less local storage
- difficult to establish theoretical conditions for convergence

## Batch mode

- adaptation performed at the end of each epoch, deterministic
- provides an accurate estimate of gradient vector
- parallelization possible

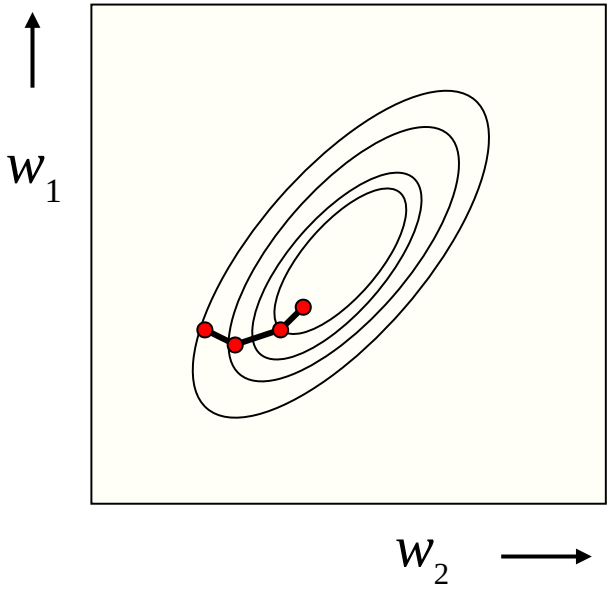
$$E_{av} = 1/(2N) \sum^{N}_{p=0} (d^{(p)} - y^{(p)})^2 \qquad \Delta w_i \propto -\partial E_{av}(t)/\partial w_i$$

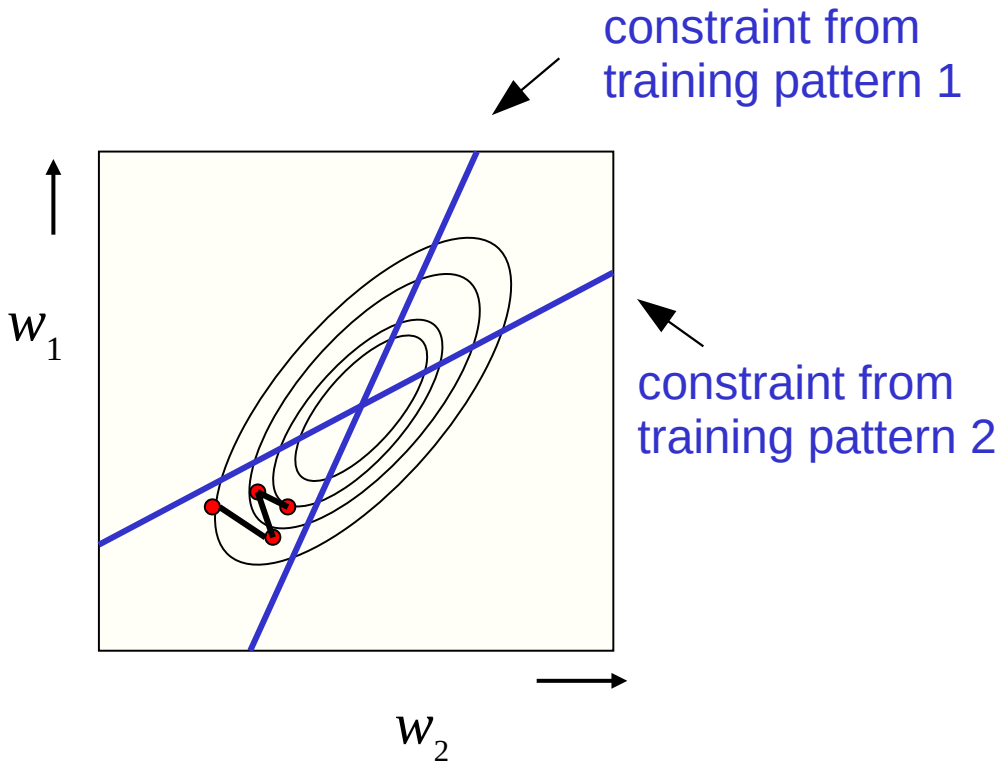
Mini-batches – best of two “worlds” (typical minibatch size 50–256)

# Online versus batch learning in param. space

Batch learning - does steepest descent on the error surface

Online learning - zig-zags around the direction of steepest descent



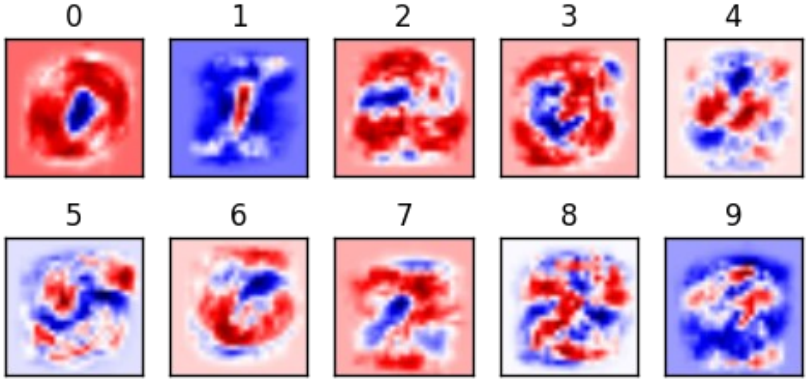



# Example: training a single-layer perceptron

- MNIST data set, 10 classes of hand-written digits, images 28x28
- Accuracy 90+ %, check also http://yann.lecun.com/exdb/mnist/

Input examples

Weights after training




https://www.oreilly.com/content/not-another-mnist-tutorial-with-tensorflow/

# Summary

- Linear models – studied in 1970s but concepts still useful
- Single-layer models are linear in parameters
- Hence, analytic solutions possible (via pseudoinverse)
- Tasks: (self-supervised) autoassociation, (supervised) regression, classification
- General Inverse model (projection to linear manifold)
- Data classification – soft assignment (softmax)
- Cross-entropy error used for classification
- Link to information theory (via entropy)
- Batch vs online learning