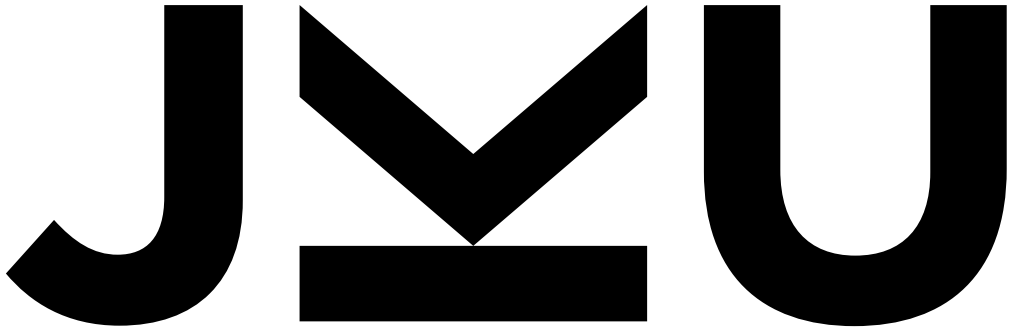
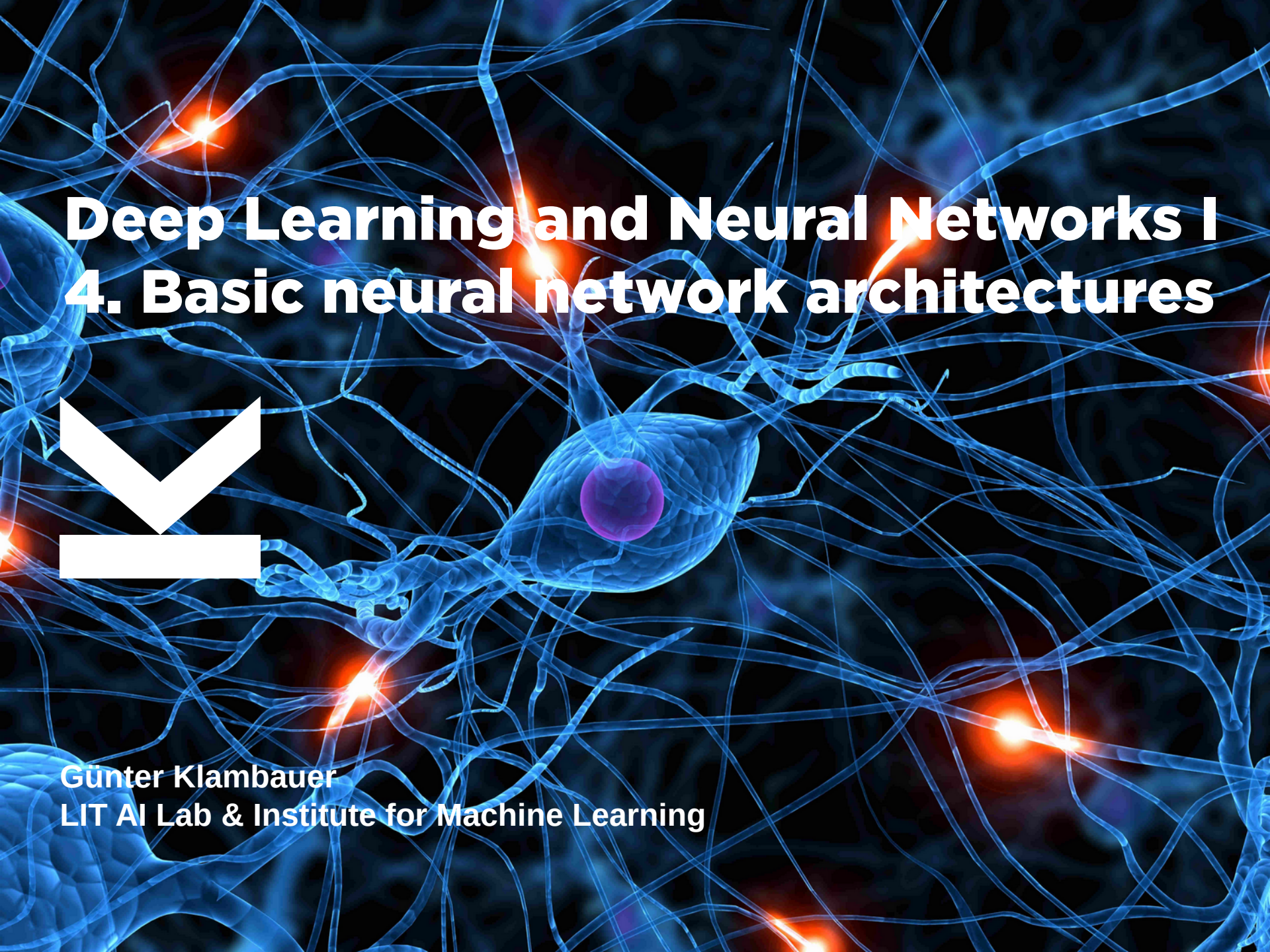


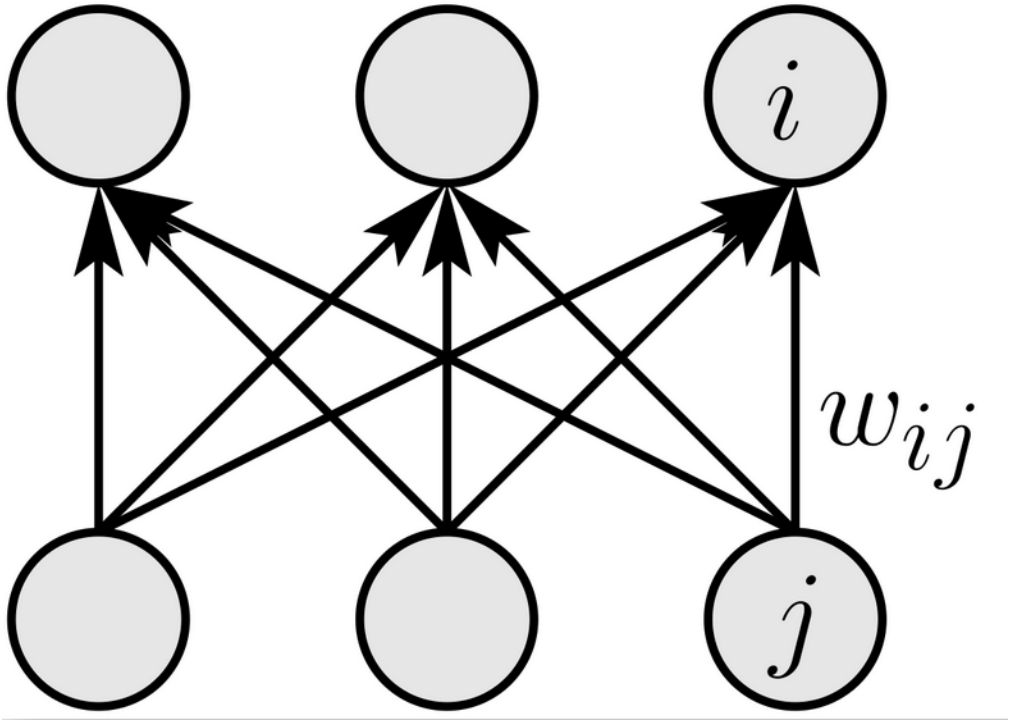
JOHANNES KEPLER
UNIVERSITY LINZ



Deep Learning and Neural Networks I
4. Basic neural network architectures

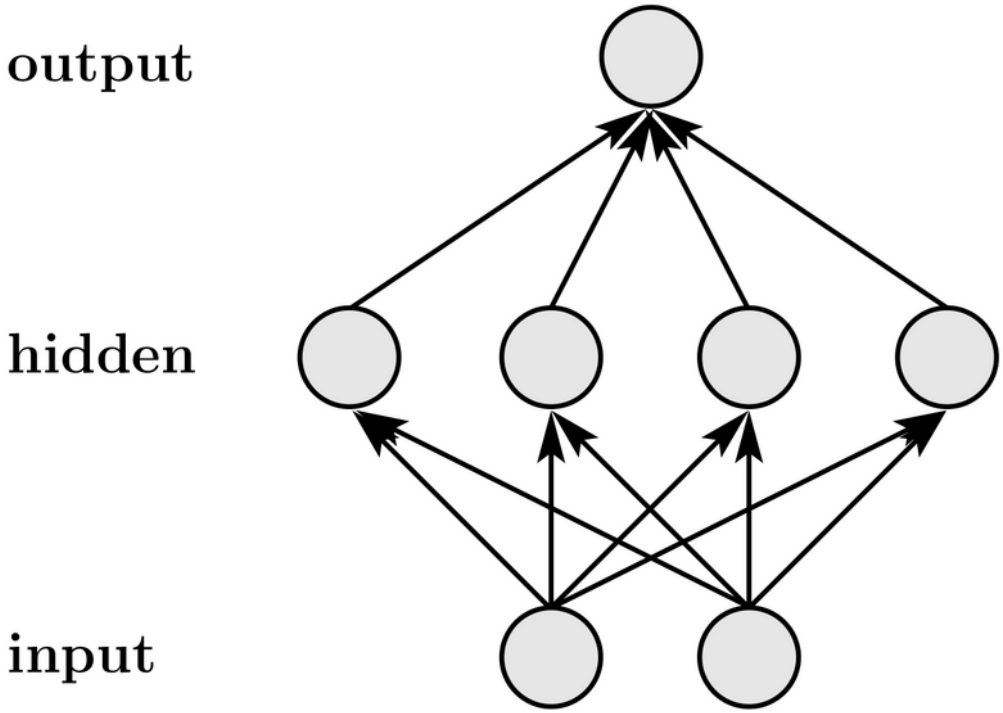
Günter Klambauer
LIT AI Lab & Institute for Machine Learning

Neural network with adaptive weights



Artificial neural networks: units and weights. The weight w_{ij} gives the weight, connection strength, or synaptic weight from unit j to unit i

Artifical neural networks: 3-layer network



- Artificial neural networks: a 3-layered net with an input, hidden, and output layer

Data for supervised learning: regression

$\boldsymbol{X}$			$\boldsymbol{y}$
$\boldsymbol{x}^1$	0.13	0.03	0.35
$\boldsymbol{x}^2$	*0.15*	0.14	0.57
	0.79	0.19	0.87
	0.48	0.28	1.21
	0.93	0.43	0.48
$\boldsymbol{x}^n$	0.43	0.41	0.70
	0.62	0.63	-0.44
	0.69	0.69	-1.05
	0.19	0.79	-1.29
	0.23	0.85	-1.11
$\boldsymbol{x}^N$	0.11	0.99	0.32

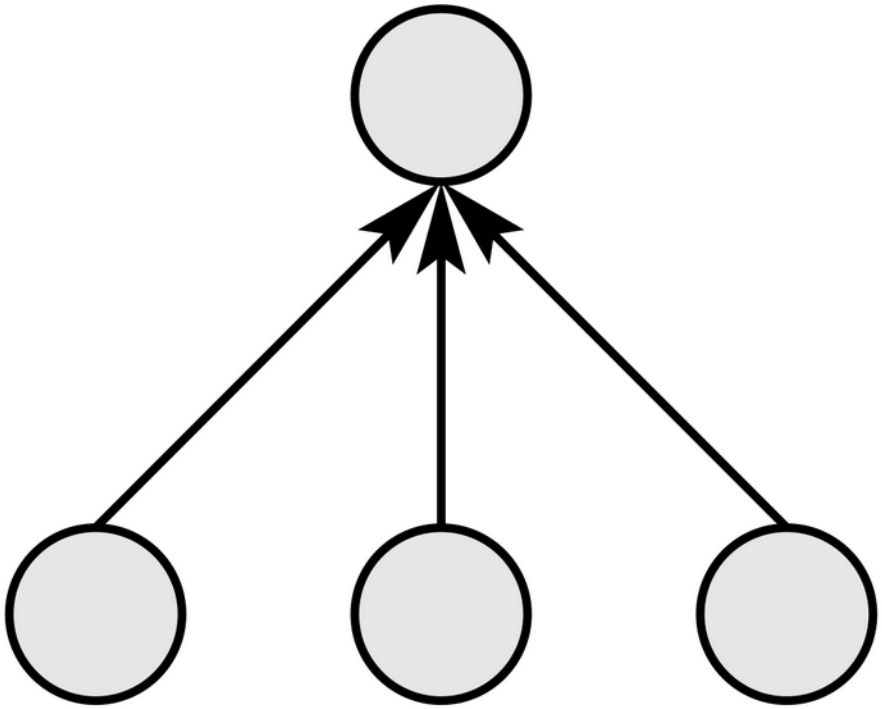
- Our supervised data set is:
  - Samples are rows in the data matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N)$$

  - Features are columns of $\boldsymbol{X}$:
    - Single features,e.g.: $x_{21} = 0.15$
    - Feature vector, e.g.: $\boldsymbol{x}_{.1}$
  - Scalar label for each data point:

$$\boldsymbol{y} = (y^1, \ldots, y^N)^T$$

Artificial neural networks: linear neuron



$$a = \boldsymbol{w}^T \boldsymbol{x}$$

$\boldsymbol{w}$

$\boldsymbol{x}$

- Inputs directly connected to output
- Input: $\boldsymbol{x}$ adaptive weights: $\boldsymbol{w}$ output (activation): a

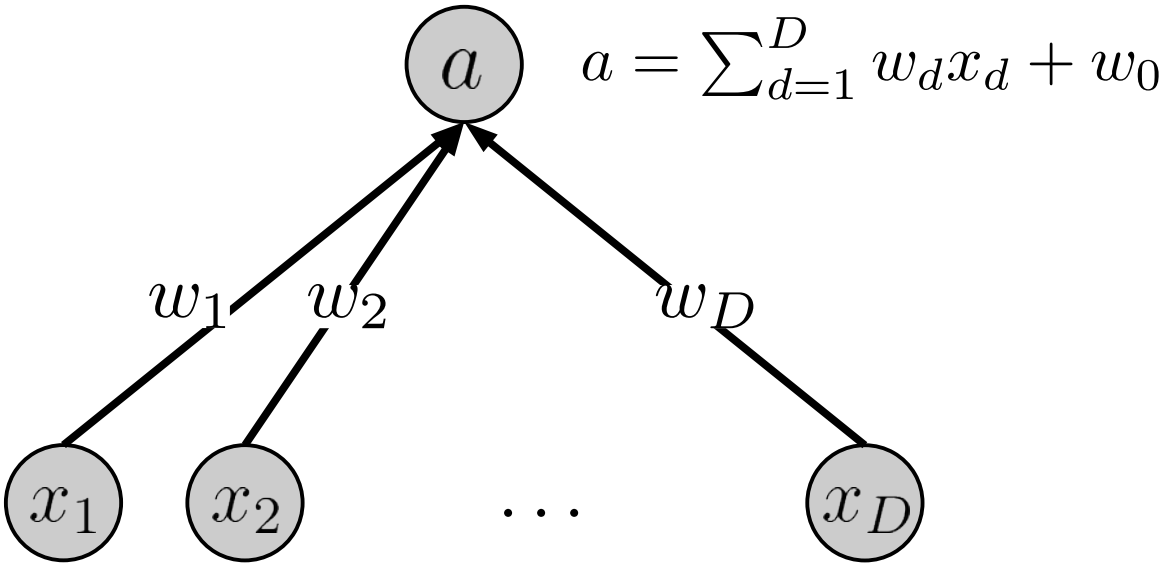
Overview

- 4. Basic neural network architectures
  - 4.1 Linear model: Neuron
  - 4.2 Perceptron
  - 4.3 Linear neuron with sigmoid activation: logistic regression
  - 4.4 Linear neuron with softmax activation: softmax regression

4.1 Linear neural network: model

4.1 Linear neural network: model

$$g(\boldsymbol{x}; \boldsymbol{w}) = a = \boldsymbol{w}^T \boldsymbol{x}$$



- Corresponds to a linear model as used for linear regression

Linear neural network: additive noise

- We assume that the target variable y is given by a deterministic function $g(\boldsymbol{x}, \boldsymbol{w})$ with additive Gaussian noise such that

$$y = g(\boldsymbol{x}, \ \boldsymbol{w}) + \epsilon,$$

  where ϵ follows $\epsilon \sim \mathcal{N}(0, \sigma)$.

- Thus we have the distribution of the target variable

$$p(y \mid \boldsymbol{x}, \boldsymbol{w}, \sigma) = \mathcal{N}(y \mid g(\boldsymbol{x}, \boldsymbol{w}), \sigma)$$

- As a prediction we could give the conditional mean

$$\mathbb{E}(y \mid \boldsymbol{x}) = \int y p(y \mid \boldsymbol{x}) \mathrm{d}y = g(\boldsymbol{x}, \boldsymbol{w})$$

Linear neural network: likelihood

- Now consider a dataset of inputs $\boldsymbol{X} = \{\boldsymbol{x}^1, \dots \boldsymbol{x}^N\}$ with corresponding target values (labels) $y^1, \dots, y^N$

  Assuming that the data points are drawn independently from each other, we obtain the following likelihood function:

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(y^n \mid g(\boldsymbol{x}^n, \boldsymbol{w}), \sigma)$$

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(y^n \mid \boldsymbol{w}^T \boldsymbol{x}^n, \sigma)$$

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y^n - \boldsymbol{w}^T \boldsymbol{x}^n)^2}{2\sigma^2}}$$

Linear NN: residual, loss function

- We take the logarithm of the likelihood function which does not change the location of the minimum:

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y^n - \boldsymbol{w}^T \boldsymbol{x}^n)^2}{2\sigma^2}}$$

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{\sigma^2} S(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y})$$

$$S(\boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{2} \sum_{n=1}^{N} (\underbrace{y^n - \boldsymbol{w}^T \boldsymbol{x}^n}_{:=\epsilon^n})^2$$

- The function: $L(y, g(\boldsymbol{x}; \boldsymbol{w})) = \frac{1}{2}(y - g(\boldsymbol{x}; \boldsymbol{w}))^2$ is called *squared loss*

Linear NN: candidate parameters

- Assume we have a candidate parameter vector $\tilde{\boldsymbol{w}}$:

$$S(\tilde{\boldsymbol{w}}) = \frac{1}{2}\sum_{n=1}^{N}(\epsilon^n)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y^n - \tilde{\boldsymbol{w}}^T\boldsymbol{x}^n\right)^2 = \frac{1}{2}\left(\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}}\right)^T\left(\boldsymbol{y} - \boldsymbol{X}\ \tilde{\boldsymbol{w}}\right).$$

  (we have dropped the dependency on data and labels)

- We can determine the *sum of residuals*
  - Low sum of residuals is equivalent to high likelihood
- **Learning:**
  - We can check different candidate parameter vectors and determine $S(\tilde{\boldsymbol{w}})$
  - Maximizing the likelihood of our data (under the assumption of Gaussian noise), is equivalent to minimizing the squared error

Linear NN: learning

- Let us *learn* the prediction task, i.e., find the parameters that maximize the likelihood or minimize the quadratic loss on the given data (*training set)*:

$$\hat{\boldsymbol{w}} = \arg\min_{\tilde{\boldsymbol{w}}} S(\tilde{\boldsymbol{w}})$$

- This approach is called *empirical risk minimization* (ERM)

$$R_{\text{emp}}(\tilde{\boldsymbol{w}}, \boldsymbol{X}, \boldsymbol{y}) = S(\tilde{\boldsymbol{w}}).$$

- Does this problem have a solution? How many?
- How could we find a solution?

Linear NN: learning

- Let us *learn* the prediction task, i.e., find the parameters that maximize the likelihood or minimize the quadratic loss on the given data (*training set)*:

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}S(\boldsymbol{w})$$

$$S(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}(y^n - \boldsymbol{w}^T\boldsymbol{x}^n)^2$$

$$\nabla_{\boldsymbol{w}} \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}, \sigma) = \frac{1}{\sigma^2}\sum_{n=1}^{N}(\boldsymbol{w}^T\boldsymbol{x}^n - y^n)\boldsymbol{x}^n$$

Linear regression: closed form solution

- Let us *learn* the prediction task, i.e., find the parameters that maximize the likelihood or minimize the quadratic loss:

$$\mathbf{0} = \frac{1}{\sigma^2} \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}^n - y^n) \boldsymbol{x}^n$$

$$\mathbf{0} = \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{w} - \boldsymbol{X}^T \boldsymbol{y}$$

$$\boldsymbol{X}^T \boldsymbol{y} = \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{w}$$

$$\hat{\boldsymbol{w}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

Linear NN: learning

- This is the solution to our optimization problem

$$\hat{\boldsymbol{w}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

  - It is a maximum-likelihood estimator
  - It minimizes the least-squares problem
  - The quantity

$$\boldsymbol{X}^+ = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T$$

    is called Moore-Penrose pseudo-inverse.
  - Note that we have used the assumption that $\boldsymbol{X}^T \boldsymbol{X}$ has full rank.
  - $\frac{1}{n} \boldsymbol{X}^T \boldsymbol{X}$ is an estimate of the covariance matrix (centered data)

Linear NN: gradient descent

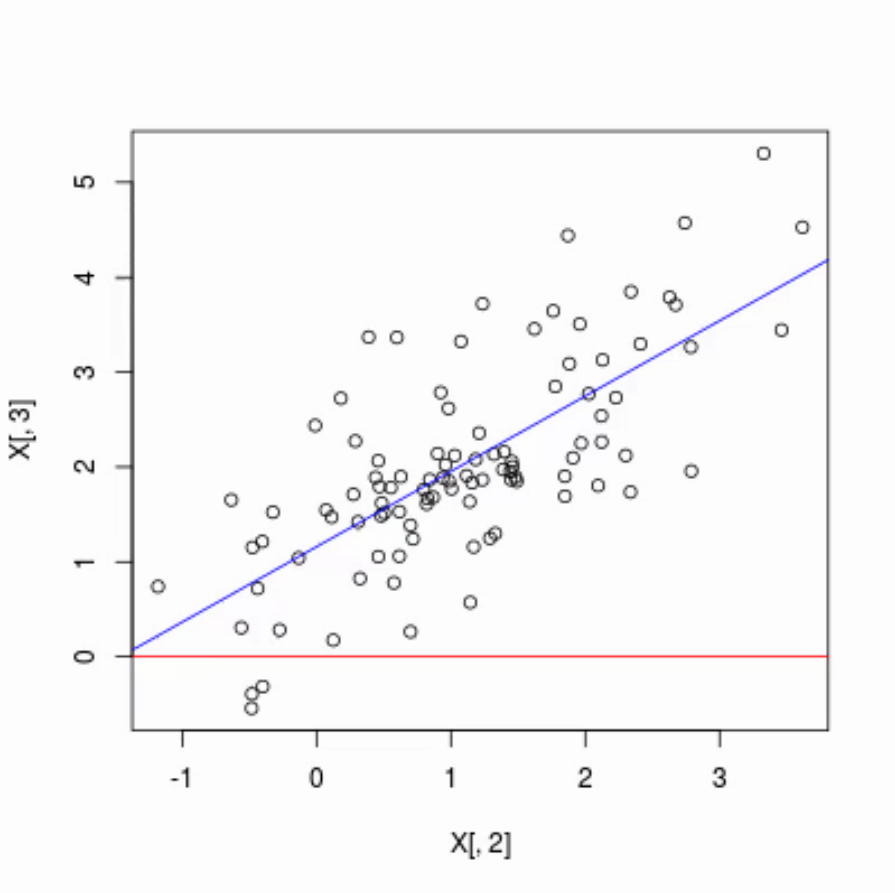
- We seek a $\hat{\boldsymbol{w}}$ that minimizes $S(\boldsymbol{w})$. To do so, we could also use a search algorithm that uses an initial guess for $\boldsymbol{w}$ and gradually improves it:

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \nabla_{\boldsymbol{w}} S(\boldsymbol{w}) \mid_{\boldsymbol{w}^{\text{old}}}$$

- This algorithm is called *gradient descent*
  - The choice of the learning rate η is critical for convergence
  - Works well for the least squared problem because it is convex
- We already know the expression for the gradient:

$$\nabla_{\boldsymbol{w}} S(\boldsymbol{w}) = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}^n - y^n) \boldsymbol{x}^n$$

Linear NN: gradient descent



Linear NN: stochastic grad. desc.

- We note that the *scaled gradient*

$$\nabla_{\boldsymbol{w}} S(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}^n - y^n) \boldsymbol{x}^n$$

  is a mean over data points.

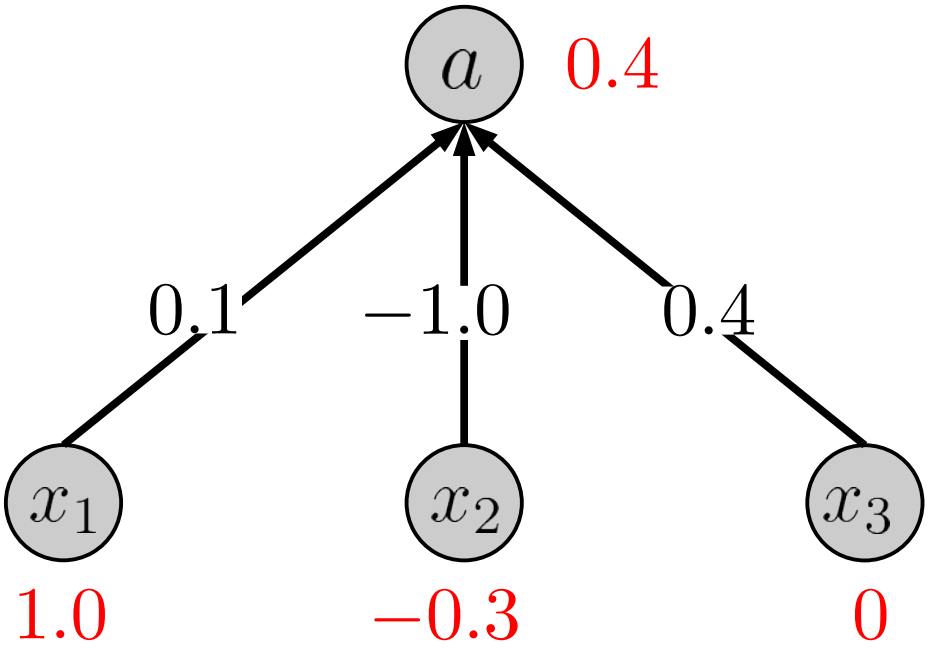
- Thus we can approximate the *scaled gradient* on a subset of datapoints, called *mini-batch*:

$$\nabla_{\boldsymbol{w}} S(\boldsymbol{w}) \approx \frac{1}{|\mathcal{B}|} \sum_{b=1}^{|\mathcal{B}|} (\boldsymbol{w}^T \boldsymbol{x}^{n_b} - y^{n_b}) \boldsymbol{x}^{n_b}$$

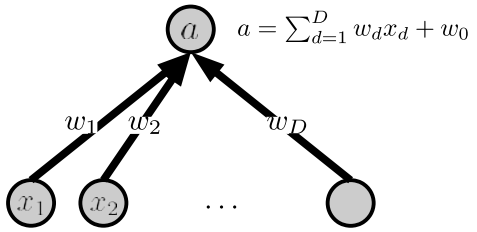
$$\mathcal{B} = \{\boldsymbol{x}_{n_1}, \ldots, \boldsymbol{x}_{n_B}\} \subset \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$$

Linear neural network: Trained network

$$g(\boldsymbol{x}^{\text{new}}; \boldsymbol{w}) = a = \boldsymbol{w}^T \boldsymbol{x}^{\text{new}}$$



Summary



- Architecture of linear neural networks
- Objective function: sum of residuals, empirical error; motivation by maximum-likelihood
- Learning: minimization of objective function
  - Different ways to minimize
  - Properties of the solution
- Goal: Trained network

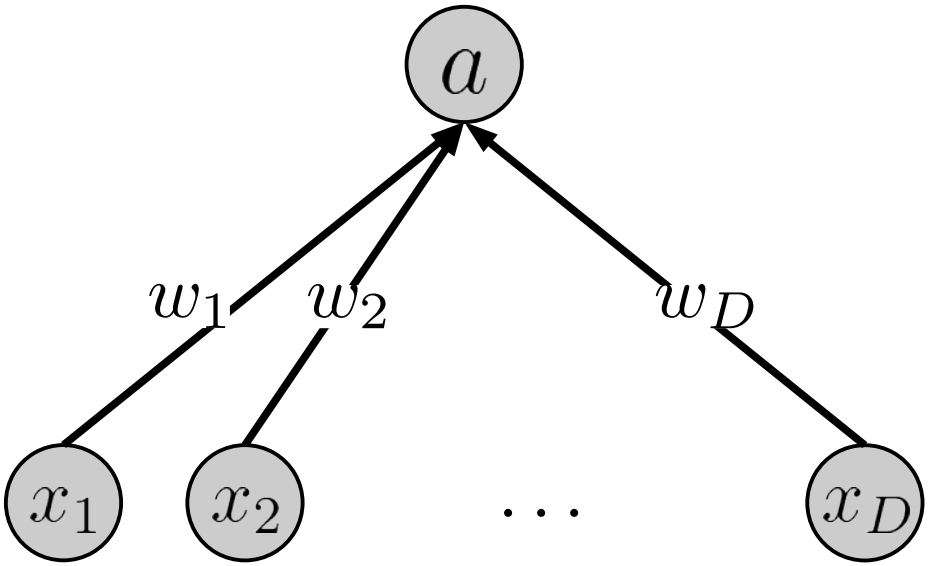
Perceptron: classification task

- Let us assume that the targets/labels only take the values -1 and 1 (classes)
- This is a supervised classification tasks
- Examples for classification tasks
  - Diseased vs healthy
  - Spam vs normal email
  - Win or loss
  - …
- What can we do in this situation? Should we just use LR as before?

X			y
x^1	0.13	0.03	1
x^2	0.15	0.14	1
	0.79	0.19	-1
	0.48	0.28	1
	0.93	0.43	-1
	0.43	0.41	-1
	0.62	0.63	1
	0.69	0.69	1
	0.19	0.79	1
	0.23	0.85	-1
x^N	0.11	0.99	-1

4.2 Perceptron

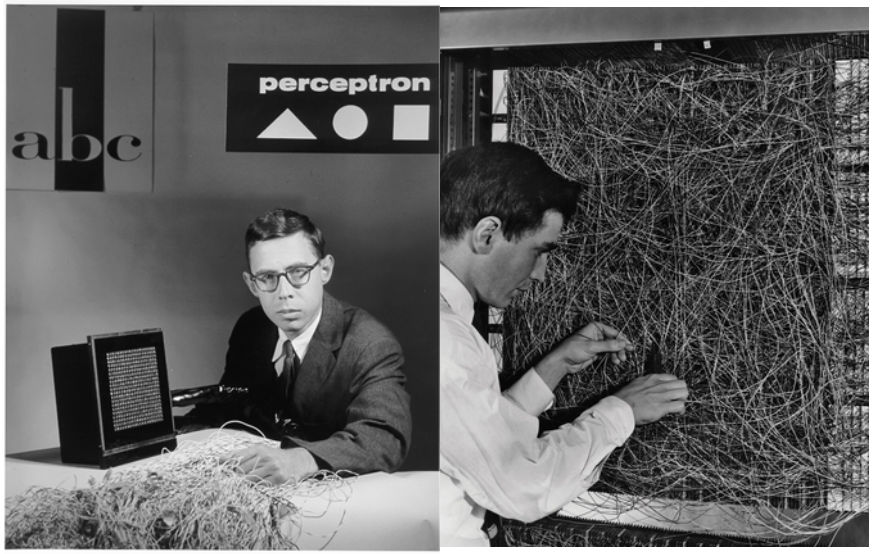
$$g(\boldsymbol{x}; \boldsymbol{w}) = a = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$$



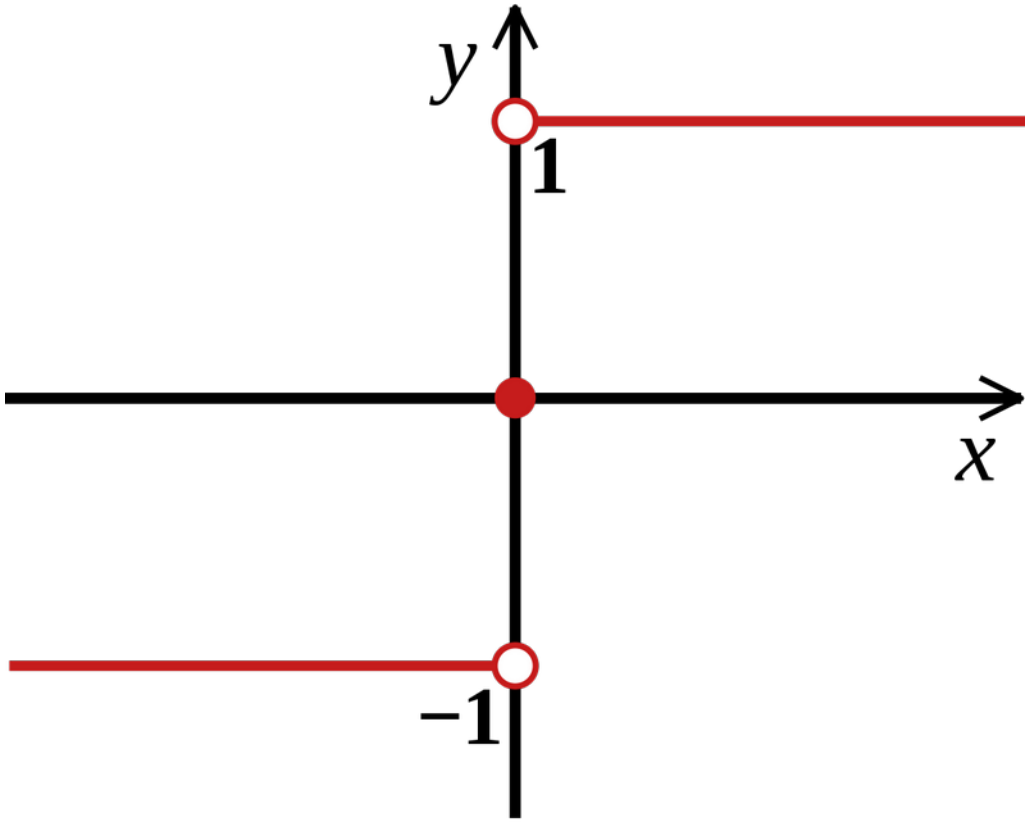
- Similar to linear neuron with linear activation

Reminder: Perceptron

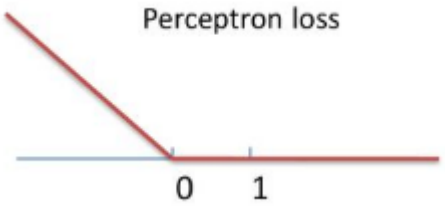
- **1958:** Frank Rosenblatt
  - Perceptron
  - Electronic device with adaptable connections
  - 400 photocells as input neurons
  - Image recognition
  - Ability to learn



Perceptron: sign function



Perceptron: loss function



- Loss function: “perceptron loss”

$$L(y, \text{sign}(\boldsymbol{w}^T \boldsymbol{x})) = \begin{cases} 0, & \text{if } y \text{ sign}(\boldsymbol{w}^T \boldsymbol{x}) \geq 0 \\ y \, \boldsymbol{w}^T \boldsymbol{x} & \text{else} \end{cases}$$

- Empirical error

$$R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w}) = - \sum_{n=1; a^n y^n = -1}^{N} y^n \boldsymbol{w}^T \boldsymbol{x}^n$$

- Note that the sum runs over all samples that are misclassified

Perceptron: learning rule

- Derivative with respect to weights:

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w}) = \nabla_{\boldsymbol{w}} - \sum_{n=1; a^n y^n = -1}^{N} y^n \boldsymbol{w}^T \boldsymbol{x}^n = - \sum_{n=1; a^n y^n = -1}^{N} y^n \boldsymbol{x}^n$$

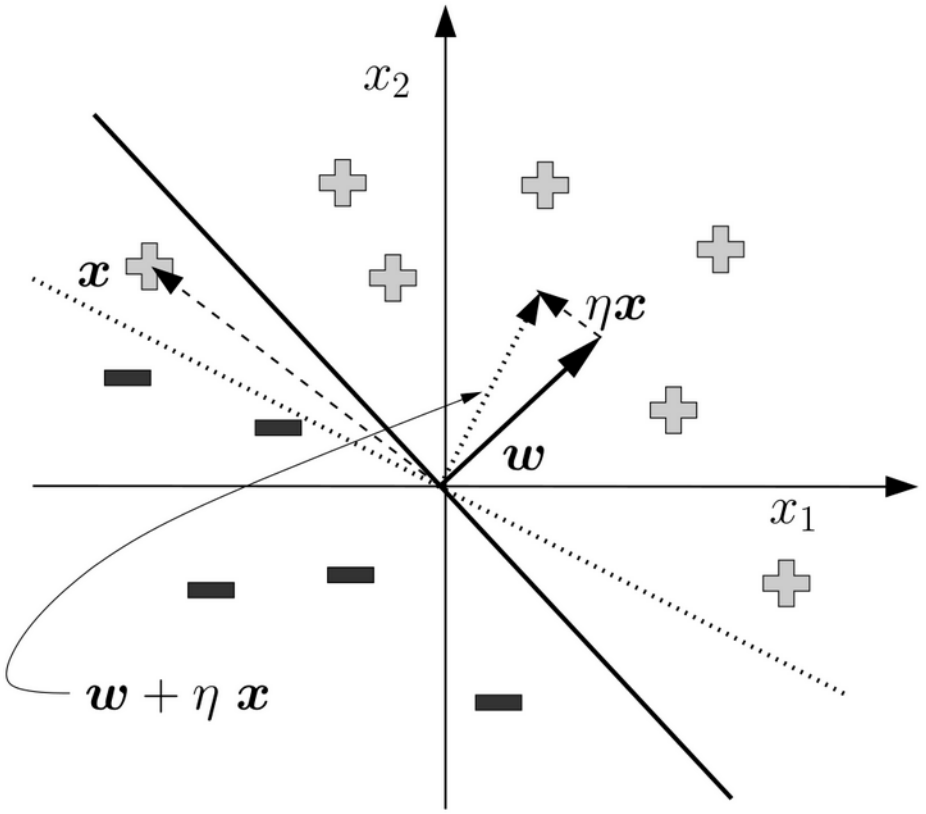
- Weight update rule:

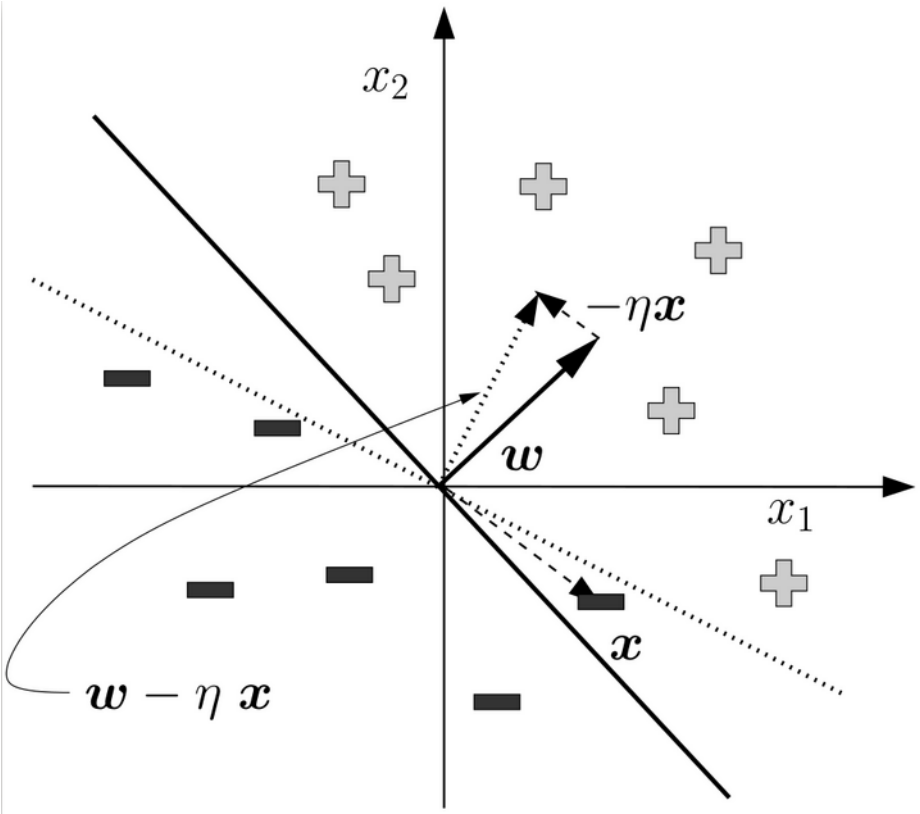
$$\Delta \boldsymbol{w} = \eta \sum_{n=1; a^n y^n = -1}^{N} y^n \boldsymbol{x}^n \qquad \text{if} \quad ay = -1 : \Delta \boldsymbol{w} = \eta y \boldsymbol{x}.$$

whole data set single sample

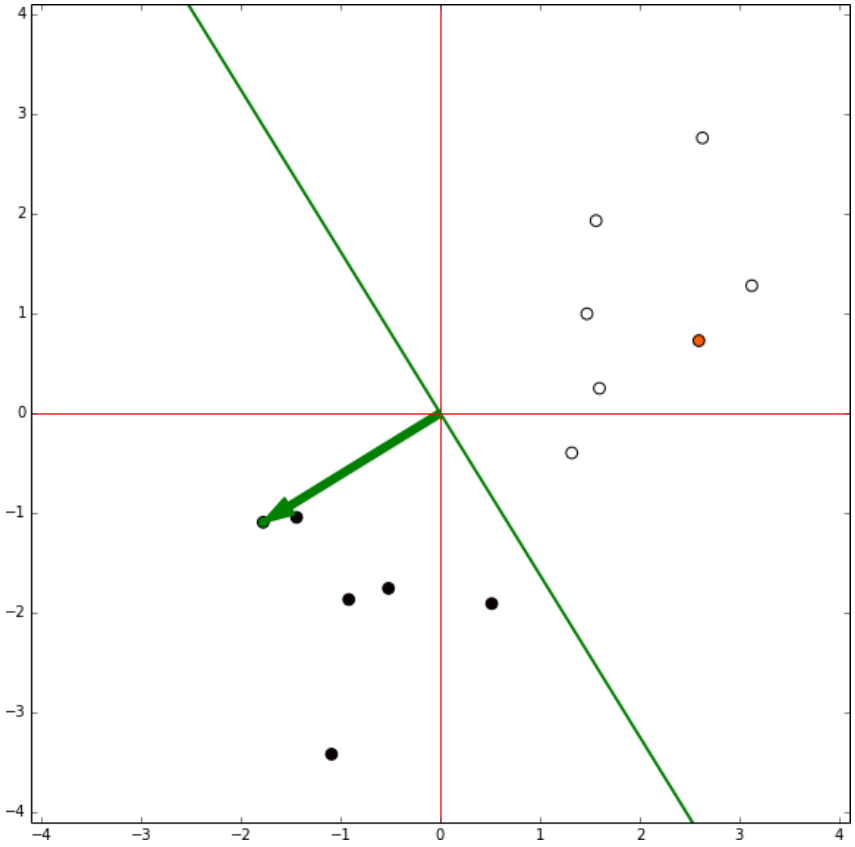
- Unique solution? Convergence?

Perceptron: Interpretation of learning rule

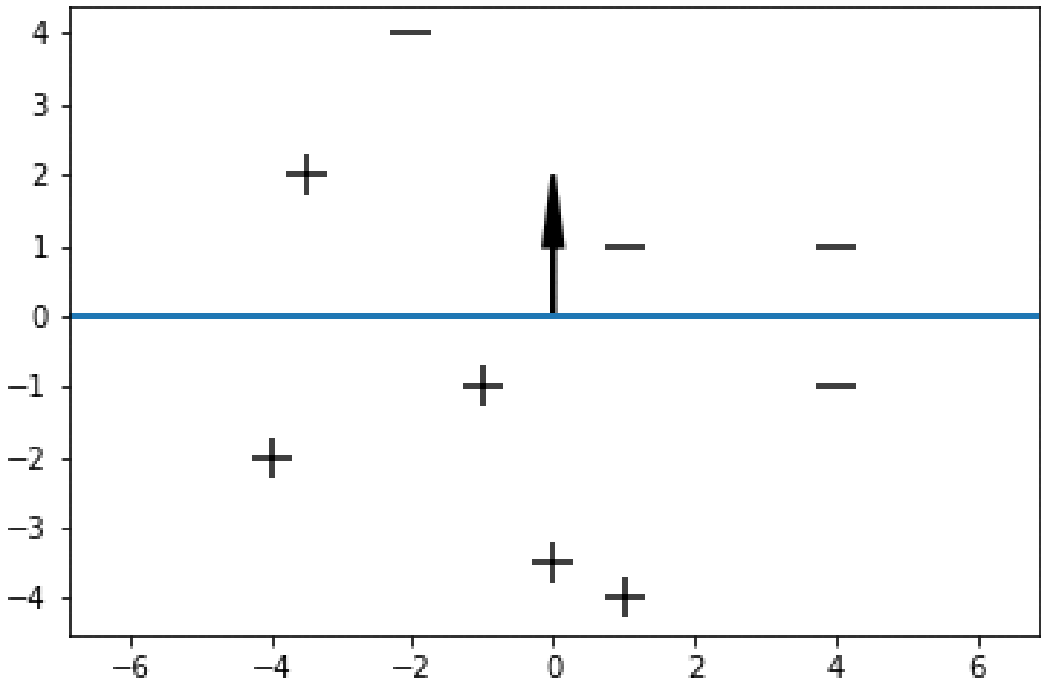




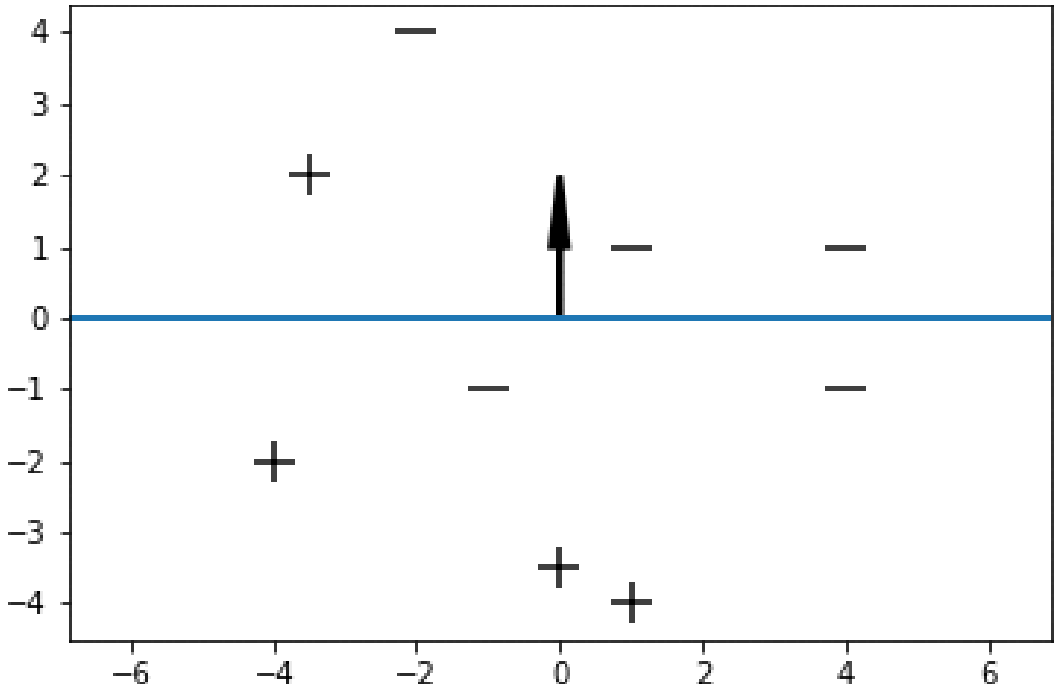
Perceptron: Interpretation of learning rule



Perceptron: separable data



Perceptron: non-separable data



Perceptron Convergence Theorem for linearly separable data (1/3)

- Since the data is linearly separable, there exists $\hat{\boldsymbol{w}}$:

$$\hat{\boldsymbol{w}}^T \boldsymbol{x}^n y^n > 0 \quad \forall n$$

- We assume that we start weights with the zero vector and update according to

$$\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} + \boldsymbol{x}^n y^n$$

  if the data point is mis-classified.

- Then the weight vector is: $\boldsymbol{w} = \sum_n \tau^n \boldsymbol{x}^n y^n$ $\boxed{\tau^n}$

- Then we multiply:

$$\hat{\boldsymbol{w}}^T \boldsymbol{w} = \sum_n \tau^n \hat{\boldsymbol{w}}^T \boldsymbol{x}^n y^n \geq \tau \min_n (\hat{\boldsymbol{w}}^T \boldsymbol{x}^n y^n)$$

Perceptron Convergence Theorem for linearly separable data (2/3)

- We remember: $\boldsymbol{w}^{\tau+1} = \boldsymbol{w}^{\tau} + \boldsymbol{x}^n y^n$ and we have

$$\begin{aligned}\|\boldsymbol{w}^{(\tau+1)}\|^2 &= \|\boldsymbol{w}^{(\tau)}\|^2 + \|\boldsymbol{x}^n\|^2 (y^n)^2 + 2\boldsymbol{w}^{(\tau)^T}\boldsymbol{x}^n y^n \\ &\leq \|\boldsymbol{w}^{(\tau)}\|^2 + \|\boldsymbol{x}^n\|^2 (y^n)^2\end{aligned}$$

  because the last term is negative.

- The change of weights satisfies:

$$\Delta\|\boldsymbol{w}\|^2 = \|\boldsymbol{w}^{(\tau+1)}\|^2 - \|\boldsymbol{w}^{(\tau)}\|^2 \leq \|\boldsymbol{x}\|^2_{\max}$$

- After τ updates we have: $\|\boldsymbol{w}\|^2 \leq \tau\|\boldsymbol{x}\|^2_{\max}$

Perceptron Convergence Theorem for linearly separable data (3/3)

- Thus the function $\hat{\boldsymbol{w}}^T \boldsymbol{w}$ is bounded below by a function that grows linearly with tau

$$\hat{\boldsymbol{w}}^T \boldsymbol{w} \geq \tau \min_n (\hat{\boldsymbol{w}}^T \boldsymbol{x}^n y^n)$$

- The length of the weight vector increases no faster than with $\tau^{1/2}$

$$\|\boldsymbol{w}\|^2 \leq \tau \|\boldsymbol{x}\|^2_{\max}$$

- Therefore, since $\hat{\boldsymbol{w}}$ is fixed, for sufficiently large time, the results would incompatible. Hence the algorithm must converge in a finite number of time steps. *qed*

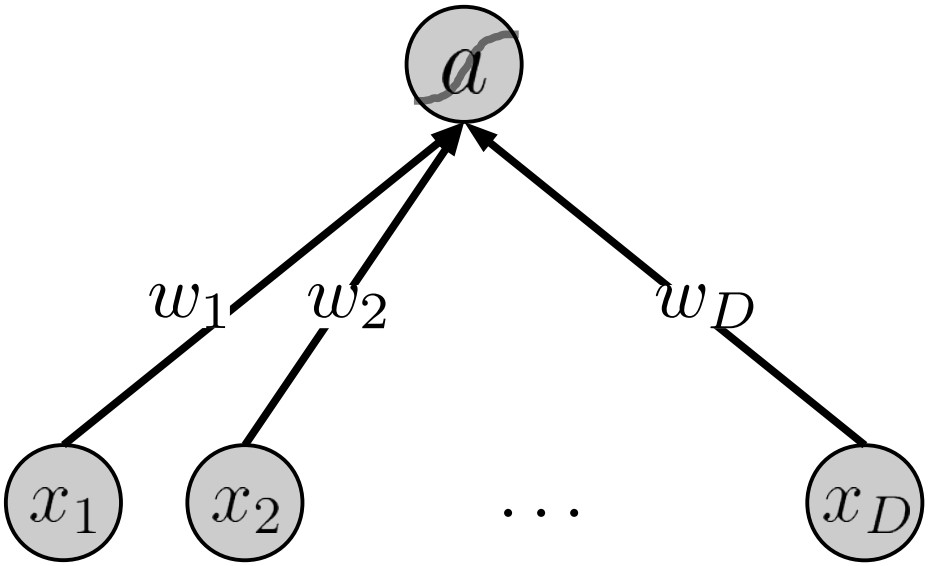
Logistic neuron: classification

- Let us assume that the targets/labels only take the values 0 and 1 (classes)
- This is a supervised classification tasks
- Examples for classification tasks
  - Diseased vs healthy
  - Spam vs normal email
  - Win or loss
  - …
- What can we do in this situation? Should we just use LR as before?

X			y
x^1	0.13	0.03	1
x^2	0.15	0.14	1
	0.79	0.19	0
	0.48	0.28	1
	0.93	0.43	0
	0.43	0.41	0
	0.62	0.63	1
	0.69	0.69	1
	0.19	0.79	1
	0.23	0.85	0
x^n	0.11	0.99	0

4.3 Logistic neuron: model

$$g(\boldsymbol{x}; \boldsymbol{w}) = a = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$



- Corresponds to a linear model as used for logistic regression

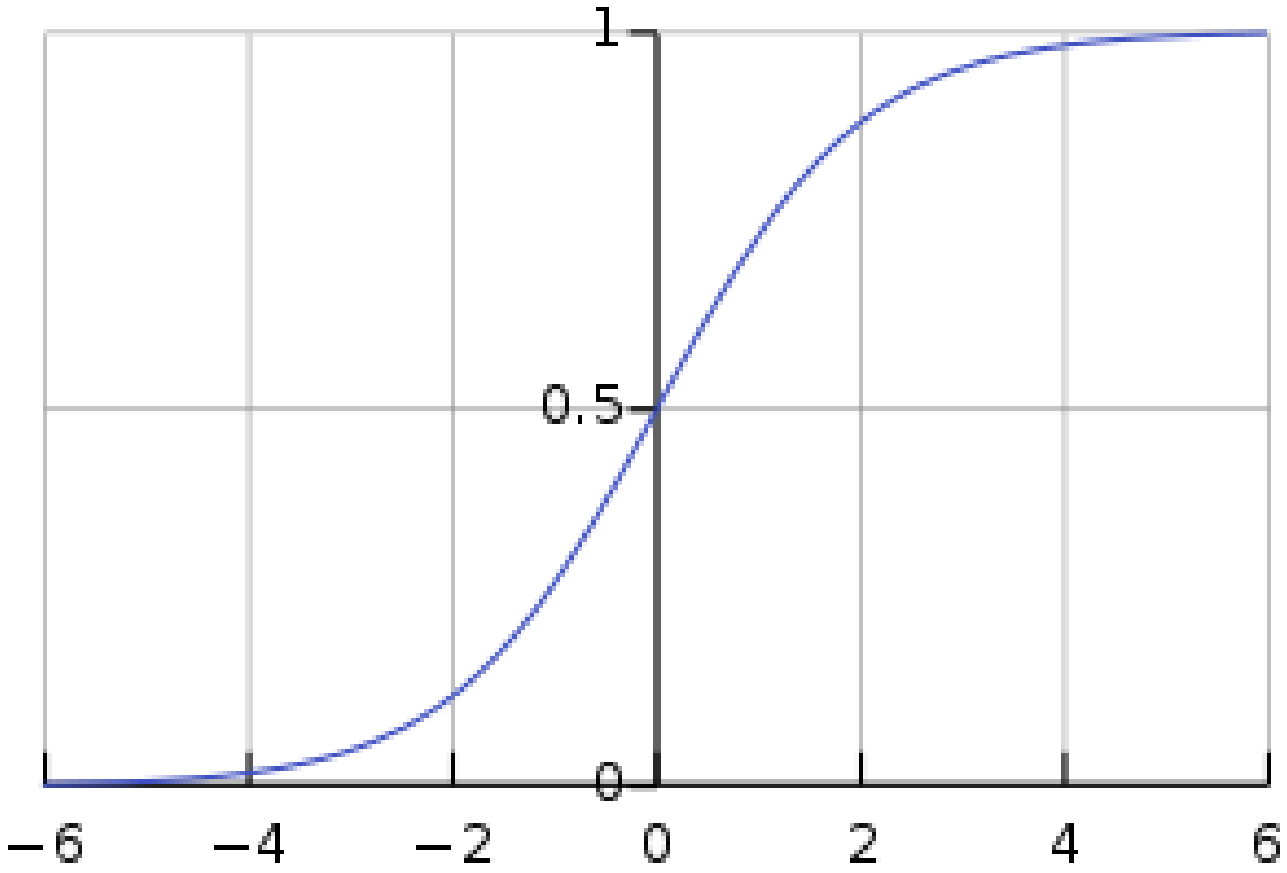
Linear neuron with sigmoid activation:

- The main idea is to fit a model of the following form

$$g(\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}} = \sigma(\boldsymbol{w}^T \boldsymbol{x}),$$

where σ is the logistic sigmoid function.

Logistic regression: The sigmoid function



Logistic regression

- Property of the sigmoid function:

$$\sigma'(x) = \frac{d}{dx}\frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Logistic regression: likelihood

- We aim at maximizing the likelihood again

$$p(y = 1 \mid \boldsymbol{x}; \boldsymbol{w}) = g(\boldsymbol{x}; \boldsymbol{w})$$

$$p(y = 0 \mid \boldsymbol{x}; \boldsymbol{w}) = 1 - g(\boldsymbol{x}; \boldsymbol{w})$$

- This can be written more compactly as:

$$p(y \mid \boldsymbol{x}; \boldsymbol{w}) = g(\boldsymbol{x}; \boldsymbol{w})^y (1 - g(\boldsymbol{x}; \boldsymbol{w}))^{(1-y)}$$

- Assuming that the data points are drawn independently, the likelihood is

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) = \prod_{n=1}^{N} g(\boldsymbol{x}^n; \boldsymbol{w})^{y^n} (1 - g(\boldsymbol{x}^n; \boldsymbol{w}))^{(1-y^n)}$$

Logistic regr: loss and error function

- The likelihood is

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) = \prod_{n=1}^{N} g(\boldsymbol{x}^n; \boldsymbol{w})^{y^n} (1 - g(\boldsymbol{x}^n; \boldsymbol{w}))^{(1-y^n)}$$

- The negative log likelihood is

$$-\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) = -\sum_{n=1}^{N} y^n \log g(\boldsymbol{x}^n; \boldsymbol{w}) + (1 - y^n) \log(1 - g(\boldsymbol{x}^n; \boldsymbol{w}))$$

- The function

$$L(y, g(\boldsymbol{x}; \boldsymbol{w})) = -(y \log g(\boldsymbol{x}; \boldsymbol{w}) + (1 - y) \log(1 - g(\boldsymbol{x}; \boldsymbol{w})))$$

  is called *binary cross-entropy*.

Logistic regression: learning

- One way to learn the model is again *gradient descent*

$$\nabla_{\boldsymbol{w}} - \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{w}) = \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w}) =$$

$$= \nabla_{\boldsymbol{w}} - \sum_{n=1}^{N} y^n \log \sigma(\boldsymbol{w}^T \boldsymbol{x}^n) + (1 - y^n) \log(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^n)) =$$

$$= - \sum_{n=1}^{N} \left(y^n \frac{1}{\sigma(\boldsymbol{w}^T \boldsymbol{x}^n)} - (1 - y^n) \frac{1}{(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^n))} \right) \nabla_{\boldsymbol{w}} \sigma(\boldsymbol{w}^T \boldsymbol{x}^n) =$$

$$= - \sum_{n=1}^{N} \left(y^n \frac{1}{\sigma(\boldsymbol{w}^T \boldsymbol{x}^n)} - (1 - y^n) \frac{1}{(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^n))} \right) \sigma(\boldsymbol{w}^T \boldsymbol{x}^n)(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^n)) \boldsymbol{x}^n =$$

$$= \sum_{n=1}^{N} \left(\sigma(\boldsymbol{w}^T \boldsymbol{x}^n) - y^n \right) \boldsymbol{x}^n$$

- Does that look familiar? Is it exactly the same as for linear regression?

Logistic regression: second derivative

- Gradient and Hessian for a single sample

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(y, \boldsymbol{x}, \boldsymbol{w}) = \left(\sigma(\boldsymbol{w}^T \boldsymbol{x}) - y\right) \boldsymbol{x}$$

$$\nabla^2_{\boldsymbol{w}} R_{\text{emp}}(y, \boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}))\boldsymbol{x}\boldsymbol{x}^T$$

Note: $a = g(\boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$

- Gradient and Hessian for multiple samples

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w}) = \sum_{n=1}^{N} \left(\sigma(\boldsymbol{w}^T \boldsymbol{x}^n) - y^n\right) \boldsymbol{x}^n = \boldsymbol{X}^T(\boldsymbol{a} - \boldsymbol{y})$$

$$\nabla^2_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}^n)(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^n))\boldsymbol{x}^n \boldsymbol{x}^{nT} = \boldsymbol{X}^T \boldsymbol{R} \boldsymbol{X}$$

where $\boldsymbol{R}$ is a diagonal matrix with $\sigma(\boldsymbol{w}^T \boldsymbol{x}^n)(1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}^n))$ as entries

Logistic regression: learning

- Again we can can view the negative log likelihood as data driven error and use *gradient descent:*

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w})|_{\boldsymbol{w}^{\text{old}}}$$

- Also *stochastic gradient descent* could be used:

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \tilde{\nabla}_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w})|_{\boldsymbol{w}^{\text{old}}}$$

- Typically, *Iteratively Reweighted Least-Squares* is used:

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w})|_{\boldsymbol{w}^{\text{old}}}$$

  where $\boldsymbol{H}$ is the Hessian matrix, i.e. the matrix of second derivatives

Intermezzo: Optimization

- Generic optimization problem:

$$\text{Let } x \in \mathbb{R}^n, f : \mathbb{R}^n \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}^m, h : \mathbb{R}^n \to \mathbb{R}. \text{ Find}$$

$$\min_x f(x)$$

$$\text{s.t.} \quad g(x) \leq 0, \; h(x) = 0$$

This would be our weights that we adapt!

This would by our empirical error that we aim to minimize

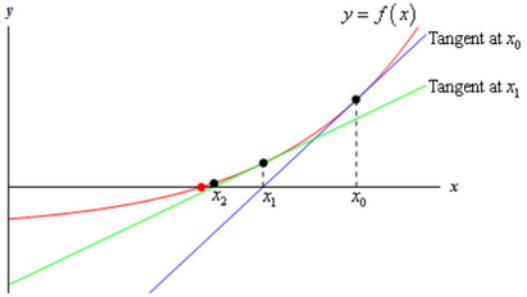
CONSTRAINTS
Typically, we do not have constraints

- Notation above is general and not the specific ML notation of this course!
- Optimization types:
  - Blackbox optimization: we only have access to $f(x)$
  - Gradient-based optimization: we have access to $f(x), \nabla f(x)$
  - Second order optimization: we have access to $f(x), \nabla f(x), \nabla^2 f(x)$

Intermezzo: Optimization Newton's method

- Finding roots (zeros) of $f(x)$:

$$x^{\text{new}} = x^{\text{old}} - \frac{f(x^{\text{old}})}{f'(x^{\text{old}})}$$



- Finding optima of $f(x)$ in 1D:

$$x^{\text{new}} = x^{\text{old}} - \frac{f'(x^{\text{old}})}{f''(x^{\text{old}})}$$

- Finding optima of $f(x)$ in n-dim:

$$x^{\text{new}} = x^{\text{old}} - \nabla^2 f(x^{\text{old}})^{-1} \nabla f(x^{\text{old}})$$

Intermezzo: Optimization Newton's method

- Taylor approximation:

$$f(x) \approx f(x_0) + (x - x_0) f'(x_0)$$

- Setting to zero and solving:

$$f(x_0) + (x - x_0) f'(x_0) = 0 \Rightarrow$$

$$x = \frac{-f(x_0) + x_0 f'(x_0)}{f'(x_0)} = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- Analogously for first derivative and multiple dimensions

Second order methods for linear and logistic regression

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \left(\nabla_{\boldsymbol{w}}^2 R(\boldsymbol{w}^{\text{old}})\right)^{-1} \nabla_{\boldsymbol{w}} R(\boldsymbol{w}^{\text{old}})$$

- Linear regression:

$$\nabla_w R(\boldsymbol{w}) = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^T\boldsymbol{y} \qquad \nabla_w^2 R(\boldsymbol{w}) = \boldsymbol{X}^T\boldsymbol{X}$$

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - (\boldsymbol{X}^T\boldsymbol{X})^{-1}(\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}^{\text{old}} - \boldsymbol{X}^T\boldsymbol{y})$$

$$\boldsymbol{w}^{\text{new}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

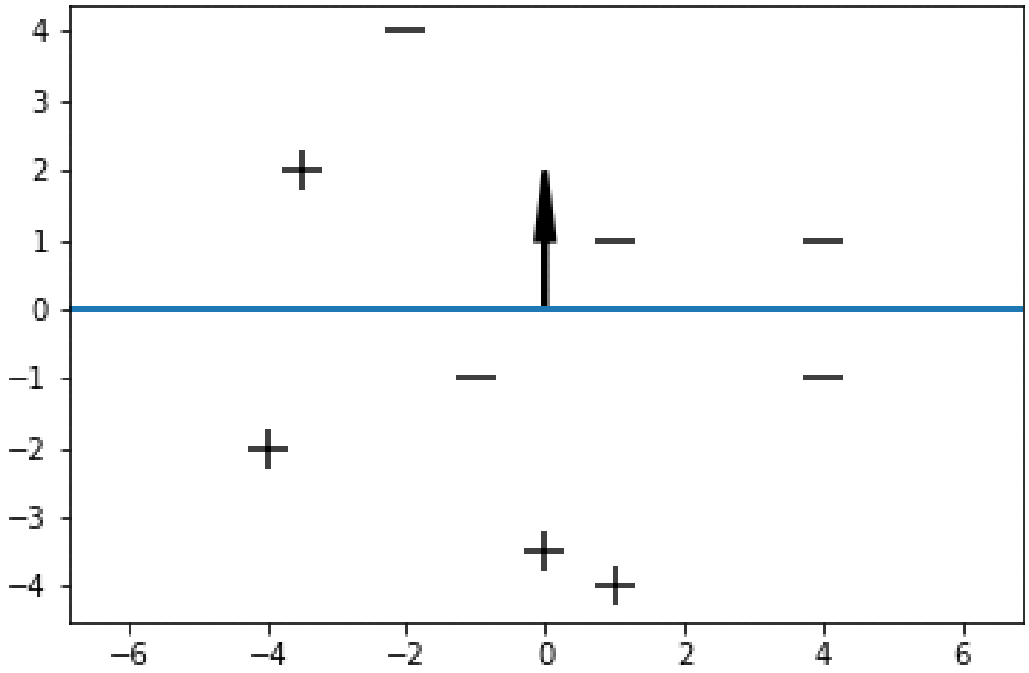
- Logistic regression:

$$\nabla_w R(\boldsymbol{w}) = \boldsymbol{X}^T(\boldsymbol{a} - \boldsymbol{y}) \qquad \nabla_w^2 R(\boldsymbol{w}) = -\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X}$$

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - (\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X})^{-1}(\boldsymbol{X}^T(\boldsymbol{a} - \boldsymbol{y}))$$

This iterative procedure is called *iterative reweighted least squares* because the matrix $\boldsymbol{R}$ gives new "weights" in each iteration.

Logistic regr: non-separable data



Logistic regression: example

- In 2019, the Austrian job center announced that a algorithm would be used to assess job-seekers
- The model is a logistic regression model using several person-centric variables
- The algorithm was show to be unfair, e.g. w.r.t. women

Anhang 1

Das vorliegende Modell ist ein logistisches Regressionsmodell, das die kurzfristige Integrationschance (d.h. die Wahrscheinlichkeit, 90 Beschäftigungstage in 7 Monaten zu erreichen) zu Geschäftsfallbeginn in Abhängigkeit von 12 erklärenden Variablen[1] und Interaktionen zwischen diesen schätzt.

Das Modell enthält alle paarweisen Interaktionsterme, alle Interaktionsterme höherer Ordnung zwischen den Merkmalen GESCHLECHT, ALTER, BERUFSGRUPPE, AUSBILDUNG und STAATENGRUPPE sowie alle Interaktionsterme höherer Ordnung zwischen den Merkmalen GESCHLECHT, ALTER, BEEINTRÄCHTIGUNG, BESCHÄFTIGUNGSVERLAUF, und GESCHÄFTSFALL-FREQUENZ.

Overview

- 4. Basic neural network architectures
  - 4.1 Linear model: Neuron
  - 4.2 Perceptron
  - 4.3 Linear neuron with sigmoid activation: logistic regression
  - 4.4 Linear neuron with softmax activation: softmax regression

4.4 Softmax regression

Softmax: classification task with multiple classes

- Let us assume that the targets/labels can have 1 of K classes
- This is a supervised classification tasks
- Examples for classification tasks
  - Image recognition: cars, dogs, cats, airplanes, ...
- What can we do in this situation? Should we just use LR as before?

X			y
x^1	0.13	0.03	A
x^2	0.15	0.14	A
	0.79	0.19	C
	0.48	0.28	D
	0.93	0.43	C
	0.43	0.41	B
	0.62	0.63	B
	0.69	0.69	A
	0.19	0.79	B
	0.23	0.85	C
x^n	0.11	0.99	B

One-hot encoding

X				y			
x^1	0.13	0.03	A	1	0	0	0
x^2	0.15	0.14	A	1	0	0	0
	0.79	0.19	C	0	0	1	0
	0.48	0.28	D	0	0	0	1
	0.93	0.43	C	0	0	1	0
	0.43	0.41	B	0	1	0	0
	0.62	0.63	B	0	0	0	0
	0.69	0.69	A	1	0	0	0
	0.19	0.79	B	0	1	0	0
	0.23	0.85	C	0	0	1	0
x^n	0.11	0.99	B	0	1	0	0

Softmax regression: model

$$g(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{a} = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{x})$$

- We have a weight matrix instead of a weight vector
- The softmax function is:

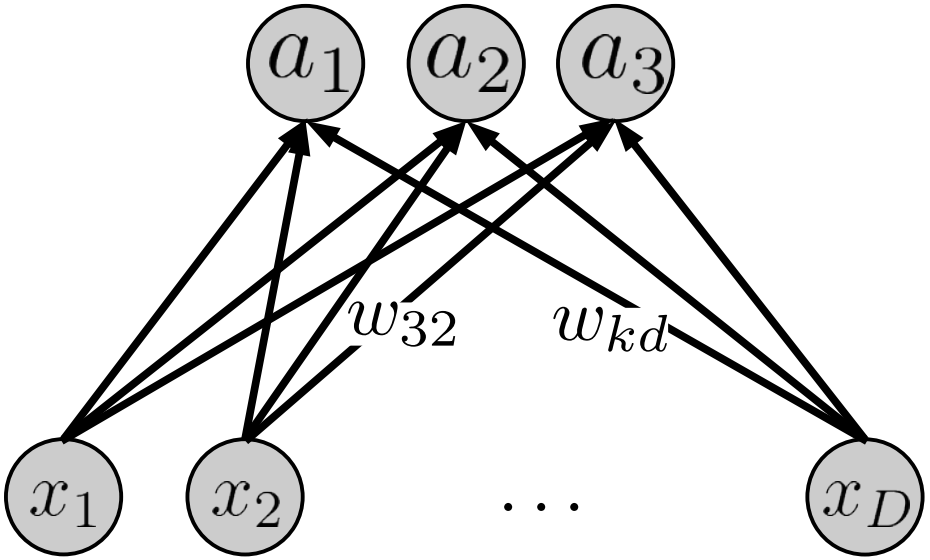
$$\mathrm{softmax}(\boldsymbol{s}) = \left(\frac{e^{s_1}}{\sum_{j=1}^{K} e^{s_j}}, \ldots, \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}, \ldots, \frac{e^{s_K}}{\sum_{j=1}^{K} e^{s_j}} \right)^T$$

- The resulting vector is a probability vector:

$$\boldsymbol{a} = \mathrm{softmax}(\boldsymbol{s})$$

with positive entries that sum to one.

Softmax regression: net



Softmax regr: loss and error

- Likelihood of label vector:

$$p(\boldsymbol{y}|\boldsymbol{a}) = \prod_{k=1}^{K} a_k^{y_k}$$

- *Categorical cross-entropy* loss:

$$-\log p(\boldsymbol{y}|\boldsymbol{a}) = -\sum_{k=1}^{K} y_k \log(a_k)$$

- Empirical error:

$$R_{\text{emp}}(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{Y}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log(a_{nk}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} (\boldsymbol{y}^n)_k \log((\boldsymbol{a}^n)_k)$$

Softmax regression: learning

- Derivative w.r.t. weight:

$$\begin{aligned}
\frac{\partial}{\partial w_{\xi d}} \left(-\sum_{k=1}^{K} y_k \log(a_k) \right) &= -\frac{\partial}{\partial w_{\xi d}} \sum_{k=1}^{K} y_k \log \left(\frac{e^{\boldsymbol{w}_k^T \boldsymbol{x}}}{\sum_{j=1}^{K} e^{\boldsymbol{w}_j^T \boldsymbol{x}}} \right) \\
&= -\frac{\partial}{\partial s_k} \frac{\partial s_k}{\partial w_{\xi d}} \sum_{k=1}^{K} y_k \log \left(\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}} \right) \\
&= (a_\xi - y_\xi) x_d
\end{aligned}$$

- Derivative w.r.t. to weight matrix:

$$\nabla_{\boldsymbol{W}} R_{\text{emp}}(\boldsymbol{W}, \boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{a} - \boldsymbol{y}) \boldsymbol{x}^T$$

Softmax regression learning

- Again, we can use gradient descent for learning
- Type of optimization problem?
- Convergence?

Comparing gradients

- Linear regression:

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(y, \boldsymbol{x}, \boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{x} - y)\boldsymbol{x}$$

- Logistic regression:

$$\nabla_{\boldsymbol{w}} R_{\text{emp}}(y, \boldsymbol{x}, \boldsymbol{w}) = \left(\sigma(\boldsymbol{w}^T \boldsymbol{x}) - y\right) \boldsymbol{x}$$

- Softmax regression

$$\nabla_{\boldsymbol{W}} R_{\text{emp}}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{W}) = (\text{softmax}(\boldsymbol{W}\boldsymbol{x}) - \boldsymbol{y})\boldsymbol{x}^T$$

Summary

- We have discussed single-layer networks
  - Linear neuron, linear regression
  - Perceptron
  - Linear neuron with sigmoid activation, log regr
  - Softmax regression
- Learning, convergence and solutions
- For many applications, these networks are sufficient