OEAW AI SUMMER SCHOOL

DEEP LEARNING I

Logistic Regression

Johannes Brandstetter, Michael Widrich
Institute for Machine Learning

Welcome to Deep Learning

- Logistic Regression
- Neural Networks
- Convolutional Neural Networks
- Autoencoders
- Generative Adversarial Networks
- Recurrent Neural Networks

Welcome to Deep Learning

- Every lecture is supported by jupyter notebooks in the exercises.
  - Python and PyTorch are used:
    - Machine Learning / Data Science can be done with many languages (R, Matlab, Python, ...).
    - In Deep Learning Python is the Go-To language.
- Notebooks are constructed in a simplistic and similar way:
  - You should be able to concentrate on the most important topics in the network architectures.
- Slides (notebooks) contain basic information (tasks) plus some more advanced background information.
  - We hope that everybody profits from that!

Lecture I: Logistic Regression - simple but powerful

Content of this lecture / exercise

- Linear Regression vs. Logistic Regression
- Loss functions
- Optimization and Gradient Descent
  - Backpropagation
- Interpreting Logistic Regression in the Deep Learning setting
  - Introduction to PyTorch (Wolfgang Waltenberger)
  - We will capitalize on the introduced setting (problem) during the whole week

Maximum Likelihood

Biased Coin Example

- You throw a coin 10 times: `H H H T H H T H H T`
- Was the coin biased?
- If so, by how much?

If we assume the coin was unbiased, how likely is it to obtain this result?

Biased Coin Example

How likely is it to obtain only 3 tails if the probability of obtaining tails / heads was 50:50?

$$p(\text{3 Tails}|\theta = 0.5) = \mathcal{B}(x = 3, p = 0.5, n = 10) \approx 0.117$$
$$p(\text{3 Tails}|\theta = 0.4) = \mathcal{B}(x = 3, p = 0.4, n = 10) \approx 0.215$$
$$p(\text{3 Tails}|\theta = 0.3) = \mathcal{B}(x = 3, p = 0.3, n = 10) \approx 0.267$$
$$p(\text{3 Tails}|\theta = 0.2) = \mathcal{B}(x = 3, p = 0.2, n = 10) \approx 0.201$$

Biased Coin Example

How likely is it to obtain only 3 tails if the probability of obtaining tails / heads was 50:50?

$$p(\text{3 Tails}|\theta = 0.5) = \mathcal{B}(x = 3, p = 0.5, n = 10) \approx 0.117$$

$$p(\text{3 Tails}|\theta = 0.4) = \mathcal{B}(x = 3, p = 0.4, n = 10) \approx 0.215$$

$$p(\text{3 Tails}|\theta = 0.3) = \mathcal{B}(x = 3, p = 0.3, n = 10) \approx 0.267$$

$$p(\text{3 Tails}|\theta = 0.2) = \mathcal{B}(x = 3, p = 0.2, n = 10) \approx 0.201$$

So given the data, the outcome is most likely if the coin has a chance of 30:70 for heads / tails ($\theta = 0.3$).

Maximum Likelihood Estimation

- **Given:** a data set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ from a distribution $p(\mathbf{x}; \theta)$, where θ represents the parameter(s) of the distribution.
  - One sample $\mathbf{x}_i$ is a vector of features.
- **Task:** find the parameter(s) θ that are most likely to produce this data.
- **Idea:** How likely is a given θ to produce the dataset?

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i; \theta)$$

- **Solution:** Find θ that maximizes $\mathcal{L}(\theta)$ (or minimizes $-\log \mathcal{L}(\theta)$, which is equivalent).

Linear Regression

Formulas

- Given a bunch of data $\mathbf{Z} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$
  - $\mathbf{x}_i$ ... vector of features
  - $\mathbf{y}_i$ ... labels
- For now, we will assume that $\mathbf{y}_i$ is a scalar (y_i).
- Find $g(\mathbf{x})$ such that $\forall i : g(\mathbf{x}_i) \approx y_i$
- Regression $\Leftrightarrow y_i \in \mathbb{R}$
- Classification $\Leftrightarrow y_i \in \{1, \ldots, k\}$
- n will denote the size of our training set: $|\mathbf{Z}| = n$

Linear Models

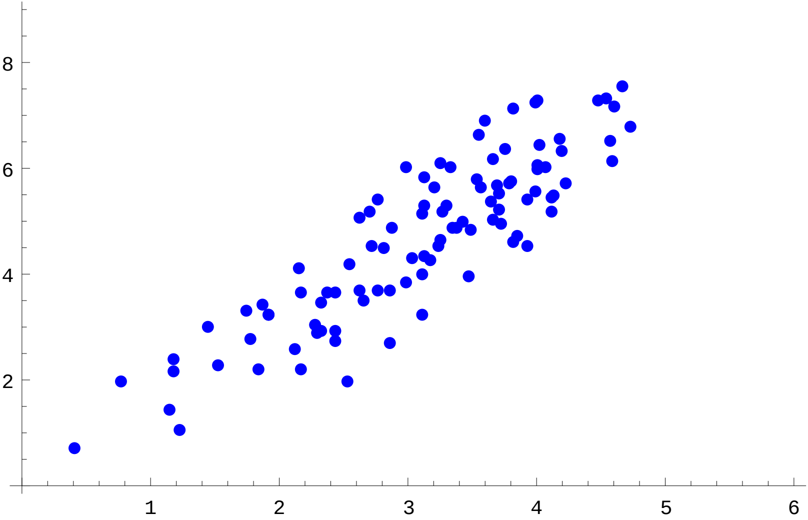
- Simplest approach: use something linear

$$g(\mathbf{x}_i) = a \cdot \mathbf{x}_i + b$$

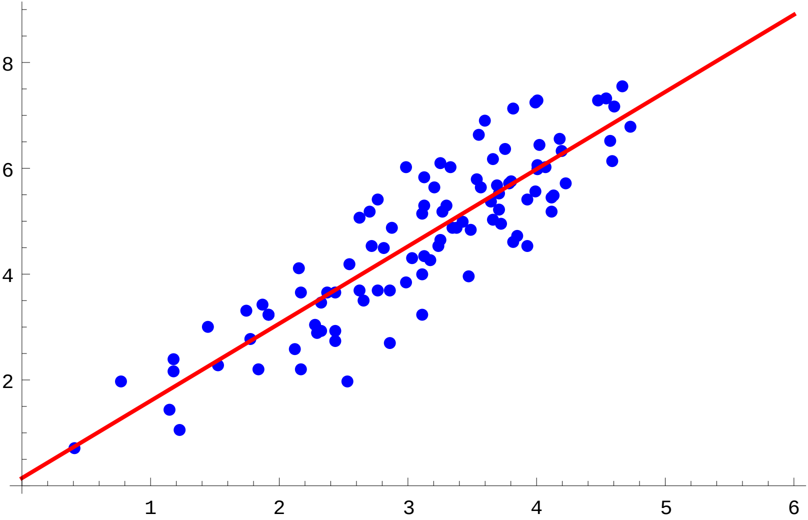
- To keep math simple, assume $\mathbf{x}_i$ is 1-dimensional (scalar)
- Idea: Minimize Squared Error ("Loss"):

$$L = \sum_{i=1}^{n} (y_i - g(x_i))^2$$

Linear Regression



Linear Regression



Linear Regression

$$g(x) = a + b \cdot x + c \cdot x^2$$

or:

$$\mathbf{x}_i = \begin{pmatrix} 1 \\ x_i \\ x_i^2 \end{pmatrix}$$

Closed form solution for minimum:

$$\min_{\mathbf{W}} \quad \frac{1}{2} \sum_{i=0}^{n} \left(y_i - \mathbf{W}^T \mathbf{x}_i \right)^2$$

Nota bene: $\mathbf{W}$ in Linear/Logistic Regression is a vector, but in neural networks it will be a matrix. Thus, the notation $\mathbf{W}$ is chosen.

Why Squared Error?

- Assume that data is generated with Gaussian noise.
- What's the (log) likelihood of observing our data?

$$
\begin{aligned}
\log \prod_i p(\mathbf{y}_i|\mathbf{x}_i) &= \sum_i \log \mathcal{N}(g(x_i), \sigma^2) \\
&= \sum_i \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - g(x_i))^2\right) \\
&= \sum_i \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2}\frac{1}{2}\sum_{i=0}^{n}(y_i - g(x_i))^2
\end{aligned}
$$

Why Squared Error?

- Assume that data is generated with Gaussian noise.
- What's the (log) likelihood of observing our data?

$$
\begin{aligned}
\log \prod_i p(\mathbf{y}_i|\mathbf{x}_i) &= \sum_i \log \mathcal{N}(g(x_i), \sigma^2) \\
&= \sum_i \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - g(x_i))^2\right) \\
&= \sum_i \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2}\frac{1}{2}\sum_{i=0}^{n}(y_i - g(x_i))^2
\end{aligned}
$$

This is the (negative) squared error function!

$\Rightarrow$ **Minimizing the negative Log-Likelihood of Gaussian noise is the same as minimizing the squared error.**

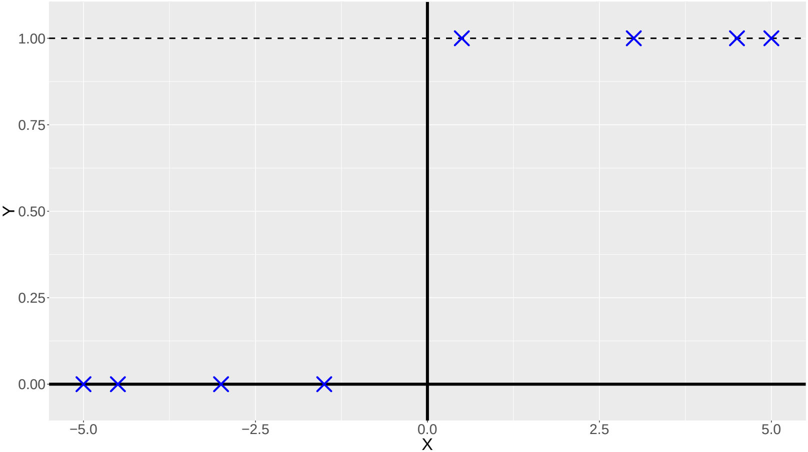
Logistic Regression

Logistic Regression

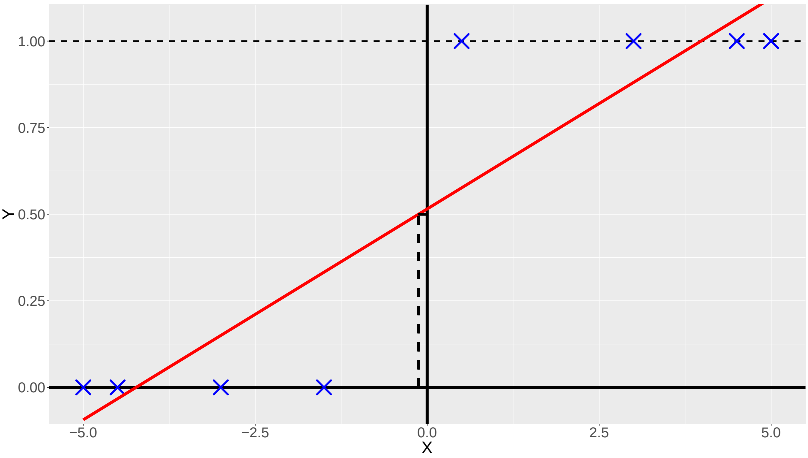
- **Given:** n datapoints $\mathbf{x}_i$ with a labels $y_i \in \{0, 1\}$
- **Task:** find $g(\mathbf{x})$ such that $g(\mathbf{x}_i) = y_i$
- $\Rightarrow$ **Classification Task**
- **First (bad) idea:** fit a linear regression line $g_{LR}(\mathbf{x}_i) = \mathbf{W}^T \mathbf{x}_i$
- **Then:**

$$y_i = \begin{cases} 0 & g_{LR}(\mathbf{x}_i) < 0.5 \\ 1 & g_{LR}(\mathbf{x}_i) \geq 0.5 \end{cases}$$

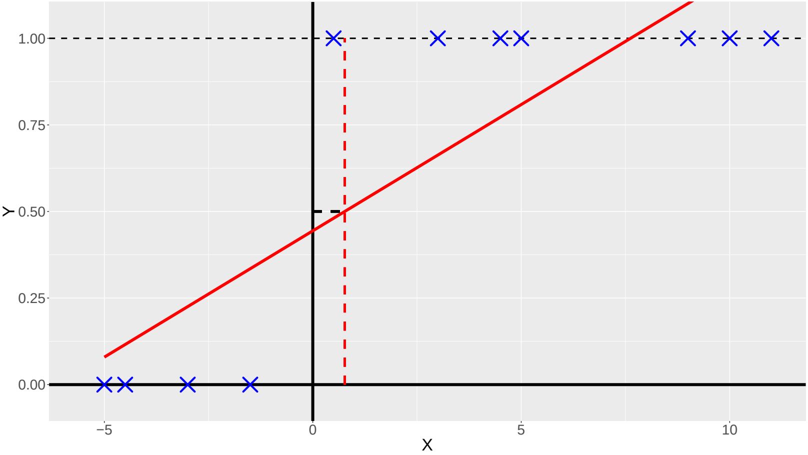
Problem with Linear Regression



Problem with Linear Regression



Problem with Linear Regression



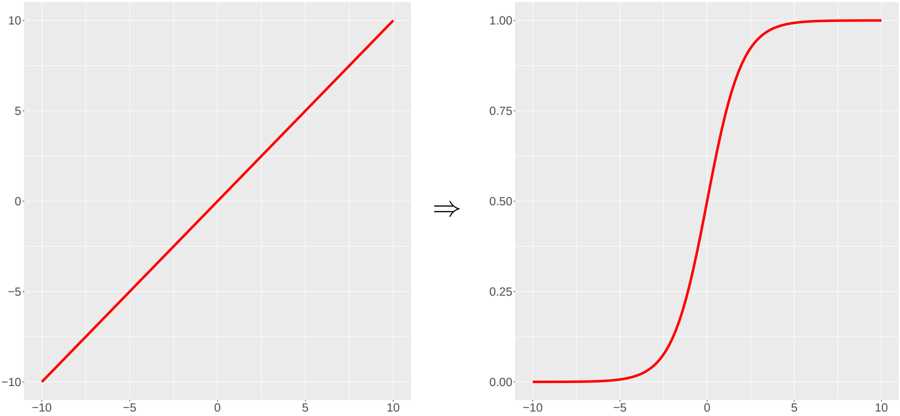
Logistic Regression

- Models relationship between a **categorical** label and some features $\mathbf{x}$.
- The relationship is not linear, instead we apply the logistic function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

- t is a linear function of features: $t = \mathbf{W}^T\mathbf{x}$
- Model: $g(\mathbf{x}) = \sigma(\mathbf{W}^T\mathbf{x})$

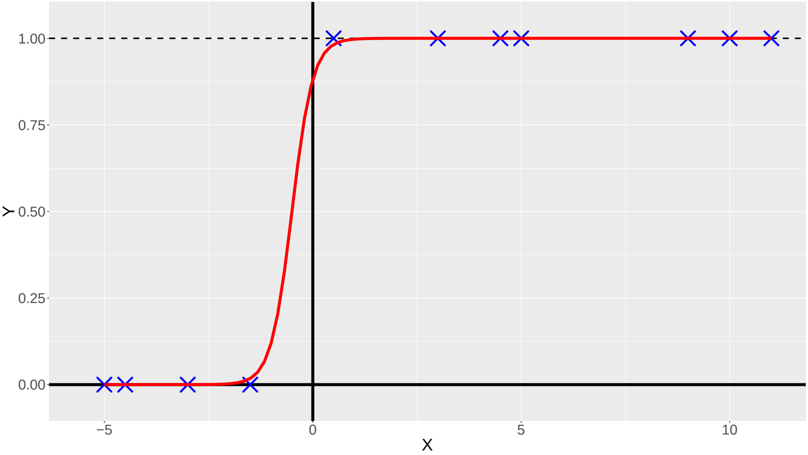
Logistic Function



Also known as **sigmoid function**.

Also known as **Fermi function** in physics.

Logistic Regression



Objective

- Likelihood function for a Bernoulli distribution:

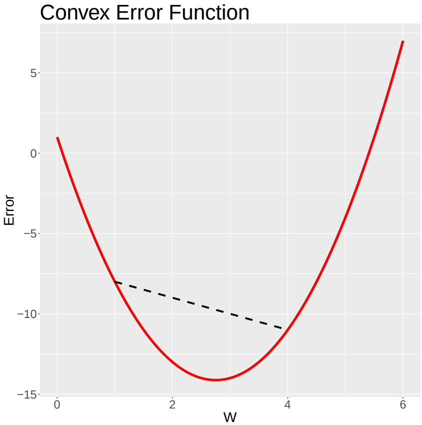
$$\mathcal{L}(\{\mathbf{x}, y\}; \mathbf{W}) = \prod_{i=1}^{n} g(\mathbf{x}_i; \mathbf{W})^{y_i} \cdot (1 - g(\mathbf{x}_i; \mathbf{W}))^{1-y_i}$$

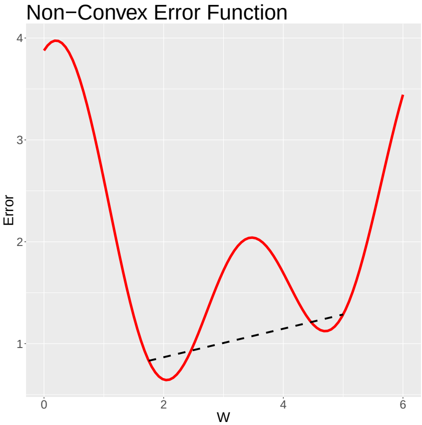
- Taking the negative logarithm, we obtain:

$$L = -\log \mathcal{L}(\{\mathbf{x}, y\}; \mathbf{W}) =$$
$$= -\sum_{i} \Big(y_i \log g(\mathbf{x}_i; \mathbf{W}) + (1 - y_i) \log(1 - g(\mathbf{x}_i; \mathbf{W})) \Big)$$

Also known as the **Cross Entropy Error**, which makes Logistic Regression a **convex problem**.

Convex vs. non-convex





Logistic Regression Problem

- **Task:**

$$\min_{\mathbf{W}} L = \underbrace{-\sum_i \left(y_i \log g(\mathbf{x}^i; \mathbf{W}) + (1 - y_i) \log(1 - g(\mathbf{x}^i; \mathbf{W})) \right)}_{\min_{\mathbf{W}}}$$

  - $g(\mathbf{x}; \mathbf{W}) = \sigma(\mathbf{W}^T \mathbf{x}) = \frac{1}{1+\mathrm{e}^{-\mathbf{W}^T \mathbf{x}}}$

- **Note:** no closed-form solution!
  You have to use methods like Gradient Descent, Newton, BFGS, Conjugate Gradient, ...

- Derivative of sigmoid function:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$$

Softmax

- Generalization of the sigmoid function
  - Also known as **Boltzmann distribution** in physics.
- Suitable for **multi-class** classification
- For K classes with $y \in \{1, \dots, K\}$ the probability of $\mathbf{x}$ belonging to class k is:

$$p(y = k|\mathbf{x}) = \frac{e^{\mathbf{W}_k^T \mathbf{x}}}{\sum_{j=1}^{K} e^{\mathbf{W}_j^T \mathbf{x}}} .$$

- With the objective:

$$\min_{\mathbf{W}} L = \min_{\mathbf{W}} -\sum_k \sum_i [y_i]_k \log p(y_i = k|\mathbf{x}; \mathbf{W})$$

where $[y_i]_k$ is the k-th entry of the *one-hot* vector $[y_i]$.

Gradient Descent

- **Given:** a function $f(x)$
- **Task:** find x that maximizes (or minimizes) $f(x)$
- **Idea:** start at some value x_0, and take a small step η in the direction in which the function decreases strongest.
  Find derivative $f' = \frac{\partial f}{\partial x}$.
  - $-f'(x_0)$ is the direction of steepest descent at x_0.
- **Solution:**
  - Iteratively calculate $x_{i+1} = x_i - \eta \cdot f'(x_i)$.
  - Each x_{i+1} should be a better solution than x_i.
  - Eventually you'll reach a (**local**) minimum.

Gradient Descent in Logistic Regression

The minimization of the loss function $L(.;\mathbf{W})$ can be done by Gradient Descent:

$$\mathbf{W}_{n+1} = \mathbf{W}_n - \eta \frac{\partial L}{\partial \mathbf{W}} \ ,$$

where η is the learning rate,

and $\mathbf{W}_0$ is some initial guess for $\mathbf{W}$.

Learning rate η

- Under certain assumptions, GD converges to a local minimum in linear time.
- The choice of η is critical:
  - If η is too large we will jump around.
  - If η is too small we will barely make progress.
  - In both cases convergence will be slow or even impossible.
  - Can we optimize for η?
- The choice of η depends on the local structure (curvature) of the optimization landscape.
  - This insight leads to second-order methods (not treated in this course).

Momentum term

- Plain GD is often slow, e.g. in flat regions or narrow valleys.
- Adding some inertia to the update rule makes the trajectory similar to that of a heavy ball rolling down the error surface.
- We provide the algorithm with a memory of recent updates:

$$\Delta \mathbf{W}_n = -\eta \frac{\partial L}{\partial \mathbf{W}} + \mu \Delta \mathbf{W}_{n-1}$$
$$\mathbf{W}_{n+1} = \mathbf{W}_n + \Delta \mathbf{W}_n$$

  - $\mu \Delta \mathbf{W}_{n-1}$ is the momentum term
  - $0 \leq \mu \leq 1$ is the momentum parameter

Gradient Checking

- Method for checking if the symbolic computation/implementation of the gradient was correct.
- Logistic Regression gradient is easy, but once we get to neural networks, you'll be glad to know this trick.
- **Idea:** compare your gradient with a numerical approximation of the gradient.

Gradient Checking

- Central difference quotient:

$$\frac{\partial L}{\partial W_{ij}} \approx \frac{L(.;\mathbf{W} + \epsilon\, \mathbf{e}_{ij}) - L(.;\mathbf{W} - \epsilon\, \mathbf{e}_{ij})}{2\,\epsilon}$$

- Central difference quotient for logistic regression ($\mathbf{W}$ is a vector):

$$\frac{\partial L}{\partial W_{i}} \approx \frac{L(.;\mathbf{W} + \epsilon\, \mathbf{e}_{i}) - L(.;\mathbf{W} - \epsilon\, \mathbf{e}_{i})}{2\,\epsilon}$$

  with $\mathbf{e}_i = (0\,0\,\ldots\,1\,\ldots\,0)^T$.

- Good choice is $\epsilon = 10^{-4}$.

Stochastic Gradient Descent

We want to minimize the error over all samples in the training set using the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$:

$$\mathcal{L} = \sum_i L(y_i, \mathbf{x}_i; \mathbf{W})$$

Problem: Calculating the gradient is expensive (we need to look at each training sample).

Idea: Use a cheaper estimate of the gradient / loss function

Solution: Use only a fraction of the training samples:

$$\mathbb{E}\left(\sum_{i=1}^{n} L(y_i, \mathbf{x}_i; \mathbf{W})\right) \approx \sum_{i=k,k\geq 1}^{m,m\leq n} L(y_i, \mathbf{x}_i; \mathbf{W})$$

Nomeclature

Stochastic Gradient Descent (SGD): use some sort of estimate of the gradient

Batch-Learning: calculate the gradient over the whole training set, then update parameters

Online Learning: estimate the gradient for a single datapoint, then update parameters (good for streaming data!)

Mini-Batch Learning: calculate the gradient, use some number k of samples to estimate the gradient, then update parameters

- $k = 1$: Online Learning
- $k = n$: Batch-Learning

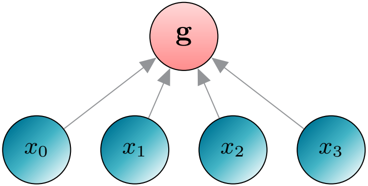
Epoch: enough updates to go through the whole training set once:

- Batch-Learning: each update is an epoch
- Online Learning: n updates are an epoch

Summary

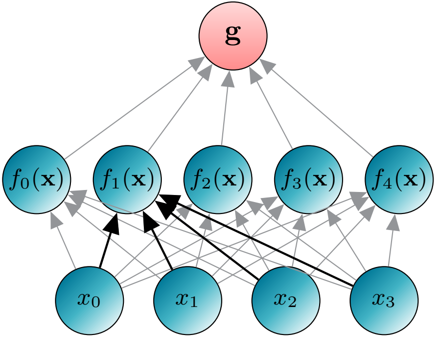
- Logistic Regression as extension of Linear Regression:
  - Sigmoid function
  - Softmax function
- Cross Entropy Error makes Logistic Regression a convex problem:
  - Gradient Descent
  - Gradient Checking
  - Stochastic Gradient descent

Relation to Neural Networks



$$g(\mathbf{x}_i) = \sigma\left(\mathbf{W}^T\mathbf{x}_i + b\right)$$

Logistic regression with **learned features**:



$$g(\mathbf{x}_i) = \sigma\left(\mathbf{W}_{(2)}^T\mathbf{f}_i + b\right)$$

What I did not talk about

- 2nd order methods
- Regularization
- Feature selection / feature construction