Machine Learning Basics III

Benjamin Roth

CIS LMU München

Outline

Outline

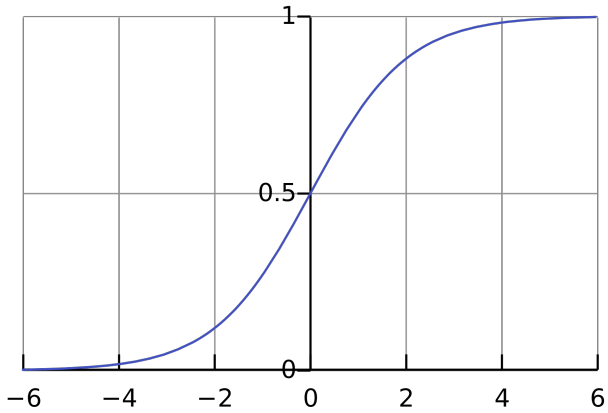
From Regression to Classification

- So far, linear regression:
  - A simple linear model.
  - Probabilistic interpretation.
  - Find optimal parameters using Maximum Likelihood Estimation.
- Can we do something similar for classification?
- $\Rightarrow$ Logistic Regression (*. . . it's not actually used for regression . . .*)

Logistic Regression

- Binary logistic model:
  Estimate the probability of a binary response $y \in \{0, 1\}$ based on features $\boldsymbol{w}$.
- Logistic Regression is a *Generalized Linear Model*:
  Linear model is related to the response variable via a **link function**.

$$p(Y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = f(\boldsymbol{\theta}^T \boldsymbol{x})$$

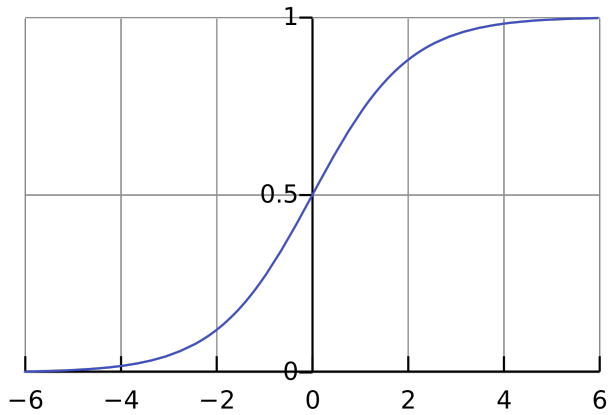


(Note: Y denotes a random variable, whereas y, $y^{(i)}, 0, 1$ denote values that the random variable can take on. If the random variable is obvious from the context, it may be omitted.)

Logistic Regression

- Recall linear regression: $p(y|\boldsymbol{x};\boldsymbol{\theta}) = N(y;\boldsymbol{\theta}^T\boldsymbol{x},\boldsymbol{I})$
  - Predicts $y \in \mathbb{R}$

- Classification: Outcome (per example) 0 or 1
  - Logistic sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$



  - Logistic Regression: Linear function + logistic sigmoid

$$p(Y = 1|\boldsymbol{x};\boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T\boldsymbol{x})$$

Binary Logistic Regression

Probability of different outcomes (for one example):

- Probability of positive outcome:

$$p(Y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

- Probability of negative outcome:

$$p(Y = 0|\boldsymbol{x}; \boldsymbol{\theta}) = 1 - p(Y = 1|\boldsymbol{x}; \boldsymbol{\theta})$$

Probability of a Training Example

- Probability for actual label $y^{(i)}$ given features $\boldsymbol{x}^{(i)}$
- Can be written for both labels (0 and 1) without case distinction
- Label exponentiation trick: use $x^0 = 1$

$$p(Y = y^{(i)} | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \begin{cases} p(Y = 1 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) & \text{if } y^{(i)} = 1 \\ p(Y = 0 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) & \text{if } y^{(i)} = 0 \end{cases}$$

$$= \begin{cases} p(Y = 1 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})^1 p(Y = 0 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})^0 & \text{if } y^{(i)} = 1 \\ p(Y = 1 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})^0 p(Y = 0 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})^1 & \text{if } y^{(i)} = 0 \end{cases}$$

$$= p(Y = 1 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})^{y^{(i)}} p(Y = 0 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta})^{1 - y^{(i)}}$$

Binary Logistic Regression

- Conditional Negative Log-Likelihood (NLL):

$$NLL(\boldsymbol{\theta}) = -\log p(\boldsymbol{y}|\boldsymbol{X};\boldsymbol{\theta})$$
$$= -\log \prod_{i=1}^{m} p(Y = y^{(i)}|\boldsymbol{x}^{(\boldsymbol{i})};\boldsymbol{\theta})$$
$$= -\log \prod_{i=1}^{m} p(Y = 1|\boldsymbol{x}^{(\boldsymbol{i})};\boldsymbol{\theta})^{y^{(i)}}(1 - p(Y = 1|\boldsymbol{x}^{(\boldsymbol{i})};\boldsymbol{\theta}))^{1-y^{(i)}}$$
$$= -\sum_{i=1}^{m} y^{(i)} \log \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(\boldsymbol{i})}) + (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(\boldsymbol{i})}))$$

- No closed form solution for minimum
- Use numerical / iterative methods.
- LBFGS, Gradient descent ...

Logistic Regression

- Logistic regression: Logistic sigmoid function applied to a a weighted linear combination of feature values.
- To be interpreted as the probability that the label for a specific example equals 1.
- Applying the model on test data: Predict $y^{(i)} = 1$ if

$$p(Y = 1|\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) > 0.5$$

- No closed form solution for maximizing NLL, iterative methods necessary.
- Next up: Gradient descent.

Outline

Optimization

- Optimization: Minimize some function $J(\boldsymbol{\theta})$ by altering $\boldsymbol{\theta}$.
- Maximize $f(\boldsymbol{\theta})$ by minimizing $J(\boldsymbol{\theta}) = -f(\boldsymbol{\theta})$
- $J(\boldsymbol{\theta})$:
  - *"criterion"*, *"objective function"*, *"cost function"*, *"loss function"*, *"error function"*
  - In a probabilistic machine learning setting often (conditional) negative log-likelihood:

    $$-\log p(\boldsymbol{X}; \boldsymbol{\theta})$$

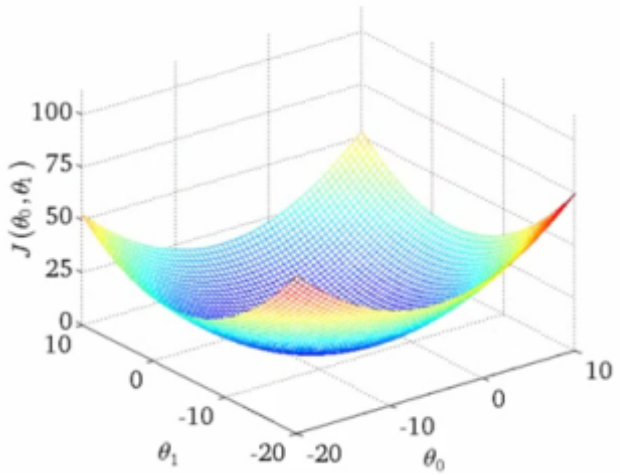
    or

    $$-\log p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{\theta})$$

    as a function of $\boldsymbol{\theta}$
  - $\boldsymbol{\theta}^* = \arg\min_\theta J(\boldsymbol{\theta})$

Optimization

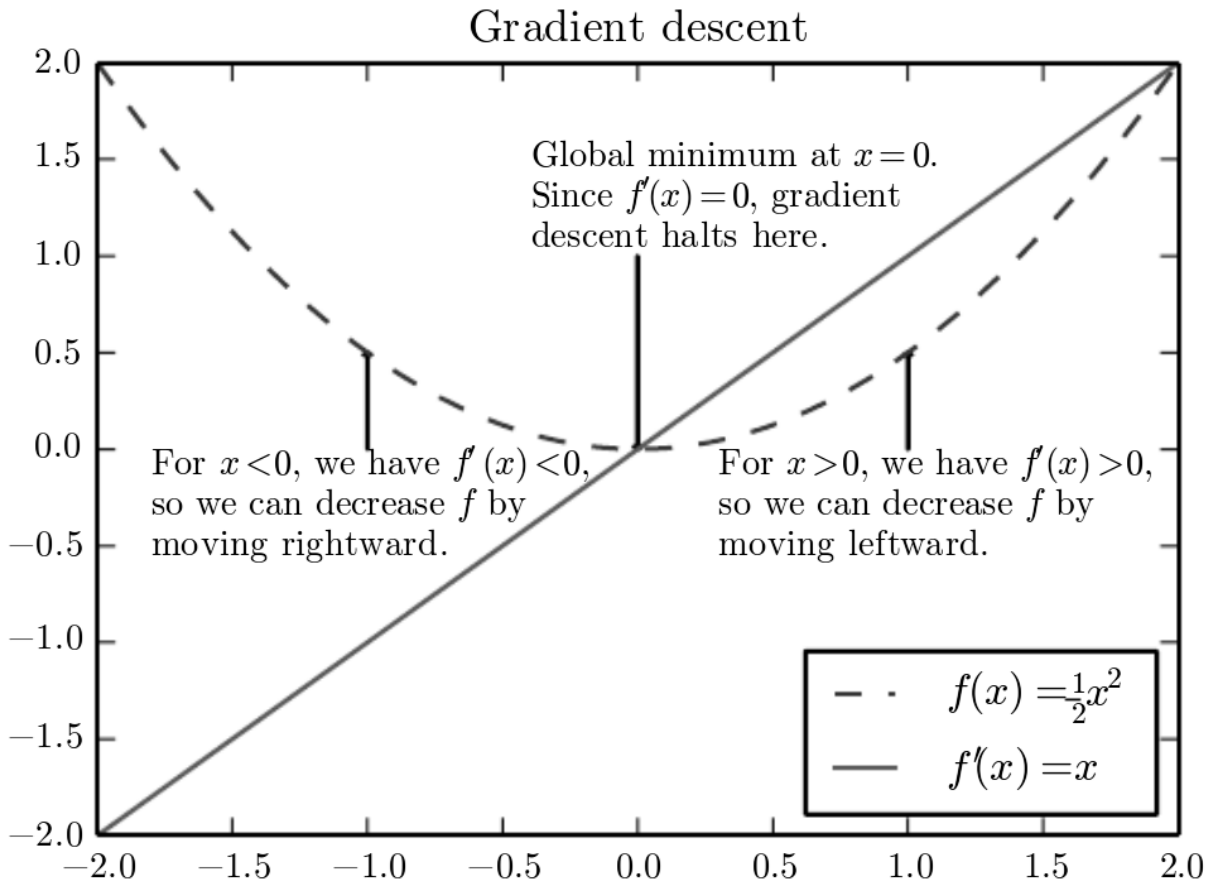
- If $J(\boldsymbol{\theta})$ is convex, it is minimized where $\nabla_\theta J(\boldsymbol{\theta}) = \mathbf{0}$
- If $J(\boldsymbol{\theta})$ is not convex, the gradient can help us to improve our objective nevertheless (and find a local optimum).
- Many optimization techniques were originally developed for convex objective functions, but are found to be working well for non-convex functions too.
- Use the fact that gradient indicates the slope of the function in the direction of steepest increase.



Gradient-Based Optimization

- Derivative: Given a small change in input, what is the corresponding change in output?

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x)$$



- $f(x - \epsilon \operatorname{sign} f'(x)) < f(x)$ for small enough ϵ

Gradient Descent

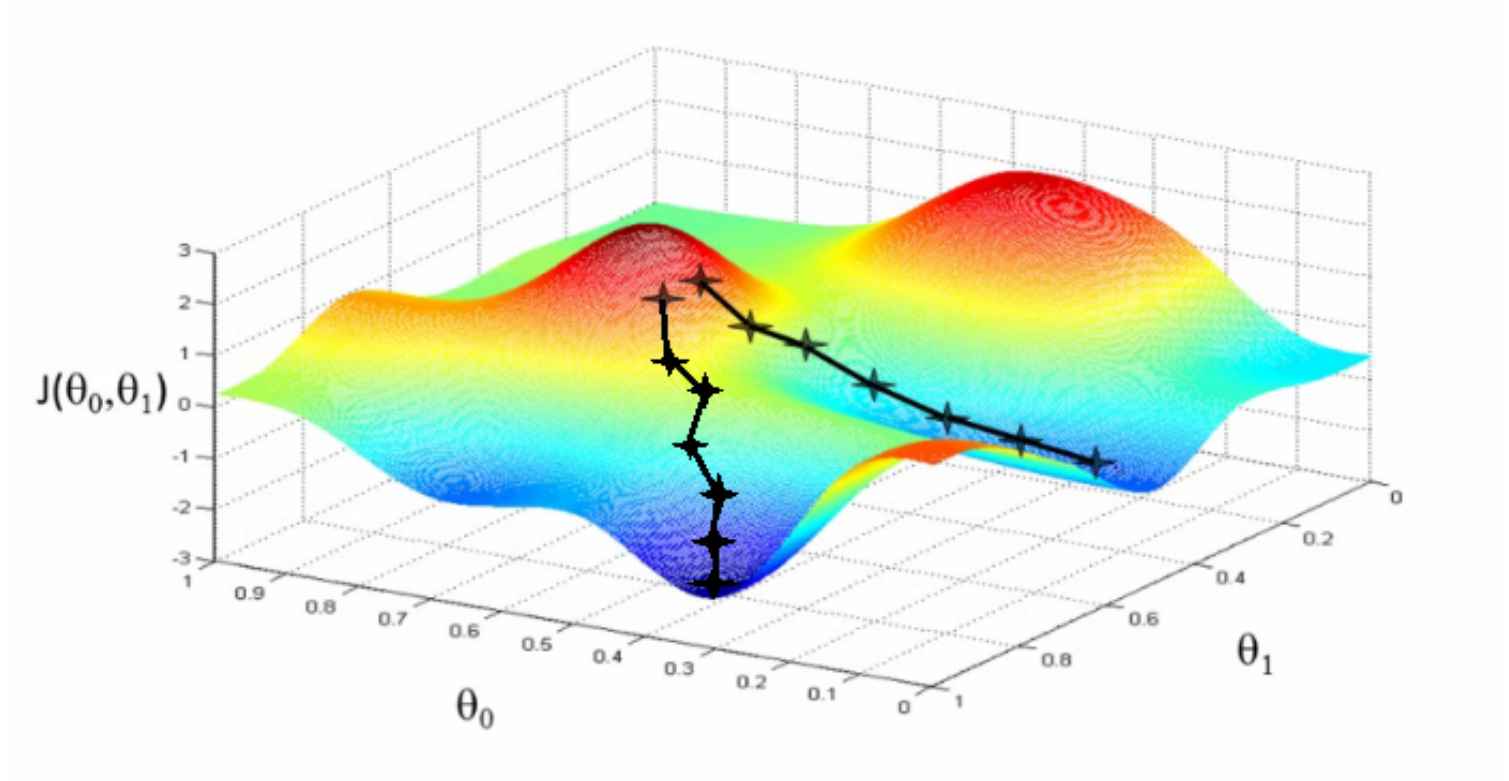
- For $J(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}$
- If partial derivative $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} > 0$, $J(\boldsymbol{\theta})$ will increase for small increases of $\boldsymbol{\theta}_j$
  $\Rightarrow$ go in opposite direction of gradient (since we want to minimize)
- Steepest descent: iterate

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

- η is the learning rate (set to small positive constant).
- Converges if $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is (close to) $\mathbf{0}$

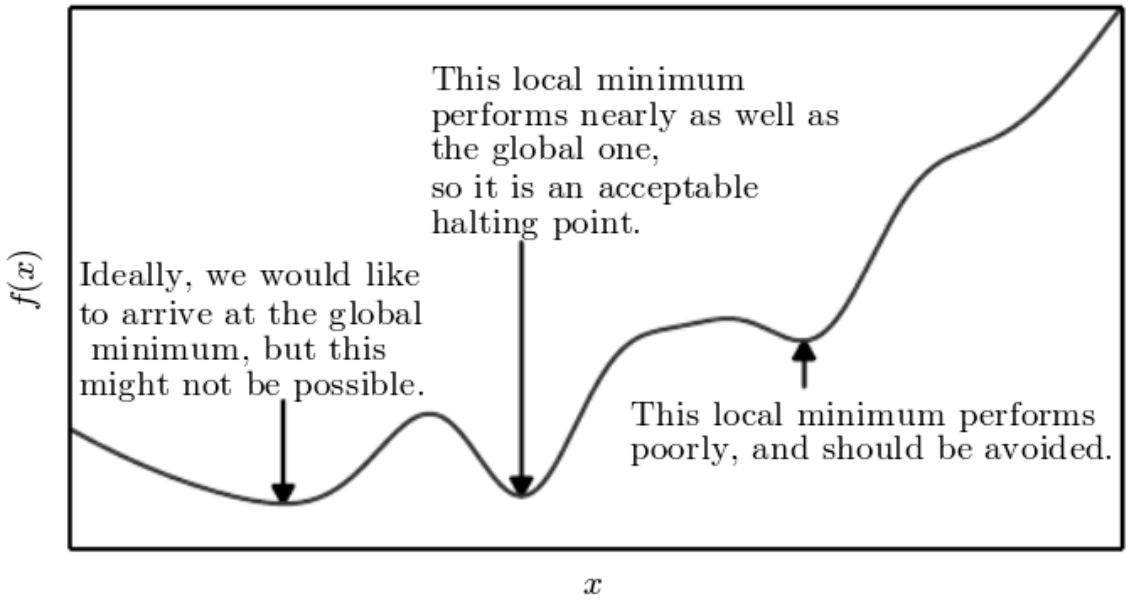
Local Minima

- If function is non-convex, different results can be obtained at convergence, depending on initialization of $\boldsymbol{\theta}$.

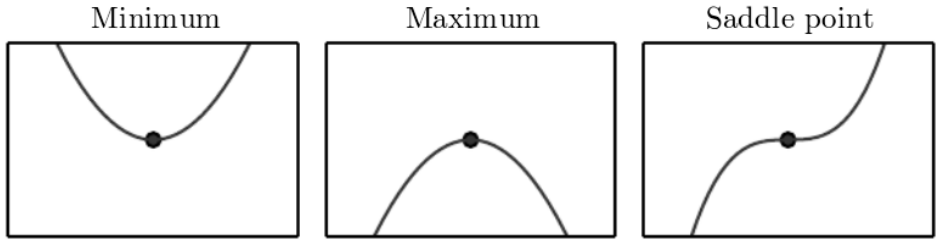


Local Minima

- Minima can be global or local:



- Critical (stationary) points: $f'(x) = 0$



- For neural networks, only good (not perfect) parameter values can be found.

Outline

Gradient Descent for Logistic Regression

$$\nabla_\theta NLL(\boldsymbol{\theta}) = -\nabla_\theta \sum_{i=1}^{m} y^{(i)} \log \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}))$$

$$= -\sum_{i=1}^{m} (y^{(i)} - \sigma(\boldsymbol{\theta}^T \boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)}$$

- The gradient descent update becomes:

$$\boldsymbol{\theta}_{t+1} := \boldsymbol{\theta}_t + \eta \sum_{i=1}^{m} (y^{(i)} - \sigma(\boldsymbol{\theta}_t^T \boldsymbol{x}^{(i)}))\boldsymbol{x}^{(i)}$$

- Note: Which feature weights are increased, which are decreased?

Derivation of Gradient for Logistic Regression

This is a great exercise! Use the following facts:

Gradient	$(\nabla_\theta f(\boldsymbol{\theta}))_j = \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j}$
Derivative of a sum	$\frac{d}{dz} \sum_i f_i(z) = \sum_i \frac{df_i(z)}{dz}$
Chain rule	$F(z) = f(g(z)) \;\Rightarrow\; F'(z) = f'(g(z))g'(z)$
Derivative of logarithm	$\frac{d \log z}{dz} = 1/z$
D. of logistic sigmoid	$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$
Partial d. of dot-product	$\frac{\partial \boldsymbol{\theta}^T \boldsymbol{x}}{\partial \theta_j} = \boldsymbol{x}_j$

Gradient Descent: Summary

- Iterative method for function minimization.
- Gradient indicates rate of change in objective function, given a local change to feature weights.
- Substract the gradient:
  - **decrease** parameters that (locally) have **positive** correlation with objective
  - **increase** parameters that (locally) have **negative** correlation with objective
- Gradient updates only have the desired properties in a small region around previous parameters $\boldsymbol{\theta}_t$. Control locality by step-size η.
- Gradient descent is slow: For relatively small step in the right direction, all of training data has to be processed.
- This version of gradient descent is often also called *batch gradient descent*.

Outline

Stochastic Gradient Descent (SGD)

- *Batch gradient descent* is slow: For relatively small step in the right direction, all of training data has to be processed.

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_\theta \sum_{i=1}^{m} \log p(y_i | \boldsymbol{x}_i; \boldsymbol{\theta})$$

- *Stochastic gradient descent* in a nutshell:
  - For each update, only use random sample $\mathbb{B}_t$ of training data (mini-batch).

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_\theta \sum_{i \in \mathbb{B}_t} \log p(y_i | \boldsymbol{x}_i; \boldsymbol{\theta})$$

  - Mini-batch size can also just be 1.

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta \nabla_\theta \log p(y_t | \boldsymbol{x}_t; \boldsymbol{\theta})$$

- $\Rightarrow$ More frequent updates.

Stochastic Gradient Descent (SGD)

- The actual gradient is *approximated* using only a sub-sample of the data.
- For objective functions that are highly non-convex, the random deviations of these approximations may even help to escape local minima.
- Treat batch size and learning rate as hyper-parameter.

Outline

Deep Feedforward Networks

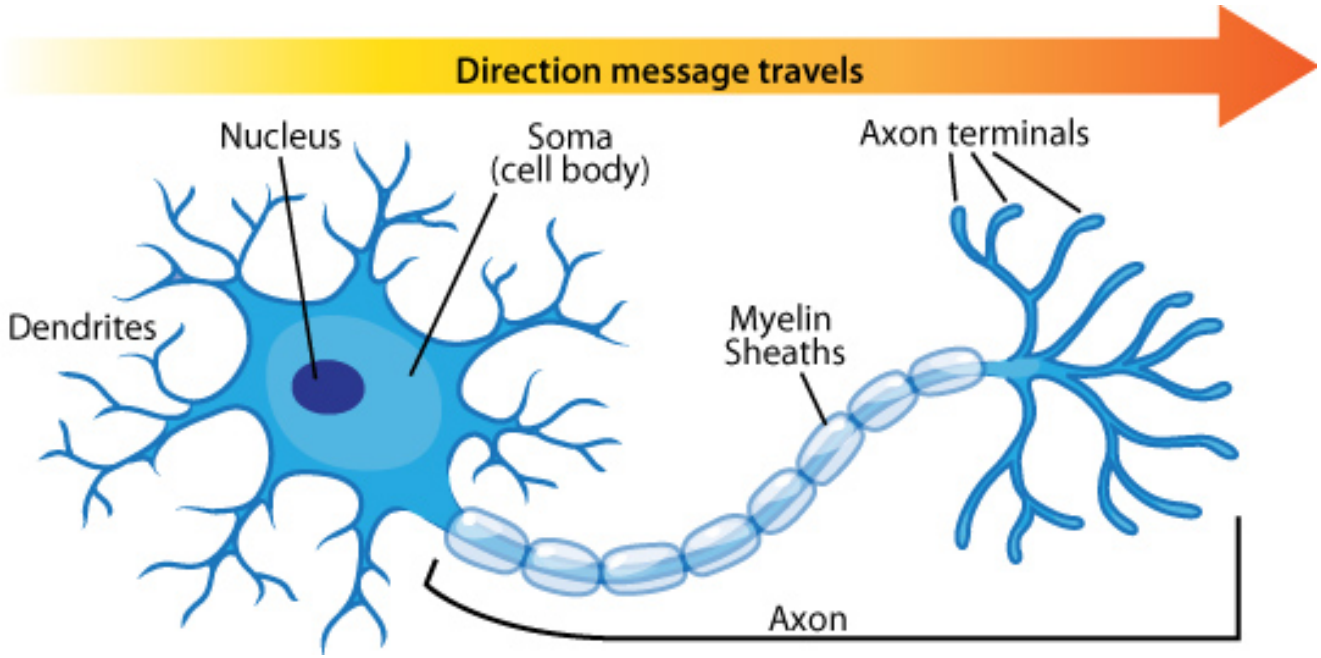
- Function approximation: find good mapping $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$
- *Network*: Composition of functions $f^{(1)}, f^{(2)}, f^{(3)}$ with multi-dimensional input and output
- Each $f^{(i)}$ represents one *layer* $f(\boldsymbol{x}) = f^{(1)}(f^{(2)}(f^{(3)}(\boldsymbol{x})))$
- *Feedforward*:
  - Input $\rightarrow$ intermediate representation $\rightarrow$ output
  - No feedback connections
  - Cf. *recurrent* networks

Deep Feedforward Networks: Training

- Loss function defined on output layer, e.g. $||\hat{y} - f(\boldsymbol{x}; \boldsymbol{\theta})||_2^1$
- Quality criterion on other layers not directly defined.
- Training algorithm must decide how to use those layers most effectively (w.r.t. loss on output layer)
- Non-output layers can be viewed as providing a feature function $\phi(\boldsymbol{x})$ of the input, that is to be learned.

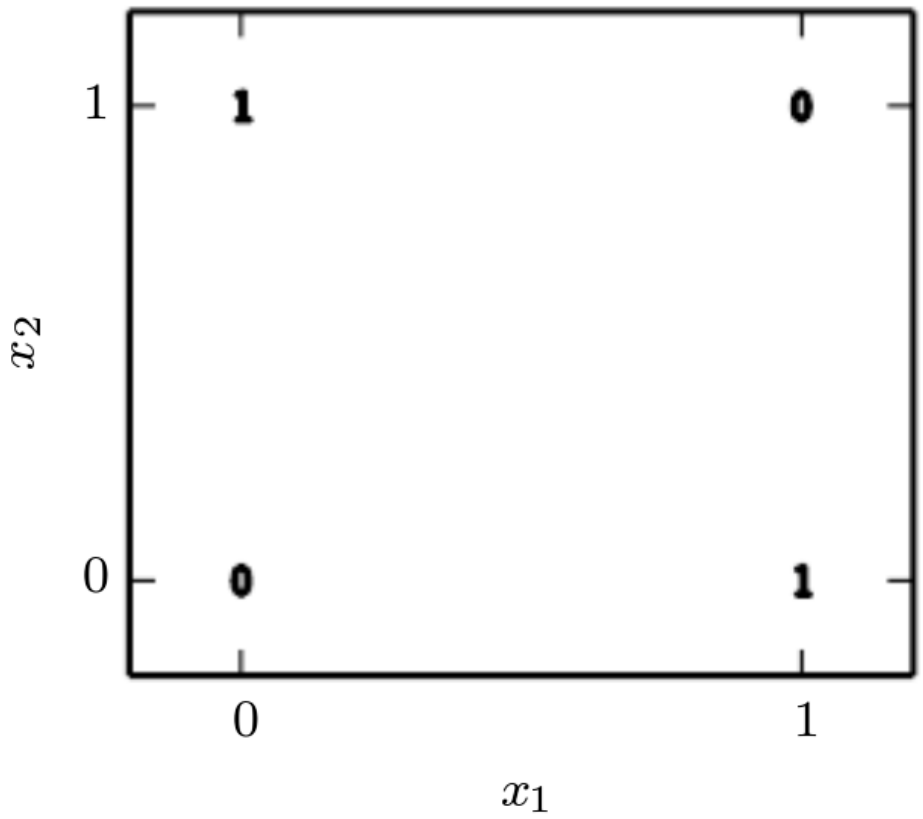
"Neural" Networks

- Inspired by biological neurons (nerve cells)
- Neurons are connected to each other, and receive and send electrical pulses.
- *"If the [input] voltage changes by a large enough amount, an all-or-none electrochemical pulse called an action potential is generated, which travels rapidly along the cell's axon, and activates synaptic connections with other cells when it arrives."* (Wikipedia)



Activation Functions with Non-Linearities

- Linear Functions are limited in what they can express.
- Famous example: XOR
- Simple layered non-linear functions can represent XOR.



Outline

Design Choices for Output Units

- Typically can be interpreted as probabilities.
  - Logistic sigmoid
  - Softmax
  - mean and variance of a Gaussian, ...
- Trained with negative log-likelihood.

Softmax

- Logistic sigmoid
  - Vector $\boldsymbol{y}$ of binary outcomes, with no contraints on how many can be 1.
  - Bernoulli distribution.
- Softmax
  - Exactly one element of $\boldsymbol{y}$ is 1.
  - Multinoulli (categorical) distribution.

$$p(Y = i|\phi(\boldsymbol{x}))$$

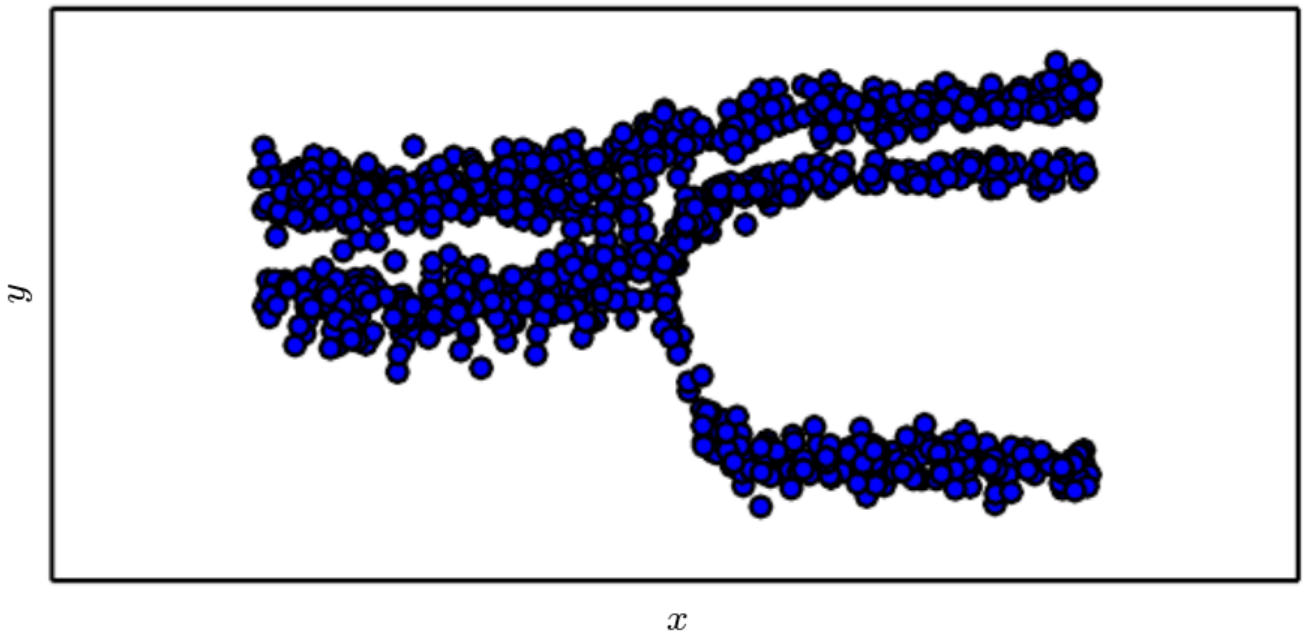
$$\sum_i p(Y = i|\phi(\boldsymbol{x})) = 1$$

$$softmax(\boldsymbol{z})_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Parametrizing a Gaussian Distribution

- Use final layer to predict parameters of Gaussian mixture model.
- Weight of mixture component: softmax.
- Means: no non-linearity.
- Precisions ($\frac{1}{\sigma^2}$) need to be positive: softplus

$$softplus(z) = ln(1 + exp(z))$$



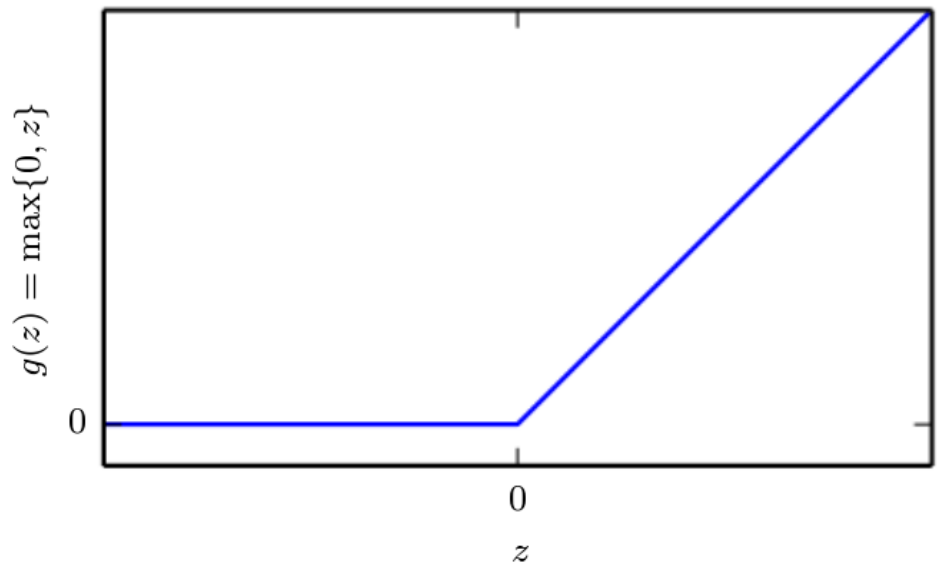
Outline

Rectified Linear Units

- Rectified Linear Unit:

$$relu(z) = max(0, z)$$

$$z = \boldsymbol{x}^T \boldsymbol{w} + b$$



- Consistent gradient of 1 when unit is *active* (i.e. if there is an error to propagate).
- Default choice for hidden units.

A Simple ReLU Network to Solve XOR

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}) = \boldsymbol{w}^T max(0, \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c})$$

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Other Choices for Hidden Units

- A good activation function aids learning, and provides large gradients.
- Sigmoidal functions (logistic sigmoid)
  - have only a small region before they flatten out in either direction.
  - Practice shows that this seems to be ok in conjunction with Log-loss objective.
  - But they don't work as well as hidden units.
  - ReLU are better alternative since gradient stays constant.
- Other hidden unit functions:
  - maxout: take maximum of several values in previous layer.
  - purely linear: can serve as low-rank approximation.

Summary

- Gradient descent: Minimize loss by iteratively substracting gradient from parameter vector.
- Stochastic gradient descent: Approximate gradient by considering small subsets of examples.
- Regularization: penalize large parameter values, e.g. by adding l2-norm of parameter vector.
- Feedforward networks: layers of (non-linear) function compositions.
- Output non-linearities: interpreted as probability densities (logistic sigmoid, softmax, Gaussian)
- Hidden layers: Rectified linear units ($max(0, z)$)