COMP0197
Applied Deep Learning

Andre Altmann

Department of Medical Physics and Biomedical Engineering

The UCL Hawkes Institute

a.altmann@ucl.ac.uk

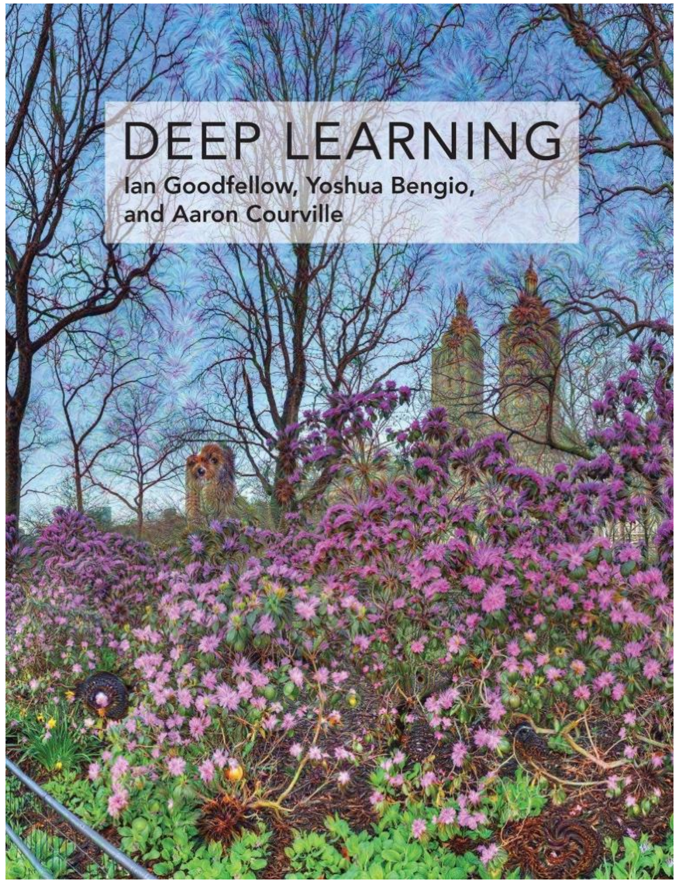
London, January 22nd 2025

Overview

A refresher on Linear Algebra and Probability Theory can be found in Chapters 2 and 3

Today:

- ➢Numerical Computation
- ➢Gradient based optimization
- ➢Parallel computing
- ➢Information Theory
- ➢Maximum Likelihood Estimation



Numerical Computation

Underflow and overflow

- We are using continuous math
- Trying to compute on a digital computer
  - Limited precision to represent numbers
  - Rounding errors potentiate
- Models that work in theory, but may fail in practice

Underflow and overflow

Underflow

- Values close to 0 are rounded to 0
  - $0.00000001 \rightarrow 0$
  - Accidental ‘division by 0’
  - Logarithm set to $-\infty$

Overflow

- Numbers with large magnitude are approximated to ∞ or $-\infty$

Example

- **softmax** function (used to turn 'outputs' 1,…,n into a probability distrib.):

$$\mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

- Assume all $x_i = c$
- Should be: $\frac{1}{n}$
- If c is very large negative: **underflow**
  - Division by 0 (undefined)
- If c is very large positive: **overflow**

Example II

- Avoid overflow
  compute: $\mathrm{softmax}(\boldsymbol{z}), with\ \boldsymbol{z} = \boldsymbol{x} - \max_i x_i$

$$\mathrm{softmax}(x)_i = \frac{\exp(x_i - \max_i x_i)\ \exp(\max_i x_i)}{\sum_{j=1}^{n} \exp(x_j - \max_i x_i)\ \exp(\max_i x_i)}$$

- Avoid underflow in the numerator.
  (problem for log $\mathrm{softmax}(x) \rightarrow -\infty$)
  make log softmax ‘stable’ using same approach
- Common underflow/overflow issues dealt with by libraries

Example III

- Working with probabilities
- $p(x) \in [0,1]$

- We often computed products of probabilities:

$$\prod p(x^{(i)})$$

- Quickly converges to something close to 0 (**underflow**)

- Work instead with *log* probabilities:

$$\sum \log p(x^{(i)})$$

Poor Conditioning

- Conditioning: *"how rapid does a function changes with small changes to the input"*
- Think: $\frac{f(x+\epsilon)}{f(x)}$ or $f(x+\epsilon) - f(x)$
    - Rounding errors can have a huge impact in poor conditioning

- Example: $f(\boldsymbol{x}) = \boldsymbol{A}^{-1}\boldsymbol{x}; \boldsymbol{A} \in \mathbb{R}^{n \times n}$ has eigenvalue decomposition $(\lambda_1, \dots, \lambda_n)$
- **Condition number**: $\max_{i,j} \left[\frac{\lambda_i}{\lambda_j}\right]$
- Large condition number → matrix inversion sensitive to errors in input

Lipschitz continuous

Deep Learning functions are complex

Can get guarantees (on bounds, convergence etc.) if functions (or their derivatives) are **Lipschitz continuous**:

$$\forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L} \|\boldsymbol{x} - \boldsymbol{y}\|_2$$

Lipschitz constant: $\mathcal{L}$

Gradient based optimization

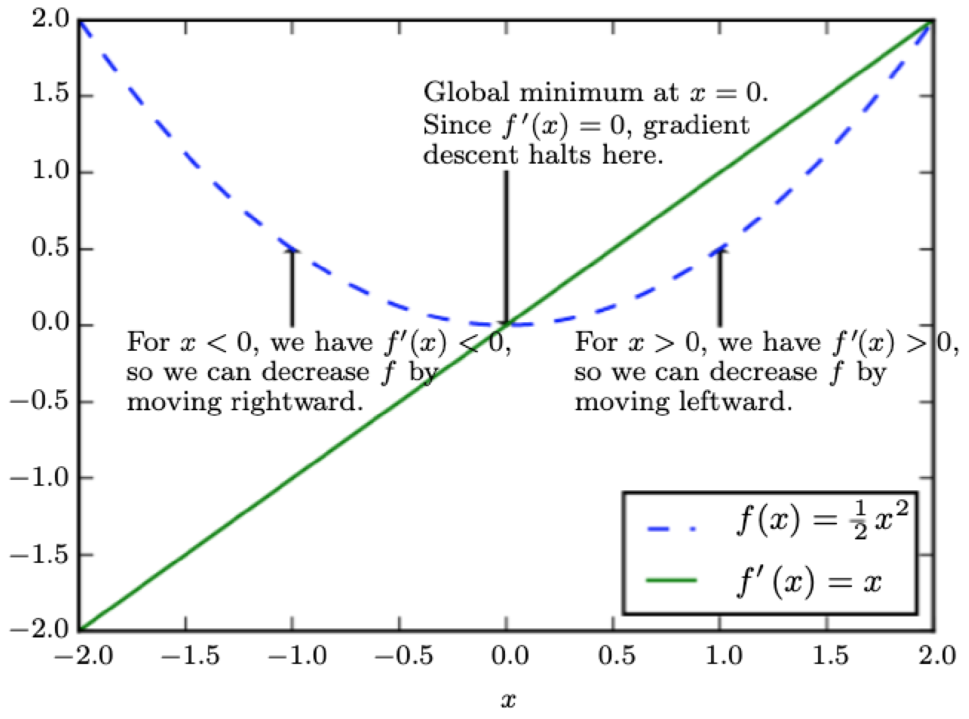
Gradient based optimization

- Learning involves maximizing or minimizing some function $f(\boldsymbol{x})$
  - **Objective function**, **criterion**, **cost function**, **loss function** or **error function**
  - $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$

- With linear regression we were *lucky* – analytical solution exists to find the **global minimum**

Gradient based optimization

- We require the **derivative** of the function
- $y = f(x)$ denoted as $f'(x)$ or $\frac{dx}{dy}$
- The derivative provides the *slope* at the point x

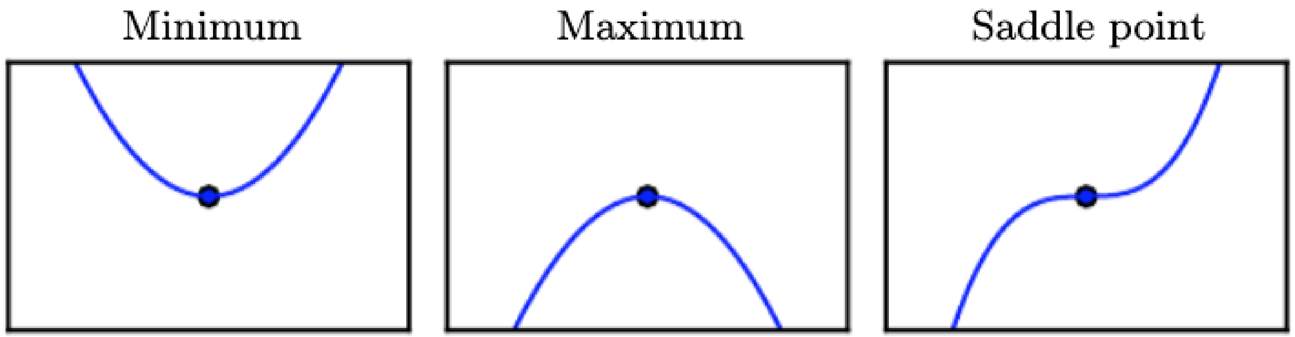


Gradient based optimization

- The slope $f'(x)$ tells us in which direction of x we must move to make $f(x)$ smaller

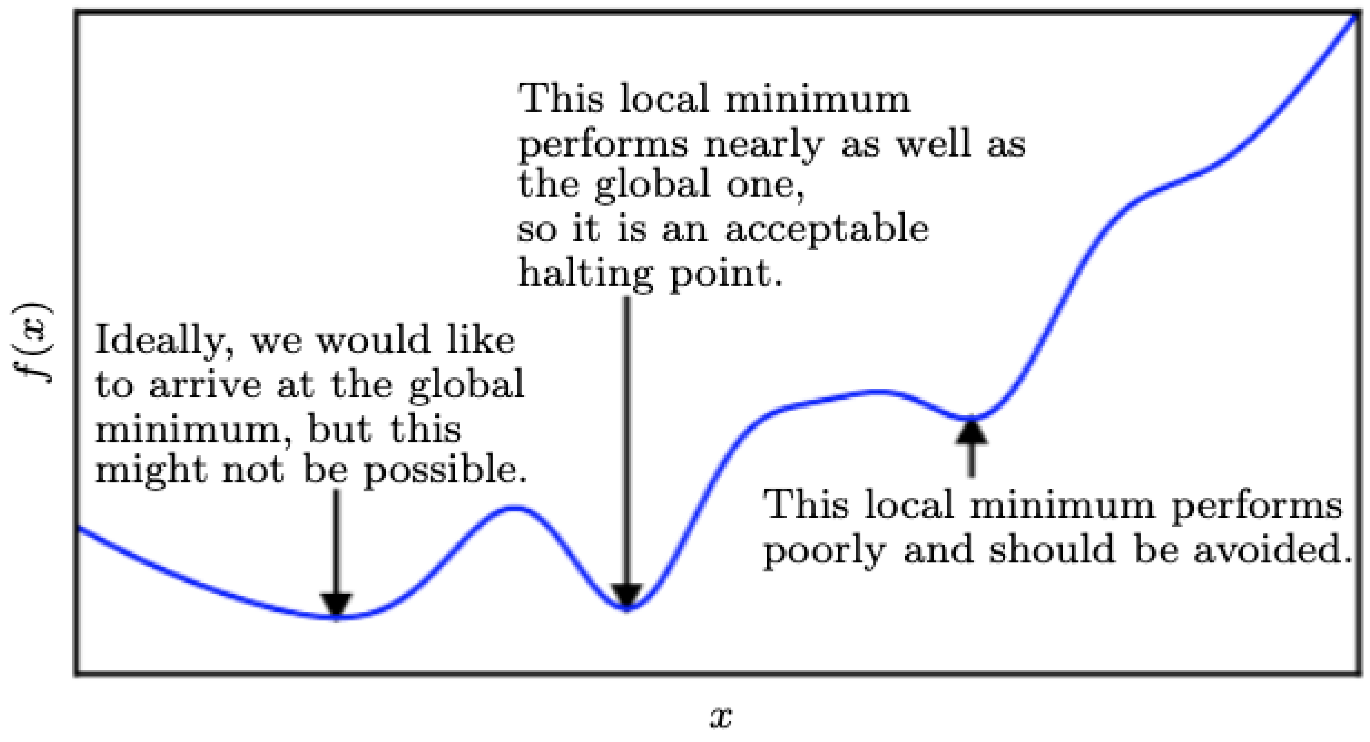
- Roughly: $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$

- Points of interest: $f'(x) = 0$



Gradient based optimization

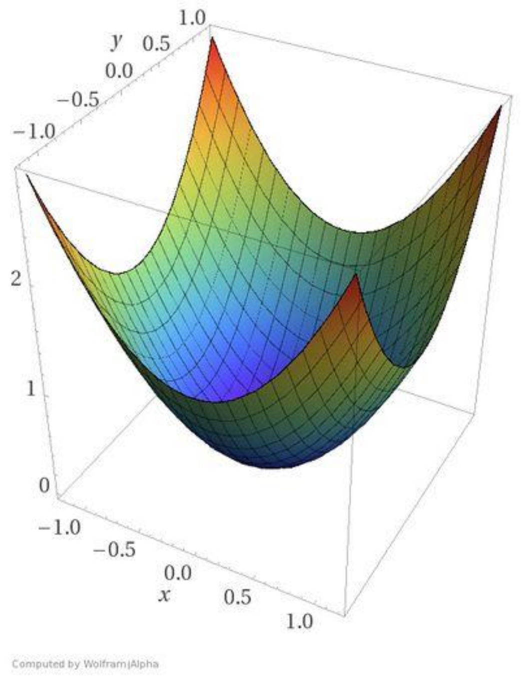
- Convex functions lack saddle points and have only one minimum
- DL functions are not ‘convex’ but have many local minima

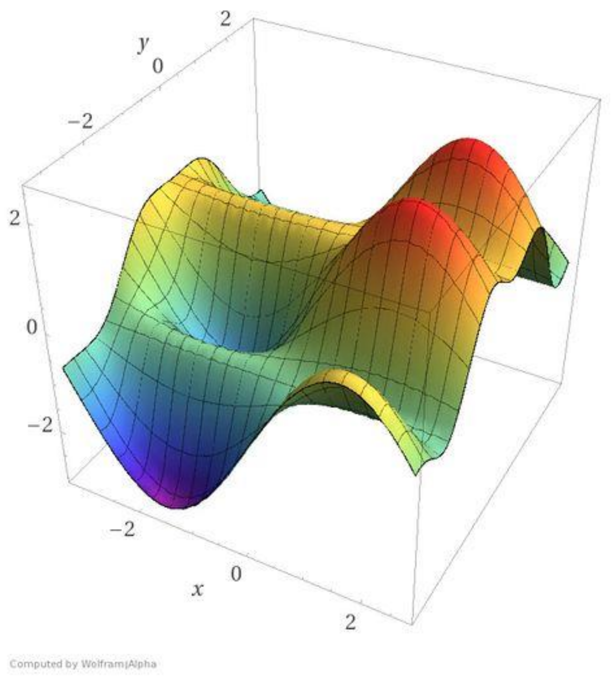


Gradient based optimization

- Functions we work with have multiple input dimensions

$$f(\boldsymbol{x}) = y$$
$$f\colon \mathbb{R}^n \to \mathbb{R}$$





Gradient based optimization

- For functions with multiple input dimensions we compute the **partial derivative** for each input dimension

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x})$$

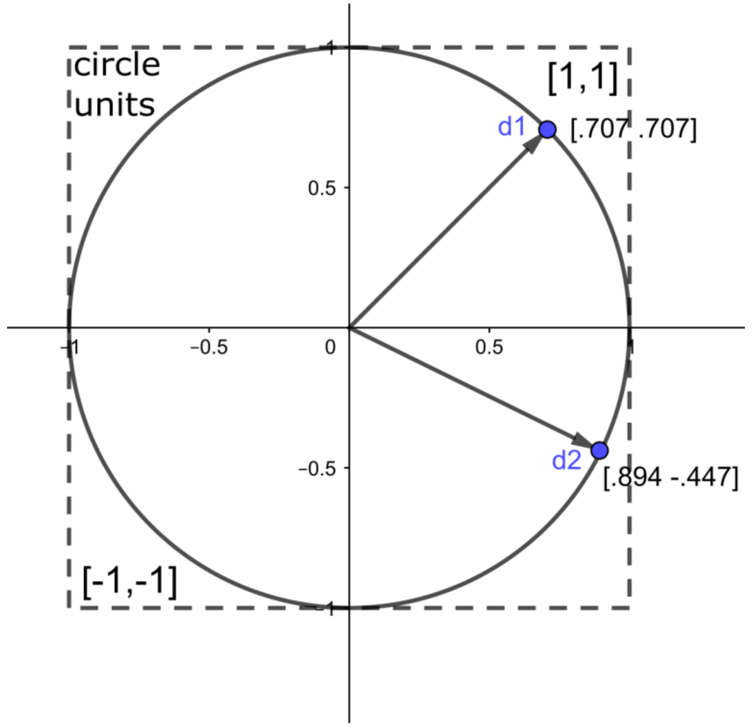
- The slope becomes a **gradient:** the vector of all partial derivatives:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

Gradient based optimization

- To find the minimum we want to move into the direction where f decreases the fastest
- We use the 'small' step trick as before
- Let $\boldsymbol{u}$ be a unit vector ($\|\boldsymbol{u}\|_2 = 1$)
- $f(\boldsymbol{x} + \alpha\boldsymbol{u})$
  - $\frac{\partial}{\partial\alpha} f(\boldsymbol{x} + \alpha\boldsymbol{u})$ at $\alpha = 0 \rightarrow \boldsymbol{u}^T \nabla_{\boldsymbol{x}} f(x)$
- To find the best direction:

$$\begin{aligned} &\min_{\boldsymbol{u},\boldsymbol{u}^T\boldsymbol{u}=1} \boldsymbol{u}^T \nabla_{\boldsymbol{x}} f(x) \\ &= \min_{\boldsymbol{u},\boldsymbol{u}^T\boldsymbol{u}=1} \|\boldsymbol{u}\|_2 \|\nabla_{\boldsymbol{x}} f(x)\|_2 \cos\theta \\ &= \min_{u} \cos\theta \end{aligned}$$



Opposite direction of the gradient.
-> (cos (180º) = -1).

Gradient based optimization

- Update rule:

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

- Learning rate: ϵ
  - Small constant
  - Evaluate $f\big(\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\big)$ for multiple ϵ (**line search**)

- Stopping criteria:
  - Change after update is very small: $f(\boldsymbol{x}) - f(\boldsymbol{x}') < c$
  - After a fixed number of iterations

Beyond gradients

- Assume multiple outputs $\boldsymbol{f}\colon\ \mathbb{R}^m \to \mathbb{R}^n$
- **Jacobian matrix** $\boldsymbol{J} \in \mathbb{R}^{n\times m}$: $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$

- **Second derivative** (curvature) of $f\colon\ \mathbb{R}^n \to \mathbb{R}$
  - Many functions – represented as **Hessian matrix:**

$$\mathbf{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x})$$

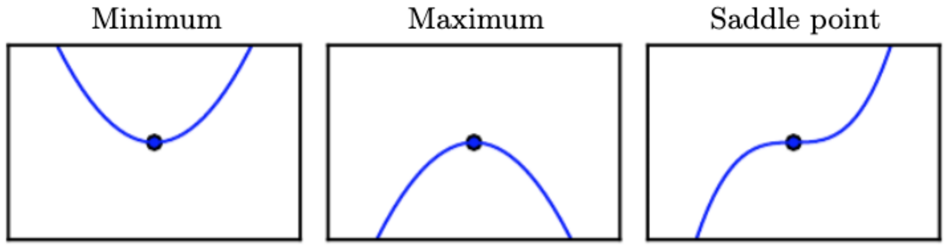
  - <u>Hessian is simply the Jacobian of the gradient.</u>

Beyond gradients

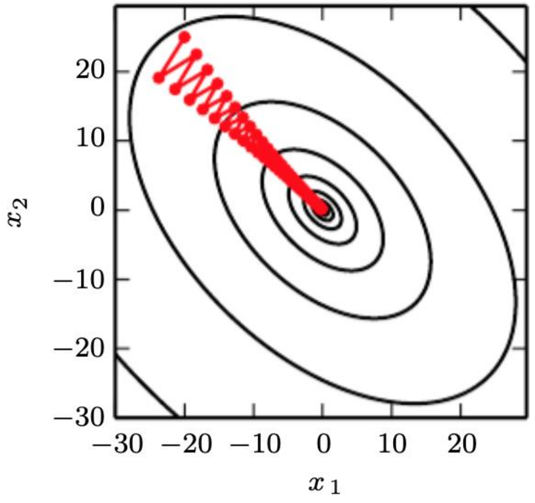
- Use **H** to optimize learning rate ($\boldsymbol{g}$ is the gradient)

$$\epsilon^* = \frac{\boldsymbol{g}^{\mathrm{T}}\boldsymbol{g}}{\boldsymbol{g}^{\mathrm{T}}\mathbf{H}\boldsymbol{g}}$$

- Decide whether extreme point is local minimum/maximum



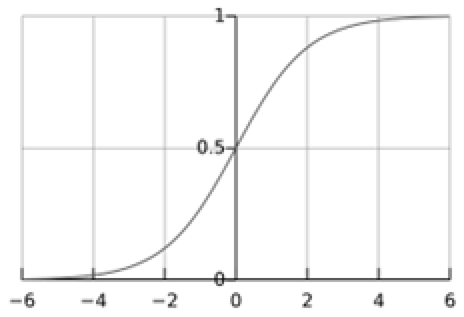
- Use in Newton's method to find critical points
- $\boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}(f)(\boldsymbol{x})^{-1}\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$

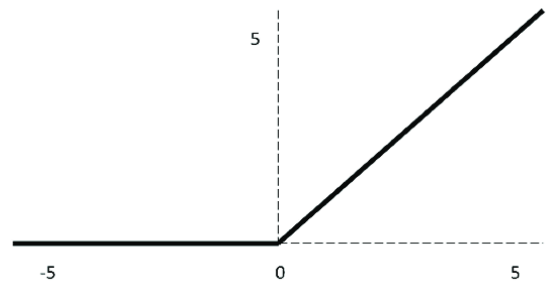


Happens when **H** has a large **conditioning** number.

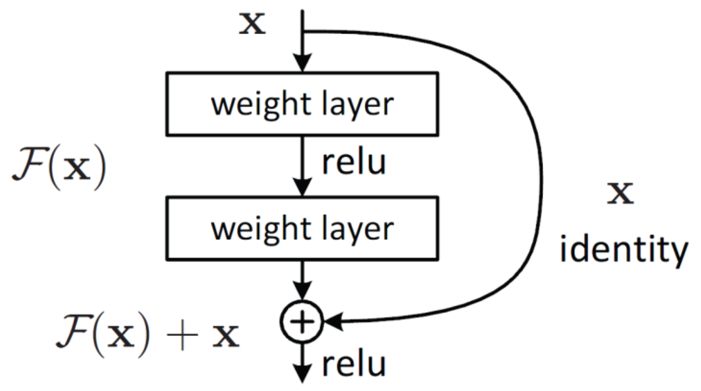
Vanishing/Exploding Gradients

- Mathematically all well-defined
- Practically: vanishing/exploding gradients due underflow/overflow





- How to avoid:
  - Choice of Activation Function (ReLu vs Sigmoid)
  - Change Architecture (ResNet)
  - Weight Initialization
  - Batch Normalization
  - Gradient Clipping



Machine learning so far …

- We were optimizing a **loss** function, under **constraints**

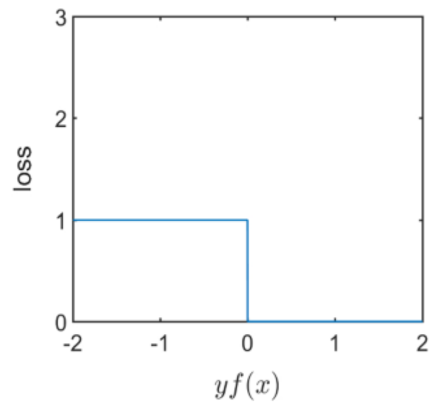
$$J(\theta) = L(\widehat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda\Omega(\boldsymbol{\theta})$$

- The loss is defined over the training dataset

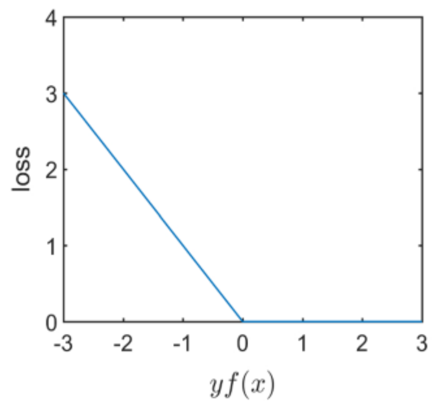
- To optimize $J(\theta)$ with gradient based methods, we prefer it to be differentiable
  - Loss has to be differentiable
  - Regularizer has to be differentiable

Examples of loss functions

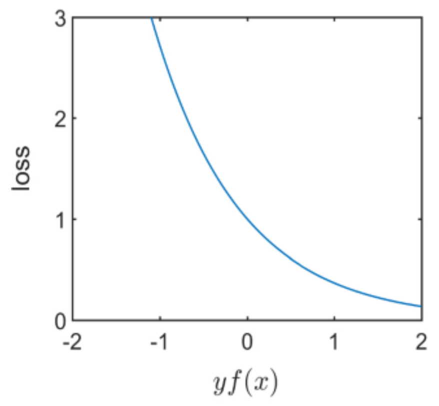
- E.g., in classification



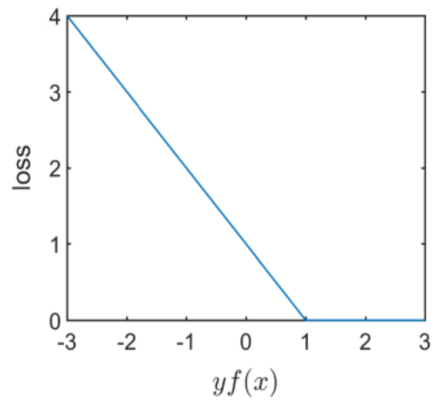
(a) 0-1 loss



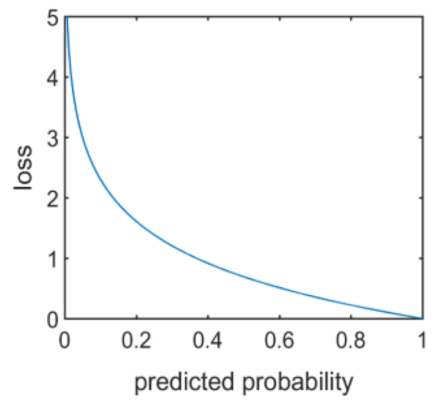
(b) perceptron loss



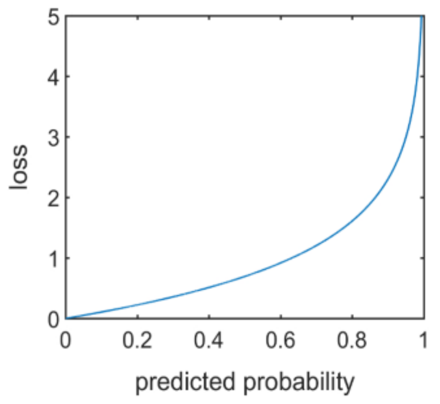
(e) exponential loss



(f) hinge loss



(c) logarithmic loss(label=1)



(d) logarithmic loss(label=-1)

Parallel Computing

Parallel Computing

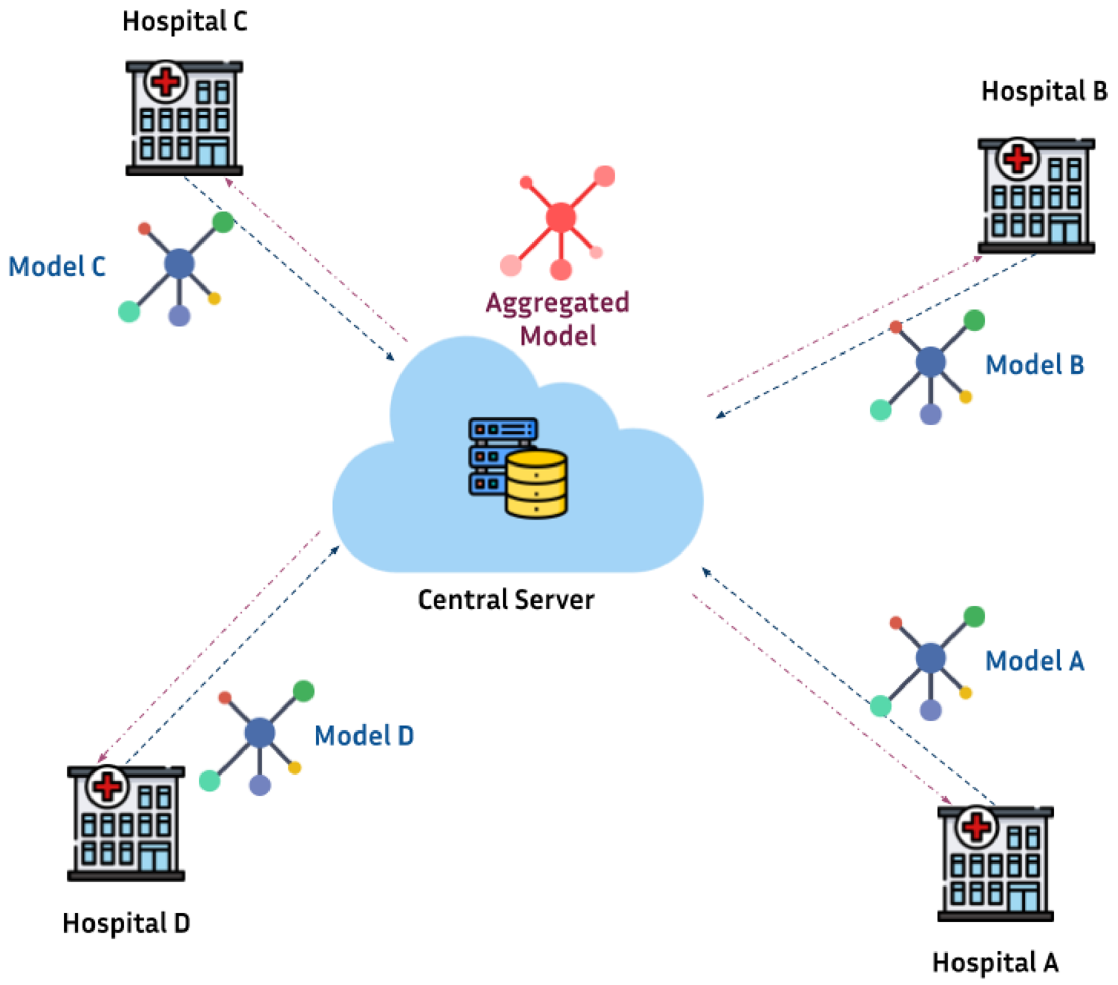
- Some tasks in ML and DL can be easily ‘*distributed*’ across multiple CPUs
  - <u>Dependency free:</u> individual folds in CV, grid search (hyperparameter tuning), apply a trained model to new data, …

- Underlying linear algebra (LA) tasks can be parallelized:
  - Matrix multiplication, inversion, singular value decomposition
  - Convolutions
  - GPUs!

Parallel Computing

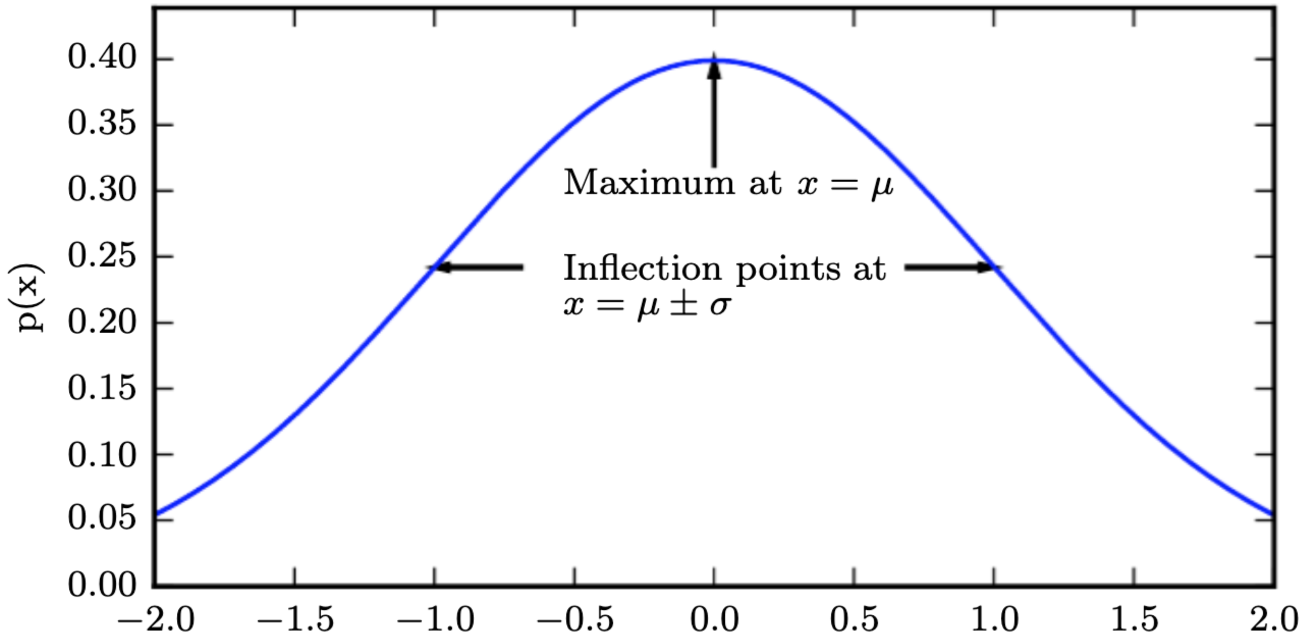
- With very large datasets data is distributed
  - Compute statistics locally (e.g., mean) and combine centrally
  - Same works for gradients used in gradient descent:
    $$\boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
  - Say, data is split into two sites, with m_1 and m_2 samples:
    $$J_k(\boldsymbol{\theta}) = \sum_{i=1}^{m_k} L\left(f\left(x^{(i)}, \theta\right), y_i\right) \to J(\boldsymbol{\theta}) = J_1(\boldsymbol{\theta}) + J_2(\boldsymbol{\theta})$$
    $$\Rightarrow \boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon\left(\nabla_{\boldsymbol{\theta}} J_1(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J_2(\boldsymbol{\theta}) \right)$$

Parallel Computing



Information Theory

Gaussian distribution



$$\mathcal{N}(x;\mu,\sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right).$$

$$\mathcal{N}(x;\mu,\beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x-\mu)^2\right).$$

precision

Information theory

- How much information present in a signal
- Many applications (noisy signals, etc.)

- Here:
  - Characterize probability distributions
  - Quantify similarity between probability distributions

- Intuition:
  - Observing a ‘*rare*’ event carries more information than a ‘*common*’ event

Information theory

- Criteria:
  - Likely events should have low information content
  - Less likely events should have higher information content
  - Independent events should have additive information

- **Self-information** (event $\mathrm{x} = x$):

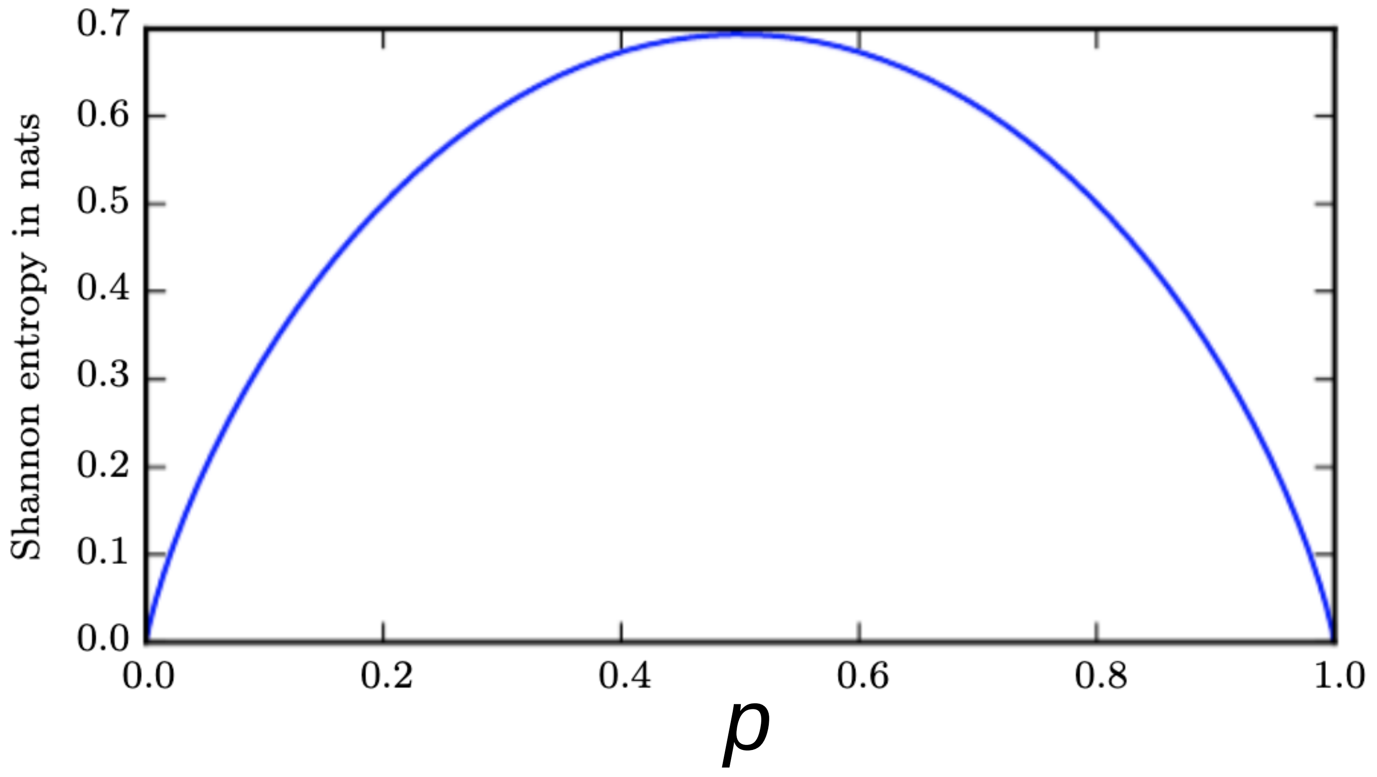
$$I(x) = -\log P(x)$$

- Unit: **nat** $= \frac{\mathbf{1}}{\boldsymbol{e}}$ (as opposed to **bits/shannons**)
- **Shannon entropy**:

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)]$$

Information theory

- Shannon entropy for a binary variable



KL-divergence

- Two distributions over x: $P(\mathrm{x})$ and $Q(\mathrm{x})$
- We can measure the difference ("*similarity*") of the distributions

$$D_{KL}(P \parallel Q) = \mathbb{E}_{\mathrm{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x}\sim P}[\log P(x) - \log Q(x)]$$

Kullback-Leibler (KL) divergence

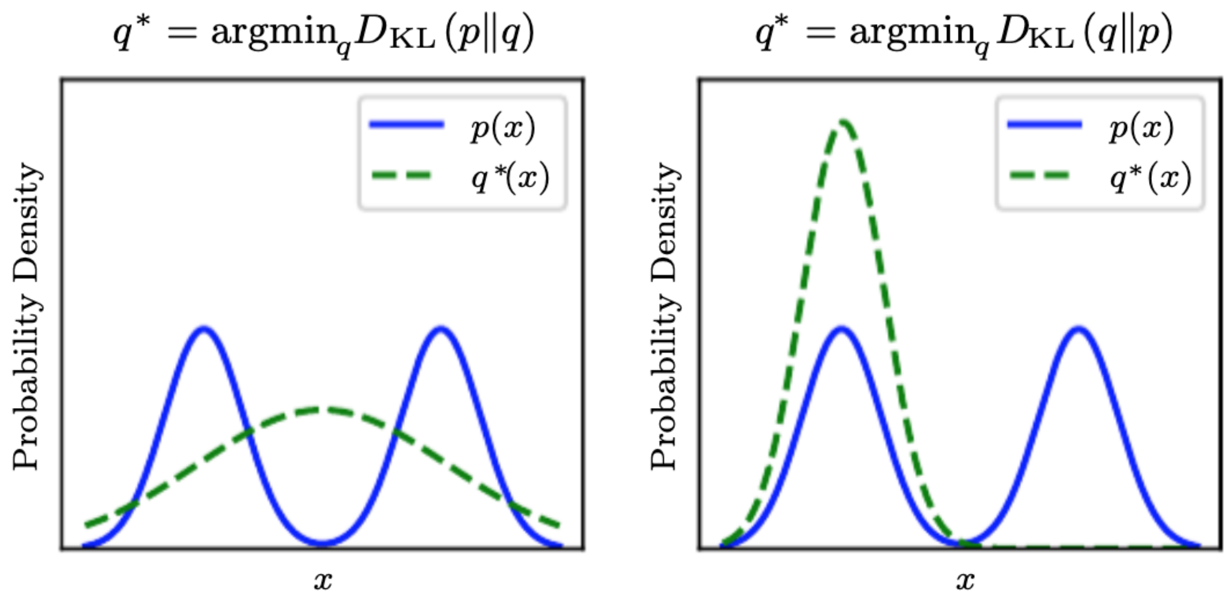
- Non-negative, only 0 iff *P* = *Q*
- "distance between distributions"

KL-divergence

- Not symmetric:

$$D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$$

- Choice important for applications



- Used for **loss/cost** function when matching distributions

Cross-entropy

Closely related to KL-divergence

Popular loss function for classification

Cross-entropy

$$H(P, Q) = H(P) + D_{KL}(P \parallel Q)$$
$$= -\mathbb{E}_{\mathrm{x} \sim P} \log Q(x)$$

- In the discrete case:

$$H(P, Q) = -\sum_{x \in X} P(x) \log Q(x) \qquad 0 \log 0 \coloneqq 0$$

Maximum Likelihood Estimation

Machine learning so far …

- We were optimizing a **loss** function, under **constraints**

$$J(\theta) = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda\Omega(\boldsymbol{\theta})$$

- Now we view the problem as finding the parameters that generated the data

- Common approach: **maximum likelihood (ML)**

Maximum Likelihood

- Set of *m* examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(m)}\}$
- Samples were drawn independently from $p_{\text{data}}(\boldsymbol{x})$
- We have a family of probability distributions:

$$p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

- ML: find parameter $\boldsymbol{\theta}$ that explains the observed data best

$$\begin{aligned}\boldsymbol{\theta}_{\text{ML}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})\end{aligned}$$

Maximum Likelihood

- Prod. of many numbers computationally problematic (underflow), use *log:*

$$\boldsymbol{\theta}_{\mathrm{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_i \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- Way to obtain estimates for μ and σ^2 for Gaussian distributions

- Another way to write this:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{x\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

Maximum Likelihood

- ML attempts to minimize the '*dissimilarity*' between the empirical distribution $\hat{p}_{\text{data}}$ and the model distribution p_{model}

- **Kullback–Leibler (KL) divergence:**
$D_{\text{KL}}(\hat{p}_{\text{data}}||p_{\text{model}}) = \mathbb{E}_{x\sim\hat{p}_{\text{data}}}[\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})]$

- To minimize KL-divergence we need only to minimize:
$-\ \mathbb{E}_{x\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\boldsymbol{x})]$

Maximum Likelihood

- We can do the same for conditional probabilities
  - Model dependence of output y on input $\boldsymbol{x}$
  - $P(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ – basis for supervised learning

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta})$$

- Assume i.i.d. samples:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P\left(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

- We can use this to train Linear Regression.

Linear Regression – ML

- We assume a conditional distribution:

$$p(y|\boldsymbol{x}) = \mathcal{N}(y; \hat{y}(\boldsymbol{x}, \boldsymbol{w}), \sigma^2)$$

- $\hat{y}(\boldsymbol{x}, \boldsymbol{w})$ provides the mean based on input and model parameters

- We assume fixed variance (independent of the input)

Linear Regression – ML

- Examples are assumed i.i.d., conditional log-likelihood:

$$\sum_{i=1}^{m} \log p\left(y^{(i)} \middle| x^{(i)}; \boldsymbol{\theta}\right)$$

$$= -m \log \sigma - \frac{m}{2} \log 2\pi - \sum_{i=1}^{m} \frac{\left\| \hat{y}^{(i)} - y^{(i)} \right\|^2}{2\sigma^2}$$

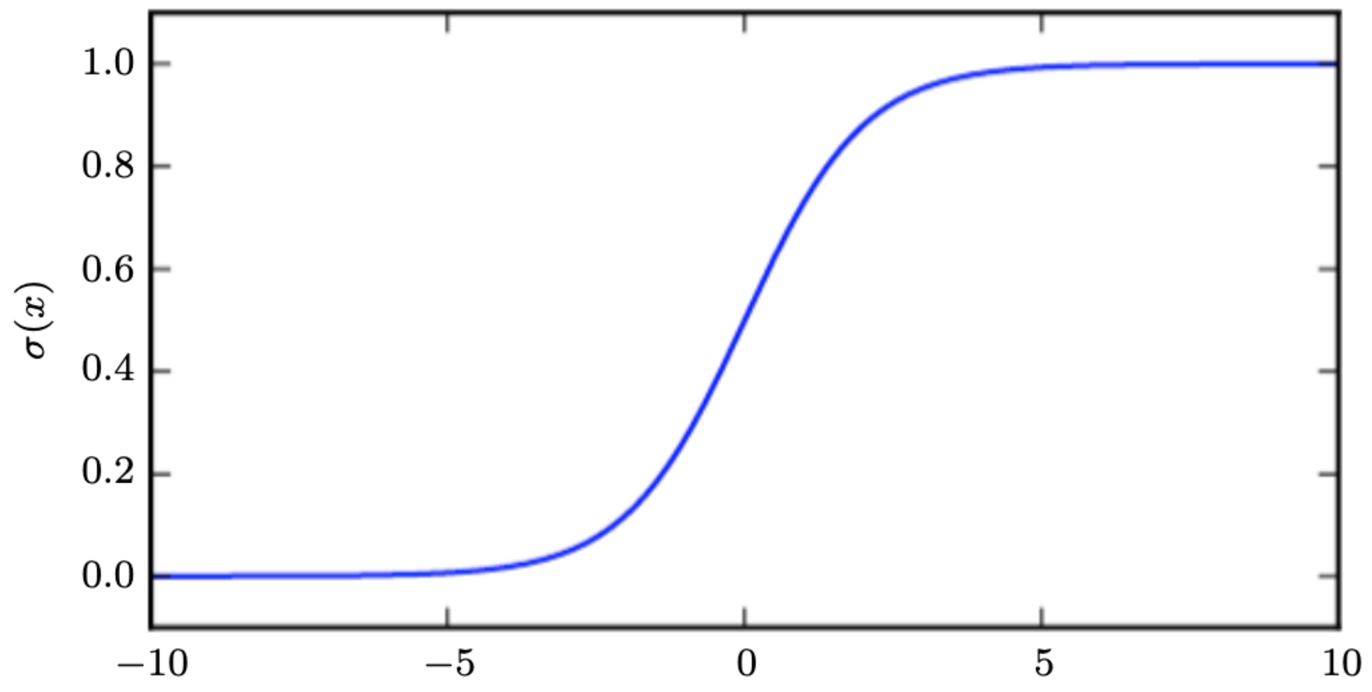
- For reference:

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m} \sum_{i=1}^{m} \left\| \hat{y}^{(i)} - y^{(i)} \right\|^2$$

Questions

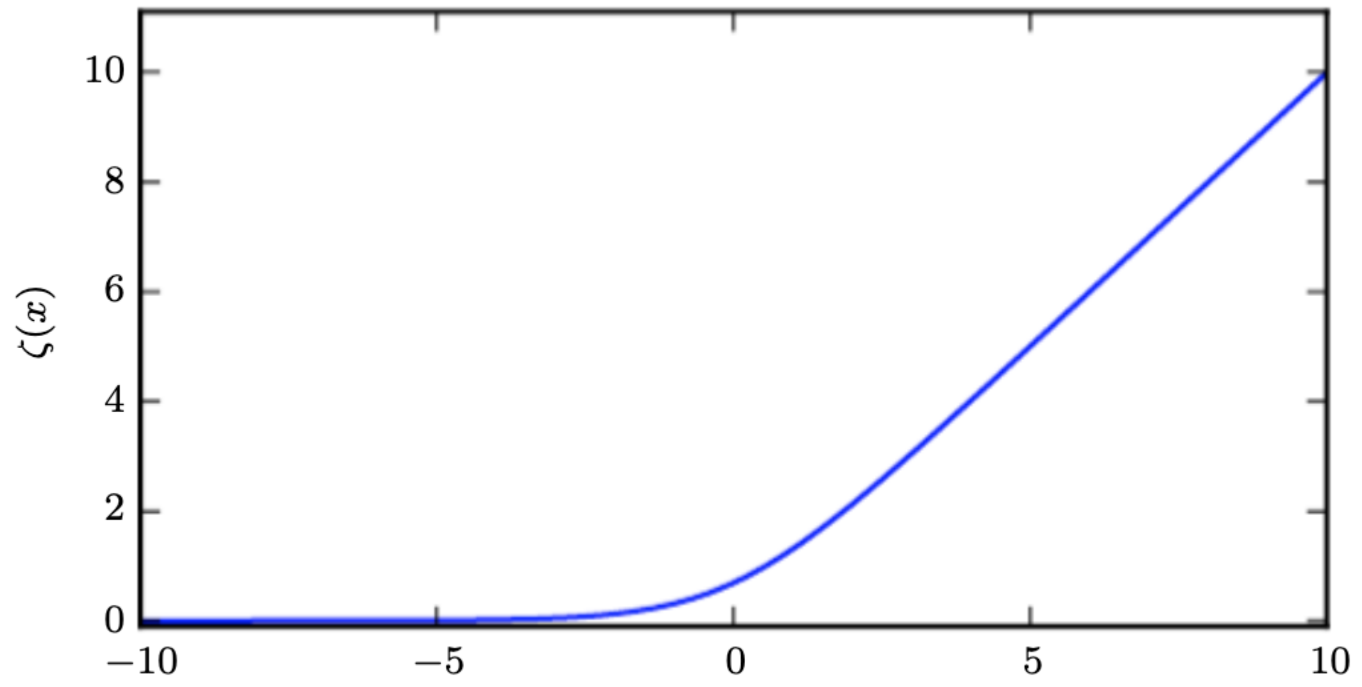
Thank you!

(logistic) sigmoid



$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

softplus



$$\zeta(x) = \log(1 + \exp(x))$$