Artificial Intelligence II: Deep learning methods

Ana Neacșu & Vlad Vasilescu

Lecture 2: Linear Networks

National University of Science and Technology POLITEHNICA Bucharest, Romania
BIOSINF Master Program

March 2024

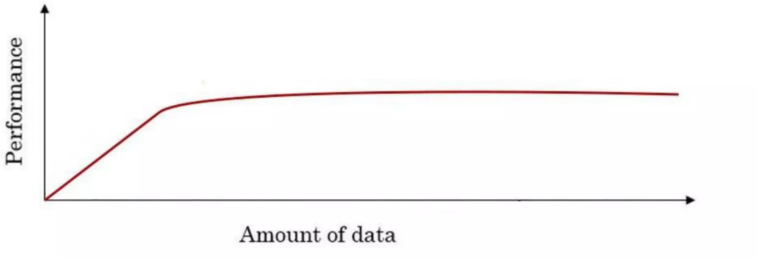
Overview

1. Why Deep Learning?
2. Convex Optimization Recap
3. What is a neural network?
4. Linear Neural Networks
5. Gradient descent

Why Deep Learning?

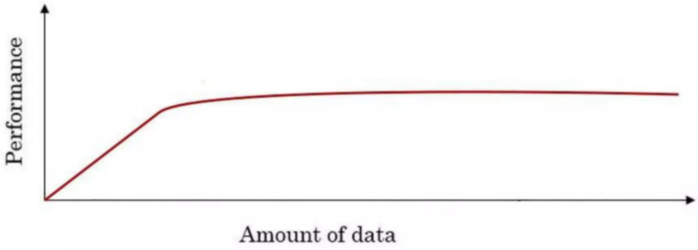
The rise of Deep Learning

- If the basic technical idea behind deep learning neural networks has been around for decades, why are they only taking off now?
- If we plot the performance of traditional algorithms such as SVM or Logistic Regression as function of the amount of data, we will get the following curve:



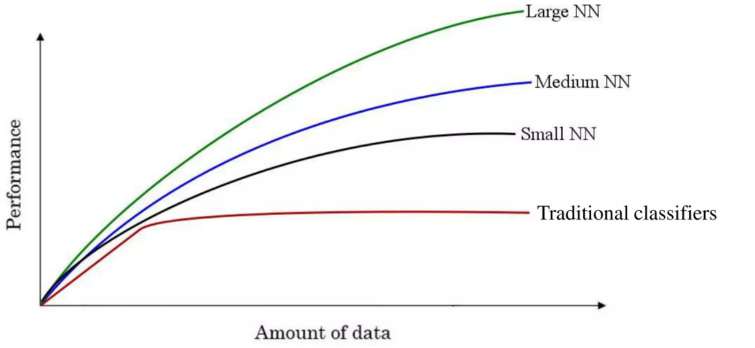
The rise of Deep Learning

- If the basic technical idea behind deep learning neural networks has been around for decades, why are they only taking off now?
- If we plot the performance of traditional algorithms such as SVM or Logistic Regression as function of the amount of data, we will get the following curve:



The rise of deep learning

- How to overcome performance plateau problem?



Why is Deep Learning working now

- Over the last 20 years we accumulated more data for applications than traditional learning algorithms were able to effectively take advantage of
- GPU (speed of processing) + Data
- Theoretical understandings of the difficulty of training deep networks (from 2006)

Libraries allow to easily implement/test/deploy neural networks :

- Torch (Lua) / PyTorch (Python/C++), Caffe(C++/Python), Caffe2 (RIP 2018)
- Microsoft CNTK
- Google Tensorflow / Keras
- Theano/Lasagne (Python, RIP 2017)
- CNTK, Chainer, Matlab, Mathematica, ...

Why is Deep Learning working now

- Over the last 20 years we accumulated more data for applications than traditional learning algorithms were able to effectively take advantage of
- GPU (speed of processing) + Data
- Theoretical understandings of the difficulty of training deep networks (from 2006)

Libraries allow to easily implement/test/deploy neural networks :

- Torch (Lua) / PyTorch (Python/C++), Caffe(C++/Python), Caffe2 (RIP 2018)
- Microsoft CNTK
- Google Tensorflow / Keras
- Theano/Lasagne (Python, RIP 2017)
- CNTK, Chainer, Matlab, Mathematica, ...

Convex Optimization Recap

The Cauchy-Schwarz inequality

Let $\mathbf{u}$, $\mathbf{v} \in \mathbb{R}^d$. The Cauchy-Schwartz inequality reads:

$$\mid \mathbf{u}^\top \mathbf{v} \mid \leq \mid\mid \mathbf{u} \mid\mid \, \mid\mid \mathbf{v} \mid\mid .$$

Some notations

- $\mathbf{u} = (u_1, \ldots, u_d)^\top$, $\boldsymbol{v} = (v_1, \ldots, v_d)^\top$ d-dimensional column vectors with real entries.
- $\mathbf{u}^\top$, transpose of $\mathbf{u}$, a d-dimensional row vector
- $\mathbf{u}^\top \mathbf{v} = \sum_{i=1}^d u_i v_i$, scalar (or inner) product of $\mathbf{u}$ and $\mathbf{v}$.
- $\mid \mathbf{u}^\top \mathbf{v} \mid$, absolute value of $\mathbf{u}^\top \mathbf{v}$
- $\mid\mid \mathbf{u} \mid\mid = \sqrt{\mathbf{u}^\top \mathbf{u}} = \sqrt{\sum_{i=1}^d u_i^2}$, Euclidean ($\ell_2$) norm of $\mathbf{u}$.

Convex Sets

A set C is **convex** if the line segment between any two points from C lies in C, i.e. for any $\mathbf{x}, \mathbf{y} \in C$ and any $\lambda \in [0, 1]$, we have:

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C$$

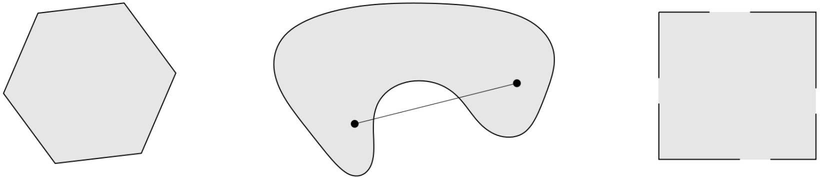


*Figure 2.2 from S. Boyd, L. Vandenberghe

Properties of convex sets

- Let $(C_i)_{i \in I}$ be convex sets, where I is a (possibly infinite) index set. Then $\cap_{i \in I} C_i$ is a convex set.
- Projections onto nonempty closed convex sets are *unique*, and *ususally* efficient to compute

$$\mathrm{P}_C(\mathbf{x}) := \mathrm{argmin}_{y \in C} ||\mathbf{y} - \mathbf{x}||$$

Convex functions

Definition: A function $f : \mathbb{R}^d \mapsto]-\infty, +\infty]$ is **convex** if:

1. $\mathbf{dom}(f) = \{x \in \mathbb{R}^d \mid f(x) < +\infty\}$ is a convex set;
2. $\forall \mathbf{x}, \mathbf{y} \in \mathbf{dom}(f), \lambda \in [0, 1] \quad f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$

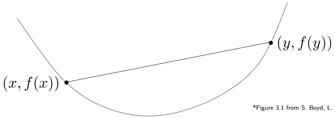


*Figure 3.1 from S. Boyd, L. Vandenberghe

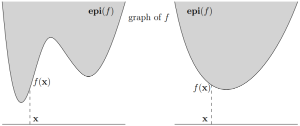
Convex functions and sets

- The **graph** of a function $f : \mathbb{R}^d \mapsto \mathbb{R}$ is defined as

$$\{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in \mathbf{dom}(f)\}$$

- The **epigraph** of a function $f : \mathbb{R}^d \mapsto \mathbb{R}$ is defined as

$$\mathbf{epi}(f) := \{(\mathbf{x}, \alpha) \in \mathbb{R}^d \times \mathbb{R} | \mathbf{x} \in \mathbf{dom}(f), \alpha \geq f(\mathbf{x})\}$$



A function is convex *iff* its epigraph is a convex set.

Examples of convex functions: $\mathbf{u} \in \mathbb{R}^d$, $b \in \mathbb{R}$

- Linear functions : $f(\mathbf{x}) = \mathbf{u}^\top \mathbf{x}$
- Affine functions : $f(\mathbf{x}) = \mathbf{u}^\top \mathbf{x} + b$
- Exponentials : $f(\mathbf{x}) = e^{\alpha \mathbf{x}}$

Question: Is norm $||\mathbf{x}||$ convex?

Convex Optimization

Convex Optimization Problems have the following form:

$$\min \quad f(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{x} \in C,$$

where

- f is a convex function
- C is a convex set

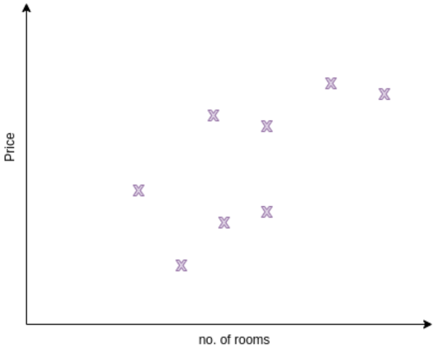
Properties :

- Every local minimizer is a **global minimizer**.
- For convex optimization problems, assuming f differentiable, all properly designed algorithms (Gradient Descent, Stochastic Gradient Descent, Projected and Proximal Gradient Descent)
  do **converge** to a global minimizer (which is not necessarily unique)!

What is a neural network?

Housing price prediction – binary case

Problem : we want to predict the price of a house based on the number of rooms.

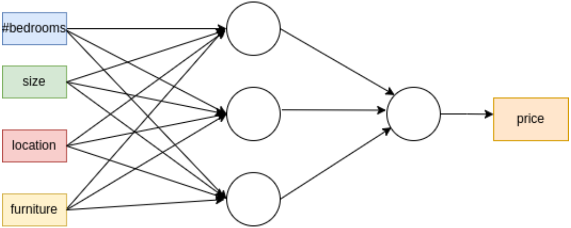


Possible solutions

- Support vector machine (SVM)
- Linear regression
- One neuron

Housing price prediction – complex case

- The previous scenario was not realistic.
- Usually, there are many more factors we should take into consideration

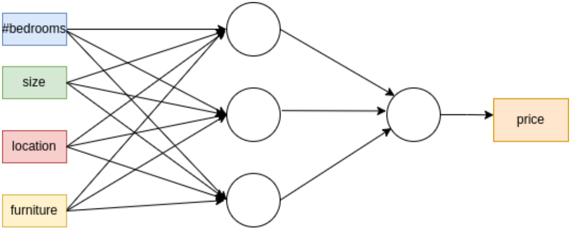


Observations:

- *A neural network can combine all this information*
- Each factor can influence differently the final decision

Housing price prediction – complex case

- The previous scenario was not realistic.
- Usually, there are many more factors we should take into consideration



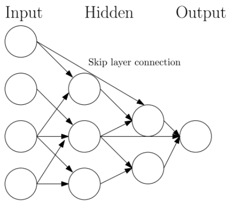
Observations:

- *A neural network can combine all this information*
- Each factor can influence differently the final decision

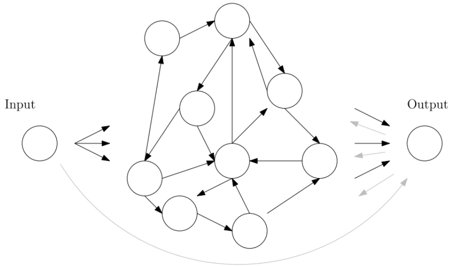
Definition

A neural network is a directed graph:

- nodes : computational units
- edges : weighted connections



Feedforward NN



Recurrent NN

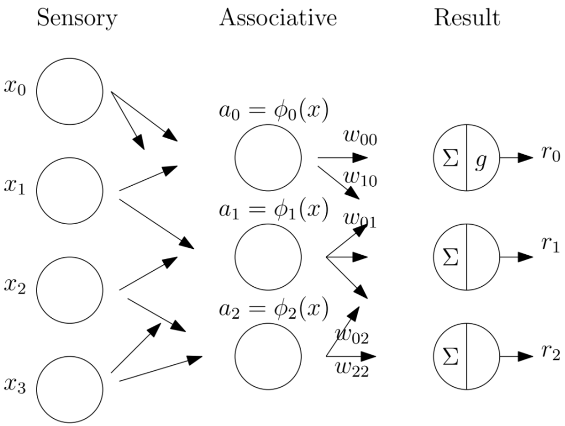
There are two possible types of graphs

- no cycle : feedforward neural network
- with at least one cycle : recurrent neural networks

Linear Neural Networks

The Perceptron (ROSENBLATT, 1958)

- Classification problem: given the pair $(x, y) \in \mathbb{R}^n \times \{-1, 1\}$
- Sensory - Associative - Response architecture, $\phi_j(x)$ with $\phi_0(x) = 1$
- The algorithm also has a geometrical interpretation



SAR Architecture

The classifier

Given fixed, predefined feature functions ϕ_j with $\phi_0(x) = 1, \forall x \in \mathbb{R}^n$, the perceptron classifies the input x as:

$$y = g(w^\top \Phi(x)) \tag{1}$$

$$g(x) = \begin{cases} -1 & \text{if} \quad x < 0 \\ +1 & \text{if} \quad x \geq 0, \end{cases} \tag{2}$$

with $\Phi(x) \in \mathbb{R}^{n_a+1}, \phi(x) = \begin{bmatrix} 1 \\ \phi_1(x) \\ \phi_2(x) \\ \vdots \end{bmatrix}.$

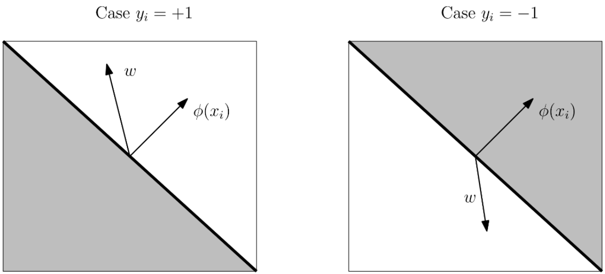
Training algorithm

Given $(x_i, y_i), y_i \in \{-1, 1\}$ the perceptron learning rule operates as follows:

$$w \leftarrow \begin{cases} w & \text{if the input is correctly classified} \\ w + \phi(x_i) & \text{if the input is incorrectly classified as} -1 \\ w - \phi(x_i) & \text{if the input is incorrectly classified as} +1 \end{cases} \tag{3}$$

Correct classification – geometrical interpretation

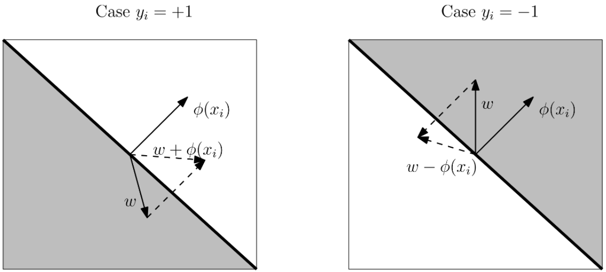
- Decision rule : $y = g(w^\top \Phi(x))$
- Algorithm : $w \leftarrow \begin{cases} w & \text{if the input is correctly classified} \\ w + \phi(x_i) & \text{if the input is incorrectly classified as } -1 \\ w - \phi(x_i) & \text{if the input is incorrectly classified as } +1 \end{cases}$



Correctly classified samples, as $+1$ and as -1

Incorrect classification – geometrical interpretation

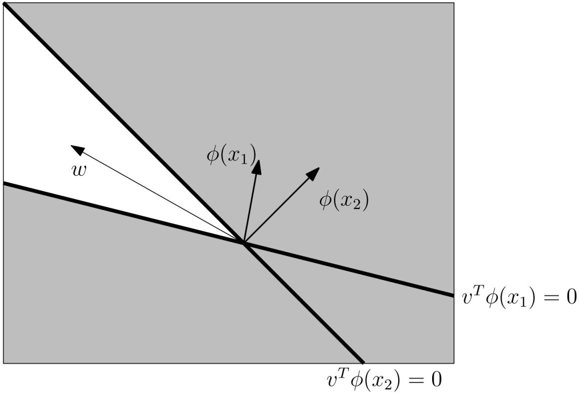
- Decision rule : $y = g(w^\top \Phi(x))$
- Algorithm : $w \leftarrow \begin{cases} w & \text{if the input is correctly classified} \\ w + \phi(x_i) & \text{if the input is incorrectly classified as} -1 \\ w - \phi(x_i) & \text{if the input is incorrectly classified as} +1 \end{cases}$



Incorrectly classified samples, $+1 \rightarrow -1$ and $-1 \rightarrow +1$

Multiple samples – Geometrical interpretation

- Decision rule : $y = g(w^\top \Phi(x))$
- Cone of feasibility : The intersection of the valid halfspaces (it may be empty)
- We consider two samples x_1, x_2 and $y_1 = +1, y_2 = -1$



The cone of feasibility for $y_1 = +1$ and $y_2 = -1$

Towards a canonical learning rule – **delta rule**

Given $(x_i, y_i), y_i \in \{-1, 1\}$ the perceptron learning rule operates as follows:

$$w = \begin{cases} w & \text{if the input is correctly classified} \\ w + \phi(x_i) & \text{if the input is incorrectly classified as} -1 \\ w - \phi(x_i) & \text{if the input is incorrectly classified as} +1 \end{cases} \tag{4}$$

$$w \leftarrow \begin{cases} w & \text{if} \quad g(w^\top \phi(x_i)) = y_i \\ w + \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) = -1 \quad \text{and} \quad y_i = +1 \\ w - \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) = +1 \quad \text{and} \quad y_i = -1 \end{cases} \tag{5}$$

$$w \leftarrow \begin{cases} w & \text{if} \quad g(w^\top \phi(x_i)) = y_i \\ w + y_i \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) \neq y_i \end{cases} \tag{6}$$

$$w = w + \frac{1}{2}(y_i - \hat{y}_i)\phi(x_i), \tag{7}$$

with $\hat{y}_i = g(w^\top \phi(x_i))$. This is called the **Delta Rule**.

Towards a canonical learning rule – **delta rule**

Given $(x_i, y_i), y_i \in \{-1, 1\}$ the perceptron learning rule operates as follows:

$$w = \begin{cases} w & \text{if the input is correctly classified} \\ w + \phi(x_i) & \text{if the input is incorrectly classified as} -1 \\ w - \phi(x_i) & \text{if the input is incorrectly classified as} +1 \end{cases} \tag{4}$$

$$w \leftarrow \begin{cases} w & \text{if} \quad g(w^\top \phi(x_i)) = y_i \\ w + \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) = -1 \quad \text{and} \quad y_i = +1 \\ w - \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) = +1 \quad \text{and} \quad y_i = -1 \end{cases} \tag{5}$$

$$w \leftarrow \begin{cases} w & \text{if} \quad g(w^\top \phi(x_i)) = y_i \\ w + y_i\phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) \neq y_i \end{cases} \tag{6}$$

$$w = w + \frac{1}{2}(y_i - \hat{y}_i)\phi(x_i), \tag{7}$$

with $\hat{y}_i = g(w^\top \phi(x_i))$. This is called the **Delta Rule**.

Towards a canonical learning rule – **delta rule**

Given $(x_i, y_i), y_i \in \{-1, 1\}$ the perceptron learning rule operates as follows:

$$w = \begin{cases} w & \text{if the input is correctly classified} \\ w + \phi(x_i) & \text{if the input is incorrectly classified as} -1 \\ w - \phi(x_i) & \text{if the input is incorrectly classified as} +1 \end{cases} \tag{4}$$

$$w \leftarrow \begin{cases} w & \text{if} \quad g(w^\top \phi(x_i)) = y_i \\ w + \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) = -1 \quad \text{and} \quad y_i = +1 \\ w - \phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) = +1 \quad \text{and} \quad y_i = -1 \end{cases} \tag{5}$$

$$w \leftarrow \begin{cases} w & \text{if} \quad g(w^\top \phi(x_i)) = y_i \\ w + y_i\phi(x_i) & \text{if} \quad g(w^\top \phi(x_i)) \neq y_i \end{cases} \tag{6}$$

$$w = w + \frac{1}{2}(y_i - \hat{y}_i)\phi(x_i), \tag{7}$$

with $\hat{y}_i = g(w^\top \phi(x_i))$. This is called the **Delta Rule**.

Convergence theorem

Definition

A binary classification problem $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ and $i \in [1 \dots N]$ is linearly separable if $\exists w \in \mathbb{R}^d$ such that

$$\forall i \quad \text{sign}(w^\top x_i) = y_i, \tag{8}$$

Theorem (Perception convergence theorem)

A binary classification problem $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ and $i \in [1 \dots N]$ is linearly separable *iff* perceptron learning rule converges to an optimal solution in a finite number of steps.

Proof : $\Leftarrow$: easy; $\Rightarrow$: we upper/lower bound $\|w_t\|_2^2$, where t is the index of the current iteration

Convergence theorem

Definition

A binary classification problem $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ and $i \in [1 \dots N]$ is linearly separable if $\exists w \in \mathbb{R}^d$ such that

$$\forall i \quad \text{sign}(w^\top x_i) = y_i, \tag{8}$$

Theorem (Perception convergence theorem)

A binary classification problem $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ and $i \in [1 \dots N]$ is linearly separable *iff* perceptron learning rule converges to an optimal solution in a finite number of steps.

Proof : $\Leftarrow$: easy; $\Rightarrow$: we upper/lower bound $\|w_t\|_2^2$, where t is the index of the current iteration

Observations

- $w_t = w_0 + \sum_{i \in \mathcal{S}_t} y_i \phi(x_i)$, with $\mathcal{S}_t$ the set of misclassified samples.
- The cost function to be minimised is: $J(w) = \frac{1}{M} \sum_i \max(0, -y_i w^\top \phi(x_i))$
- The solution:

$$w_t = w_0 + \sum_i \frac{1}{2}(y_i - \hat{y}_i)\phi(x_i),$$

where $(y_i - \hat{y}_i)$ is called the **prediction error**.

ADALINE

- Introduced by Widrow & Hoff in 1962.
- Let us consider the linear regression problem analytically.

Problem : Given $(x_i, y_i), x_i \in \mathbb{R}^{n+1}, y_i \in \mathbb{R}$, minimize

$$J(w) = \frac{1}{N} \sum_i \| y_i - w^\top x_i \|^2 .$$

Here-above we assume that $\forall i \quad x_i[0] = 1$ and $w[0]$ accounts for the bias term, $n \in \mathbb{N}^*$ is the input dimension, while $N \in \mathbb{N}^*$ is the total number of samples. Analytically, we can vectorize the expression: Assume $\mathbf{X} = [x_0|x_1|\ldots]$, then $J(w) = \| y - \mathbf{X}^\top w \|^2$.
$\nabla_w J(w) = 0 \Leftrightarrow \; -2(y - \mathbf{X}^\top w)^\top \mathbf{X}^\top = 0 \Rightarrow \mathbf{X}\mathbf{X}^\top w = \mathbf{X}$y

- $\mathbf{X}\mathbf{X}^\top$ non-singular : $w = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}y$
- $\mathbf{X}\mathbf{X}^\top$ singular (e.g. points along a line in 2D) $\rightarrow$ infinite no. solutions
  - Use regularized least square method to solve the problem:

$$\textbf{min} \quad G(w) = J(w) + \alpha w^\top w$$

  - $\nabla_w G(w) = 0 \Rightarrow (\mathbf{X}\mathbf{X}^\top + \alpha I_d)w = \mathbf{X}y$
  - if $\alpha \in (0, +\infty[\Rightarrow (\mathbf{X}\mathbf{X}^\top + \alpha I_d)$ is non-singular

Observation : We need to compute $\mathbf{X}\mathbf{X}^\top$ over the whole training set!

Linear regression with stochastic gradient descent

Problem : Given $(x_i, y_i), x_i \in \mathbb{R}^{n+1}, y_i \in \mathbb{R}$, minimize

$$J(w) = \frac{1}{N} \sum_i || \, y_i - w^\top x_i \, ||^2 \, .$$

Here-above we assume that $\forall i \quad x_i[0] = 1$ and $w[0]$ accounts for the bias term.

Algorithm 1: Stochastic Gradient Descent Algorithm

Output: Trained weights w

Initialize w_0 randomly;
for $t = 1$ **to** T **do**
    for $i = 1$ **to** N **do**
        $\hat{y}_i \leftarrow w_t^\top x_i$;
        $w_t \leftarrow w_{t-1} - \epsilon \nabla_w J(w_{t-1}) = w_{t-1} + \epsilon (y_i - \hat{y}_i) x_i$;
    endfor
endfor

where $\epsilon \in]0, \infty[$ is the learning rate.

Gradient descent

Batch Gradient Descent

The cost function to be minimized is

$$J(w, x, y) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(w, x_i, y_i),$$

where $\mathcal{L}$ is the *loss function*, e.g. $\mathcal{L}(w, x_i, y_i) = ||y_i - w^\top x_i||^2$

Batch gradient descent

- compute the gradient of the cost $J(w)$ over the whole training set
- performs one step in direction of $-\nabla_w J(w, x, y)$, so the update rule becomes:

$$w_{t+1} = w_t - \epsilon_t \nabla_w J(w, x, y)$$

Algorithm 2: Batch Gradient Descent Algorithm

Output: Trained weights w

Initialize w_0 randomly;

for $t = 1$ **to** T **do**

  Compute gradient $\nabla_w J(w_t; x, y)$ using the entire dataset;

  Update $w_{t+1} \leftarrow w_t - \epsilon \nabla_w J(w_t; x, y)$;

endfor

Stochastic Gradient Descent

The cost function to be minimized is

$$J(w, x, y) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(w, x_i, y_i),$$

where $\mathcal{L}$ is the *loss function*, e.g. $\mathcal{L}(w, x_i, y_i) = ||y_i - w^\top x_i||^2$

Stochastic gradient descent (SGD)

- one sample at a time, noisy estimate of $\nabla_w J$
- performs one step in direction of $-\nabla_w \mathcal{L}(w, x_i, y_i)$

$$w_{t+1} = w_t - \epsilon_t \, \nabla_w \mathcal{L}(w, x, y)$$

- converges faster than gradient descent

Minibatch gradient descent

The cost function to be minimized is

$$J(w, x, y) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(w, x_i, y_i),$$

Minibatch gradient descent

- noisy estimate of the true gradient with M samples, with M being the *minibatch size*
- randomize $\mathcal{I}$ with $|\mathcal{I}| = M$, one set at a time

$$w_{t+1} = w_t - \epsilon_t \, \frac{1}{M} \sum_{j \in \mathcal{I}} \nabla_w \mathcal{L}(w, x_j, y_j)$$

- creates a smoother estimate than SGD
- great for parallel architectures (GPU)

Observation : if the batch size is too large, there is a generalization gap.

Why use gradient descent?

Convexity

A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if:

1. $\iff \forall x_1, x_2 \in \mathbb{R}^n, \forall \alpha \in [0;1] f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2)$
2. with f twice diff.,
   $\iff \forall x \in \mathbb{R}^n, \mathbf{H} = \nabla^2 f(x)$ is positive semidefinite, i.e.
   $\forall x \in \mathbb{R}^n, x^\top \mathbf{H} x \geq 0$.

Observations

- For a convex function f, all local minima are global minima
- Under mild conditions gradient descent and stochastic gradient descent converge
- Typically, $\sum \epsilon_t = \infty$ and $\sum \epsilon_t^2 \leq \infty$

Linear Regression – Recap

Problem : Given $(x_i, y_i), x_i \in \mathbb{R}^{n+1}, y_i \in \mathbb{R}$

- We assume that $x[0] = 1$ to encompass the bias term.
- We have a **linear model :** $\hat{y} = w^\top x$
- We consider ℓ_2 **loss :** $\mathcal{L}(\hat{y}, y) = ||\hat{y} - y||^2$.
- Solve using **gradient descent**

$$\nabla_w \mathcal{L}(w, x_i, y_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w} = -(y_i - \hat{y}_i)x_i$$

Observations

- Other choices for the loss function may also be considered, e.g. Huber loss, MAE, MSE ...
- We can also include a regularization term (we'll discuss this later).
- Linear regression with ℓ_2 loss is convex:

$$\begin{aligned}
\mathcal{L}(w) =& \frac{1}{2}(w^\top x_i - y_i)^2 \\
\nabla_w \mathcal{L} =& (w^\top x_i - y_i)x_i \\
\nabla_w^2 \mathcal{L} =& x_i x_i^\top \quad \forall x \in \mathbb{R}^n x^\top x_i x_i^\top x = \|x_i^\top x\|^2 \geq 0
\end{aligned}$$

Linear classification – Recap

Consider the Maximum Likelihood in binary classification task:

Problem : Given $(x_i, y_i), x_i \in \mathbb{R}^{n+1}, y_i \in \{0, 1\}$
We consider our samples to be independent and we consider the conditional probability to be parametrized by w, $P(y = 1|x) = p(x, w)$.
The conditional likelihood of the labels is:

$$L(w) = \prod_i P(y = y_i|x_i) = \prod_i p(x_i; w)^{y_i} (1 - p(x_i; w))^{1-y_i}$$

Usually, we prefer to minimize the averaged *negative log-likelihood*:

$$J(w) = -\frac{1}{N} \log(L(w)) = \frac{1}{N} \sum_i -y_i \log(p(x_i; w)) - (1 - y_i) \log(1 - p(x_i; w))$$

Binary classification

Problem : Given $(x_i, y_i), x_i \in \mathbb{R}^{n+1}, y_i \in \{0, 1\}$

- **Linear logit model :** $g(x) = w^\top x$
- Use **Sigmoid** transfer function : $\hat{y}(x) = \sigma(g(x)) = \sigma(w^\top x)$, where

$$\sigma : \mathbb{R} \mapsto [0, 1], \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

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

- **Cross-entropy loss** (also called negative log-likelihood) :

$$\mathcal{L}(\hat{y}, y) = -y\text{log}(\hat{y}) - (1 - y)\text{log}(1 - \hat{y})$$

- The gradient of CE is easy to compute :

$$\nabla_w \mathcal{L}(w, x_i, y_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w} = -(y_i - \hat{y}_i)x_i$$

Question: Is this problem convex ?

Should we use ℓ_2 as a loss?

Consider ℓ_2 loss $\mathcal{L} = \frac{1}{2}||\hat{y} - y||^2$ and the "linear" model

$$\hat{y} = \sigma\left(w^\top x_i\right)$$

Let us compute the gradient w.r.t. w:

$$\nabla_w \mathcal{L}(w, x_i, y_i) = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w} = -(\hat{y}_i - y_i)\, \sigma\left(w^\top x_i\right)\left(1 - \sigma\left(w^\top x_i\right)\right) x_i$$

Observations

- If $x_i \to$ strongly misclassified (e.g. $y_i = 1, w^\top x_i \to +\infty$), then $\sigma\left(w^\top x_i\right)\left(1 - \sigma\left(w^\top x_i\right)\right) \approx 0$ and $\nabla_w \mathcal{L}(w, x_i, y_i) \approx 0$
$\Rightarrow$ step-size is very small.
- If we use CE loss, then $\nabla_w \mathcal{L}(w, x_i, y_i)$ is proportional with the error.

Multiclass Classification (Softmax regression)

Problem : Given $(x_i, y_i), x_i \in \mathbb{R}^{n+1}, y_i \in \{0, \dots, K-1\}$, where $K > 2$ is the total number of classes.
Assume that the samples are independent and the conditional probability for a class c is $P(y = c|x) = \frac{e^{(w_c^\top x)}}{\sum_k e^{(w_k^\top x)}}$, parameterized by $(w_k)_{1 \le k \le K}$.
The conditional likelihood of the labels is:

$$L(w) = \prod_i P(y = y_i|x_i)$$

We can also minimize the averaged negative log-likelihood:

$$J(w) = -\frac{1}{N}\log(L(w)) = -\frac{1}{N}\sum_i \log(P(y = y_i|x_i))$$

Usually we use one-hot encoding of the target class (i.e. $y_i = [0, \dots, 0, 1, 0 \dots 0]$), the cost function can be expressed as:

$$J(w) = -\frac{1}{N}\log(L(w)) = -\frac{1}{N}\sum_i \sum_c y_{c_i} \log(P(y = c_i|x_i))$$

Softmax vs. CE loss – numerical stability issues

Large exponentials

- if we compute naively the softmax $\rightarrow \exp(\cdot)$ may have large values
- We can use the following property:

$$\text{softmax}(g_1, g_2, \dots) = \text{softmax}(g_1 - g', g_2 - g', \dots) = \frac{e^{g_i - g'}}{\sum_j e^{g_j - g'}}$$

,
where g' can be a constant, but usually $g' = \max(x)$, in order to have $g_j - g' \leq 0$

Avoiding some exponentials with the log-sum-exp trick

$$\log(\textstyle\sum_j e^{g_j}) = g' + \log(\textstyle\sum_j e^{g_j - g'})$$

- No need to compute $\log(\hat{y}_i) = \log(\text{softmax}_j(x))$:

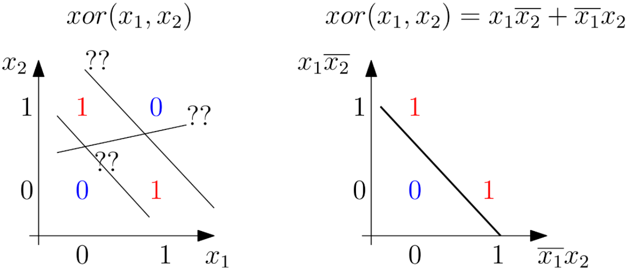
$$\log(\hat{y}_i) = \log\left(\frac{e^{g_i - g'}}{\sum_j e^{g_j - g'}}\right) = g_i - g' - \log\left(\sum_j e^{g_j - g'}\right)$$

.

- This is why in practice we use CE loss with logits rather than Softmax + negative log-likelihood.

The limitations of Linear classification

Perceptrons and logistic regression perform linear separation in a **predefined**, **fixed** feature space.



XOR and its transformation

Can we learn these features ?