

Introduction to Deep Learning (IT3320E)

1 - Preliminaries, Machine Learning, Artificial Neural Network

Hung Son Nguyen

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Sept. 27, 2022

1 Overview of Statistical Learning

- Formulating The Learning Problem
- Loss functions
- Defining Learning Algorithms

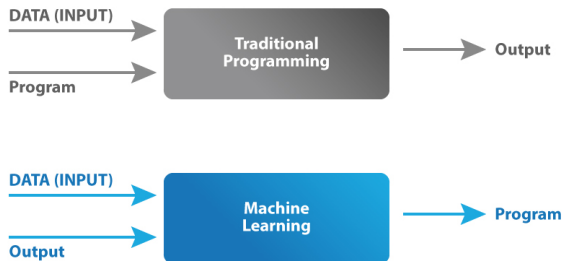
2 Gradient Descent

3 Back propagation algorithm

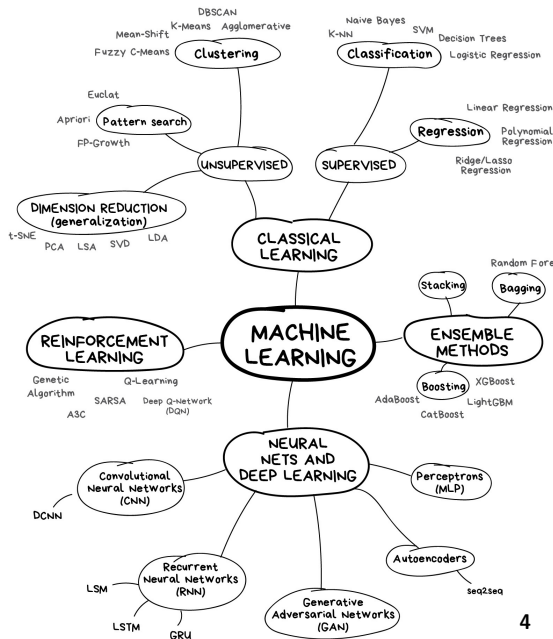
- Network Training
- Deep Learning

- **Linear Algebra:** familiarity with vector spaces, matrix operations (e.g. inversion, singular value decomposition (SVD)), inner products and norms, etc.
- **Calculus:** limits, derivatives, measures, integrals, etc.
- **Probability Theory:** probability distributions, conditional and marginal distribution, expectation, variance, etc.

- **Arthur Samuel, 1959** defined Machine Learning as “Field of study that gives computers the capability to learn without being explicitly programmed”.
- **Traditional Programming:** Data and program is run on the computer to produce the output.
- **Machine Learning:** Data and output is run on the computer to create a program. This program can be used in traditional programming.



- **Supervised ML:** (inductive learning) Training data includes desired outputs.
- **Unsupervised ML:** Training data does not include desired outputs. Example is clustering. It is hard to tell what is good learning and what is not.
- **Reinforcement learning:** Rewards from a sequence of actions. AI likes it, it is the most ambitious type of learning.
- **Ensemble Learning:** techniques that create multiple models and then combine them to produce improved results.
- **Deep Learning:** uses multiple layers to progressively extract higher-level features from the raw input.



There are tens of thousands of machine learning algorithms and hundreds of new algorithms are developed every year. Each of them has three components:

- **Representation:** how to represent knowledge. Examples include decision trees, sets of rules, instances, graphical models, neural networks, support vector machines, model ensembles and others.
- **Evaluation:** the way to evaluate candidate programs (hypotheses). Examples include accuracy, prediction and recall, squared error, likelihood, posterior probability, cost, margin, entropy k-L divergence and others.
- **Optimization:** the way candidate programs are generated known as the search process. For example combinatorial optimization, convex optimization, constrained optimization.

All machine learning algorithms are combinations of these three components. A framework for understanding all algorithms.

Table 1: The three components of learning algorithms.

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
<i>K</i> -nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

Model:	Also known as “hypothesis”, a machine learning model is the mathematical representation of a real-world process. A machine learning algorithm along with the training data builds a machine learning model.
Feature:	A feature is a measurable property or parameter of the data-set.
Feature Vector:	It is a set of multiple numeric features. We use it as an input to the machine learning model for training and prediction purposes.
Training:	An algorithm takes a set of data known as “training data” as input. The learning algorithm finds patterns in the input data and trains the model for expected results (target). The output of the training process is the machine learning model.
Prediction:	Once the machine learning model is ready, it can be fed with input data to provide a predicted output.
Target (Label):	The value that the machine learning model has to predict is called the target or label.
Overfitting:	When a massive amount of data trains a machine learning model, it tends to learn from the noise and inaccurate data entries. Here the model fails to characterise the data correctly.
Underfitting:	It is the scenario when the model fails to decipher the underlying trend in the input data. It destroys the accuracy of the machine learning model. In simple terms, the model or the algorithm does not fit the data well enough.

1. Start Loop

- **Understand the domain, prior knowledge and goals.** Talk to domain experts. Often the goals are very unclear. You often have more things to try than you can possibly implement.
- **Data integration, selection, cleaning and pre-processing:** The most time consuming part. It is important to have high quality data. The more data you have, the more it sucks because the data is dirty. Garbage in, garbage out.
- **Learning models.** The fun part. This part is very mature. The tools are general.
- **Interpreting results.** Sometimes it does not matter how the model works as long as it delivers results. Other domains require that the model is understandable. You will be challenged by human experts.
- **Consolidating and deploying discovered knowledge.** The majority of projects that are successful in the lab are not used in practice. It is very hard to get something used.

2. End Loop

It is mandatory to learn a programming language, preferably Python, along with the required analytical and mathematical knowledge. Here are the five mathematical areas that you need to brush up before jumping into solving Machine Learning problems:

- **Linear algebra for data analysis:** Scalars, Vectors, Matrices, and Tensors
- **Mathematical Analysis:** Derivatives and Gradients
- **Probability theory and statistics**
- **Multivariate Calculus**
- **Algorithms and Complex Optimizations**

- Python is hands down the best programming language for Machine Learning applications due to the various benefits mentioned in the section below.
 - **Numpy, OpenCV, and Scikit** are used when working with images
 - **NLTK** along with Numpy and Scikit again when working with text
 - **Librosa** for audio applications
 - **Matplotlib, Seaborn, and Scikit** for data representation
 - **TensorFlow and Pytorch** for Deep Learning applications
 - **Scipy** for Scientific Computing
 - **Django** for integrating web applications
 - **Pandas** for high-level data structures and analysis
- Other programming languages that could to use for Machine Learning Applications are R, C++, JavaScript, Java, C#, Julia, Shell, TypeScript, and Scala.

- Linear Regression
- Logistic Regression
- Decision Tree
- SVM
- Naive Bayes
- kNN
- Random Forest
- Dimensionality Reduction Algorithms
- Gradient Boosting algorithms
 - GBM
 - XGBoost
 - LightGBM
 - CatBoost

Section 1

Overview of Statistical Learning

Main ingredients:

- \mathcal{X} : the input space, \mathcal{Y} : the output space;
- ρ : the unknown distribution on $\mathcal{X} \times \mathcal{Y}$
- $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ a loss function measuring the discrepancy $\ell(y, y')$ between any two values $y, y' \in \mathcal{Y}$.

We would like to minimize the expected risk:

$$\begin{aligned} & \underset{f: \mathcal{X} \rightarrow \mathcal{Y}}{\text{minimize}} \quad \mathcal{E}(f) \\ & \text{where} \quad \mathcal{E}(f) = \int_{\mathcal{X} \times \mathcal{Y}} \ell(f(x), y) d\rho(x, y) \end{aligned}$$

The expected prediction error incurred by a predictor $f : \mathcal{X} \rightarrow \mathcal{Y}$

Input space

Linear Spaces:

- Vectors
- Matrices
- Functions
- ...

Structured Spaces:

- Strings
- Graphs
- Probabilities
- Points on a manifold

Output space

Linear Spaces:

- $\mathcal{Y} = \mathbb{R}$: Regression
- $\mathcal{Y} = \{1, \dots, T\}$: Classification
- $\mathcal{Y} = \mathbb{R}^T$: Multi-task learning
- ...

Structured Spaces:

- Strings
- Graphs
- Probabilities
- Orders (i.e. Ranking)

- Informally: the distribution ρ on $\mathcal{X} \times \mathcal{Y}$ encodes the probability of getting a pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ when observing (sampling from) the unknown process.
- Throughout the course we will assume $\rho(x, y) = \rho(y | x) \cdot \rho_{\mathcal{X}}(x)$, where

$\rho_{\mathcal{X}}(x)$ **the marginal distribution on \mathcal{X}**

$\rho(y | x)$ **the conditional distribution on \mathcal{Y} given $x \in \mathcal{X}$**

- $\rho(y | x)$ characterizes the relation between a given input x and the possible outcomes y that could be observed.
- In noisy settings it represents the uncertainty in our observations.
- Example: $y = f_*(x) + \varepsilon$, with $f_* : \mathcal{X} \rightarrow \mathbb{R}$ is the **true function** and $\varepsilon \sim \mathcal{N}(0, \sigma)$ is the Gaussian distributed noise. Then:

$$\rho(y | x) = \mathcal{N}(f_*(x), \sigma)$$

Definition of Statistical Learning

- $y = f(\mathbf{x}) + \varepsilon$
- f represents the information that \mathbf{x} provides about y
- **Definition:** Statistical Learning refers to a set of techniques/ approaches to estimate the function
- **Questions:** Why should we estimate and what are the techniques to estimate ?

- **Prediction:** the average, or expected value, of the squared expected value difference between the predicted $\hat{y} = \hat{f}(\mathbf{x})$ and actual value of y :

$$\begin{aligned}\mathbb{E}(y - \hat{y})^2 &= \mathbb{E}[f(\mathbf{x}) + \varepsilon - \hat{f}(\mathbf{x})]^2 \\ &= \underbrace{[f(\mathbf{x}) - \hat{f}(\mathbf{x})]^2}_{\text{reducible}} + \underbrace{\text{Var}(\varepsilon)}_{\text{irreducible}}\end{aligned}$$

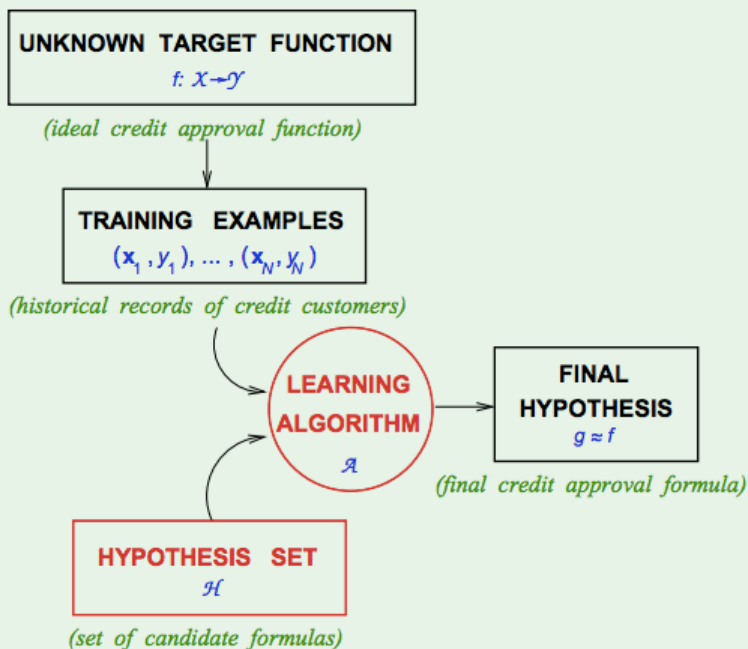
- **Inference:** We are often interested in understanding the association between output and the inputs:
 - Which predictors are associated with the response?
 - What is the relationship between the response and each predictor?
 - Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

Loss functions and Cost functions

Loss function is any function $L : \mathcal{Y} \times \mathcal{Y}' \rightarrow \mathbb{R}^+$ that evaluates how well our algorithm models our dataset. Usually, the loss function is applied to the designed output and the predicted output.

The cost function is the average loss over the entire training dataset:

$$\text{Cost}(f) = \frac{1}{n} \sum_i^n L(y_i, \hat{y}_i), \text{ where } \hat{y}_i = f(\mathbf{x}_i)$$



- Cross Entropy Loss function for 2 classes $\mathcal{Y} = \{0, 1\}$:

$$L_{CE} : \{0, 1\} \times [0, 1] \rightarrow \mathbb{R}$$

$$L_{CE}(y, \hat{y}) = -y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})$$

- Cross Entropy Loss function for multiclass $\mathcal{Y} = \{1, \dots, C\}$

$$L_{CE} : \{0, 1\}^C \times [0, 1]^C \rightarrow \mathbb{R}$$

$$L_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^C y_i \ln(\hat{y}_i)$$

where $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_C)$ is the class distribution, i.e. $\hat{y}_1 + \dots + \hat{y}_C = 1$, returned by the model.

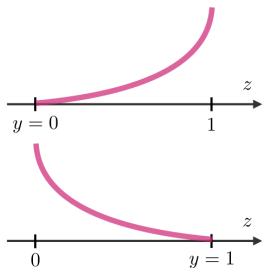
- Hinge Loss (for binary classification: $\mathcal{Y} = \{-1, 1\}$)

$$L_H : \{-1, 1\} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$L_H(y, \hat{y}) = \max\{0, 1 - y \cdot \hat{y}\}$$

Cross entropy

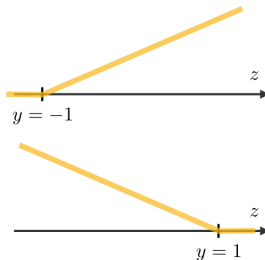
$$-y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})$$



Neural Network

Hinge loss

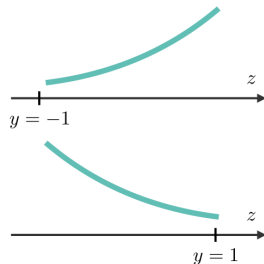
$$\max\{0, 1 - y \cdot \hat{y}\}$$



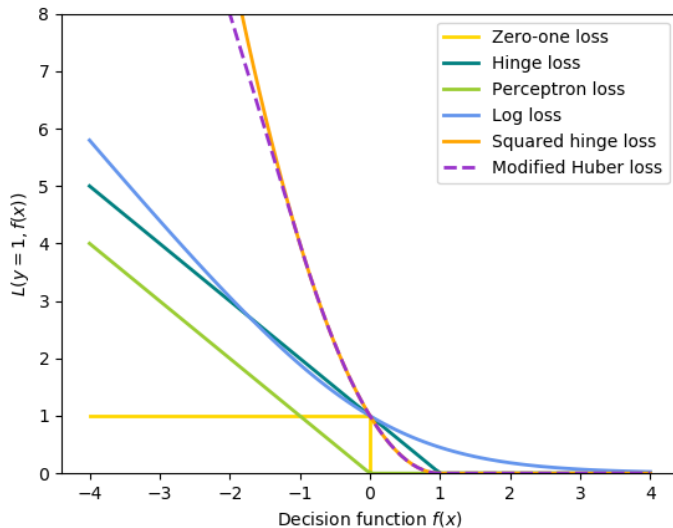
SVM

Logistic loss

$$\log(1 + \exp(-y \cdot \hat{y}))$$



Logistic regression



- Kullback-Leibler (KL)-divergence loss function

$$\begin{aligned}\mathbf{KL}(q_{\theta}||p) &= \sum_d q(d) \log \left(\frac{q(d)}{p(d)} \right) = \sum_d q(d) (\log q(d) - \log p(d)) \\ &= \underbrace{\sum_d q(d) \log q(d)}_{-\text{entropy}} - \underbrace{\sum_d q(d) \log p(d)}_{\text{cross-entropy}}\end{aligned}$$

$$\begin{aligned}\mathbf{KL}(p||q_{\theta}) &= \sum_d p(d) \log \left(\frac{p(d)}{q(d)} \right) = \sum_d p(d) (\log p(d) - \log q(d)) \\ &= \sum_d p(d) \log p(d) - \sum_d p(d) \log q(d)\end{aligned}$$

- Loss functions used in regression task, i.e. $\mathcal{Y} = \mathbb{R}$

- MAE (absolute error) or L_1

$$L_1(y, \hat{y}) = \|y - \hat{y}\|$$

- MSE (square error) or L_2

$$L_2(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

- ε -intensive

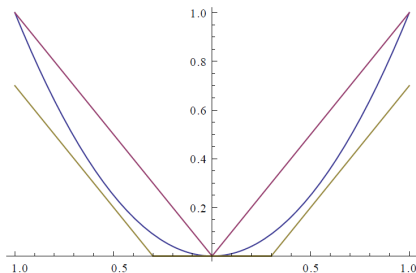
$$V_\varepsilon(y, \hat{y}) = \max(|y - \hat{y}| - \varepsilon, 0)$$

- Huber Loss is a modification of MSE:

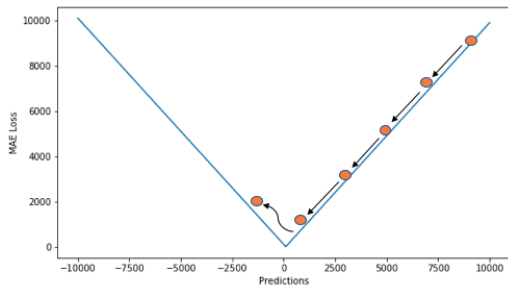
$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

- Log-Cosh Loss

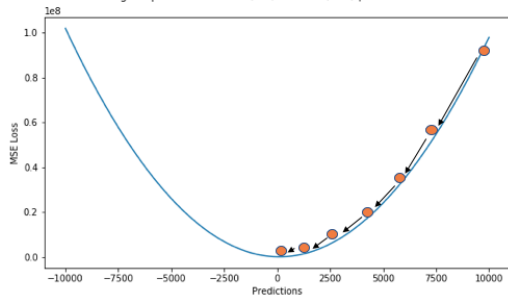
$$L_{\log}(y, \hat{y}) = \frac{1}{a} \ln(\cosh a(\hat{y} - y)) = \frac{1}{a} \ln \frac{e^{a(\hat{y}-y)} + e^{-a(\hat{y}-y)}}{2}$$



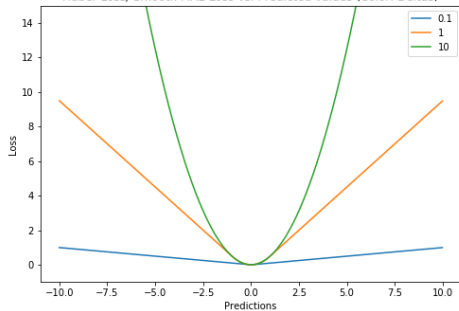
Range of predicted values: (-10,000 to 10,000) | True value: 100



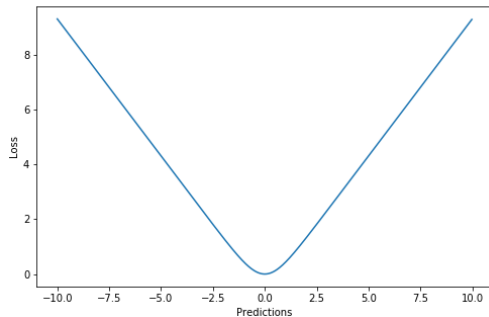
Range of predicted values: (-10,000 to 10,000) | True value: 100



Huber Loss/ Smooth MAE Loss vs. Predicted values (Color: Deltas)



Log-Cosh Loss vs. Predictions



- The relation between \mathcal{X} and \mathcal{Y} encoded by the distribution is unknown in reality. The only way we have to access a phenomenon is from finite observations.
- The goal of a learning algorithm is therefore to find a good approximation $f_n : \mathcal{X} \rightarrow \mathcal{Y}$ for the minimizer of expected risk

$$\inf_{f: \mathcal{X} \rightarrow \mathcal{Y}} \mathcal{E}(f)$$

from a finite set of examples $S = \{(\mathbf{x}_i, y_i) : i = 1, \dots, n\}$ sampled independently from ρ .

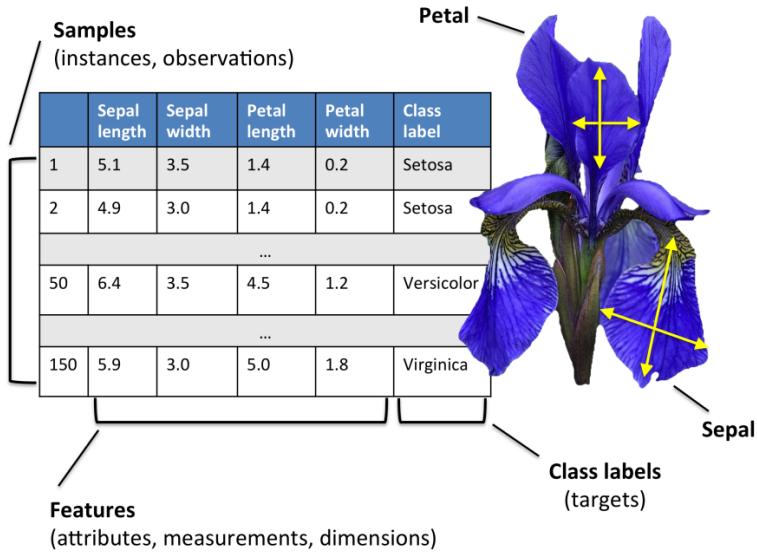
Let $\mathcal{S} = \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$ be the set of all finite datasets on $\mathcal{X} \times \mathcal{Y}$. A learning algorithm is a map

$$A : \mathcal{S} \rightarrow \mathcal{F}$$

$$S \mapsto A(S) : \mathcal{X} \rightarrow \mathcal{Y}$$

In case $S = \{(x_i, y_i) : i = 1, \dots, n\}$, we will denote:

$$f_n = A(\{(x_i, y_i) : i = 1, \dots, n\})$$



Definition: Classifier Learning

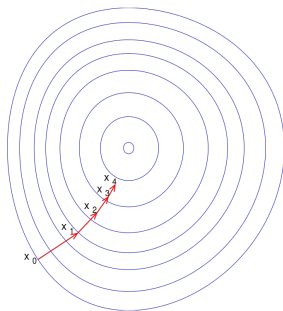
Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$ where $\mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^D$ is a feature vector and $y_i \in \mathcal{Y}$ is a class label, learn a function $f : \mathbb{R}^D \rightarrow \mathcal{Y}'$ that accurately predicts the class label y for any feature vector \mathbf{x} .

Function f is also called *the model*.

Section 2

Gradient Descent

Gradient descent is an iterative optimization algorithm for finding the minimum of a function. How? Take step proportional to the negative of the gradient of the function at the current point.



Gradient descent on a series of level sets

If we consider a function $f(\boldsymbol{\theta})$, the **gradient descent update** can be expressed as:

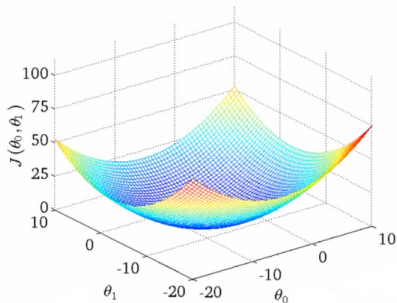
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} f(\boldsymbol{\theta}) \quad (1)$$

for each parameter θ_j .

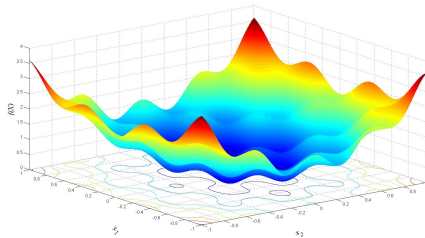
The size of the step is controlled by **learning rate** α .

Gradient Descent for 1-d function $f(\theta)$.

Turns out that if the function is **convex** gradient descent will converge to the **global minimum**. For **non-convex** functions, it may converge to **local minima**.



Convex Function



Non-Convex Function

Gradient descent is often used in machine learning to **minimize a cost function**, usually also called *objective* or *loss* function and denoted $L(\cdot)$ or $J(\cdot)$.

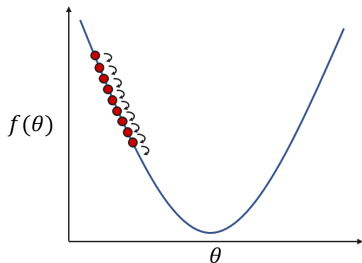
The cost function depends on the model's parameters and is a proxy to evaluate model's performance. Generally speaking, in this framework minimizing the cost equals to maximizing the effectiveness of the model.

In principle, to perform a single update step you should run through all your training examples. This is known as **batch gradient descent**.

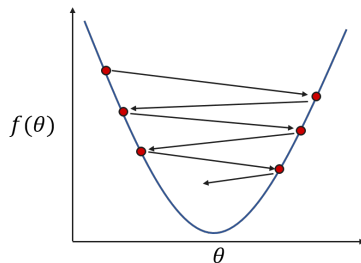
A different strategy is the one of **minibatch stochastic gradient descent**. In this case, only a small subset of the training dataset is considered at each update step.

In the extreme case in which only a random example of the training set is considered to perform the update step, we talk of **stochastic gradient descent**.

Choosing the the right **learning rate** α is essential to correctly proceed towards the minimum. A step *too small* could lead to an extremely *slow* convergence. If the step is *too big* the optimizer could *overshoot* the minimum or even *diverge*.



Learning Rate too small



Learning Rate too big

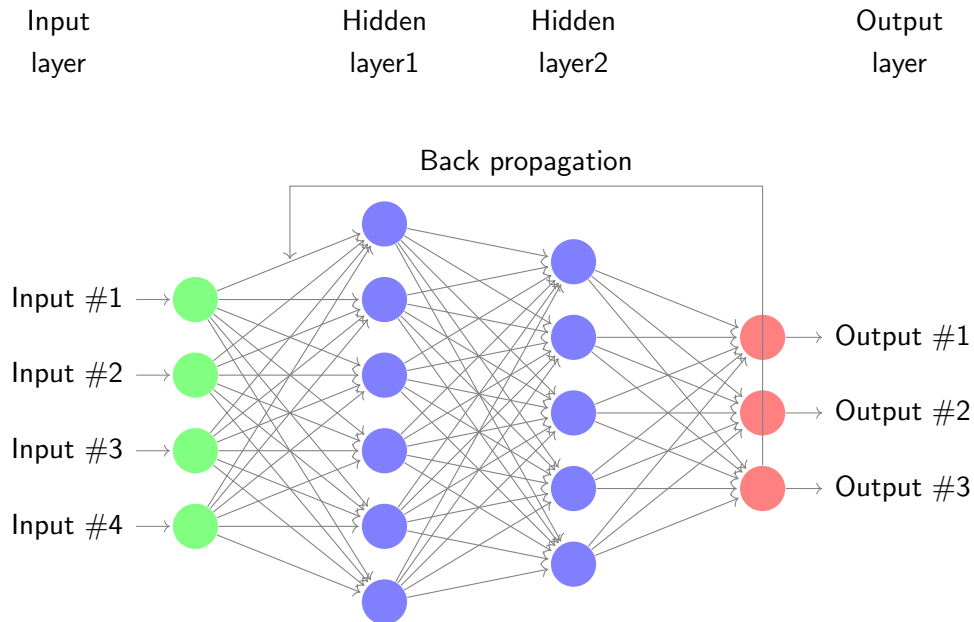
In practice, it's quite rare to see the procedure described above (so called **vanilla SGD**) used for optimization in the real-world.

Conversely, a number of cutting-edge optimizers [1,2,3] are commonly used. However, these advanced optimization techniques are out of the scope of this short overview.

- [1] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [2] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [3] M. D. Zeiler. Adadelata: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Section 3

Back propagation algorithm




A multilayer perceptron represents an adaptable model $y(\cdot, w)$ able to map D -dimensional input to C -dimensional output:

$$y(\cdot, w) : \mathbb{R}^D \rightarrow \mathbb{R}^C, x \mapsto y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix}. \quad (2)$$

In general, a $(L + 1)$ -layer perceptron consists of $(L + 1)$ layers, each layer l computing linear combinations of the previous layer $(l - 1)$ (or the input).

On input $x \in \mathbb{R}^D$, layer $l = 1$ computes a vector $y^{(1)} := (y_1^{(1)}, \dots, y_{m^{(1)}}^{(1)})$ where

$$y_i^{(1)} = f(z_i^{(1)}) \quad \text{with } z_i^{(1)} = \sum_{j=1}^D w_{i,j}^{(1)} x_j + w_{i,0}^{(1)}. \quad (3)$$

 i^{th} component is called “unit i ”

where f is called activation function and $w_{i,j}^{(1)}$ are adjustable weights.

What does this mean?

Layer $l = 1$ computes linear combinations of the input and applies an (non-linear) activation function ...

The first layer can be interpreted as generalized linear model:

$$y_i^{(1)} = f \left(\left(w_i^{(1)} \right)^T x + w_{i,0}^{(1)} \right). \quad (4)$$

Idea: Recursively apply L additional layers on the output $y^{(1)}$ of the first layer.

In general, layer l computes a vector $y^{(l)} := (y_1^{(l)}, \dots, y_{m^{(l)}}^{(l)})$ as follows:

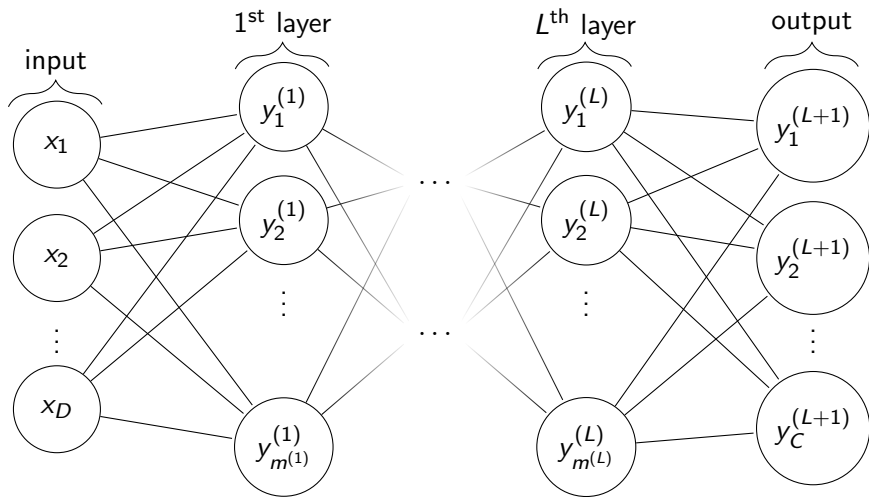
$$y_i^{(l)} = f(z_i^{(l)}) \quad \text{with } z_i^{(l)} = \sum_{j=1}^{m^{(l-1)}} w_{i,j}^{(l)} y_j^{(l-1)} + w_{i,0}^{(l)}. \quad (5)$$

Thus, layer l computes linear combinations of layer $(l - 1)$ and applies an activation function ...

Layer $(L + 1)$ is called output layer because it computes the output of the multilayer perceptron:

$$y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix} := \begin{pmatrix} y_1^{(L+1)} \\ \vdots \\ y_C^{(L+1)} \end{pmatrix} = y^{(L+1)} \quad (6)$$

where $C = m^{(L+1)}$ is the number of output dimensions.



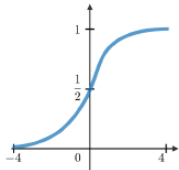
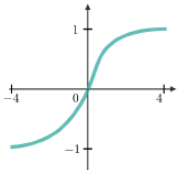
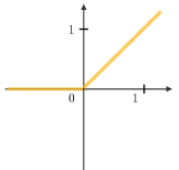
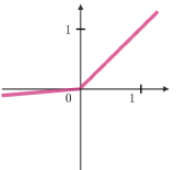
How to choose the activation function f in each layer?

- Non-linear activation functions will increase the expressive power: Multilayer perceptrons with $L + 1 \geq 2$ are universal approximators [?]
- Depending on the application: For classification we may want to interpret the output as posterior probabilities:

$$y_i(x, w) \stackrel{!}{=} p(c = i|x) \quad (7)$$

where c denotes the random variable for the class.

Activation layers compute non-linear activation function elementwise on the input volume. The most common activations are **ReLU**, **sigmoid** and **tanh**.

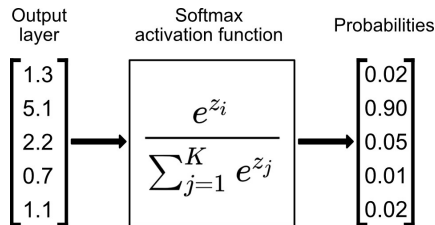
Sigmoid	Tanh	ReLU	Leaky ReLU
$f(z) = \frac{1}{1 + e^{-z}}$	$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$f(z) = \max(0, z)$	$f(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Nonetheless, more complex activation functions exist [?, ?].

For classification with $C > 1$ classes, layer $(L + 1)$ uses the softmax activation function:

$$y_i^{(L+1)} = \sigma(z^{(L+1)}, i) = \frac{\exp(z_i^{(L+1)})}{\sum_{k=1}^C \exp(z_k^{(L+1)})}. \quad (8)$$

Then, the output can be interpreted as posterior probabilities.



By now, we have a general model $y(\cdot, w)$ depending on W weights.

Idea: Learn the weights to perform

- regression,
- or classification.

We focus on classification.

Given a training set

C classes:

1-of- C coding scheme

$$U_S = \{(\mathbf{x}_n, t_n) : 1 \leq n \leq N\}, \quad (9)$$

learn the mapping represented by U_S ...

by minimizing the squared error

$$E(w) = \sum_{n=1}^N E_n(w) = \sum_{n=1}^N \sum_{i=1}^C (y_i(\mathbf{x}_n, w) - t_{n,i})^2 \quad (10)$$

using iterative optimization.

We distinguish ...

Stochastic Training: A training sample (\mathbf{x}_n, t_n) is chosen at random, and the weights w are updated to minimize $E_n(w)$.

Batch and Mini-Batch Training: A set $M \subseteq \{1, \dots, N\}$ of training samples is chosen and the weights w are updated based on the cumulative error $E_M(w) = \sum_{n \in M} E_n(w)$.

Of course, online training is possible, as well.

Problem: How to minimize $E_n(w)$ (stochastic training)?

- $E_n(w)$ may be highly non-linear with many poor local minima.

Framework for iterative optimization: Let ...

- $w[0]$ be an initial guess for the weights (several initialization techniques are available),
- and $w[t]$ be the weights at iteration t .

In iteration $[t + 1]$, choose a weight update $\Delta w[t]$ and set

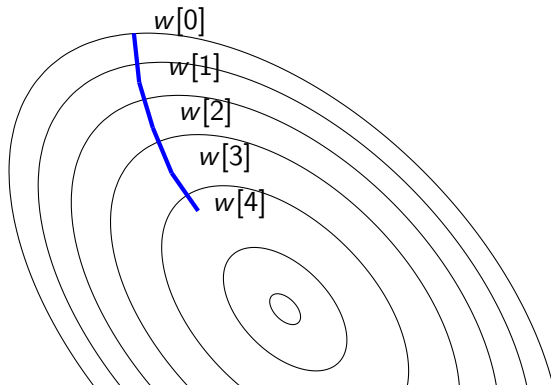
$$w[t + 1] = w[t] + \Delta w[t] \quad (11)$$

Remember:

Gradient descent minimizes the error $E_n(w)$ by taking steps in the direction of the negative gradient:

$$\Delta w[t] = -\gamma \frac{\partial E_n}{\partial w[t]} \quad (12)$$

where γ defines the step size.



Problem: How to evaluate $\frac{\partial E_n}{\partial w[t]}$ in iteration $[t + 1]$?

- “Error Backpropagation” algorithm allows to evaluate $\frac{\partial E_n}{\partial w[t]}$ in $\mathcal{O}(W)$!
- **Feed-forward step:** Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- **Backward step:** Calculate the error in the outputs and travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
- Similar algorithm allows to evaluate the Hessian $\frac{\partial^2 E_n}{\partial w[t]^2}$ such that second-order optimization can be used.

Further details ...

- See the original paper “Learning Representations by Back-Propagating Errors,” by Rumelhart et al. [?].

For an input vector \mathbf{x}_n do a forward step to compute the activations and outputs for all layers in the network (as described in previous slides):

- The first layer:

$$y_i^{(1)} = f \left(\left(w_i^{(1)} \right)^T \cdot \mathbf{x}_n + w_{i,0}^{(1)} \right).$$

- Layer l computes linear combinations of layer $(l - 1)$ and applies an activation function

$$z_i^{(l)} = \sum_{j=1}^{m^{(l-1)}} w_{i,j}^{(l)} y_j^{(l-1)} + w_{i,0}^{(l)} \quad \text{and then} \quad y_i^{(l)} = f \left(z_i^{(l)} \right).$$

for $l = 2, \dots, L + 1$

- 1 Calculate the error functions δ starting from the output units:

$$\delta_k^{(L+1)} = 2(y_k^{(L+1)} - t_k) \cdot f'(z_k^{L+1})$$

- 2 Calculate the remaining error functions by working backwards using the backpropagation algorithm

$$\delta_j^{(l)} = f'(z_k^l) \cdot \sum_k w_{k,j}^{(l+1)} \delta_k^{(l+1)}$$

- 3 Estimate the required derivatives $\nabla E = \left(\frac{\partial E_n}{\partial w_{k,j}^{(l)}} = \delta_k^{(l)} \cdot y_j^{(l-1)} \right)$

Note that the bias term for a layer l , the input is $z = 1$ so $\frac{\partial E_n}{\partial b_k^{(l)}} = \delta_k^{(l)}$.

- ④ Change the weights based on estimated gradients by $-\gamma \cdot \nabla E$:

$$w[t + 1] = w[t] - \gamma \frac{\partial E_n}{\partial w[t]}$$

where γ defines the step size.

- ⑤ Go back to forward step and repeat until a number of iterations or a desired minimum.

Multilayer perceptrons are called deep if they have more than three layers: $L + 1 > 3$.

Motivation: Lower layers can automatically learn a hierarchy of features or a suitable dimensionality reduction.

- No hand-crafted features necessary anymore!

However, training deep neural networks is considered very difficult!

- Error measure represents a highly non-convex, “potentially intractable” [?] optimization problem.

Possible approaches:

- Different activation functions offer faster learning, for example

$$\max(0, z) \quad \text{or} \quad |\tanh(z)|; \quad (13)$$

- unsupervised pre-training can be done layer-wise;
- ...

Further details ...

- See “Learning Deep Architectures for AI,” by Y. Bengio [?] for a detailed discussion of state-of-the-art approaches to deep learning.

Most prominent advantages of Backpropagation are:

- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Disadvantages of using Backpropagation

- The actual performance of backpropagation on a specific problem is dependent on the input data.
- Backpropagation can be quite sensitive to noisy data
- You need to use the matrix-based approach for backpropagation instead of mini-batch.

The multilayer perceptron represents a standard model of neural networks. They ...

- allow to tailor the architecture (layers, activation functions) to the problem;
- can be trained using gradient descent and error backpropagation;
- can be used for learning feature hierarchies (deep learning).

Deep learning is considered difficult.