https://github.com/leduckhai

Lê Đức Khải

Royal Machine Learning Summary

Written on October 1, 2019

Last updated on July 22, 2021

Messages from the author:

I have been writing this summary in order to systematize all knowledge in the domain of Machine Learning, which is acquired from online lectures, blogs and papers (references included below).

The “selling point” of the summary is that it is neither too short as a cheatsheet, nor too long as a lecture. So this file is perfect for newbies wanting to learn AI faster, sergeants preparing for interviews and veterans researching awesome things.

This summary is updated every month. So check https://github.com/leduckhai monthly to get the latest version.

https://github.com/leduckhai

CONTENTS:

- Slack variable
- Hinge loss
- Weight decay
- Bias trick
- Gradient descent for soft margin SVM

2.3. Kernel SVM

- ____

2.4. Multi-class SVM

- _____

3. Generative learning

- Goal of generative learning
- Gaussian Discriminant Analysis (GDA)
- Naive Bayes

4. Trees and ensemble methods

- Classification and Regression Trees (CART)
- Random forest
- Boosting

5. Other non-parametric methods

- K-nearest neighbors (KNN)

6. Clustering

6.1. Expectation-Maximization

- Latent variables
- Expectation-Maximization algorithm

6.2. K-means clustering

- K-means clustering algorithm
- Distortion function

6.3. Hierarchical clustering

- Hierarchical clustering algorithm
- Types of algorithm

6.4. Clustering assessment metrics

- Silhouette coefficient
- Calinski-Harabaz index

7. Dimension reduction

7.1. Principal component analysis

- Eigenvalue, eigenvector
- Spectral theorem
- PCA algorithm

7.2. Independent component analysis

- Assumptions
- Bell and Sejnowski ICA algorithm

B. Deep Learning

1. Neural Network

https://github.com/leduckhai

1.1. Perceptron Learning Algorithm (PLA)
1.2. Logistic regression
1.3. Softmax regression
1.4. Multi-layer perceptron

2. Convolutional Neural Network

2.1. General
- Architecture of a traditional CNN
- Types of layer
- Tuning hyperparameters
- Activation functions (commonly used)

2.2. Object detection
- Types of models
- Data labeling
- Intersection over Union (IoU)
- Anchor boxes
- Non-max suppression
- YOLO algorithm
- R-CNN algorithm

2.3. Face verification and recognition
- Types of models
- One-shot learning
- Siamese Network
- Triplet loss

3. Recurrent Neural Network

3.1. Traditional RNN
- Architecture of a traditional RNN
- Forward propagation
- Backpropagation through time
- Pros and cons of a typical RNN architecture
- Vanishing and exploding gradient
- Gradient clipping
- Types of RNNs

3.2. Gated Recurrent Unit (GRU)
- Types of gates
- Equations of GRU

3.3. Long Short-Term Memory units (LSTM)
- Equations of LSTM

3.4. Bidirectional RNN (BRNN)
- Functional principles

3.5. Deep RNN (DRNN)
- Functional principles

3.6. Natural Language Processing & Word Embeddings
3.7. Sequence models & Attention mechanism

https://github.com/leduckhai

4. Deep Learning Tips and Tricks

4.1. Data processing
- Data augmentation
- Batch normalization

4.2. Training a neural network
- Epoch
- Mini-batch gradient descent
- Loss function
- Cross-entropy loss
- Backpropagation
- Updating weights

4.3. Parameter tuning
- Weights initialization: Xavier initialization, Transfer learning
- Optimizing convergence: Learning rate, Adaptive learning rates

4.4. Regularization
- Dropout
- Weight regularization
- Early stopping

4.5. Good practices
- Overfitting small batch
- Gradient checking

5. Important scientific paper

5.1. Residual Neural Network (ResNet)

https://github.com/leduckhai

References:

- **CS229 – Machine Learning Cheatsheet, Shervine Amidi:**
  https://stanford.edu/~shervine/teaching/cs-229/
- **CS230 – Deep Learning Cheatsheet, Shervine Amidi:**
  https://stanford.edu/~shervine/teaching/cs-230/
- **DeepLearning.ai Courses Notes, Mahmoud Badry:**
  https://github.com/mbadry1/DeepLearning.ai-Summary
- **Machine Learning Cơ Bản, Vũ Hữu Tiệp:**
  https://machinelearningcoban.com/

A.Machine Learning

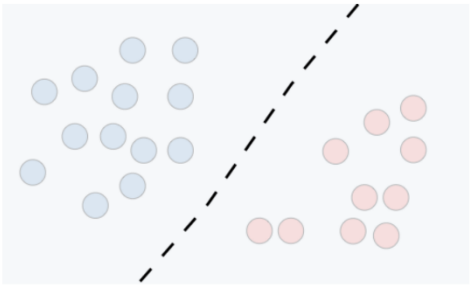
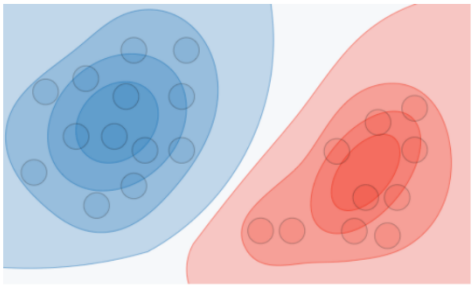
1. General Concepts

1.1.Overview

- Type of prediction: Given a set of N data points $\{x^{(1)}, \dots, x^{(N)}\}$ associated to a set of outcomes $\{y^{(1)}, \dots, y^{(N)}\}$, we want to build a classifier that learns how to predict y from x:

	Regression	Classification
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

- Type of model:

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

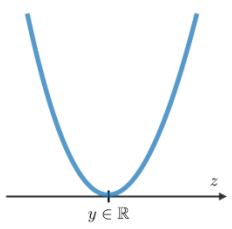
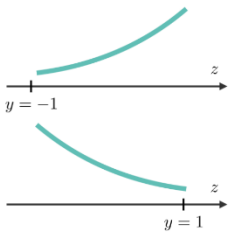
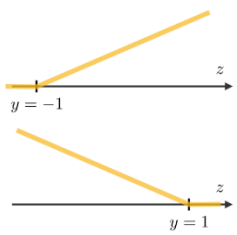
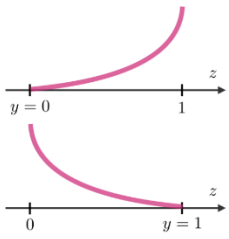
- Hypothesis: The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$ the model prediction output is $h_\theta(x^{(i)})$.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_C x_C$$

$$h(x) = \sum_{i=0}^{C} \theta_i x_i = \theta^T x$$

  - C: Number of classes
  - The θ_i's: are the parameters (also called weights) parameterizing the space of linear functions mapping from X to Y

- Loss function: A loss function is a function $L: (z, y) \in R \times Y \longmapsto L(z, y) \in R$ that takes as inputs the predicted value z corresponding to the real data value y and outputs how different they are.

Least squared error	Logistic loss	Hinge loss	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0, 1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

- Cost function: The cost function J is commonly used to assess the performance of a model, and is defined with the loss function L as follows:

$$J(\theta) = \sum_{i=1}^{m} L\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

- Likelihood: The likelihood of a model $L(\theta)$ given parameters θ is used to find the optimal parameters θ through likelihood maximization. We have:

$$\theta^{opt} = \arg\max_{\theta} L(\theta)$$

  Remark: in practice, we use the log-likelihood $l(\theta) = \log(L(\theta))$ which is easier to optimize.

- Newton's algorithm: is a numerical method that finds θ such that $l'(\theta) = 0$. Its update rule is as follows:

$$\theta \leftarrow \theta - \frac{l'(\theta)}{l''(\theta)}$$

  Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:

$$\theta \leftarrow \theta - \frac{\nabla_\theta l(\theta)}{\nabla_\theta^2 l(\theta)}$$

1.2. Gradient descent

- Univariate gradient descent: $x_{t+1} = x_t - \eta f'(x_t)$
  - $f'(x_t)$: derivative of loss function f at x_t
  - η: learning rate ($\eta > 0$)
  - t: iteration (starting from 1)

- Multivariate gradient descent: $\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$
  - $\theta_t = \theta_t(\theta_0, \theta_1, \dots, \theta_C)$: multivariate function at iteration t

- Numerical gradient: $f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$

  $\rightarrow f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$

- Gradient descent algorithm:
  - Determine loss function f
  - Calculate gradient of f after θ
  - Calculate gradient descent, stopping criteria is gradient $\approx \vec{0}$, which means $\frac{\|\nabla_\theta f(\theta_t)\|}{\dim(\nabla_\theta f(\theta_t))} \approx 0$.

- Gradient descent with momentum:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta f(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t$$

  - γ: constant, ranging from 0 to 1, common default value is 0.9
  - v_t: velocity at iteration t

- Stochastic Gradient Descent (SGD): its algorithm is summarized as follows:
  - Step 1: Epoch running from 0 to 10:
    - Shuffle data
    - Step 2: In an epoch, iteration runs N times (with N being number of data points):
      - Calculate stochastic gradient (gradient of i-th data point)
      - Calculate gradient descent: $\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$
      - Check θ for each 10 iterations whether it's equal to its last check

- Stopping criteria:
  - Iteration limitation: can make sure that the program stops sooner, but probably before the minimum is found.
  - Compare gradients of each 2 consecutive iterations: can slow down the algorithm's speed and consume too much computation capacity (Batch Gradient Descent)
  - Compare values of loss function at each 2 consecutive iterations: the algorithm sometimes stops at saddle points, which overlooks minima.
  - Compare gradients of each N consecutive iterations: can speed up the algorithm, reduce memory consumption and determine roots more precisely.

1.3.Linear models

- Linear regression: we assume here that $y|x;\theta \sim N(\mu,\sigma^2)$ (normal distribution)
  - Normal equations: By noting $X_{N\times C}$ the design matrix, $y_{N\times 1}$ the ground truth, the value of $\theta_{C\times 1}$ that minimizes the cost function is a closed-form solution such that:
$$\theta = (X^T X)^{-1} X^T y$$
  - Cost function:
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$
    - $x^{(i)}, y^{(i)}$: the input value and ground truth at the i-th datapoint
  - Least Mean Squares algorithm: By noting α the learning rate, the update rule of the Least Mean Squares (LMS) algorithm for a training set of m data points, which is also known as the Widrow-Hoff learning rule, is as follows:
$$\begin{cases} \theta_j \leftarrow \theta_j - \alpha \dfrac{\delta}{\delta\theta_j} J(\theta) \\ \dfrac{\delta}{\delta\theta_j} J(\theta) = (h_\theta(x) - y)x_j \end{cases} \Rightarrow \begin{cases} \forall j, \quad \theta_j \leftarrow \theta_j + \alpha \displaystyle\sum_{i=1}^{N} \left[y^{(i)} - h_\theta(x^{(i)})\right] x_j^{(i)} \\ \theta \leftarrow \theta + \alpha \displaystyle\sum_{i=1}^{N} \left[y^{(i)} - h_\theta(x^{(i)})\right] x^{(i)} \end{cases}$$
    - j: j-th class in C classes
    - $\alpha > 0$: learning rate

    Remark: the update rule is a particular case of the gradient ascent.

  - Locally Weighted Regression (LWR): the model does not learn a fixed set of parameters as is done in ordinary linear regression. Rather parameters θ are computed individually for each query point $x^{(q)}$. LWR weights each training example in its cost function by non-negative weight $w^{(i)}(x)$, which is defined with parameter $\tau \in R$ as:
$$w^{(i)}(x) = \exp\left(-\frac{\left(x^{(i)} - x^{(q)}\right)^2}{2\tau^2}\right)$$
$$\Rightarrow \text{Cost function: } J(\theta) = \sum_{i=1}^{N} w^{(i)} (\theta^T x^{(i)} - y^{(i)})^2$$

- Logistic regression:
  - Sigmoid function g:
$$\forall z \in R, \quad g(z) = \frac{1}{1+e^{-z}} \in (0,1)$$
  - Logistic regression: We assume here that $y_{N\times 1}|x_{N\times 1}; \theta_{N\times 1} \sim Bernoulli(\phi)$
$$\phi_{1\times 1} = p(y=1|x;\theta) = \frac{1}{1+\exp(-\theta^T x)} = g(\theta^T x)$$

    Remark: Logistic regressions do not have closed form solutions.

- Softmax regression: A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes. By convention, at class C we set $\theta_C = 0$, which makes the Bernoulli parameter ϕ_i of each class i be such that:

$$\phi_i = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^{C} \exp(\theta_j^T x)}$$

- Generalized linear models:
  - Exponential family: A class of distributions is said to be in the exponential family if it can be written in terms of a natural parameter (also called the canonical parameter or link function) η, a sufficient statistic $T(y)$ and a log-partition function $a(\eta)$:

$$p(y;\eta) = b(y)\exp(\eta T(y) - a(\eta))$$

    Remark: we will often have $T(y) = y$. Also, $\exp(-a(\eta))$ can be seen as a normalization parameter that will make sure that the probabilities sum to one.

    The most common exponential distributions:

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1+\exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^{η}	$\frac{1}{y!}$
Geometric	$\log(1-\phi)$	y	$\log\left(\frac{e^{\eta}}{1-e^{\eta}}\right)$	1

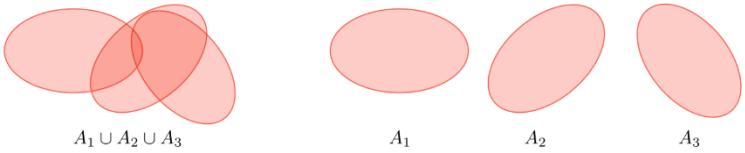
  - Assumptions of GLMs: Generalized Linear Models (GLM) aim at predicting a random variable y as a function of $x \in R^{n+1}$ and rely on 3 assumptions:
    - $y|x;\theta \sim ExpFamily(\eta)$: Given x and θ, the distribution of y follows some exponential family distribution, with parameter η.
    - $h_\theta(x) = E[y|x;\theta]$: The prediction $h(x)$ output by learned hypothesis h to satisfy $h(x) = E[y|x]$.
    - $\eta = \eta^T x$: The natural parameter η and the inputs x are related linearly.

    Remark: ordinary least squares and logistic regression are special cases of generalized linear models.

1.4. Learning theory

- Hoeffding inequality:
  - Union bound: Let $A_1, \dots, A_k$ be k events. We have:

$$P(A_1 \cup \dots \cup A_k) \leq P(A_1) + \dots + P(A_k)$$



  - Hoeffding inequality: Let $Z_1, \dots, Z_m$ be N variables drawn from a Bernoulli distribution of parameter ϕ. Let $\hat{\phi}$ be their sample mean and $\gamma > 0$ fixed. We have:

$$P\left(\left|\phi - \hat{\phi} > \gamma\right|\right) \leq 2\exp(-2\gamma^2 N)$$

    Remark: this inequality is also known as the Chernoff bound.

- Probably Approximately Correct (PAC):
  - Training error: For a given classifier h, we define the training error $\hat{\epsilon}(h)$, also known as the empirical risk or empirical error, to be as follows:

$$\hat{\epsilon}(h) = \frac{1}{N}\sum_{i=1}^{N} 1_{\{h(x^{(i)}) \neq y^{(i)}\}}$$

  - Probably Approximately Correct (PAC): PAC is a framework under which numerous results on learning theory were proved, and has the following set of assumptions:
    - The training and testing sets follow the same distribution
    - The training examples are drawn independently

- Vapnik-Chervonenkis dimension:
  - Shattering: Given a set $S = \{x^{(1)}, \dots, x^{(d)}\}$, and a set of classifiers $\mathcal{H}$, we say that $\mathcal{H}$ shatters S if for any set of labels $\{y^{(1)}, \dots, y^{(d)}\}$, we have:

$$\exists h \in \mathcal{H}, \forall i \in [\![1, d]\!], h(x^{(i)}) = y^{(i)}$$

  - Upper bound theorem: Let $\mathcal{H}$ be a finite hypothesis class such $|\mathcal{H}| = k$ and let δ and the sample size N be fixed. Then, with probability of at least $1 - \delta$, we have:

$$\epsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + 2\sqrt{\frac{1}{2N}\log\frac{2k}{\delta}}$$

  - VC dimension: The Vapnik-Chervonenkis (VC) dimension of a given infinite hypothesis class $\mathcal{H}$, noted $VC(\mathcal{H})$ is the size of the largest set that is shattered by $\mathcal{H}$.

    Remark: the VC dimension of $\mathcal{H} = \{set\ of\ linear\ classifiers\ in\ 2\ dimensions\}$ is 3.

- Theorem (Vapnik): Let $\mathcal{H}$ be given, with $VC(\mathcal{H}) = d$ and N the number of training examples. With probability at least $1 - \delta$, we have:

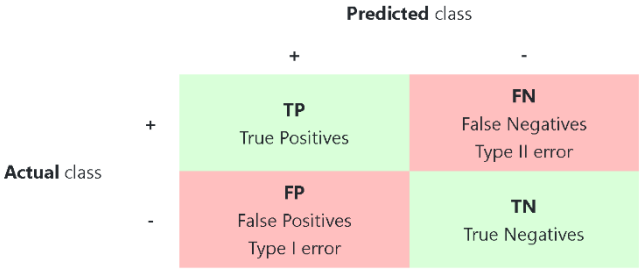
$$\epsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + O\left(\sqrt{\frac{d}{N}\log\left(\frac{N}{d}\right) + \frac{1}{N}\log\left(\frac{1}{\delta}\right)}\right)$$

1.5. Classification/regression metrics

Classification metrics:

In a context of a binary classification, here are the main metrics that are important to track in order to assess the performance of the model.

- Confusion matrix:
  - The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:



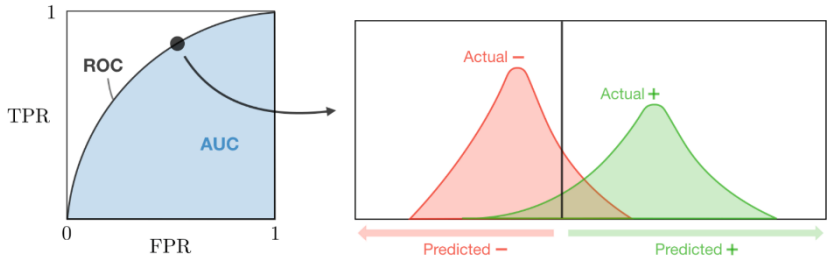
  - Main metrics: The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

- ROC and AUC:
  - The receiver operating curve (ROC), is the plot of TPR versus FPR by varying the threshold:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{TP}{TP + FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN + FP}$	1-specificity

  - AUC: The area under the receiving operating curve (AUC or AUROC), is the area below the ROC as shown in the following figure:



Regression metrics:

- Basic regression metrics and coefficient of determination:
  - Basic metrics: Given a regression model f, the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$\mathrm{SS}_{\mathrm{tot}} = \sum_{i=1}^{m} (y_i - \bar{y})^2$	$\mathrm{SS}_{\mathrm{reg}} = \sum_{i=1}^{m} (f(x_i) - \bar{y})^2$	$\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{m} (y_i - f(x_i))^2$

  - Coefficient of determination: The coefficient of determination, often noted R^2 or r^2, provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

  - Main metrics: The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{\mathrm{SS}_{\mathrm{res}} + 2(n+1)\widehat{\sigma}^2}{m}$	$2\left[(n+2) - \log(L)\right]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

where L is the likelihood and $\hat{\sigma}^2$ is an estimate of the variance associated with each response.

1.6. Model selection and diagnostics

When selecting a model, we distinguish 3 different parts of the data that we have as follows:

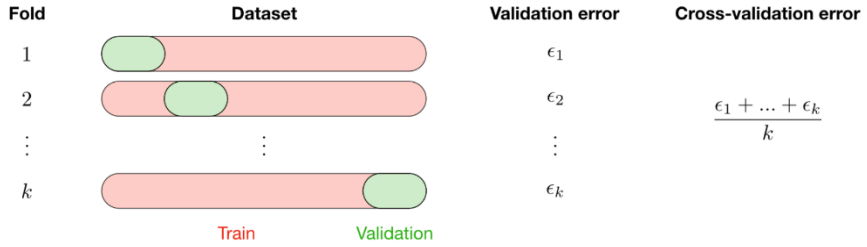
Training set	Validation set	Testing set
• Model is trained • Usually 80% of the dataset	• Model is assessed • Usually 20% of the dataset • Also called hold-out or development set	• Model gives predictions • Unseen data

Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:

- Cross-validation:
  - Cross-validation: is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

k-fold	Leave-p-out
• Training on $k-1$ folds and assessment on the remaining one • Generally $k = 5$ or 10	• Training on $n-p$ observations and assessment on the p remaining ones • Case $p = 1$ is called leave-one-out

  - K-fold cross-validation: The most commonly used method is called k-fold cross-validation and splits the training data into k folds to validate the model on one fold while training the model on the $k-1$ other folds, all of this k times. The error is then averaged over the k folds and is named cross-validation error.

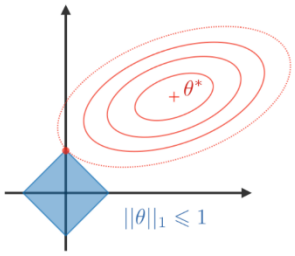
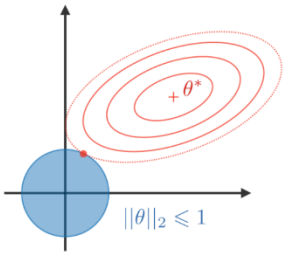
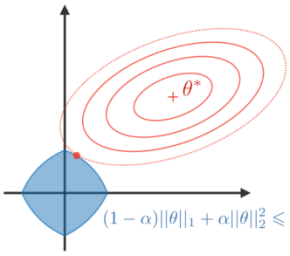


- Regularization: The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. Regularized cost function is defined as:

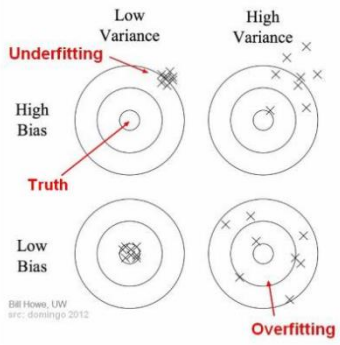
$$J_{reg}(\theta) = J(\theta) + \lambda R(\theta)$$

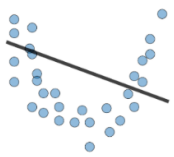
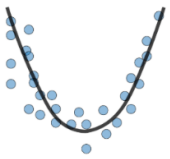
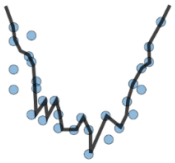
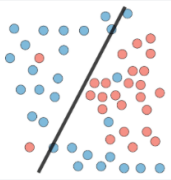
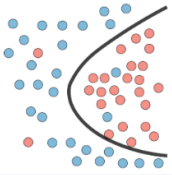
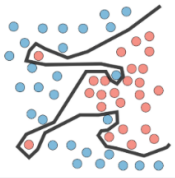
where: $\lambda \in R$: regularizaton parameter

The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
• Shrinks coefficients to 0 • Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$... + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

- Bias/variance tradeoff:
  - Bias: is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.
  - Variance: is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.
  - Bias/variance tradeoff: The simpler the model, the higher the bias, and the more complex the model, the higher the variance.



	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

- Error analysis: is analyzing the root cause of the difference in performance between the current and the perfect models.
- Ablative analysis: is analyzing the root cause of the difference in performance between the current and the baseline models.

2. Support Vector Machine (SVM)

2.1. Hard margin SVM

Neural Networks	Support Vector Machine	Property
PLA	Hard Margin SVM	2 classes are linearly separable
Logistic Regression	Soft Margin SVM	2 classes are almost linearly separable
Softmax Regression	Multi-class SVM	Multi-class classification with linear boundary
Multi-layer Perceptron	Kernel SVM	Non-linearly separable classes

- Margin: is the distance from the nearest point to the classify hyperplane

$$margin = \min_i \frac{y^{(i)}\left(w^T x^{(i)} + b\right)}{\|w\|}$$

- Optimal margin classifier: h is such that:

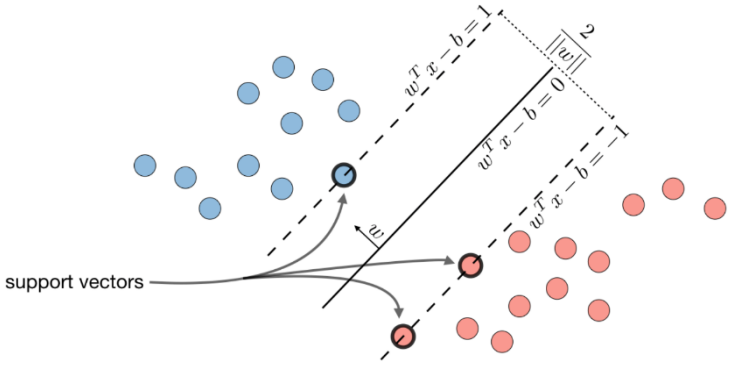
$$h(x) = sign(w^T x + b)$$

- SVM optimization problem: $(w, b) \in R^n \times R$ is the solution of the following optimization problem:

$$\min \left(\frac{1}{2}\|w\|^2\right) \text{ such that } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1$$

which also means:

$$(w, b) = \arg\min_{w,b} \frac{1}{2}\|w\|^2 \text{ subject to: } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \; \forall i = 1, 2, \ldots, N$$



Remark: the decision boundary is defined as $w^T x + b = 0$

2.2. Soft margin SVM

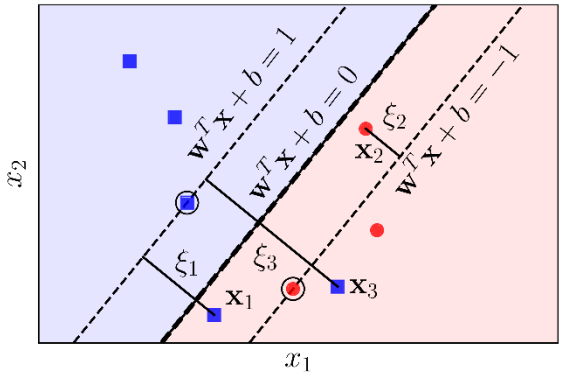
- Slack variable: Slack variable $\xi^{(i)} = \left|w^T x^{(i)} + b - y^{(i)}\right| \geq 0$ and a positive constant C is added to the soft margin SVM optimization problem:

$$\min \left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi^{(i)}\right) \text{ such that } y^{(i)}\left(w^T x^{(i)} + b\right) + \xi^{(i)} \geq 1$$

which also means:

$$(w, b, \xi) = \arg\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi^{(i)}$$

$$\text{subject to: } y^{(i)}\left(w^T x^{(i)} + b\right) + \xi^{(i)} \geq 1 \text{ and } \xi^{(i)} \geq 0 \; \forall i = 1, 2, \dots, N$$



- Hinge loss: The hinge loss is used in the setting of SVMs and is defined as follows:

$$L(z, y) = [1 - yz]_+ = \max(0, 1 - yz)$$

$$\Longrightarrow \quad L(w, b) = \sum_{i=1}^{N} L_i = \sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\left(w^T x^{(i)} + b\right)\right)$$

- Weight decay: The new loss function $J(w, b)$ is regularized as:

$$J(w, b) = L(w, b) + \lambda R(w, b) = \sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\left(w^T x^{(i)} + b\right)\right) + \frac{\lambda}{2}\|w\|^2$$

where $\lambda > 0$: regularization parameter
$R(w, b)$: regularization function

- Bias trick: Using the bias trick to turn weight w into new weight $\bar{w}$ for simpler gradient calculation:

$$J(\bar{w}) = \underbrace{\sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\bar{w}^T\bar{x}^{(i)}\right)}_{\text{hinge loss}} + \underbrace{\frac{\lambda}{2}\|w\|^2}_{\text{regularization}}$$

where: $\bar{x}^{(i)} = \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix},\ x^{(i)} \in R^n,\ \bar{x}^{(i)} \in R^{n+1}$

$\bar{w} = [w^T, b]^T \in R^{n+1}$

- Gradient descent for soft margin SVM:
  - Gradient of loss function:

$$\nabla J(\bar{w}) = \sum_{i \in \mathrm{H}} -y^{(i)}\bar{x}^{(i)} + \lambda \begin{bmatrix} W \\ 0 \end{bmatrix}$$

where: $u_{1\times N} = \left[y^{(1)}\bar{w}^T\bar{x}^{(1)},\ y^{(2)}\bar{w}^T\bar{x}^{(2)}, \dots,\ y^{(N)}\bar{w}^T\bar{x}^{(N)}\right]$

$H = \{i : u^{(i)} < 1\}$

  - Update the gradient:

$$\bar{w} \leftarrow \bar{w} - \eta\left(\sum_{i \in \mathrm{H}} -y^{(i)}\bar{x}^{(i)} + \lambda \begin{bmatrix} W \\ 0 \end{bmatrix}\right)$$

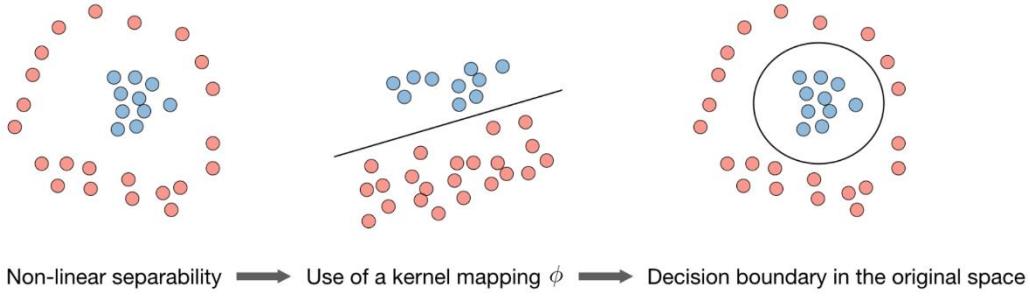
where: $\eta > 0$ is learning rate

2.3.Kernel SVM

- Kernel SVM: Given a feature mapping ϕ, we define the kernel K as follows:

$$K(x,z) = \phi(x)^T \phi(z)$$

In practice, the kernel K defined by $K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ is called the Gaussian kernel and is commonly used



- Remark: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ, which is often very complicated. Instead, only the values $K(x,z)$ are needed.

- Lagrangian: Lagrangian $\mathcal{L}(w,b)$ is defined as follows:

$$\mathcal{L}(w,b) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

Remark: the coefficients β_i are called the Lagrange multipliers.

2.4. Multi-class SVM

- asdf

3. Generative learning

- Goal of generative learning: A generative model first tries to learn how the data is generated by estimating $P(x|y)$, which we can then use to estimate $P(y|x)$ by using Bayes' rule.

- Gaussian Discriminant Analysis:
  - Setting: The Gaussian Discriminant Analysis assumes that y and $x|y=0$ and $x|y=1$ are such that:
    - $y \sim Bernoulli(\phi)$
    - $x|y=0 \sim \mathcal{N}(\mu_0, \Sigma)$
    - $x|y=1 \sim \mathcal{N}(\mu_1, \Sigma)$
  - Estimation: The following table sums up the estimates that we find when maximizing the likelihood:

$\widehat{\phi}$	$\widehat{\mu_j} \quad (j=0,1)$	$\widehat{\Sigma}$
$\frac{1}{m}\sum_{i=1}^{m} 1_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}}$	$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T$

- Naive Bayes:
  - Assumption: The Naive Bayes model supposes that the features of each data point are all independent:

$$P(x|y) = P(x_1, x_2, \dots|y) = P(x_1|y)P(x_2|y)\dots = \prod_{i=1}^{N} P(x_i|y)$$

  - Solutions: Maximizing the log-likelihood gives the following solutions:

$$P(y=k) = \frac{1}{m} \times \#\{j|y^{(i)}=k\} \text{ and } P(x_i=l|y=k) = \frac{\#\{j|y^{(i)}=k \text{ and } x_i^{(j)}=l\}}{\#\{j|y^{(i)}=k\}}$$

with $k \in \{0,1\}$ and $l \in [\![1,\ L]\!]$

Remark: Naive Bayes is widely used for text classification and spam detection.

4. Trees and ensemble methods

These methods can be used for both regression and classification problems.

- Classification and Regression Trees (CART): commonly known as decision trees, can be represented as binary trees. They have the advantage to be very interpretable.

- Random forest: It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

  Remark: random forests are a type of ensemble methods.

- Boosting: The idea of boosting methods is to combine several weak learners to form a stronger one:

Adaptive boosting	Gradient boosting
• High weights are put on errors to improve at the next boosting step • Known as Adaboost	• Weak learners are trained on residuals • Examples include XGBoost

5. Other non-parametric methods

- K-nearest neighbors (KNN): is a non-parametric approach where the response of a data point is determined by the nature of its k neighbors from the training set. It can be used in both classification and regression settings.

  Remark: the higher the parameter k, the higher the bias; and the lower the parameter k, the higher the variance.

$k = 1$ $k = 3$ $k = 11$

6. Clustering

6.1. Expectation-Maximization

- Latent variables: Latent variables are hidden/unobserved variables that make estimation problems difficult, and are often denoted z. Here are the most common settings where there are latent variables:

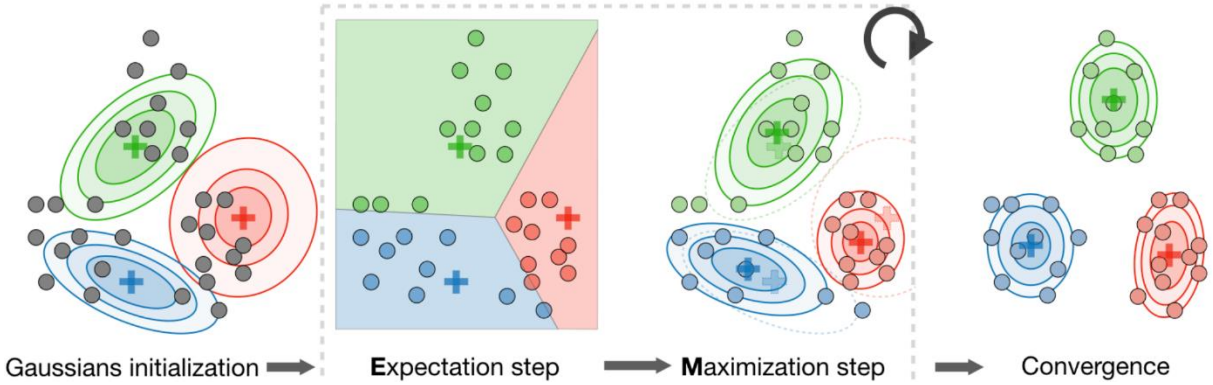
Setting	Latent variable z	$x\|z$	Comments
Mixture of k Gaussians	$\text{Multinomial}(\phi)$	$\mathcal{N}(\mu_j, \Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Factor analysis	$\mathcal{N}(0, I)$	$\mathcal{N}(\mu + \Lambda z, \psi)$	$\mu_j \in \mathbb{R}^n$

- Expectation-Maximization algorithm: gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:
  - E-step: Evaluate the posterior probability $Q_i\left(z^{(i)}\right)$ that each data point $x^{(i)}$ came from a particular cluster $z^{(i)}$ as follows:

$$Q_i\left(z^{(i)}\right) = P\left(z^{(i)}|x^{(i)};\theta\right)$$

  - M-step: Use the posterior probabilities $Q_i\left(z^{(i)}\right)$ as cluster specific weights on data points $x^{(i)}$ to separately re-estimate each cluster model as follows:

$$\theta_i = \arg\max_\theta \sum_i \int_{z^{(i)}} Q_i\left(z^{(i)}\right) \log\left(\frac{P\left(x^{(i)}, z^{(i)};\theta\right)}{Q_i(z^{(i)})}\right) dz^{(i)}$$



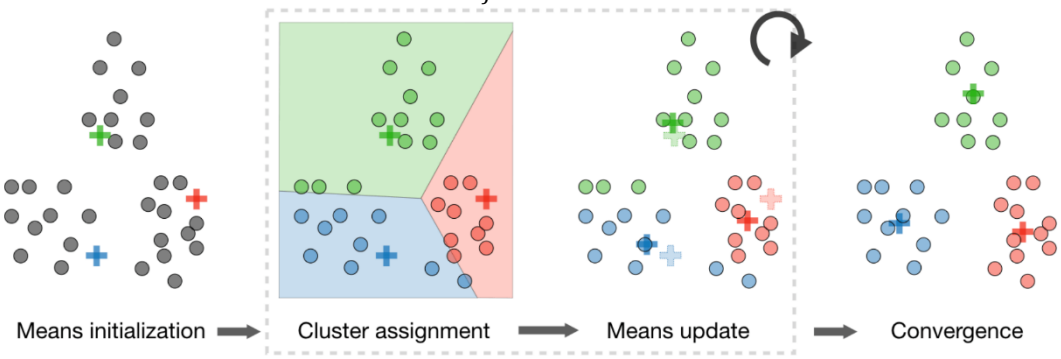
6.2. K-means clustering

We note $c^{(i)}$ the cluster of data point i and μ_j the center of cluster j.

- K-means clustering algorithm: After randomly initializing the cluster centroids $\mu_1, \mu_2, \dots, \mu_k \in R^n$, the k-means algorithm repeats the following step until convergence:

$$\mu_j = \frac{\sum_{i=1}^{N} 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^{N} 1_{\{c^{(i)}=j\}}}$$

$$c^{(i)} = \arg\min_j \left\| x^{(i)} - \mu_j \right\|^2$$



- Distortion function: In order to see if the algorithm converges, we look at the distortion function defined as follows:

$$J(c, \mu) = \sum_{i=1}^{N} \left\| x^{(i)} - \mu_{c^{(i)}} \right\|^2$$

6.3. Hierarchical clustering

- Hierarchical clustering algorithm: It is a clustering algorithm with an agglomerative hierarchical approach that build nested clusters in a successive manner.

- Types of algorithm: There are different sorts of hierarchical clustering algorithms that aims at optimizing different objective functions, which is summed up in the table below:

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

6.4. Clustering assessment metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

- Silhouette coefficient: By noting a and b the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient s for a single sample is defined as follows:

$$s = \frac{b - a}{\max(a, b)}$$

- Calinski-Harabaz index:
  - By noting k the number of clusters, B_k and W_k the between and within-clustering dispersion matrices respectively defined as:

$$B_k = \sum_{j=1}^{k} n_{c^{(i)}} \left(\mu_{c^{(i)}} - \mu\right) \left(\mu_{c^{(i)}} - \mu\right)^T$$

$$W_k = \sum_{i=1}^{N} \left(x^{(i)} - \mu_{c^{(i)}}\right) \left(x^{(i)} - \mu_{c^{(i)}}\right)^T$$

  - The Calinski-Harabaz index $s(k)$ indicates how well a clustering model defines its clusters, such that the higher the score, the more dense and well separated the clusters are. It is defined as follows:

$$s(k) = \frac{Tr(B_k)}{Tr(W_k)} \times \frac{N - k}{k - 1}$$

7. Dimension reduction

7.1. Principal component analysis

It is a dimension reduction technique that finds the variance maximizing directions onto which to project the data.

- Eigenvalue, eigenvector: Given a matrix $A \in R^{n \times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in R^n \{0\}$, called eigenvector, such that we have:

$$Az = \lambda z$$

- Spectral theorem: Let $A \in R^{n \times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in R^{n \times n}$. By noting $\Lambda = diag(\lambda_1, \dots, \lambda_n)$, we have:

$$\exists \Lambda \; diagonal, \;\; A = U \Lambda U^T$$

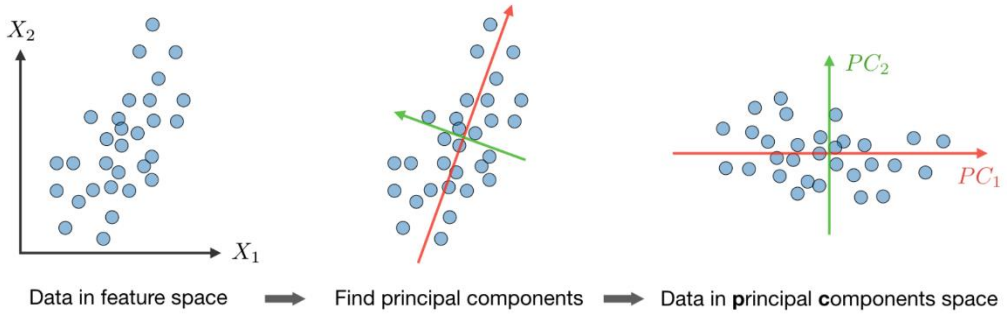
  Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A.

- PCA algorithm: The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on k dimensions by maximizing the variance of the data as follows:
  - Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

    where $\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)}$ and $\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} \left(x_j^{(i)} - \mu_j \right)^2$
  - Step 2: Compute $\Sigma = \frac{1}{N} \sum_{i=1}^{N} x^{(i)} x^{(i)^T} \in R^{n \times n}$, which is symmetric with real eigenvalues.
  - Step 3: Compute $u_1, \dots, u_k \in R^n$ the k orthogonal principal eigenvectors of Σ, i.e. the orthogonal eigenvectors of the k largest eigenvalues.
  - Step 4: Project the data on $span_R(u_1, \dots, u_k)$

  This procedure maximizes the variance among all k-dimensional spaces.



7.2. Independent component analysis

It is a technique meant to find the underlying generating sources.

- Assumptions: We assume that our data x has been generated by the n-dimensional source vector $s = (s_1, \dots, s_n)$, where s_i are independent random variables, via a mixing and non-singular matrix A as follows:

$$x = As$$

  Remark: The goal is to find the unmixing matrix $W = A^{-1}$.

- Bell and Sejnowski ICA algorithm: This algorithm finds the unmixing matrix W by following the steps below:
  - Step 1: Write the probability of $x = As = W^{-1}s$ as:

$$p(x) = \prod_{i=1}^{n} p_s(w_i^T x) \cdot |W|$$

  - Step 2: Write the log likelihood given our training data $\{x^{(i)}, i \in [\![1, N]\!]\}$ and by noting g the sigmoid function as:

$$l(W) = \sum_{i=1}^{N} \left(\sum_{j=1}^{n} \log\left(g'\left(w_j^T x^{(i)}\right)\right) + \log|W| \right)$$

  - Step 3: The stochastic gradient ascent learning rule is such that for each training example $x^{(i)}$, we update W as follows:

$$W \leftarrow W + \alpha \left(\begin{pmatrix} 1 - 2g\left(w_1^T x^{(i)}\right) \\ 1 - 2g\left(w_2^T x^{(i)}\right) \\ \vdots \\ 1 - 2g\left(w_n^T x^{(i)}\right) \end{pmatrix} x^{(i)^T} + (W^T)^{-1} \right)$$

A. Deep Learning

1. Neural Network

1.1. Perceptron Learning Algorithm (PLA)

- Loss function: $J(w) = \sum_{x_i \in M}(-y_i w^T x_i)$
  - M: Misclassified dataset (these dataset elements change over w)
- Derivative of loss function for a single point: $\nabla_w J(w, x_i, y_i) = -y_i x_i$
- Algorithm:
  - Choose random initiation w with its elements approximate to 0.
  - Check random-ordered data x:
    - If x_i is truly classified, aka $sgn(w^T x_i) = y_i$, skip it
    - If x_i is misclassified, update w after SGD: $w_{t+1} = w_t + y_i x_i$
  - Check x if there is still misclassified point with w. If there is, go back second step. If not, stop the program.
- Discussion:
  - PLA generates unlimited roots.
  - PLA is always converged.
  - Input data of PLA need to be linearly separable.
  - Pocket algorithm is applied for data which contain noise.

1.2. Logistic regression

- Output of logistic regression: $f(x) = \theta(w^T x)$
  - θ: logistic function
- Sigmoid function:
  - $f(s) = \frac{1}{1+e^{-s}} \triangleq \sigma(s)$ with $\sigma(s) \in (0,1)$
  - $\sigma'(s) = \frac{e^{-s}}{(1+e^{-s})^2} = \sigma(s)(1 - \sigma(s))$
- Tanh function:
  - $\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$ with $\tanh(s) \in (-1,1)$
  - $\tanh(s) = 2\sigma(2s) - 1$
- Loss function of logistic regression at 1 single data point:

$$J(w; x_i, y_i) = -(y_i \log z_i + (1 - y_i) \log(1 - z_i))$$

where $z_i = \theta(w^T x_i)$

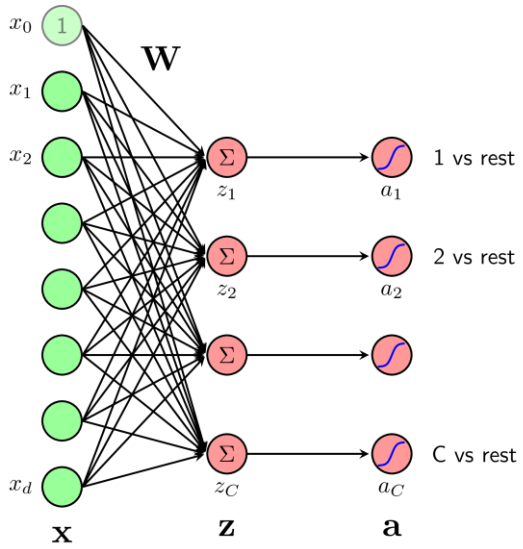
- Gradient of loss function:

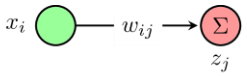
$$\nabla J_w(w; x_i, y_i) = (z_i - y_i)x_i$$

- Update of new w:

$$w_{t+1} = w_t + \eta(y_i - z_i)x_i$$

1.3. Softmax regression





w_{0j}: biases, don't forget!

d: data dimension

C: number of classes

$\mathbf{x} \in \mathbb{R}^{d+1}$

$\mathbf{W} \in \mathbb{R}^{(d+1)\times C}$

$z_i = \mathbf{w}_i^T \mathbf{x}$

$\mathbf{z} = \mathbf{W}^T \mathbf{x} \in \mathbb{R}^C$

$a_i = \text{sigmoid}(z_i) \in \mathbb{R}$

$0 < a_i < 1$

- One-vs-rest:
  - $a_i = \sigma(z_i) = \sigma(w_i^T x)$
  - $a_i = P(y_k = i | x_k; W) = \frac{e^{z_i}}{\sum_j^C e^{z_j}}$
    - C: number of classes
    - P: probability of one-vs-rest
    - $W(i,j)$: model-parameters-matrix of all classes (i: data dimension, j: class)
    - $X(i,j)$: input matrix of all datapoints (i: data dimension, j: datapoint)
    - $Y(i,j)$: label matrix (i: class, j: datapoint)
  - Stable softmax: $a_i = \frac{e^{z_i - c}}{\sum_j^C e^{z_j - c}}$
    - c: constant, commonly $c = \max(z_i)$
- One-hot coding:
  - Cross-entropy loss function for a single datapoint:

$$J(W; x_i, y_i) = -\sum_{j=1}^{C} y_{ji} \log(a_{ji}) = -\sum_{j=1}^{C} y_{ji} \log\left(\frac{\exp(w_j^T x_i)}{\sum_{k=1}^{C} \exp(w_k^T x_i)}\right)$$

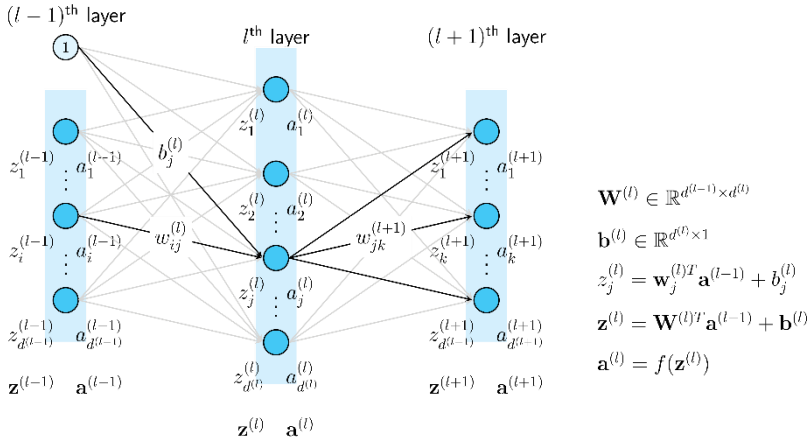
- Optimization of softmax regression:
  - Gradient of cross-entropy loss function for a single datapoint:

$$\frac{dJ_i(W)}{dW} = x_i[e_{1i}, e_{2i}, \dots, e_{Ci}] = x_i e_i^T = x_i(a_i - y_i)^T$$

  - Update: $W_{t+1} = W_t + \eta x_i (y_i - a_i)^T$

1.4. Multi-layer perceptron

- Representation figure:



- Number of layers: $L = N_{Hidden\ layer} + N_{Output\ layer}$
-
- Algorithm (for mini-batch gradient descent):
  - ❖ Feedforward:
    - Hidden layer:
      - $A^{(0)} = X$
      - $Z^{(l)} = W^{(l)T} A^{(l-1)} + b^{(l)},\ \ l = 1,2, \dots, L$
      - $A^{(l)} = \text{f}(Z^{(l)})$ (ReLU commonly as activation function)
    - Output layer:
      - $\hat{Y} = A^{(L)} = softmax\left(Z^{(L)}\right)$ ($\hat{Y}$: predicted label matrix)
    - Loss function:

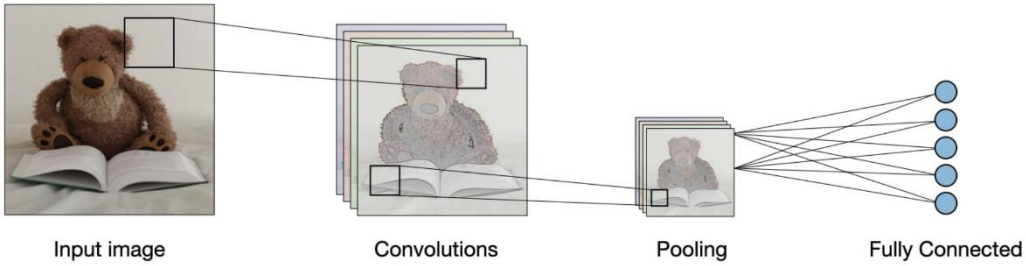
$$J \triangleq J(W, b; X, Y) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} Y_{ji} \odot \log\left(\hat{Y}_{ji}\right)$$

  - ❖ Backpropagation:
    - $E^{(L)} = \frac{dJ}{dZ^{(L)}} = \frac{1}{N}\left(\hat{Y} - Y\right)$
    - $\frac{dJ}{dW^{(L)}} = A^{(L-1)} E^{(L)\,T}$
    - $\frac{dJ}{db^{(L)}} = \sum_{n=1}^{N} e_n^{(L)}$
    - $E^{(l)} = \left(W^{(l+1)} E^{(l+1)}\right) \odot f'\left(Z^{(l)}\right),\ \ l = L-1, L-2, \dots, 1$
    - $\frac{dJ}{dW^{(l)}} = A^{(l-1)} E^{(l)T}$
    - $\frac{dJ}{db^{(l)}} = \sum_{n=1}^{N} e_n^{(l)}$
  - ❖ Update of weights and biases:
    - $W_{t+1}^{(l)} = W_t^{(l)} - \eta \frac{dJ}{dW^{(l)}} = W_t^{(l)} - \eta A^{(l-1)} E^{(l)T}$
    - $b_{t+1}^{(l)} = b_t^{(l)} - \eta \frac{dJ}{db^{(l)}} = b_t^{(l)} - \eta \sum_{n=1}^{N} e_n^{(l)}$

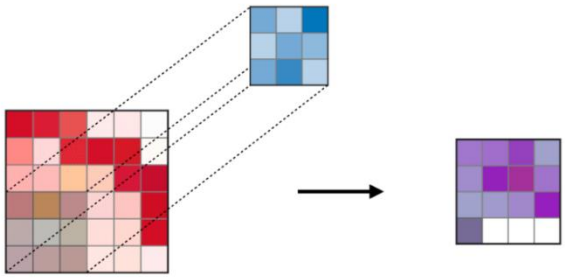
2. Convolutional Neural Network

2.1. General

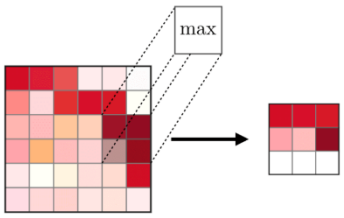
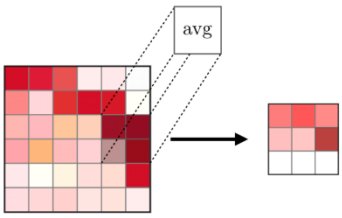
- Architecture of a traditional CNN: are generally composed of the following layers



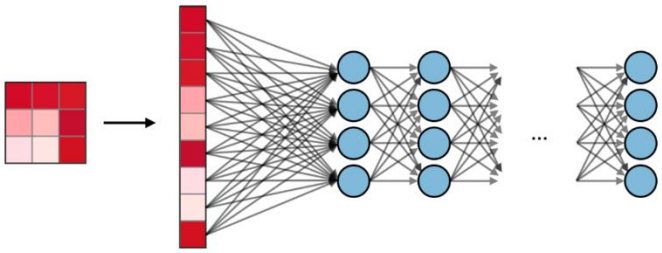
- Types of layer:
  - Convolution layer (CONV): uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size F and stride S. The resulting output O is called feature map or activation map.



  - Pooling layer (POOL): is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map • Used in LeNet

- Fully connected layer (FC): operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



- Tuning hyperparameters:
  - Size of output layer:

$$N = H_{new} \times W_{new} \times D_{new} = \left(\frac{H - F + 2P}{S} + 1\right) \times \left(\frac{W - F + 2P}{S} + 1\right) \times K$$

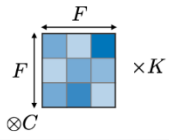
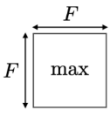
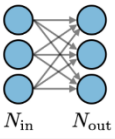
where:

H, W, D: height, width and depth of the input tensor

K: number of kernels, whose size is: $F \times F \times D$

S, P: stride and padding

  - Complexity of the model: In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

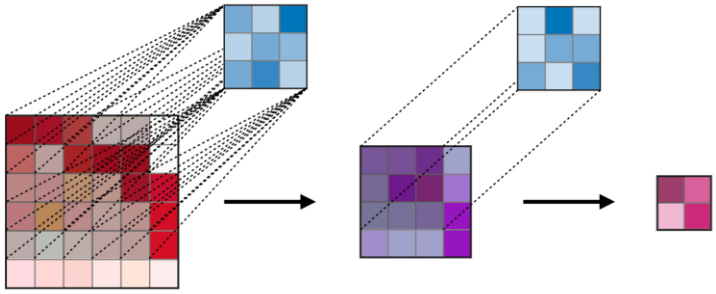
	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	• One bias parameter per filter • In most cases, $S < F$ • A common choice for K is $2C$	• Pooling operation done channel-wise • In most cases, $S = F$	• Input is flattened • One bias parameter per neuron • The number of FC neurons is free of structural constraints

  - Receptive field: The receptive field at layer k is the area denoted $R_k \times R_k$ of the input that each pixel of the k-th activation map can 'see'. By calling F_j the filter size of layer j and S_i

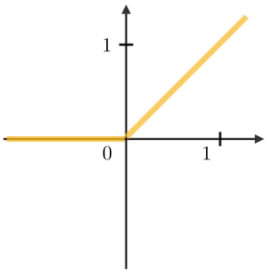
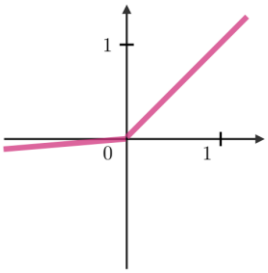
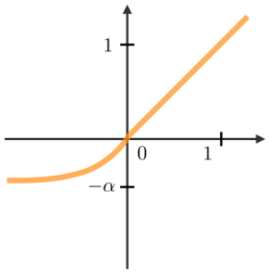
the stride value of layer i and with the convention $S_0 = 1$, the receptive field at layer k can be computed with the formula:

$$R_k = 1 + \sum_{j=1}^{k} (F_j - 1) \prod_{i=0}^{j-1} S_i$$

In the example below, we have $F_1 = F_2 = 3$, and $S_1 = S_2 = 1$, which gives $R_2 = 1 + 2 \cdot 1 + 2 \cdot 1 = 5$.



- Activation functions (commonly used):
  - Rectified Linear Unit: The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

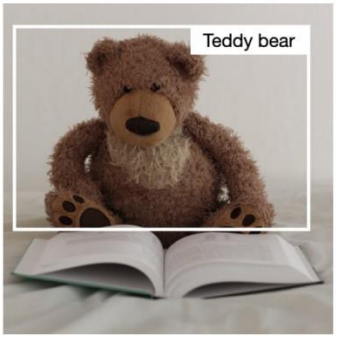
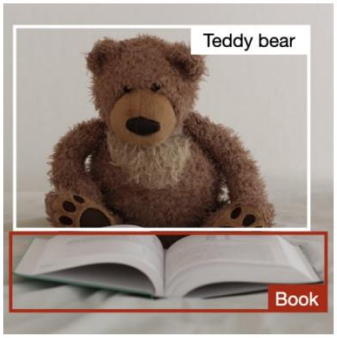
ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
• Non-linearity complexities biologically interpretable	• Addresses dying ReLU issue for negative values	• Differentiable everywhere

  - Softmax: The softmax step can be seen as a generalized logistic function that takes as input a vector of scores $x \in R^n$ and outputs a vector of output probability $x \in R^n$ through a softmax function at the end of the architecture. It is defined as:

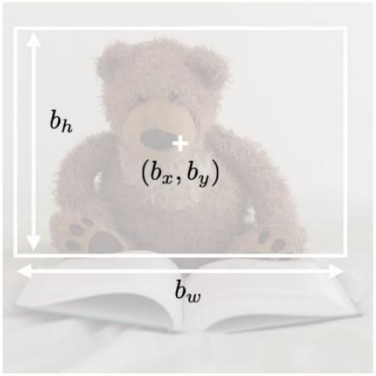
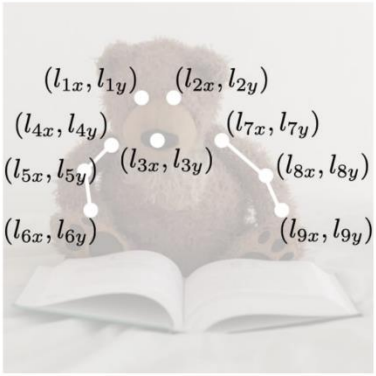
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

2.2. Object detection

- Types of models:

Image classification	Classification w. localization	Detection
		
• Classifies a picture • Predicts probability of object	• Detects an object in a picture • Predicts probability of object and where it is located	• Detects up to several objects in a picture • Predicts probabilities of objects and where they are located
Traditional CNN	Simplified YOLO, R-CNN	YOLO, R-CNN

- Data labeling:

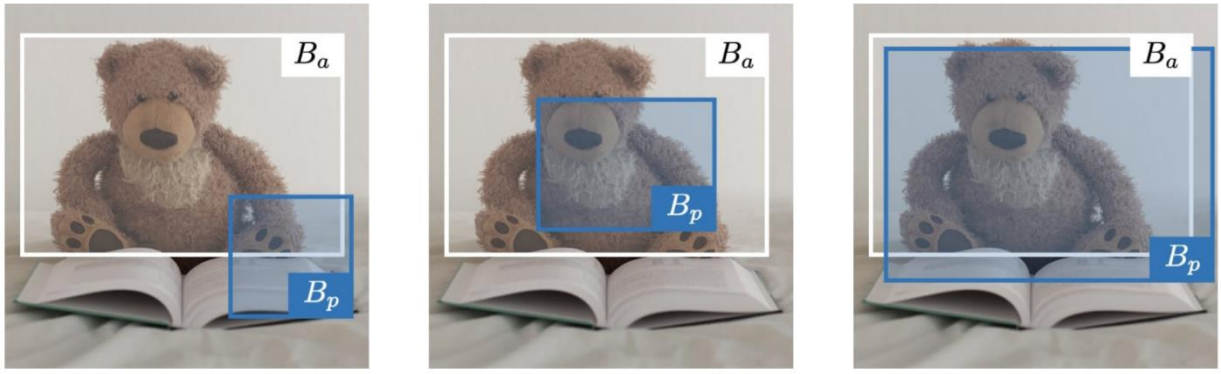
Bounding box detection	Landmark detection
• Detects the part of the image where the object is located	• Detects a shape or characteristics of an object (e.g. eyes) • More granular
	
Box of center (b_x, b_y), height b_h and width b_w	Reference points (l_{1x}, l_{1y}), ..., (l_{nx}, l_{ny})

- Intersection over Union (IoU): quantifies how correctly positioned a predicted bounding box B_p is over the actual bounding box B_a

$$IoU(B_p, B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$

Remarks: $IoU \in [0,1]$

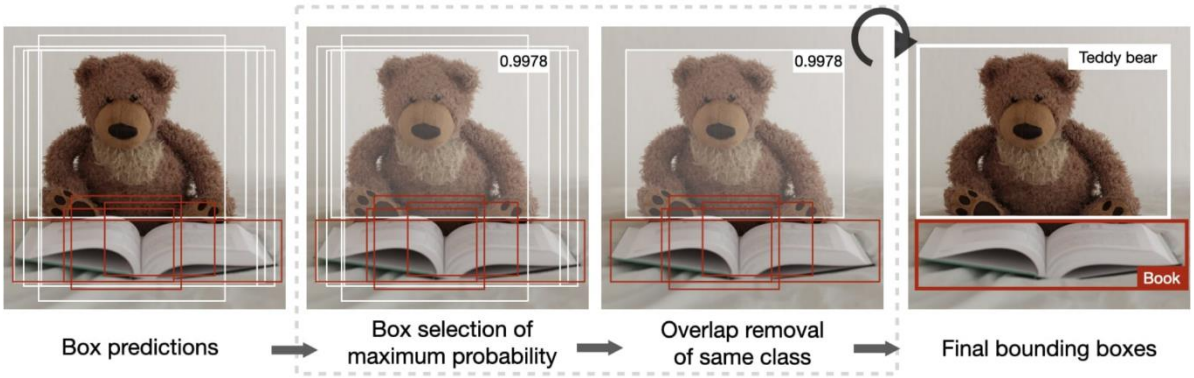
$IoU \geq 0.5$ is considered as good



$\text{IoU}(B_p, B_a) = 0.1$ $\text{IoU}(B_p, B_a) = 0.5$ $\text{IoU}(B_p, B_a) = 0.9$

- Anchor boxes: Anchor boxing is a technique used to predict overlapping bounding boxes. In practice, the network is allowed to predict more than one box simultaneously, where each box prediction is constrained to have a given set of geometrical properties. For instance, the first prediction can potentially be a rectangular box of a given form, while the second will be another rectangular box of a different geometrical form.

- Non-max suppression: is a technique to remove duplicate overlapping bounding boxes of a same object by selecting the most representative ones.
  - ❖ Algorithm:
    - Each output prediction is $[p_c\ b_x\ b_y\ b_h\ b_w]^T$. Discard all boxes with $p_c \leq 0.6$.
    - While there are any remaining boxes:
      - Pick the box with the largest p_c output that as a prediction
      - Discard any remaining box with $IoU \geq 0.5$ with the box output in the previous step



- YOLO algorithm:
  - Step 1: Divide the input image into a $G \times G$ grid
  - Step 2: For each grid cell, run a CNN that predicts y of the following form:

$$y = [\underbrace{p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_p}_{\text{repeated k times}}, \dots]^T \in R^{G \times G \times k \times (5+p)}$$

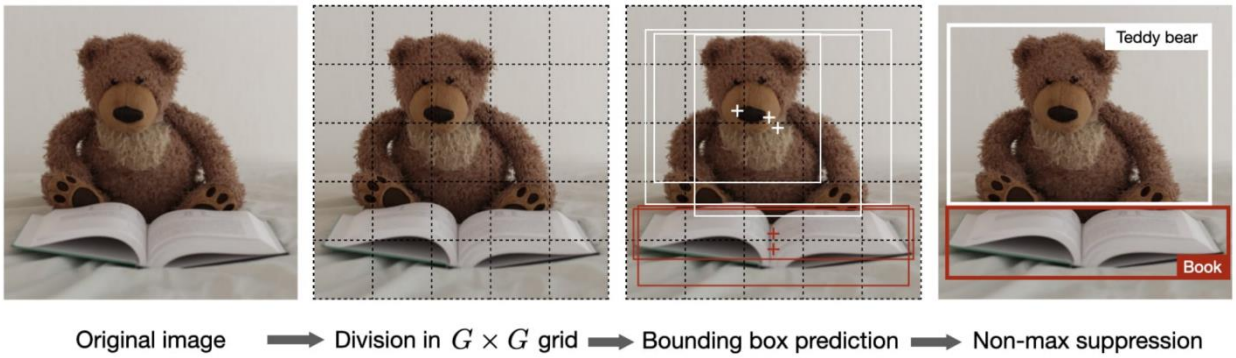
where: p_c is the probability of detecting an object

b_x, b_y, b_h, b_w are the properties of the detected bounding box

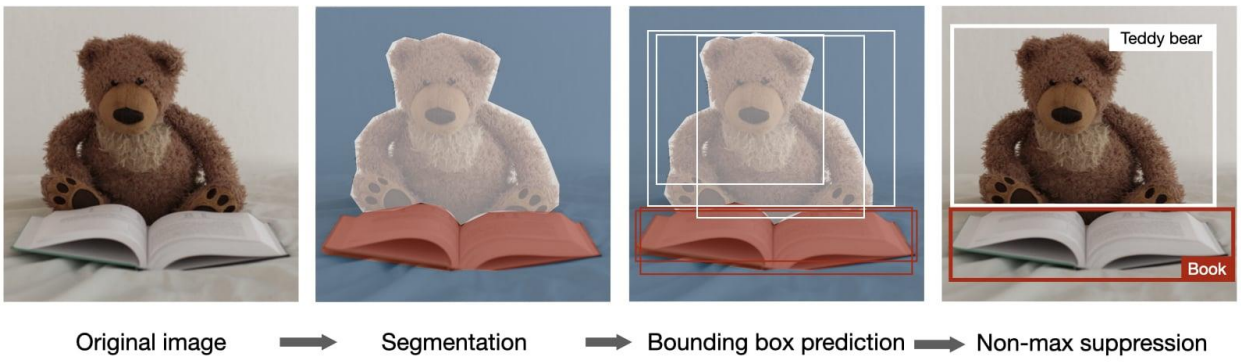
$c_1, c_2, \dots, c_p$ is a one-hot representation of which of the p classes were detected

k is the number of anchor boxes

- Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



- R-CNN algorithm: Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.

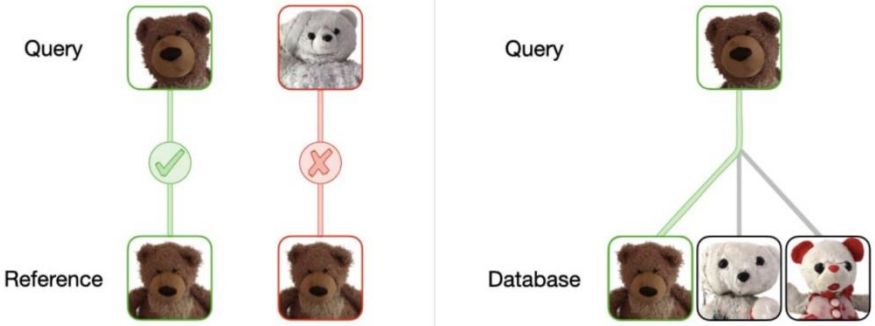


Remark: The original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.

2.3. Face verification and recognition

- Types of models:

Face verification	Face recognition
• Is this the correct person? • One-to-one lookup	• Is this one of the K persons in the database? • One-to-many lookup

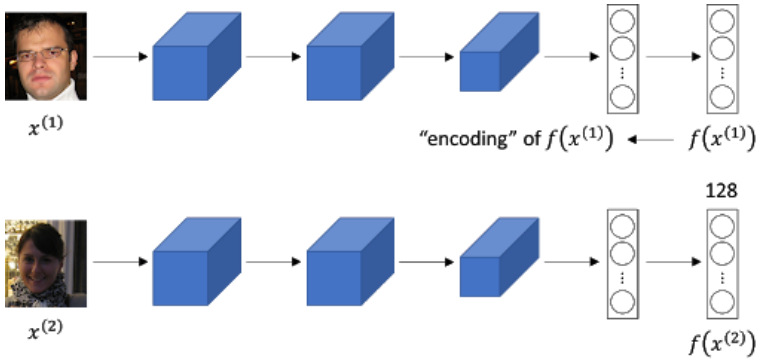


- One-shot learning: is a face verification algorithm that uses a limited training set to learn a similarity function that quantifies how different two given images are:

$$\text{Degree of differences between images} = \text{d}(\text{image 1}, \text{image 2})$$

$$\text{d}(\text{image 1}, \text{image 2}) = \begin{cases} \text{"same person"}, & \text{d} \leq \tau \\ \text{"different person"}, & \text{d} > \tau \end{cases}$$

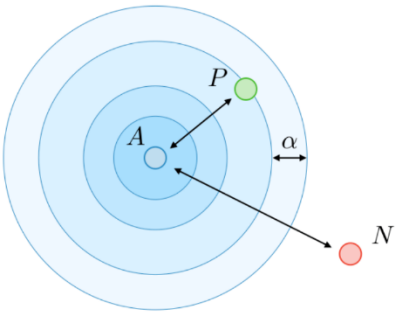
- Siamese Network: aims at learning how to encode images to then quantify how different two images are. For a given input image $x^{(i)}$, the encoded output is often noted as $f(x^{(i)})$.



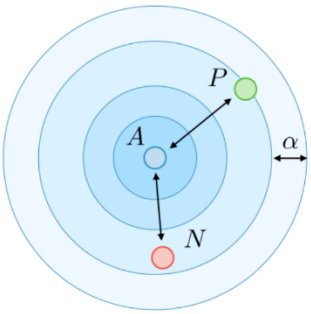
$$d(x^{(1)}, x^{(2)}) = ||f(x^{(1)}) - f(x^{(2)})||_2^2$$

- Triplet loss: The triplet loss l is a loss function computed on the embedding representation of a triplet of images A (anchor), P (positive) and N (negative). By calling $\alpha \in R^+$ the margin parameter, this loss is:

$$l(A, P, N) = \max\left(d(A, P) - d(A, N) + \alpha, 0\right)$$



$\ell(A, P, N) = 0$

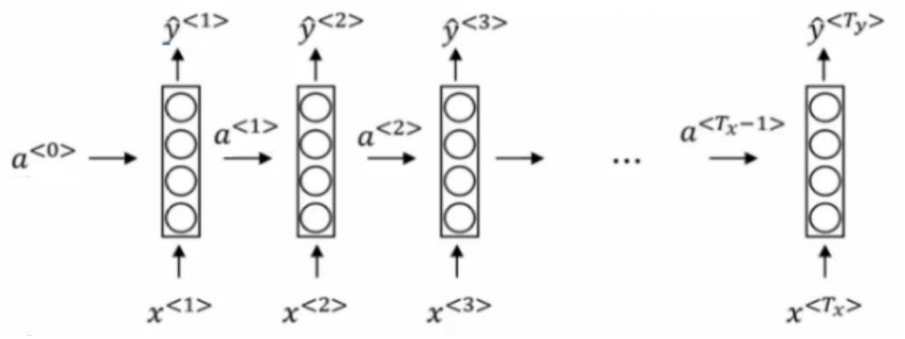


$\ell(A, P, N) > 0$

3. Recurrent Neural Network

3.1. Traditional RNN

- Architecture of a traditional RNN: Previous outputs are used as inputs while having hidden states



Notations:

- X, $x^{<i>}$: Input data and its i-th element vector
- Y, $y^{<i>}$: Actual output and its i-th element vector
  Example of named entity recognition:
  X = "Jisoo of BlackPink is so cute". $x^{<1>}$ = "Jisoo". $x^{<2>}$ = "of".
  Y = 1 0 1 0 0 0. $y^{<1>}$ = 1. $y^{<2>}$ = 0.
- $\hat{Y}$, $\hat{y}^{<i>}$: Predicted output matrix and i-th predicted output vector
- T_x, T_y: Size of the input sequence and size of the output sequence

- Forward propagation: For each timestep t, the activation $a^{<t>}$ and the predicted output $\hat{y}^{<i>}$ are expressed as:

$$a^{<0>} = \vec{0}$$
$$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a) \longrightarrow \boldsymbol{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)}$$
$$\hat{y}^{<1>} = g_2(W_{ya}a^{<1>} + b_y) \longrightarrow \boldsymbol{\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

where:

- $W_{ax} \in R^{N_a \times N_x}$, $W_{aa} \in R^{N_a \times N_a}$, $W_{ya} \in R^{N_y \times N_a}$
- $b_a, b_y \in R^{N_a \times 1}$
- N_a, N_x, N_y: The number of nodes in a, x, y
- g_1, g_2: The activation functions, g_1 is often a tanh/ReLU, g_2 is often a sigmoid/softmax

Simplified RNN notation:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \longrightarrow \begin{cases} a^{<t>} = g_1(W_a.[a^{<t-1>}, x^{<t>}] + b_a) \\ W_a = [W_{aa} \vdots W_{ax}] \end{cases}$$

$$\hat{y}^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \longrightarrow \hat{y}^{<t>} = g_2(W_y a^{<t>} + b_y)$$

where:

- $W_a \in R^{N_a \times (N_a + N_x)}$, is W_{aa} and W_{ax} stacked horizontally
- $[a^{<t-1>}, x^{<t>}] \in R^{(N_a + N_x) \times 1}$, is $a^{<t-1>}$ and $x^{<t>}$ stacked vertically

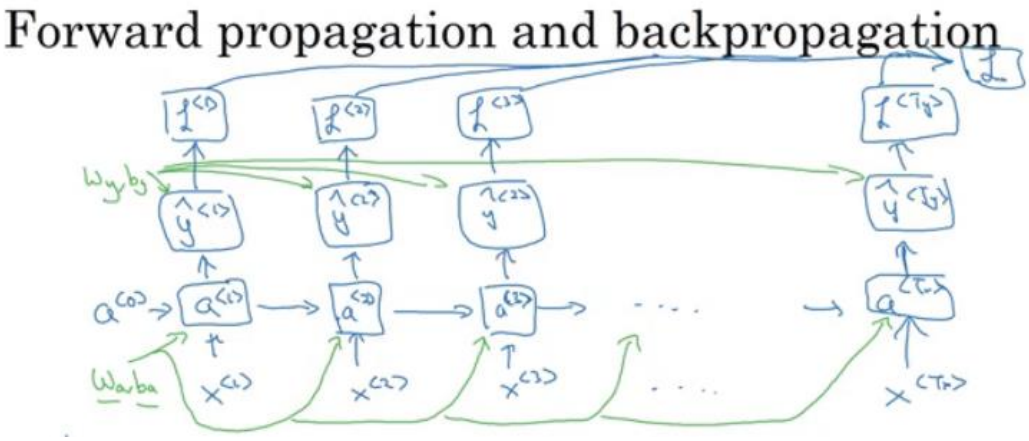
- Backpropagation through time: The backpropagation here is called backpropagation through time because we pass activation a from one sequence element to another like backwards in time

  Cross-entropy loss function:

$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log(1 - \hat{y}^{<t>})$$

$$\rightarrow L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

  W_a, b_a, W_y and b_y are shared across each element in a sequence:

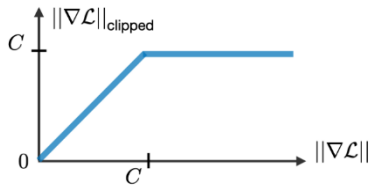


- Pros and cons of a typical RNN architecture:

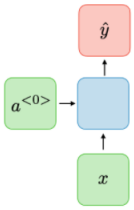
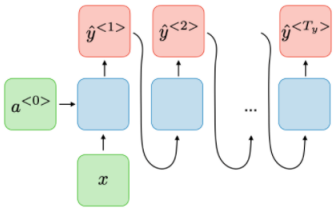
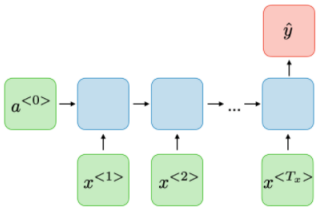
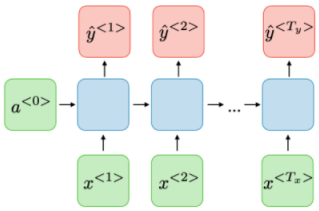
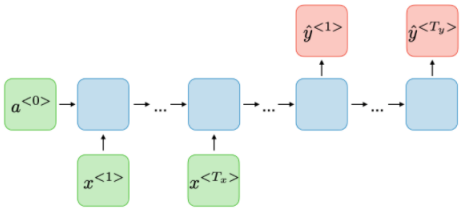
Advantages	Drawbacks
• Possibility of processing input of any length • Model size not increasing with size of input • Computation takes into account historical information • Weights are shared across time	• Computation being slow • Difficulty of accessing information from a long time ago • Cannot consider any future input for the current state

- Vanishing and exploding gradient: are often encountered in the context of RNNs. The reason why is that it is difficult to capture long-term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

- Gradient clipping: is used to cope with the exploding gradient problem by capping the maximum value for the gradient.



- Types of RNNs:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

3.2. Gated Recurrent Unit (GRU)

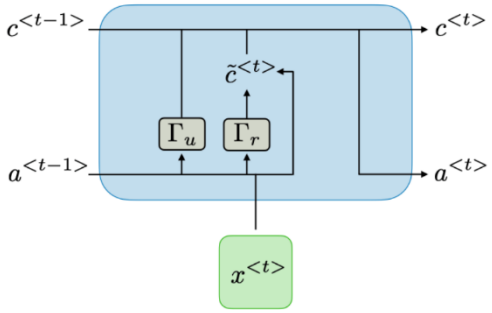
- Types of gates: In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted Γ and are equal to:

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

where W, U, b are coefficients specific to the gate and σ is the sigmoid function

Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

- Equations of GRU:



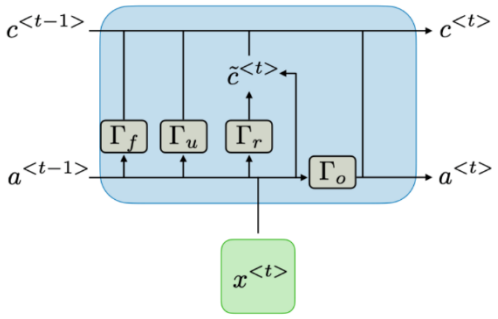
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[a^{<t-1>}, x^{<t>}] + b_r)$$
$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$$
$$c^{<t>} = \Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>}$$
$$a^{<t>} = c^{<t>}$$

where:

- $\star$ is the element-wise multiplication between two vectors
- $\Gamma_u, \Gamma_r, b_u, b_r, b_c, a^{<t>} \in R^{N_a \times 1}$ and N_a is the number of nodes in a
- $\tilde{c}^{<t>} \in R^{N_a \times 1}$: the new candidate values at layer t, to be added to $c^{<t>}$
- $c^{<t>} \in R^{N_a \times 1}$: the memory cell at layer t, tells to whether memorize something or not

3.3.Long Short-Term Memory units (LSTM)

- Equations of LSTM:



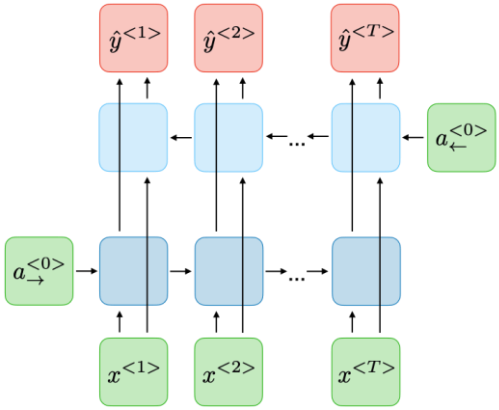
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_r = \sigma(W_r[a^{<t-1>}, x^{<t>}] + b_r)$$
$$\Gamma_f = \sigma\big(W_f[a^{<t-1>}, x^{<t>}] + b_f\big)$$
$$\Gamma_o = \sigma(W_r[a^{<t-1>}, x^{<t>}] + b_o)$$
$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$$
$$c^{<t>} = \Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$$
$$a^{<t>} = \Gamma_o \star \tanh(c^{<t>})$$

where:

- $\star$ is the element-wise multiplication between two vectors
- $\Gamma_u, \Gamma_r, \Gamma_f, \Gamma_o, b_u, b_r, b_f, b_o, b_c, a^{<t>} \in R^{N_a \times 1}$ and N_a is the number of nodes in a
- $\tilde{c}^{<t>} \in R^{N_a \times 1}$: the new candidate values at layer t, to be added to $c^{<t>}$
- $c^{<t>} \in R^{N_a \times 1}$: the memory cell at layer t, tells to whether memorize something or not

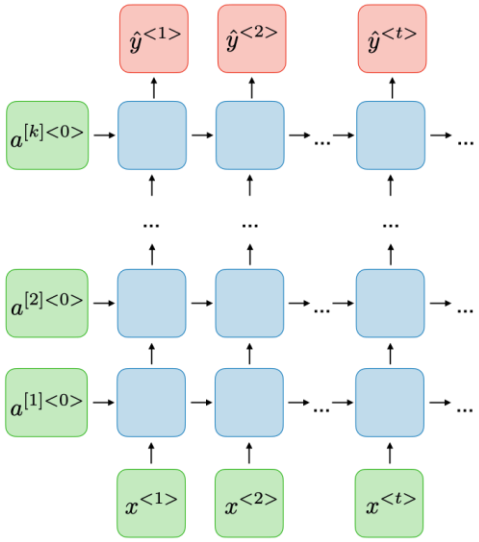
3.4. Bidirectional RNN (BRNN)

- Functional principles:
  - BRNN is an acyclic graph.
  - Part of the forward propagation goes from left to right, and part goes from right to left. It learns from both sides.
  - To make predictions we use $\hat{y}$ by using the two activations that come from left and right.



3.5. Deep RNN (DRNN)

- Functional principles:
  - Stack some RNN layers to make a deeper network
  - Stacking 3 layers is already considered deep and expensive to train



4. Deep Learning Tips and Tricks

4.1.Data processing

- Data augmentation: to get more data from the existing ones

Original	Flip	Rotation	Random crop
			
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

- Batch normalization: It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

4.2. Training a neural network

- Epoch: one iteration where the model sees the whole training set to update its weights.

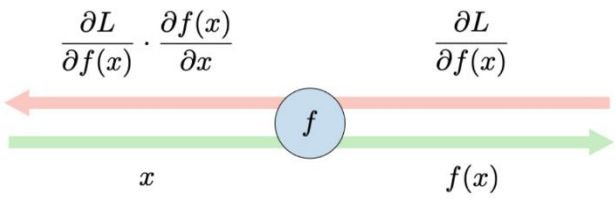
- Mini-batch gradient descent: during the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter (batch size) that we can tune.

- Loss function: In order to quantify how a given model performs, the loss function L is usually used to evaluate to what extent the actual outputs y are correctly predicted by the model outputs z.

- Cross-entropy loss: In the context of binary classification in neural networks, the cross-entropy loss $L(z, y)$ is commonly used and is defined as follows:

$$L(z, y) = -[y \log(z) + (1 - y) \log(1 - z)]$$

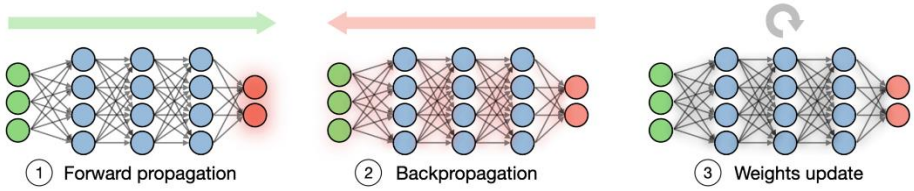
- Backpropagation: is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.



Using this method, each weight is updated with the rule:

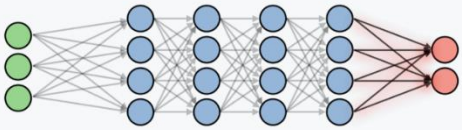
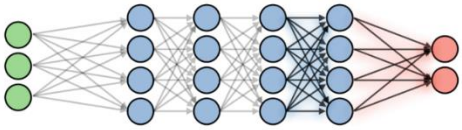
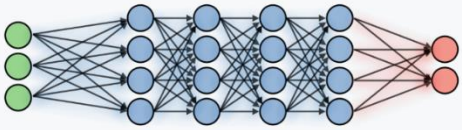
$$w \leftarrow w - \alpha \frac{\delta L(z, y)}{\delta w}$$

- Updating weights: In a neural network, weights are updated as follows:
  - Step 1: Take a batch of training data and perform forward propagation (feedforward) to compute the loss.
  - Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
  - Step 3: Use the gradients to update the weights of the network.



4.3. Parameter tuning

- Weights initialization:
  - Xavier initialization: Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.
  - Transfer learning: It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

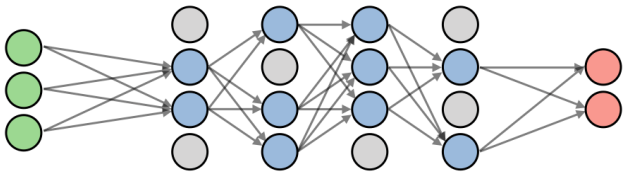
- Optimizing convergence:
  - Learning rate: The learning rate, often noted α or η, indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.
  - Adaptive learning rates: Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful:

Method	Explanation	Update of w	Update of b
Momentum	• Dampens oscillations • Improvement to SGD • 2 parameters to tune	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	• Root Mean Square propagation • Speeds up learning algorithm by controlling oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	• Adaptive Moment estimation • Most popular method • 4 parameters to tune	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \longleftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD

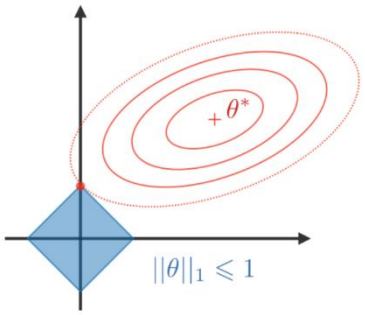
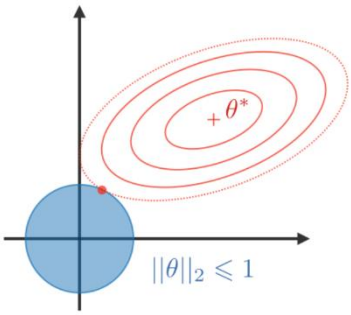
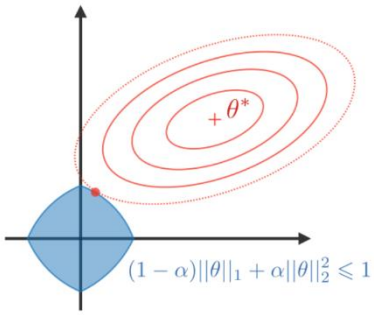
4.4. Regularization

- Dropout: to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.

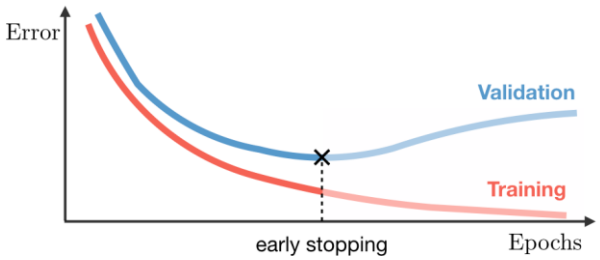


Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter $1 - p$.

- Weight regularization: In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights:

LASSO	Ridge	Elastic Net
• Shrinks coefficients to 0 • Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$\ldots + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

- Early stopping: to stop the training process as soon as the validation loss reaches a plateau or starts to increase.



4.5. Good practices

- Overfitting small batch: When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

- Gradient checking: Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

Type	Numerical gradient	Analytical gradient
Formula	$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Comments	• Expensive; loss has to be computed two times per dimension • Used to verify correctness of analytical implementation • Trade-off in choosing h not too small (numerical instability) nor too large (poor gradient approximation)	• 'Exact' result • Direct computation • Used in the final implementation

5. Important scientific paper

5.1. Residual Neural Network (ResNet)

- ResNet: in reality, when training a model with high number of layers, the training error cannot be decreased anymore after n-th layer, causing “vanishing gradients” problem. ResNet creates residual blocks, following the equation:

$$a^{[l+2]} = g\left(a^{[l]} + z^{[l+2]}\right)$$