AI Notes

Huu Duc Nguyen M.Sc.

29 February 2022

Contents

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CS	Computer Science
CV	Computer Vision
RL	Reinforcement Learning
NLP	Natural Language Processing
GPU	Graphics Processing Unit
CPU	Central Processing Unit
prob.	probability
params.	parameters
algor.	algorithms
a.k.a.	also known as
w.r.t.	with regard to
no.	number of
func.	function
vs.	versus
pdf.	Probability Density Function
MLE	Maximum Likelihood Estimation
MAP	Maximum A Posteriori
MoG	Mixture of Gaussians
SVM	State Vector Machine
SGD	Stochastic Gradient Descent
NAG	Nestorov Accelerated Gradient
RMSprop	Root mean squared prop
Adam	Adaptive moment estimation
PCA	Principal Component Analysis
LDA	Linear Discriminant Analysis
KL	Kullback–Leibler
KKT	Karush-Kuhn-Tucker
RBF	Radial basic function
MLP	Multi-Layer Perceptron
ReLU	Rectified Linear Unit
BPTT	Backpropagation through time

RNN	Recurrent Neural Network
LSTM	Long short-term memory
CNN	Convolutional Neural Network
GNN	Graph Neural Network
CONV	Convolutional
FC	Fully Connected
VAE	Variational Auto-Encoders
GAN	Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
CGAN	Conditional Generative Adversarial Network
ResNet	Residual Network
BatchNorm	Batch Normalization
IN	Instance Normalization
AdaIN	Adaptive Instance Normalization
NAS	Neural Architecture Search
DPM	Deformable Part Model
HOG	Histogram of Oriented Gradients
CS	conditioned stimuli

1 Introduction

Artificial Intelligence (AI) is the study field that leverages the ability of machines to mimic the problem-solving skill of human. In other words, AI pushing machines to think, act like people, in a rational way. It lies in the core of countless novel applications in real life, self-driving cars, virtual assistant, face recognition, etc. Machine Learning (ML) is a sub-field of conditioned stimuli (CS) and AI, that "gives computers the ability to learn without being explicitly programmed" (Wikipedia). Deep Learning (DL) is then a subset of ML algorithms, that take advantage of the great amount of collected data and new powerful computational hardware (Fig. 1.1). Advanced applications which relate to Natural Language Processing (NLP), Computer Vision (CV), robotic learning, etc., are with in this DL subset.

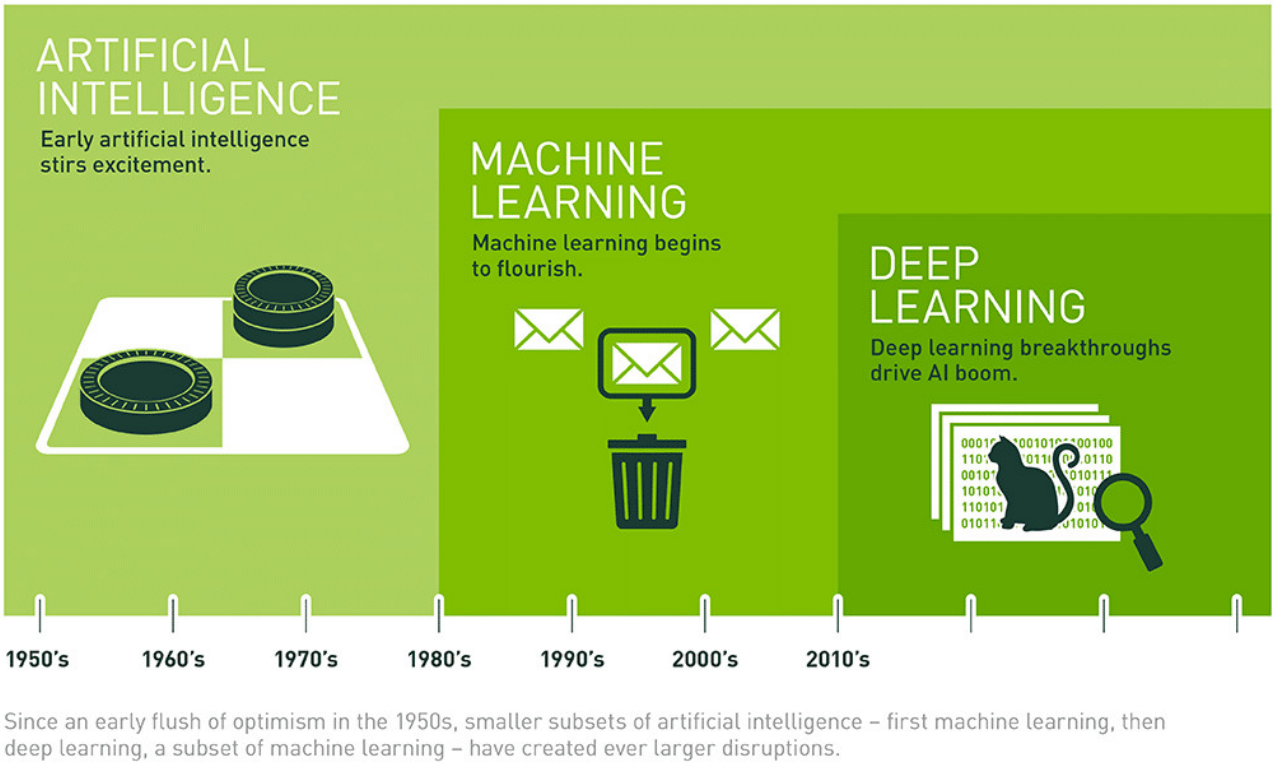


Figure 1.1: The relation between AI, ML and DL (src).

It's important to understand that there are more to AI and ML than just neural networks. In the end, to create a meaningful and working neural network, one should have strong background in the basics of ML as well.

These notes are my way of keeping record of what I have learn in the field. The structure of the notes is as follows:

- Chap. 2 introduces common ideas in ML.
- Chap. 3 presents the mathematics background on probabilities, matrix.
- [**TODO: chapter 3**] explain basic concepts, the branching of different classes in ML. Later chapters presents each smaller branches.

[**TODO: The structure of the notes**]

2 Overview of Machine Learning

A machine learning algorithm is an algorithm that has the ability to *learn* from the data. A computer program is said to **learn**, if its performance at tasks in T, measured by P, improves with experience E (in which the experience is equivalent to the data). [GBC16]

2.1 Tasks

A *task* is usually described by how the ML model process a single *data point*. This section presents some common ML tasks. [Vu18]

2.1.1 Classification

The task is to specify a label for the given data point. The labels are usually members of a list. E.g.:

- Hand-written digit classification
  - The data point: images of hand-written numbers with their labels
  - The task: tell which number is in a unseen image
  - Labels: there are 10 possible labels, i.e., $\{0, 1, \ldots, 9\}$.
- Hand-written letter classification
  - The data point: images of hand-written letters with their labels
  - The task: tell which letter is in a unseen image
  - Labels: there are 26 (or more) possible labels, i.e., $\{a, b, \ldots, z\}$.

2.1.2 Regression

If the desired output is a real value, instead of a label in a list, then it's a regression problem. E.g.:

- Input data: a person image $\longrightarrow$ Output: predicts the age of the person
- Input data: a feature vector $\longrightarrow$ Output: generates an image
- Input data: dimension of a block of gold $\longrightarrow$ Output: the price of it

2.1.3 Clustering

This is the task of grouping relevant data points based on some relationship between them.

E.g., find the pattern in customer shopping behaviors.

2.1.4 Others

Other worth-mentioning tasks:

- Recommendation System
- Machine Translation
- Completion
- Ranking
- Information Retrieval
- Denoising

2.2 Performance

Usually, the dataset is divided into *training set* and *test set*. The model uses the training set to tune/update the model parameters (params.), and the test set to examine the performance.

Online training is the approach when new data will continuously arise and introduce for the model to learn. E.g. in Reinforcement Learning (RL). *Offline training* is the opposite case where the model learns from the a fixed training set.

2.3 Experience Types

2.3.1 Supervised Learning

Data with labels: Supervised Learning is the approach that predict the outputs of new data points based on pairs of known inputs and outputs. This is the most common type of ML algorithms (algor.).

2.3.2 Unsupervised Learning

Data without labels: On the opposite, with unsupervised Learning, there isn't known output, just inputs. Unsupervised Learning algor. will carry on some tasks which base on the characteristics of the dataset, e.g. clustering, dimension reduction.

2.3.3 Semi-supervised Learning

Some data with, some without labels

2.4 Model Parameters and Loss Function

Each ML model is described by a set of model params.. E.g., in the problem of finding a line passing through points in the 2D plane, the model params. are a, b in the line equation $y = ax+b$. The of training aim is to find the model params. which leads to the best performance or the minimum loss. For classification problems, it can be having the least number of incorrect classified data points. For regression problems, it can be having the smallest difference with the actual output. It is then equivalent to having an optimization problem, in which we try to minimize that loss/cost function.

3 Probabilities

Many ML algorithms and problems concern with the probability (prob.) or Probability Density Function (pdf.) of the outputs from complex inputs. We all aware of some basic examples of the output prob. from simple inputs, e.g., flipping a coin, rolling a dice. With these simple inputs, due to uncertainty, one could only say about the prob. that the output could be, e.g., flipping a coin has 50% of getting head, 50% of getting tail. Comparing to flipping a coin, inputs for ML algorithms are usually more complex, in higher dimensions, in different modals, etc. The inputs can be datasets, images, audio, etc. However, as with the simple inputs, we concern ourselves with the prob. of the outputs, e.g.:

- Given a dog image, output the prob. of the breed of the dog.
- Image segmentation: given an image, output the prob. of object class for each pixel.
- Cancer detection: given a medical image, output the prob. of the patient having cancer.

This chapter presents important prob. problems relating to ML. For mathematics foundation on prob., check the mathematics notes.

[TODO: Add image: image with prob. of being cat or dog]

3.1 Parameter Estimation

Many of ML problems are boiled down to finding *statistical models*. Those models predict the prob. of the data belongs to one class (in classification problem), prob. of events that will happen, etc. It all end up with finding a suitable set of params. for these *statistical models*.

3.1.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) finds the parameters that maximize the prob. of the existing data.

Model parameter θ: $$\theta = \arg\max_{\theta} p(x_1, x_2, \ldots, x_N|\theta) \tag{3.1}$$

Assuming independent variables: $$\theta = \arg\max_{\theta} \prod_{n=1}^{N} p(x_n|\theta) \tag{3.2}$$

Maximum log-likelihood: $$\theta = \arg\max_{\theta} \sum_{n=1}^{N} [\log p(x_n|\theta)] \tag{3.3}$$

Minimum negative log-likelihood: $$\theta = \arg\min_{\theta} \sum_{n=1}^{N} [-\log p(x_n|\theta)] \tag{3.4}$$

3.1.2 Maximum A Posteriori

Sometimes, prior knowledge of the pdf. is presented. E.g., the prob. of getting head when flipping a coin is around 50%. Maximum A Posteriori (MAP) takes advantage of the prior knowledge $p(\theta)$ on the parameters θ by applying Bayes rule.

$$\theta = \arg\max_{\theta} \prod_{n=1}^{N} p(x_n|\theta)p(\theta) \tag{3.5}$$

MLE (Sec. 3.1.1) suffers when there is not enough data ⇒ use MAP

3.2 Naive Bayes Classifier

The word *naive* implies having the independence assumption on the variables.

$$c = \arg\max_{c\in\mathbb{C}} p(c|x) \tag{3.6}$$

$$= \arg\max_{c\in\mathbb{C}} p(x|c)\, p(c) \tag{3.7}$$

If x is:

- continuous variable ⇒ Gaussian Naive Bayes
- feature vector ⇒ Multinomial Naive Bayes
- binary vector ⇒ Bernoulli Naive Bayes

Minimize the expected loss: $\mathbb{E}[L] = \sum_k \sum_j \int_{R_j} L_{kj}\, p(x, C_k)\, dx$ by choosing region R_j such that $\mathbb{E}[L] = \sum_k L_{kj}\, p(C_k|x)$

3.3 Views on the Decision Problem

3.3.1 Generative Methods

First determine the class-conditional densities and separately infer the prior class prob. ⇒ Bayes theorem ⇒ class membership

$$p(x|C_k)\, p(C_k) \Rightarrow y_k(x)$$

E.g., Mixture of Gaussians

3.3.2 Discriminative Methods

First solve the inference problem of determined the posterior class prob.

NOTE: Frequentist versus Bayesian

- Frequentist: the params. is fixed $p(\mathcal{D}|\theta)$
- Bayesian: the params. depends on the given data $p(\theta|\mathcal{D})$

3.3.3 Classification with Loss Functions

For classification problem with more than 2 classes, the output's predicted class $\mathcal{C}_k$ is the one with the greatest posterior prob. among all classes $\mathcal{C}_j$, which can still be wrong:

$$p(\mathcal{C}_k|x) > p(\mathcal{C}_j|x) \qquad \forall j \neq k$$

$$p(x|\mathcal{C}_k)p(\mathcal{C}_k) > p(x|\mathcal{C}_j)p(\mathcal{C}_j) \qquad \forall j \neq k$$

Loss function differentiate the possible decisions and the possible true classes. E.g.:

- Decisions: *sick* or *healthy*
- Classes: patient is *sick* or *healthy*

The cost may be symmetric. In this case however, it is asymmetric:

loss(decision=healthy | patient=sick) >> loss(decision=sick | patient=healthy)

The classification problem could be formalized by introducing loss matrix L_{kj}

$$L_{kj} = \textit{loss for decision } \mathcal{C}_j \textit{ if truth is } \mathcal{C}_k$$

E.g., cancer diagnosis:

$$L_{cancer\ diagnosis} = \begin{array}{cc} & \textbf{Decision} \\ & \begin{array}{cc} cancer & normal \end{array} \\ \textbf{Truth} \begin{array}{c} cancer \\ normal \end{array} & \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix} \end{array}$$

Loss functions maybe different for different actors:

$$L_{stocktrader}(subprime) \neq L_{bank}(subprime)$$

3.3.4 Minimize the Expected Loss

But regardless, the optimal solution is the one that minimizes the *expected loss*. It is *expected* loss because it depends on the true class, which is unknown.
$\Rightarrow$ ***Minimize the expected loss***

E.g., 2 class $C_1,\ C_2$, 2 decisions $\alpha_1,\ \alpha_2$.

The loss: $L(\alpha_j|C_k) = L_{kj}$.

The expected loss is equal to the risk R.

$$\mathbb{E}_{\alpha_1}[L] = R(\alpha_1|x) = L_{11}\,p(C_1|x) + L_{21}\,p(C_2|x)$$
$$\mathbb{E}_{\alpha_2}[L] = R(\alpha_2|x) = L_{12}\,p(C_1|x) + L_{22}\,p(C_2|x)$$

Choose α_1 if $R(\alpha_1|x) < R(\alpha_2|x)$

3.4 Probability Density Estimation

The following approaches' aim is to estimate an unknown pdf..

- ***Non-parametric approaches:*** represent the pdf. without parameter, usually as a lookup table. Using lookup table is fast, but could also be memory-consuming.
  Among the presented ones, only histogram is non-parametric.
- ***Parametric approaches:*** within sufficiently small region $\mathcal{R}$, the prob. can be estimated as $p = \int\limits_{\mathcal{R}} p(y)dy \approx p(x)V = \frac{K}{N.V}$, where K is the number of data points in the region, V is the volume of the region.

3.4.1 Histogram

The prob. of a bin:

$$p_i = \frac{n_i}{N.\Delta_i} \tag{3.8}$$

in which n_i is the number of data points in that bin, N is the total number of data point, Δ_i is the width of the bin, often $\Delta_i = \Delta$.

Notes:

- Δ serves as the **smoothing factor**
- With D as the dimensions of the data points. The number of bins grow exponentially with $\mathcal{O}(k^D)$

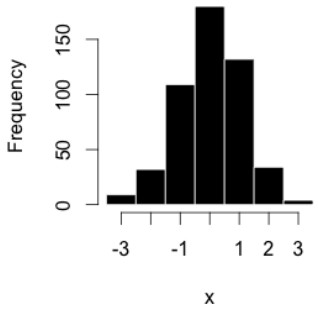


Figure 3.1: Example of a histogram, $\Delta = 1$ (src).

3.4.2 Kernel Methods

The kernel methods fix the volume V and determine the number of data points K. The volume V is the space restricted within a parzen window $k(u)$ that satisfies $k(u) \geq 0$.

A hyper-space cube:
$$k(u) = \begin{cases} 1 & if \quad |u_i| \leq \frac{1}{2}h, \quad i = 1, 2, \ldots, D \\ 0 & else \end{cases} \tag{3.9}$$

The number of points inside:
$$K = \sum_{n=1}^{N} k(x - x_n) \tag{3.10}$$

The region *volume*:
$$V = \int k(u)du = h^D \tag{3.11}$$

The probability:
$$\Rightarrow p(x) \approx \frac{K}{N.V} = \frac{1}{N.h^D} \sum_{n=1}^{N} k(x - x_n) \tag{3.12}$$

NOTE: the above parzen window is asymmetric.
The ***symmetric Gaussian kernel is a better substitution***.

$$\text{A Gaussian kernel:} \qquad k(u) = \frac{1}{\sqrt{2\pi h^2}} exp\left(\frac{-u^2}{2h^2}\right) \tag{3.13}$$

$$\text{The region } volume\text{:} \qquad V = \int k(u)du = 1 \tag{3.14}$$

$$\text{The probability:} \qquad \Rightarrow p(x) \approx \frac{1}{N}\sum_{n=1}^{N} \frac{1}{(2\pi)^{D/2}h} exp\left(\frac{-||x - x_n||^2}{2h^2}\right) \tag{3.15}$$

For Kernel methods, h is the ***smoothing factor***.

Generalization: $k(u) \geq 0$, $\int k(u)du = 1$.

Size of the hypersphere is proportional to h^2.

3.4.3 K-Nearest Neighbor

When you fix the number of data points K and determine the volume V, it leads to K-Nearest Neighbor.

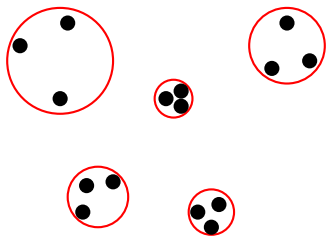


Figure 3.2: K-Nearest Neighbor with $K = 3$.

$$p(x) \approx \frac{K}{NV} \tag{3.16}$$

Here, K is the ***smoothing factor***.

- **Too much bias ⇒ too smooth**
- **Too much variance ⇒ NOT smooth enough**

⇒ combine parametric methods to a mixture model

Mixture distribution = multi-parametric model

3.4.4 Mixture of Gaussians

Mixture of Gaussians (MoG), as **Generative Model**, is defined from the sum prob. of elemental Gaussians: $p(x|\theta) = \sum_{j=1}^{M} p(x|\theta_j)p(j)$, where $p(x|\theta_j)$ is a **mixture component**, $p(j) = \pi_j$ is the **weight of the component**

$$p(x|\theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left[\frac{-(x-\mu_j)^2}{2\sigma_j^2}\right], \quad p(j) = \pi_j, \quad \sum \pi_j = 1 \tag{3.17}$$

$$p(x|\theta_j) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma_j|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)\right] \tag{3.18}$$

3.4.5 K-Means Clustering

There are 3 steps. Step 2 and 3 are repeated until there is no change.

1. Pick K centroids
2. Assign sample to the centroid
3. Adjust centroids

NOTE: Check machinelearningcoban.com.

- This leads to a local optimum, depends on initialization.
- It's sensitive to ***outliers***, detects ***spherical clusters only***.
- Application: e.g., image compression.

3.4.6 EM Clustering

Assuming N data points and K Gaussians.

- **E-Step:** Fix the Gaussians, find $\gamma_j(x)$, which represent the **responsibility of component j for data point x**.

$$\gamma_j(x_n) = \frac{\pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)} \qquad \forall j = 1, 2, \ldots, K, \quad n = 1, 2, \ldots, N \tag{3.19}$$

- **M-Step:** Fix $\gamma_j(x)$, update the Gaussians.

$$\hat{N}_j = \sum_{n=1}^{N} \gamma_j(x_n) \tag{3.20}$$

$$\hat{\mu}_j = \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(x_n) x_n \tag{3.21}$$

$$\hat{\pi}_j = \frac{\hat{N}_j}{N} \tag{3.22}$$

$$\hat{\Sigma}_j = \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(x_n) \left(x_n - \hat{\mu}_j\right) \left(x_n - \hat{\mu}_j\right)^T \tag{3.23}$$

Notes:

- EM is short for Expectation-Maximization
- Regularization with $\Sigma + \sigma_{min} I$
- Initialization μ_j with K-Means
- Compare with K-Means
  - Hard-assignment $\Leftrightarrow$ K-Means: each data point to 1 class
  - Soft-assignment $\Leftrightarrow$ EM Clustering: each data point $\Rightarrow$ prob. to fall into many classes
  - With **more params.**, EM **needs more iteration**.

4 Basic ML Problems

4.1 Linear Regression

This is the simplest regression problem in ML.

Problem statement: Given data points $\mathbf{x}_i \in \mathbb{R}^D$ and their labels $y_i \in \mathbb{R}$, find the *line* that fits these data points. The line is represented via parameters $\mathbf{w} = [w_0, w_1, \ldots, w_n]^T$. For each data point $\mathbf{x}$ and its label y.

Approach:

$$\bar{\mathbf{x}} = [1, x_0, \ldots, x_n] \qquad \text{(x bar - extended variable)}$$

$$y \approx \hat{y} = \bar{\mathbf{x}}.\mathbf{w} \qquad \text{(y hat - predicted label)}$$

$$\Rightarrow \frac{1}{2}e^2 = \frac{1}{2}\left(y - \bar{\mathbf{x}}.\mathbf{w}\right)^2 \qquad \text{(the squared error between true and predicted labels)}$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \bar{\mathbf{x}}_i.\mathbf{w}\right)^2 \qquad \text{(the loss function for all points)} \tag{4.1}$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \bar{\mathbf{x}}_i.\mathbf{w}\right)^2 \qquad \text{(the average loss)} \tag{4.2}$$

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\,\mathcal{L}(\mathbf{w}) \qquad \text{(the weights that minimize the loss function)} \tag{4.3}$$

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}||\mathbf{y} - \overline{\mathbf{X}}.\mathbf{w}||_2^2 \qquad \text{(using matrix form)} \tag{4.4}$$

with $\overline{\mathbf{X}} = \begin{bmatrix} \bar{\mathbf{x}}_1 \\ \bar{\mathbf{x}}_2 \\ \vdots \\ \bar{\mathbf{x}}_n \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

Solution:

$$\frac{\partial\mathcal{L}(\mathbf{w}^*)}{\partial\mathbf{w}} = \overline{\mathbf{X}}^T\left(\overline{\mathbf{X}}\mathbf{w}^* - \mathbf{y}\right) = 0 \tag{4.5}$$

$$\Leftrightarrow \overline{\mathbf{X}}^T\overline{\mathbf{X}}\mathbf{w}^* = \overline{\mathbf{X}}^T\mathbf{y} \tag{4.6}$$

$$\Leftrightarrow \mathbf{w}^* = \left(\overline{\mathbf{X}}^T\overline{\mathbf{X}}\right)^{\dagger}\overline{\mathbf{X}}^T\mathbf{y} \tag{4.7}$$

in which, $A^{\dagger}$ (A dagger) is the pseudo inverse of a matrix, because A might not be inverse-able.

$$\boxed{A^{\dagger} = \left(A^T A\right)^{-1} A^T}$$

NOTE:

- Sensitive to outliers ⇒ pre-processing
- Multi-variable:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}||\mathbf{y} - \overline{\mathbf{X}}.\mathbf{w}||_2^2$$
$$\Rightarrow \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \frac{1}{N}\overline{\mathbf{X}}^T(\overline{\mathbf{X}}\mathbf{w} - \mathbf{y})$$

4.2 Linear Discriminant Functions

Problem statement: of general classification problem

- Given: training set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ with target values (labels) $\mathbf{T} = \{\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_N\}$.
- Goal: take a new input $\mathbf{x}$ and assign it to one of K classes C_k
  ⇒ Learn a discriminant function $y(x)$ to perform the classification

Approach:

- 2-class problem: Binary target values: $t_n \in \{0, 1\}$
  E.g. of discriminant function: $y(x) > 0 \Rightarrow$ class C_1, else C_2
- K-class problem: 1-of-K coding scheme, e.g.: $\mathbf{t}_n = [0, 1, 0, 0, 0]^T$
- Extension to multiple classes: one-vs-all, one-vs-one classifiers

4.2.1 Least-Squares Classification

With above problem statement, consider K classes described by linear models:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T\mathbf{x} + w_{k0} \qquad k = 1, \ldots, K$$
$$\Rightarrow \mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^T\widetilde{\mathbf{x}} \qquad \text{(using vector notation)}$$
$$\mathbf{Y}(\widetilde{\mathbf{X}}) = \widetilde{\mathbf{X}}\widetilde{\mathbf{W}} \qquad \text{(for the entire dataset)}$$

In which output $\mathbf{y}$ in 1-of-K notation, and can be compared to the target value

$$\mathbf{t} = [t_1, \ldots, t_k]^T$$

Solution: minimize the sum-of-squares error $E(\mathbf{w})$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{y} - \mathbf{t})^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(\mathbf{w}_k^T\mathbf{x}_n - t_{kn}\right)^2$$
$$\Rightarrow \widetilde{\mathbf{W}}^* = \left(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}\right)^{-1}\widetilde{\mathbf{X}}^T T = \widetilde{\mathbf{X}}^{\dagger}T$$

NOTE: Least-squares is very sensitive to outliers!
The error function penalizes predictions that are "too correct".

4.2.2 Generalized Linear Discriminant

$$y_k(x) = \sum_{j=1}^{M} w_{kj}\phi_j(x) + w_{k0} = \sum_{j=0}^{M} w_{kj}\phi_j(x), \qquad \phi_0(x) = 1 \tag{4.8}$$

4.2.3 Generalized Linear Models

- Linear Model:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{4.9}$$

- Generalized Linear Model:

$$y(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + w_0) \tag{4.10}$$

  In which $g(\cdot)$ is called an ***activation function*** and may be nonlinear

4.3 Logistics Regression

Though named as "regression", logistics regression is used for classification problem, just like Linear Discriminant (Sec. 4.2).

Problem: In many of binary classification problems (classification problem with 2 classes), it's rather hard (or even impossible) to be certain about the class of the output, and the training data is not linear separable. Instead of a hard threshold, we could have a soft one, represent by the prob. belonging to either classes.

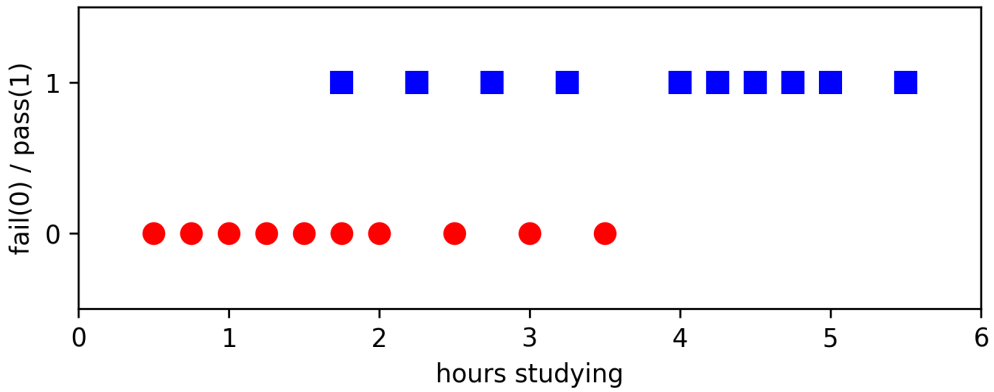


Figure 4.1: Example of exam results based on study hours (src). Given the number of hours, instead of predicting whether fail or pass, the model predicts the prob. that the student will pass, or fail.

The sigmoid function gives a nice nonlinear transition for the prob.. The further the data point is from the threshold, the higher the prob. it belongs to one class, and small to the others.

$$f(s) = \frac{1}{1+e^{-s}} \triangleq \sigma(s) \tag{4.11}$$

$$s = \ln\left(\frac{\sigma}{1-\sigma}\right) \tag{4.12}$$

$$\sigma'(s) = \sigma(s)\left(1-\sigma(s)\right) \tag{4.13}$$

Thus, if we use the sigmoid function as the activation function for Eq. 4.10:

$$s = \mathbf{w}^T\mathbf{x} \tag{4.14}$$

$$y = \sigma(s) \tag{4.15}$$

$$\Rightarrow \frac{\partial y}{\partial \mathbf{w}} = \frac{\partial y}{\partial s}\frac{\partial s}{\partial \mathbf{w}} = y(1-y)\mathbf{x} \tag{4.16}$$

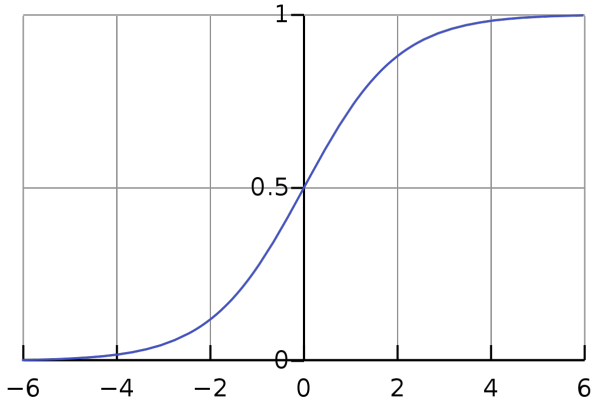


Figure 4.2: Sigmoid function (src).

Approach:

- Design of the error function.
  Assume that the prob. of data point $\mathbf{x}$ falls into class 1 is $f(\mathbf{w}^T\mathbf{x})$ and class 0 is $1-f(\mathbf{w}^T\mathbf{x})$.
  With $z_i = f(\mathbf{w}^T\mathbf{x}_i)$:

$$\begin{cases} P(y_i=1|\mathbf{x}_i;\mathbf{w}) &= f(\mathbf{w}^T\mathbf{x}_i) = z_i \\ P(y_i=0|\mathbf{x}_i;\mathbf{w}) &= 1-f(\mathbf{w}^T\mathbf{x}_i) = 1-z_i \end{cases} \Leftrightarrow P(y_i|\mathbf{x}_i;\mathbf{w}) = z_i^{y_i}(1-z_i)^{1-y_i} \tag{4.17}$$

With the aim to find the model params. that maximizes the data prob. (Subsec. 3.1.1):

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} P(\mathbf{y}|\mathbf{X};\ \mathbf{w})$$

$$\iff \mathbf{w}^* = -\arg\min_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{X};\ \mathbf{w}) \qquad \text{(negative log-likelihood)}$$

$$P(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_{i=1}^{N} P(y_i|\mathbf{x}_i;\mathbf{w}) = \prod_{i=1}^{N} z_i^{y_i}(1-z_i)^{1-y_i} \qquad \text{(independence assumption)}$$

$$\Rightarrow -\log P(\mathbf{y}|\mathbf{X};\ \mathbf{w}) = -\sum_{i=1}^{N}[y_i \log z_i + (1-y_1)\log(1-z_i)] \qquad \text{(the cross entropy error)}$$

- Optimization with Stochastic Gradient Descent (SGD):

$$\mathbf{w} = \mathbf{w} - \eta.\frac{\partial J}{\partial \mathbf{w}}, \quad (\eta \text{ as the learning rate}) \tag{4.18}$$

$$J(\mathbf{w}, \mathbf{x}_i, y) = -\left(y_i \log z_i + (1-y_i)\log(1-z_i)\right), \quad \text{data point } (\mathbf{x}_i, y_i) \tag{4.19}$$

$$\Rightarrow \frac{\partial J}{\partial \mathbf{w}} = -\left(\frac{y_i}{z_i} - \frac{1-y_i}{1-z_i}\right)\frac{\partial z_i}{\partial \mathbf{w}} = \frac{z_i - y_i}{z_i(1-z_i)}\frac{\partial z_i}{\partial \mathbf{w}} \tag{4.20}$$

$$\frac{\partial z_i}{\partial \mathbf{w}} = z_i(1-z_i)\mathbf{x}_i \qquad \text{(the beauty of sigmoid function, Eq. 4.16)} \tag{4.21}$$

$$\Rightarrow \frac{\partial J}{\partial \mathbf{w}} = (z_i - y_i)\mathbf{x}_i \tag{4.22}$$

$$\Rightarrow w = w + \eta(y_i - z_i)\mathbf{x}_i \qquad \text{(replace into Eq. 4.18)} \tag{4.23}$$

NOTE: Require less params., only D with D as the number of (no.) dimensions, compared to Gaussians with $\left[\frac{M(M+5)}{2} + 1\right]$ params..

4.4 Softmax Regression

Softmax Regression is the generalization of Logistics Regression for multiple-class classification problem. Thus, it is also known as: **Multinomial Logistics Regression, Maximum Entropy Classifier**. Logistics Regression can only be applied for binary classification problem. Given C classes, we would need multiple one-vs-all or one-vs-one classifiers. Softmax regression offers a better alternative.

$$z_i = \mathbf{w}^T\mathbf{x}_i \tag{4.24}$$

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C}\exp(z_j)} \tag{4.25}$$

$$\begin{cases} a_i > 0, & a_i \text{ represents the prob. the input belongs to class } C_i \\ \sum a_i = 1 & \\ z_m > z_n \iff a_m > a_n & \text{(order)} \end{cases} \tag{4.26}$$

There are cases that one output z_i is significantly greater than the others, which will lead to numerical error while coding. In these cases, the following Softmax version is more stable:

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C}\exp(z_j)} = \frac{\exp(z_i - c)}{\sum_{j=1}^{C}\exp(z_j - c)}, \quad c = \max_i z_i \tag{4.27}$$

For the one hot coding representation, for each input $\mathbf{x}$, the softmax regression model calculate a output vector $\mathbf{a} = \mathbf{W}^T\mathbf{x}$. The loss function will be build to represent the difference between output $\mathbf{a}$ and the real label $\mathbf{y}$. Vector $\mathbf{a}$ itself is a prob. distribution of the input belongs to different classes. Instead of choosing the squared error function, cross-entropy is a better alternative to represent the difference between two prob. distributions (Sec. **??**). As mentioned before, $\mathbf{q} > 0$, thus, we can't choose $\mathbf{q} = \mathbf{y}$.

$$J(\mathbf{W}, \mathbf{x}_i, \mathbf{y}_i) = -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(a_{ij}) \qquad \text{for each data point } (\mathbf{x}_i, \mathbf{y}_i) \tag{4.28}$$

$$\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}} = \mathbf{x}_i\mathbf{e}_i^T = \mathbf{x}_i(\mathbf{a}_i - \mathbf{y}_i)^T \qquad \text{(the gradient)} \tag{4.29}$$

$$\mathbf{W} = \mathbf{W} + \eta\mathbf{x}_i(\mathbf{y}_i - \mathbf{a}_i)^T \qquad \text{(SGD, machinelearningcoban.com)} \tag{4.30}$$

4.5 SVM

SVM

4.5.1 Hard Margin SVM

Problem: Give N data points $\mathbf{x}_i$ and their labels $y_i \in \{-1, 1\}$, find the line that separate these points into two classes while maximize the margin.

$$margin = \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n) + b}{||\mathbf{w}||_2} \tag{4.31}$$

$$\Rightarrow (\mathbf{w}, b) = \arg\max_{\mathbf{w},b}\left\{\min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n) + b}{||\mathbf{w}||_2}\right\} \tag{4.32}$$

$$\Rightarrow (\mathbf{w}, b) = \arg\max_{\mathbf{w},b}\left\{\frac{1}{||\mathbf{w}||_2}\min_n \left[y_n(\mathbf{w}^T\mathbf{x}_n + b)\right]\right\} \tag{4.33}$$

For simplicity, we can assume that: $\min_n \left[y_n(\mathbf{w}^T\mathbf{x}_n + b)\right] = 1$. Thus, the original problem can be transform to:

Problem: Find $(\mathbf{w}, b) = \arg\min_{\mathbf{w},b}\frac{1}{2}||\mathbf{w}||_2^2$ that subject to:

$$1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) \le 0, \quad \forall n = 1, 2, \ldots, N$$

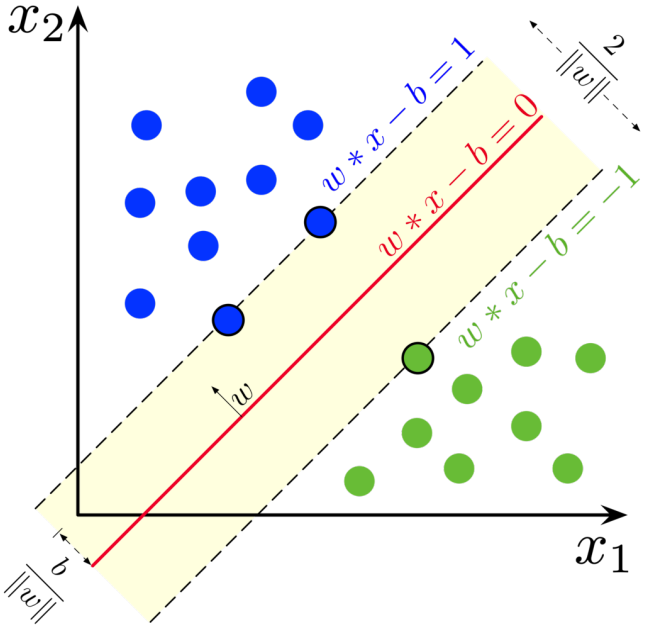


Figure 4.3: Example of SVM (src).

This is a well-studied optimization problem, solved with Lagrange's method.

Approach: The Lagrange's method

The primal formulation of SVM:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}||\mathbf{w}||_2^2 + \sum_{n=1}^{N} \lambda_n [1 - y_n(\mathbf{w}^T \mathbf{x}_n + b)], \quad \lambda_n \geq 0 \quad \forall n \tag{4.34}$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda})}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \lambda_n y_n \mathbf{x}_n \tag{4.35}$$

$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \boldsymbol{\lambda})}{\partial \mathbf{b}} = 0 \Rightarrow \sum_{n=1}^{N} \lambda_n y_n = 0 \tag{4.36}$$

The dual formulation of SVM:

$$\Rightarrow g(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \tag{4.37}$$

$$\text{Set } \mathbf{V} = [y_1 \mathbf{x}_1, y_2 \mathbf{x}_2, \ldots, y_N \mathbf{x}_N] \tag{4.38}$$

$$\Rightarrow g(\boldsymbol{\lambda}) = -\frac{1}{2} \boldsymbol{\lambda}^T \mathbf{V}^T \mathbf{V} \boldsymbol{\lambda} + 1^T \boldsymbol{\lambda} \qquad \text{is } \underline{\textbf{\textit{concave}}} \text{ with } \quad \lambda_i \geq 0, \quad \sum_{n=1}^{N} \lambda_n y_n = 0 \tag{4.39}$$

$\Rightarrow$ Find λ by solving Quadratic Programming

Then use Karush-Kuhn-Tucker (KKT) conditions to find $\mathbf{w}, b$:

$$\mathbf{S} = \{\mathbf{n} \mid \lambda_\mathbf{n} \neq \mathbf{0}\} \Rightarrow \begin{cases} b = \frac{1}{N_S} \sum_{n \in S} \left(y_n - \sum_{m \in S} \lambda_m y_m \mathbf{x}_m^T \mathbf{x}_n \right) \\ \mathbf{w} = \sum_{m \in S} \lambda_m y_m \mathbf{x}_m \end{cases}$$

4.5.2 Soft Margin SVM

- Hard margin SVM:

$$(\mathbf{w}, b) = \arg\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2^2$$
$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- Soft margin SVM:

$$(\mathbf{w}, b, \xi) = \arg\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{n=1}^{N}\xi_n$$
$$\text{subject to} \quad \begin{cases} y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \\ \xi_n \geq 0 \end{cases} \quad \forall n$$

C and the margins are in-proportional: $\begin{cases} C\uparrow \quad \Rightarrow \quad margin\downarrow \\ C\downarrow \quad \Rightarrow \quad margin\uparrow \end{cases}$

The *"Soft"* constraints:

$$y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \quad \Longleftrightarrow \quad 1 - \xi_n - y_n(\mathbf{w}^T\mathbf{x}_n + b) \leq 0 \quad \forall n$$

Problem: Find $(\mathbf{w}, b, \xi) = \arg\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{n=1}^{N}\xi_n$ that subject to:

$$\begin{cases} 1 - \xi_n - y_n(\mathbf{w}^T\mathbf{x}_n + b) \leq 0 \\ -\xi_n \leq 0 \end{cases} \quad \forall n$$

Approach: Lagrange's method with $\lambda_i \geq 0$ and $\mu_i \geq 0$:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\lambda_n[1 - \xi_n - y_n(\mathbf{w}^T\mathbf{x}_n + b)] - \sum_{n=1}^{N}\mu_n\xi_n \quad (4.40)$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = 0 \iff \mathbf{w} = \sum_{n=1}^{N}\lambda_n y_n \mathbf{x}_n \quad (4.41)$$

$$\frac{\partial\mathcal{L}}{\partial b} = 0 \iff \sum_{n=1}^{N}\lambda_n y_n = 0 \quad (4.42)$$

$$\frac{\partial\mathcal{L}}{\partial\xi_n} = 0 \iff \lambda_n = C - \mu_n \quad (4.43)$$

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\mathbf{w},b,\boldsymbol{\xi}} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \quad (4.44)$$

$$\Rightarrow g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{n=1}^{N}\lambda_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m y_n y_m \mathbf{x}_n^T\mathbf{x}_m = g(\boldsymbol{\lambda}) \quad (4.45)$$

$$\Rightarrow \boldsymbol{\lambda} = \arg\max_{\boldsymbol{\lambda}} g(\boldsymbol{\lambda}) \quad \text{subject to} \quad \begin{cases} \sum_{n=1}^{N}\lambda_n y_n = 0 \\ 0 \leq \lambda_n \leq C \quad \forall n \end{cases} \quad (4.46)$$

After finding $\boldsymbol{\lambda}$:

$$\mathcal{M} = \{n \mid 0 < \lambda_n < C\} \qquad \Rightarrow b = \frac{1}{N_M} \sum_{n \in \mathcal{M}} \left(y_n - \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x}_n \right) \tag{4.47}$$

$$\mathcal{S} = \{m \mid 0 < \lambda_m \leq C\} \qquad \Rightarrow \mathbf{w} = \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m \tag{4.48}$$

- $\begin{cases} \lambda_n = 0 \\ \xi_n = 0 \end{cases} \Rightarrow$ **Safe points**
- $\begin{cases} 0 < \lambda_n < C \\ \xi_n = 0 \end{cases} \Rightarrow$ **Marginal points**
- $\lambda_n = C \quad \begin{cases} \xi_n \leq 1 & \Rightarrow \textbf{still correct} \\ \xi_n > 1 & \Rightarrow \textbf{incorrect} \end{cases}$

4.5.3 Kernel Support Vector Machine

Mercer's Conditions: kernel function (func.) must theoretically satisfy.

In practice, however, some acceptable kernels don't satisfy the Mercer's conditions.

$$\sum_{n=1}^{N} \sum_{m=1}^{N} k(x_n, x_m) c_n c_m \geq 0 \quad \forall c_i \in \mathbb{R}, i = 1, 2, \ldots, N \tag{4.49}$$

E.g. kernel functions:

$k(x, z) = x^T z$ **linear**

$k(x, z) = (\gamma x^T z + r)^d$ **polynomial**

$k(x, z) = \exp(-\gamma ||x - z||_2^2), \quad \gamma > 0$ **RBF**

$k(x, z) = \tanh(\gamma x^T z + r)$ **sigmoid**

5 Ensembles

Ensemble of Models are also known as addictive models.

5.1 Error Reduction

[TODO: Explanation]

E.g.: bagging

$$y_{COM}(x) = \frac{1}{M}\sum_{m=1}^{M} y_m(x) \tag{5.1}$$

$$y(x) = h(x) + \varepsilon(x) \tag{5.2}$$

$$\mathbb{E}_x = [y_m(x) - h(x)]^2 = \mathbb{E}_x\left[\varepsilon_m(x)^2\right] \tag{5.3}$$

$$\Rightarrow \mathbb{E}_{AV} = \frac{1}{M}\sum_{m=1}^{M} \mathbb{E}_x\left[\varepsilon_m(x)^2\right] \tag{5.4}$$

$$\mathbb{E}_{COM} = \mathbb{E}_x\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(x) - h(x)\right\}^2\right] = \mathbb{E}_x\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \varepsilon_m(x)\right\}^2\right] \tag{5.5}$$

$$\Rightarrow \text{if } \begin{cases} \text{errors have 0 mean:} & \mathbb{E}_x[\varepsilon_m(x)] = 0 \\ \text{errors are uncorrelated:} & \mathbb{E}_x[\varepsilon_m(x)\varepsilon_j(x)] = 0 \text{ (unrealistic??)} \end{cases} \Rightarrow \mathbb{E}_{COM} = \tfrac{1}{M}\mathbb{E}_{AV}$$

However, in general, $\mathbb{E}_{COM} < \mathbb{E}_{AV}$

NOTE: The weak classifiers have to be unstable algorithms, i.e., decision trees, neural network; **NOT** nearest neighbor, SVM, linear regression.

5.2 Bagging

- also known as (a.k.a.) Bootstrap aggregating
- Average $\approx 63\%$

[TODO: Image]

- Split the given data set into train and test set
- Assume train set with n data points
- Pick n' data points into each bag D_i, $\quad \frac{n'}{n} < 1 (\approx 60\%)$

- Create m data bags: $D_1, D_2, \ldots, D_m$
- Learn a model M_i from each bag D_i
- The final output is the average of models' outputs:
$$y = \frac{1}{m} \sum_{i=1}^{m} f(M_i)$$
- **Random with replacement:** bag i can have multiple times data point $\mathbf{x}_i$
- **Simple, easy to implement, commonly used**

NOTE: Practically, resampling with replacement is ***usually unnecessary***, since SGD and random initialization usually makes the models sufficiently independent.

5.3 Boosting

- Adaboost
- Gradient boosting
- Xgboost (extreme gradient boosting)
- Gentle Boost (cross entropy error)

5.3.1 Adaboost

1. First weak classifier
2. Calculate error $J \Rightarrow \varepsilon$
   in case of normalization $w \Rightarrow \sum w = 1 \Rightarrow \varepsilon = J$
3. Calculate amount of say α from ε
4. Update w with α (normalize w)
5. 2nd weak classifier with new weight w or sample on new w

- Ensembles of Classifiers: K independent classifiers, error prob. < 0.5
- Suitable for unstable algorithms (i.e., decision trees, neural networks)
- Not good with stable methods (i.e., nearest neighbors, SVMs, linear regression)

[TODO: EXPLANATION?]

$$-g(x_i) = -\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)} \tag{5.6}$$

$$\Rightarrow \text{Regression model:} \quad h_i \text{ for } (x_i, -g(x_i)) \tag{5.7}$$

$$\Rightarrow F := F + \rho h, \quad \rho = 1 \tag{5.8}$$

[TODO: EXPLANATION?]

Problem:

- Given: N data points and their labels (x_n, t_n)
- Goal: find M weak classifiers $h_m(x)$

$$w_n^{(0)} = \frac{1}{N} \quad \text{initial data weights} \tag{5.9}$$

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I\left(h_m(x) \neq t_n\right) \quad \text{weighted error function} \tag{5.10}$$

$$\varepsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I\left(h_m(x) \neq t_n\right)}{\sum_{n=1}^{N} w_n^{(m)}} \quad \text{(normalized) weighted error} \tag{5.11}$$

$$\alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right) \quad \text{classifier weights} \tag{5.12}$$

$$w_n^{(m+1)} = w_n^{(m)}.\exp\left\{\alpha_m I\left(h_m(x) \neq t_n\right)\right\} \quad \text{updated data weights} \tag{5.13}$$

$$H(x) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right) \quad \text{final classification} \tag{5.14}$$

[TODO: EXPLANATION?] **Adaboost**

$$F(x) = sign\left(\sum_{m=1}^{M} \theta_m f_m(x)\right) \tag{5.15}$$

$$w(x_i, y_i) = \frac{1}{n} \quad \text{initial weights for each data} \tag{5.16}$$

$$\varepsilon_m = \mathbb{E}_{\varepsilon_m}\left[1_{y \neq f(x)}\right] \tag{5.17}$$

$$\Rightarrow \theta_m = \frac{1}{2}\ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right) \quad \text{Update rule} \tag{5.18}$$

$$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i)\exp[-\theta_m y_i f_m(x_i)]}{z_m} \quad \text{update weights} \tag{5.19}$$

(z_m is the normalization factor) (5.20)

[TODO: Explanation:] **Exponential error used in Adaboost**

- Fast convergence
- No penalty for too correct
- Less robust to outlier

[TODO: Add function graph]

5.3.2 Gradient Boosting (Regression)

$$(x_i, y_i) \text{ and first model } F_1(x) \tag{5.21}$$

$$\Rightarrow \text{fit } F_2(x) \text{ to } (x_i, y_i - F_1(x_i)), \quad y_i - F_1(x_i) = h(x_i) \text{ as the residuals} \tag{5.22}$$

$$J = \sum \mathcal{L}(y_i, F(x_i)) \tag{5.23}$$

$$\Rightarrow F_{i+1} = F_i + h_i \tag{5.24}$$

$$\mathcal{L} = \frac{(y_i - F_i)^2}{2} \Rightarrow -g = y_i - F_i(x_i) \tag{5.25}$$

$$\mathcal{L} = |y - F| \qquad \text{(absolute loss)} \tag{5.26}$$

$$\mathcal{L} = \begin{cases} \frac{1}{2}(y - F)^2 & \text{if } |y - F| \geq \delta \\ \delta\left(|y - F| - \frac{\delta}{2}\right) & \text{if } |y - F| < \delta \end{cases} \qquad \text{(Huber loss)} \tag{5.27}$$

The two loss functions Eq. 5.26 and Eq. 5.27 are less sensitive to outliers

5.4 Information

Mutual Information: YouTube, YouTube.

5.5 Decision Trees

Iterative Dichotomiser 3: MLCoBan Detail Example calculation: medium.com, blog

CART: Gini Index: blog

Random forest: YouTube

[**TODO: Add image**] root node, non-leaf node has 2 (or more) child node, leaf/terminal node. All non-leaf nodes have 2 child nodes ⇒ binary decision tree

Iterative Dichotomiser 3 (ID3) only for categorical attribute (discrete)

Classification and Regression Tree (CART) for both categorical and continuous.

6 questions:

- Bin / multi valued
- when node → leaf
- Deal impure nodes?
- how to select query
- pruned?

- missing attribute?

Impurity measures:

- Misclassification: $i(s_j) = 1 - \max_k p(C_k|s_j)$
- Information gain: C classes with N_C as the number of members in each class

$$H(S) = -\sum_{c=1}^{C} \frac{N_C}{N} \log\left(\frac{N_C}{N}\right) \qquad \text{entropy at a node} \tag{5.28}$$

Choose attribute $X \Rightarrow K$ child nodes: $S_1, S_2, \ldots, S_k$ with m_k elements

$$H(x, S) = \sum_{k=1}^{K} \frac{m_k}{N} H(S_k) \qquad \text{entropy sum with weights} \tag{5.29}$$

$$\Rightarrow G(x, S) = H(S) - H(x, S) \qquad \text{information gain} \tag{5.30}$$

$$x^* = \arg\max_x G(x, S) = \arg\min_x H(x, S) \tag{5.31}$$

Reduction in entropy = gain in information

- Gini Index:

$$i(s_j) = \sum_{k \neq l} p(C_k|s_j) p(C_l, s_j) \qquad \text{variance impurity at node } s_j \tag{5.32}$$

$$= \frac{1}{2}\left[1 - \sum_k p^2(C_k, s_j)\right] \tag{5.33}$$

$$H(x, s_j) = \sum_{k=1}^{K} \frac{m_k}{N} i(s_j) \tag{5.34}$$

$$\Rightarrow x^* = \min H(x, s_j) \tag{5.35}$$

- Chi-square: $= \sqrt{\frac{(actual - expected)^2}{expected}}$, $\arg\max$
  CHAID: Chi-square Automatic Interaction Detector
- Reduction in Variance, $\arg\min$

$$i(s_j) = \frac{\sum(x - \bar{x})^2}{n} \tag{5.36}$$

[TODO: Add image]

Stopping and Pruning is more IMPORTANT

5.5.1 Stopping Conditions

- Prevent over-fitting = **Prepruning**
- $H(\mathcal{S}) = 0$: entropy $= 0 \Rightarrow \forall$ points $\in$ each class
- no. members in each node $<$ a certain threshold
  The leaf node's class = the dominant class
- Distance from node $\rightarrow$ root $= c \Rightarrow$ to limit the tree depth

- The total no. of node exceed a threshold
- Further expansion induces insignificant entropy reduction

5.5.2 Pruning

Post-pruning

1. Validation Set: Prune a node if it increases the ***precision*** for VS (Validation Set). This is also called Reduction error pruning method
2. Regularized loss function

$$\mathcal{L} = \sum_{k=1}^{K} \frac{|\mathcal{S}_k|}{\mathcal{S}} H(\mathcal{S}_k) + \lambda K \tag{5.37}$$

   - Minimum error: sensitive to no. classes, the least accurate in practice
   - Pessimistic: the most crude and the quickest, no need for validation set, but extra caution needed
   - Error complexity: $R(t) = r(t)p(t) + \alpha N_t$, with $r(t)$ as the error, N_t as the no. leafs
   - Critical value: the value we choose to decide the query at each nodes
     $\Rightarrow$ depends on how the tree is created
   - Reduced error: ...

   NOTE: The last three error functions are more stable and accurate

5.5.3 Computational Complexity

Given:

$$\begin{cases} N \text{ data points} \\ D \text{ dimensions} \end{cases} \Rightarrow \begin{cases} \text{Storage: } \mathcal{O}(N) \\ \text{Test time: } \mathcal{O}(\log N) \\ \text{Training time: } \mathcal{O}(DN^2 \log N) \end{cases}$$

Explanation:

- Test time: in the worst case scenario, each leaf has one data
  $\Rightarrow$ After k step, the number of nodes are: $2^k = N$
  $\Rightarrow k \in \mathcal{O}(\log N)$
- Training time:
  maximum $\mathcal{O}(DN \log N)$ times each node
  maximum N nodes

5.5.4 Summary

- Simple
- Interpretable results
- Resistance to overfitting
- **Memory consumption ⇒ suitable for problems with little data**
- Noisy weak classifiers not generated well
- Sensitive to outliers ***??***
- Expensive learning step

5.6 Random Forest

- Handle missing values while maintaining accuracy
- Won't overfit
- Not good with regression
- Have little control to modify

Steps:

- Sample from training set
- Choose $m < M$ (input features). At each node, select m random data from M to decide query attribute to split the node
- Grow tree to the largest, no pruning
- Predict data output:
  - classification ⇒ majority vote
  - regression ⇒ average

Choose randomly K attributes:
Training time: $\mathcal{O}(KN^2 \log N)$, $K \ll D$ $(K = \sqrt{N_\delta})$
Typically: $\begin{cases} K = 10 & \text{root node} \\ K = 100d & \text{level-}d\text{ node} \end{cases}$

5.7 Bayesian Model Averaging

Given H different models with prior prob. $p(h)$, the final output would be the weighted average of these models:

$$p(X) = \sum_{h=1}^{H} p(X|h)p(h) \tag{5.38}$$

5.8 Distillation

Ensemble models is usually more robust than single models. However, the big question is:
Can we make a single model that is as good as an ensemble?

⇒ ***Distillation:*** train on the ensemble's predictions as "soft" targets [HVD+15]

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{5.39}$$

Intuition: more knowledge in soft targets than hard labels!

6 Feature Engineering

[**TODO: Add explanation**]
Dimensionality Reduction is an important technique in ML. Actual feature vectors can be in great dimension, in great number. Thus, filtering is crucial for storage, calculation. Dimension reduction is necessary and also useful in data compression.

[**TODO:**]

6.1 Principle Component Analysis

Principal Component Analysis (PCA) is a method for Feature Extraction

Learning Resources:

- setosa.io: for Visualization

6.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA)

blog

6.3 Word Representation

The idea of represent a word as a vector, but not as one-hot coding. With this representation

- similar words have similar vector values
- similar word's relationships have similar vector values

6.3.1 The trigram (n-gram method)

- Hugh amount of n-tuples of words ⇒ predict relative prob.
- Problems: scalability, observability

6.3.2 Word Embedding

$$\mathbf{x}_{V\times 1} \longrightarrow \mathbf{W}_{V\times d} \longleftrightarrow \mathbf{h}_{D\times 1} \tag{6.1}$$

1 of K encoding

6.3.3 word2vec

- CBOW: **syntactic (grammar)**
  Only care which word occurs
  Don't care about order of occurrence
- SKIP Gram **semantic (meaning)**
  less weight to more distance word

[**TODO: Add image**]

6.3.4 Hierarchical Softmax

Organizing words in binary search tree

7 Error Functions

As mention in Sec. 2.4, the loss function is the metric to assess the performance of the ML model with a certain task. Different loss functions fit with different tasks. In many cases, the design of the loss function is what makes the differences.

[**TODO: Add graph and explanation**]

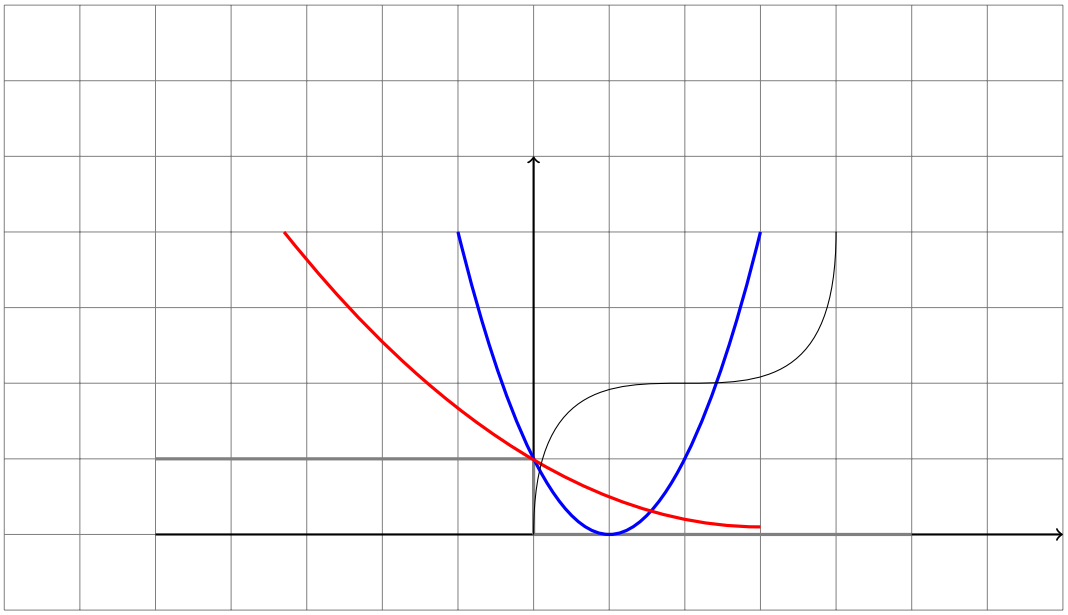


Figure 7.1: Different error functions: Ideal Miss-classification Error (gray line)

7.1 Ideal Miss-classification Error

Gradient = 0 $\Rightarrow$ can't use gradient descent.
It simply counts incorrectly classified points.

7.2 Squared Error - L_2 Loss

- Leads to closed form solutions
- Sensitive to outliers
- Penalize "too correct" data points

7.3 Cross Entropy Error

- Concave function ⇒ unique minimum exists
- Robust to outliers, error increases only roughly linear
- No closed-form solution, requires iterative method

7.4 Squared Error on Sigmoid / Tanh

- No penalty for "too correct" points
- Zero gradient for confidently incorrect classifications

⇒ **Do NOT** use L_2 loss with sigmoid outputs, instead, use cross-entropy.

7.5 Hinge Error

- Robust to outliers
- Zero error for points outside margin ⇒ sparsity
- Not differentiable around $z_n = 1$

NOTE: Want the correct class to have a score that is higher than incorrect class by a fixed margin Δ.

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \tag{7.1}$$

in which, s_j is other classes score, s_{y_i} is real class score.

7.6 L_1, L_0 Loss

Median, no wrong points

$$L_1 = \sum |t - y| \tag{7.2}$$

$$L_2 = \sum (t - y)^2 \tag{7.3}$$

7.7 Average Loss

Mathematically, dividing the loss by the amount of data N (in each batch or epoch) doesn't have any effect on the result. However, it's usually advisable to take the average to have more meaningful judgment and avoiding overflow when there are numerous data points.

8 Neural Network

Deep neural network takes care of the complex feature engineering process (Chap. 6). E.g., in classical computer vision, for people detection, the following process is applied:

1. Input: Image
2. Low-level feature extraction: Histogram of Oriented Gradients (HOG)
3. Mid-level features: Deformable Part Model (DPM)
4. Classifier: SVM
5. Output: final label

Using deep neural network, the process is truncated to simply:

1. Input: Image
2. Network training (end-to-end)
3. Output: final label

Different tasks require special expertise to design feature extraction, e.g., designing a program playing gammon would need someone knowing the rules, tips and tricks. On the other hands, we can assured that the human-proposed features are sufficient and helpful. Deep neural network alleviate the human-effort of hand-designing feature extraction process. In addition, it also learns prioritize important features for specific task.

8.1 General

Forward pass:

$$\mathbf{y}^{(0)} = \mathbf{x} \tag{8.1}$$

$$\mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{y}^{(k-1)}, \qquad k = 1, \ldots, l \tag{8.2}$$

$$\mathbf{y}^{(k)} = g_k(\mathbf{z}^{(k)}) \tag{8.3}$$

$$\mathbf{y} = \mathbf{y}^{(l)} \tag{8.4}$$

$$E = L(\mathbf{t}, \mathbf{y}) + \lambda\Omega(\mathbf{W}) \tag{8.5}$$

Backward pass:

$$h \leftarrow \frac{\partial E}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} L(\mathbf{t},\ \mathbf{y}) + \lambda \frac{\partial}{\partial \mathbf{y}} \Omega \tag{8.6}$$

$$\text{for } k = l \leftarrow 1: \tag{8.7}$$

$$h \leftarrow \frac{\partial E}{\partial \mathbf{z}^{(k)}} = h \odot g(\mathbf{y}^{(k)}) \tag{8.8}$$

$$\frac{\partial E}{\partial \mathbf{w}^{(k)}} = h\mathbf{y}^{(k-1)T} + \lambda \frac{\partial \Omega}{\partial \mathbf{w}^{(k)}} \tag{8.9}$$

$$h \leftarrow \frac{\partial E}{\partial \mathbf{y}^{(k-1)}} = \mathbf{W}^{(k)T} h \tag{8.10}$$

8.2 Gradient Descent

NOTE: Just use ADAM??

8.2.1 Vanilla Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t) \tag{8.11}$$

Check derivative:

$$f'(x) \approx \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon} \quad \text{(numerical gradient)} \tag{8.12}$$

8.2.2 Momentum

- Init: $v_{dW_0} = 0, v_{db_0} = 0$
- Calculate dW, db
- Update W, b

$$\Rightarrow \begin{cases} v_{dW} & = \beta v_{dW} + (1-\beta) dW \\ v_{db} & = \beta v_{db} + (1-\beta) db \end{cases} \Rightarrow \begin{cases} W & = W - \alpha v_{dW} \\ b & = b - \alpha v_{db} \end{cases} \tag{8.13}$$

  The above formulas are to calculate the moving average of v_{dW} and v_{db}.
- Tips: Choose $\beta_1 = 0.9$, implying taking average of the last 10 steps.
- Reference source: DeepLearning.AI.

8.2.3 Nesterov Accelerated Gradient

Nestorov Accelerated Gradient (NAG):

$$v_t = \gamma v_{t-1} + \eta \nabla_t J\left(\theta - \gamma v_{t-1}\right) \tag{8.14}$$

8.2.4 RMSprop

Root mean squared prop (RMSprop):

- Init $s_{dW_0} = 0, s_{db_0} = 0$
- Calculate dW, db
- Update W, b

$$\begin{cases} s_{dW} &= \beta s_{dW} + (1-\beta) dW^2 \\ s_{db} &= \beta s_{db} + (1-\beta) db^2 \end{cases} \Rightarrow \begin{cases} W &= W - \alpha \frac{dW}{\sqrt{s_{dW}} + \varepsilon} \\ b &= b - \alpha \frac{db}{\sqrt{s_{db}} + \varepsilon} \end{cases} \tag{8.15}$$

- Tips: choose $\beta_2 = 0.999, \quad \varepsilon = 10^{-7}$
- Reference source: DeepLearning.AI.

8.2.5 Adam

Adaptive moment estimation (Adam) is basically the combination of Momentum and RMSprop.

- Init $v_{dW_0}, s_{dW_0}, v_{db_0}, s_{db_0} = 0$
- Calculate dW, db
- Update W, b

$$\begin{cases} v_{dW} &= \beta_1 v_{dW} + (1-\beta_1) dW \\ v_{db} &= \beta_1 v_{db} + (1-\beta_1) db \\ s_{dW} &= \beta_2 s_{dW} + (1-\beta_2) dW^2 \\ s_{db} &= \beta_2 s_{db} + (1-\beta_2) db^2 \end{cases} \Rightarrow \begin{cases} v_{dW}^{cor.} &= \frac{v_{dW}}{1-\beta_1^t} \\ v_{db}^{cor.} &= \frac{v_{db}}{1-\beta_1^t} \\ s_{dW}^{cor.} &= \frac{s_{dW}}{1-\beta_2^t} \\ s_{db}^{cor.} &= \frac{s_{db}}{1-\beta_2^t} \end{cases} \Rightarrow \begin{cases} W &= W - \alpha \frac{v_{dW}^{cor.}}{\sqrt{s_{dW}^{cor.}} + \varepsilon} \\ b &= b - \alpha \frac{v_{db}^{cor.}}{\sqrt{s_{db}^{cor.}} + \varepsilon} \end{cases} \tag{8.16}$$

- Tips: choose $\beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \varepsilon = 10^{-7}$
- Reference source: DeepLearning.AI.

8.3 Normalization

8.3.1 BatchNorm

Batch Normalization (BatchNorm) normalizes the mean and standard deviation for each individual feature channel [IS15]. Given an input batch x from batch size N, no. channel C, height H and width W:

$$BN(x) = \gamma\left(\frac{x-\mu(x)}{\sigma(x)}\right) + \beta, \quad \begin{array}{l}\text{input } x \in \mathbb{R}^{N\times C\times H\times W}\\ \text{mean and standard deviation } \mu(x), \sigma(x) \in \mathbb{R}^C\\ \text{affine params. to learn } \gamma, \beta \in \mathbb{R}^C\end{array} \tag{8.17}$$

$$\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw} \tag{8.18}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_n hcw - \mu_c(x))^2 + \epsilon} \tag{8.19}$$

Intuition:

- γ and β are basically the learned standard deviation and mean of the output feature map.
- BatchNorm layer simply transforms the output to a different mean and standard deviation.
- Why BatchNorm helps?:
  - Check these videos: CodeEmporium, DeepLearningAI
  - Speeds up training speed: by transforming the parameter space to be more even, instead of being overwhelmed by some parts of params.. This is basically the same as using Adam instead of conventional SGD
  - Allows suboptimal starts: same reason as above.
  - Regularizes the model (slightly)

8.3.2 IN

Instance Normalization (IN) [ULV+16] show significant improvement over BatchNorm, especially for the style transfer problem in CV:

- BatchNorm computes mean and deviation for each channel: $\mu(x), \sigma(x) \in \mathbb{R}^C$ (Eq. 8.19)
- IN computes mean and standard deviation independently for each channel and each sample: $\mu(x), \sigma(x) \in \mathbb{R}^{N\times C}$

$$IN(x) = \gamma\left(\frac{x-\mu(x)}{\sigma(x)}\right) + \beta \tag{8.20}$$

$$\mu_n c(x) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw} \tag{8.21}$$

$$\sigma_n c(x) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_n hcw - \mu_n c(x))^2 + \epsilon} \tag{8.22}$$

- At test time, IN layers are applied unchanged, while BatchNorm layers usually replace mini-batch statistics with population statistics.

- *Conditional instance normalization* (CIN) is an extension of IN [DSK16]

$$CIN(x;s) = \gamma^s \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta^s \tag{8.23}$$

- It's interesting that the affine parameters γ, β can completely change the style of the output image.

8.3.3 AdaIN

Adaptive Instance Normalization (AdaIN) simply aligns the mean and variance of content input x to match those of style input y. Here, the affine parameters are no longer learnable, but adaptively computed from the style input [HB17]

$$AdaIN(x,y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y) \tag{8.24}$$

NOTE: Using concatenation alone would not disentangle the content and style information for later learning step.

8.4 Convolutional Operator

8.4.1 Convolution

[LBB+98]

8.4.2 Transposed Convolution

This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations here [DV16] and the Deconvolutional Networks paper [ZKT+10].

8.5 Tips and Tricks

- Shuffling
- Data Augmentation: reshape, rescale, crops, zooming, change color (color PCA)
- Normalizing the inputs
  Convergence is the fastest if
  - The mean of each input variable = 0
  - Scale ⇒ same covariance

Mean cancellation ⇒ Kullback–Leibler (KL) expansion ⇒ covariance equalization (if possible)

- Leaky Rectified Linear Unit (ReLU) is better a bit than ReLU, ELU
- Weights initialization: Xavier-Glorot:

$$W \sim U\left(0, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$$

- Batch Norm(alization): Normalize after each layer
  ⇒ learn the moving average
- Drop out
  NOTE: When in inferencing (after training), must multiply the activation output with the prob. that the weights are set to 0

9 Network Structures

This chapter presents some common network structures and their capabilities.

9.1 Base Structures

9.1.1 LeNet

LeNet5 is probably the earliest Convolutional Neural Network (CNN) network, proposed by LeCun et al. (1998) [LBB+98].

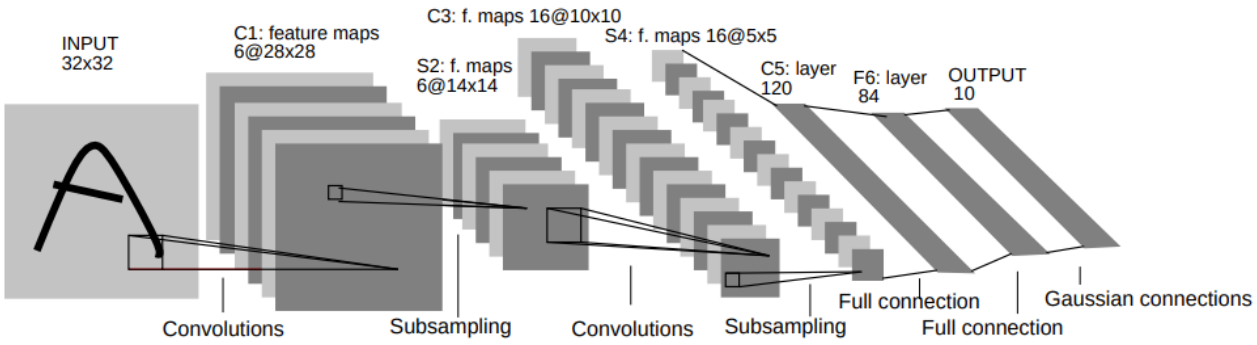


Figure 9.1: LeNet5 Architecture. [LBB+98]

Example coding with `pytorch` (src):

```
import torch.nn as nn

class LeNet5(nn.Module):
  def __init__(self, num_classes):
    super(ConvNeuralNet, self).__init__()
    self.conv1 = nn.Sequential(
    nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
    nn.BatchNorm2d(6),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size = 2, stride = 2))
    self.conv2 = nn.Sequential(
    nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size = 2, stride = 2))
    self.fc1 = nn.Sequential(
    nn.Linear(400, 120),
```

```
    nn.ReLU())
    self.fc2 = nn.Sequential(
    nn.Linear(120, 84),
    nn.ReLU())
    self.fc3 = nn.Linear(84, num_classes)

  def forward(self, x):
    ...
    return y
```

9.1.2 AlexNet

The classic CNN architecture for image classification on the CIFAR10 dataset [KSH12].

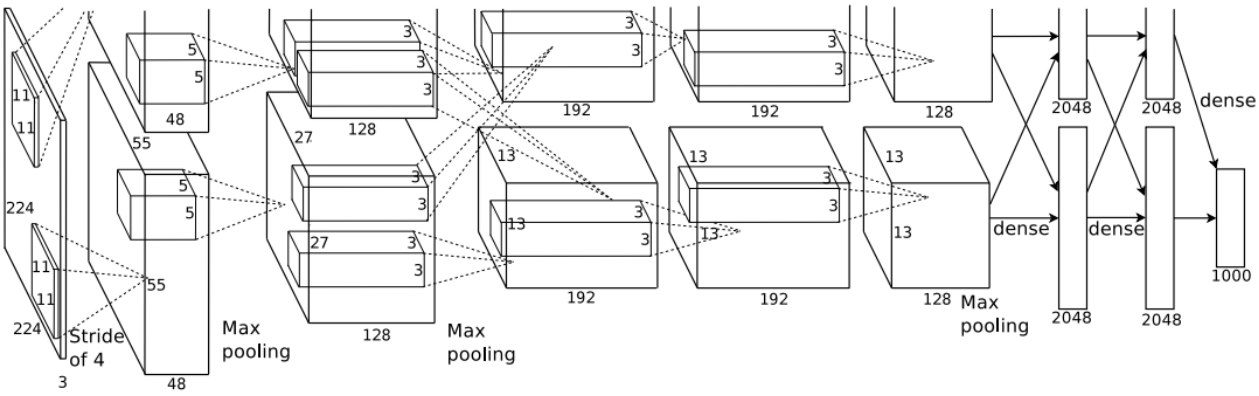


Figure 9.2: AlexNet architecture. [KSH12]

Example coding with `pytorch` (src):

```
class AlexNet(nn.Module):
  def __init__(self, num_classes=10):
    super(AlexNet, self).__init__()
    self.layer1 = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
    nn.BatchNorm2d(96),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size = 3, stride = 2))
    ...
    self.fc1 = nn.Sequential(
    nn.Dropout(0.5),
    nn.Linear(4096, 4096),
    nn.ReLU())
    self.fc2= nn.Sequential(
    nn.Linear(4096, num_classes))

```

```
  def forward(self, x):
    ...
    return y
```

9.1.3 VGG Net

The runner-up at the ILSVRC 2014 competition by Simonyan and Zisserman (2014) [SZ14]:

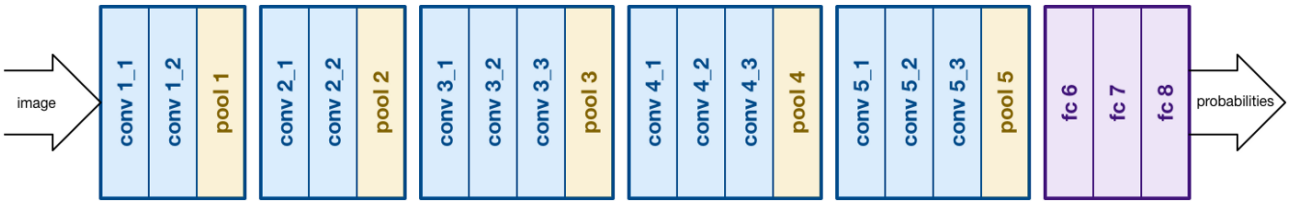


Figure 9.3: The VGG Net architecture with 13 Convolutional (CONV) layers and 3 Fully Connected (FC) layers. [SZ14]

Example coding with `pytorch` (src):

```
  class VGG16(nn.Module):
  def __init__(self, num_classes=10):
  super(VGG16, self).__init__()
  self.layer1 = nn.Sequential(
  nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
  nn.BatchNorm2d(64),
  nn.ReLU())
  ...
  self.fc1 = nn.Sequential(
  nn.Dropout(0.5),
  nn.Linear(4096, 4096),
  nn.ReLU())
  self.fc2= nn.Sequential(
  nn.Linear(4096, num_classes))

  def forward(self, x):
  ...
  return y
```

9.1.4 Residual Network

The Residual Network (ResNet) by He et al. (2016) [HZR+16] tackles the vanishing gradient problem for deep neural network with the residual block (Fig. 9.4). It creates a highway pass for the gradient to go to the early layers.

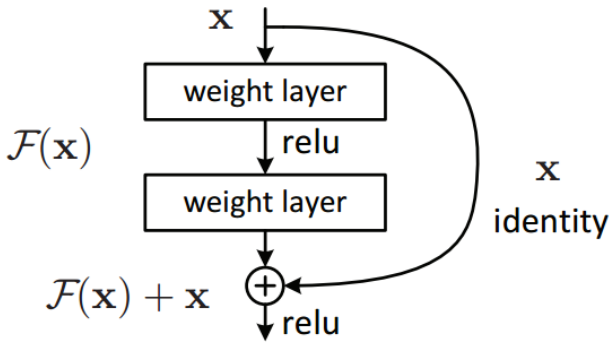


Figure 9.4: Residual block [HZR+16].

Example coding with `pytorch` (src):

```
class ResidualBlock(nn.Module):
  def __init__(self, in_channels, out_channels,
      stride = 1, downsample = None):
    super(ResidualBlock, self).__init__()
    self.conv1 = nn.Sequential(
    nn.Conv2d(in_channels, out_channels, kernel_size = 3,
    stride = stride, padding = 1),
    nn.BatchNorm2d(out_channels),
    nn.ReLU())
    self.conv2 = nn.Sequential(
    nn.Conv2d(out_channels, out_channels, kernel_size = 3,
    stride = 1, padding = 1),
    nn.BatchNorm2d(out_channels))
    self.downsample = downsample
    self.relu = nn.ReLU()
    self.out_channels = out_channels

  def forward(self, x):
    residual = x
    out = self.conv1(x)
    out = self.conv2(out)
    if self.downsample:
      residual = self.downsample(x)
    out += residual
    out = self.relu(out)
    return out
```

```
class ResNet(nn.Module):
  def __init__(self, block, layers, num_classes = 10):
    super(ResNet, self).__init__()
    self.inplanes = 64
    self.conv1 = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size = 7, stride = 2, padding = 3),
```

```
    nn.BatchNorm2d(64),
    nn.ReLU())
    self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1)
    self.layer0 = self._make_layer(block, 64, layers[0], stride = 1)
    self.layer1 = self._make_layer(block, 128, layers[1], stride = 2)
    self.layer2 = self._make_layer(block, 256, layers[2], stride = 2)
    self.layer3 = self._make_layer(block, 512, layers[3], stride = 2)
    self.avgpool = nn.AvgPool2d(7, stride=1)
    self.fc = nn.Linear(512, num_classes)

  def _make_layer(self, block, planes, blocks, stride=1):
    downsample = None
    if stride != 1 or self.inplanes != planes:
      downsample = nn.Sequential(
        nn.Conv2d(self.inplanes, planes, kernel_size=1, stride=stride),
        nn.BatchNorm2d(planes),
        )
    layers = []
    layers.append(block(self.inplanes, planes, stride, downsample))
    self.inplanes = planes
    for i in range(1, blocks):
      layers.append(block(self.inplanes, planes))
    return nn.Sequential(*layers)

  def forward(self, x):
    x = self.conv1(x)
    x = self.maxpool(x)
    x = self.layer0(x)
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.avgpool(x)
    x = x.view(x.size(0), -1)
    x = self.fc(x)
    return x

model = ResNet(ResidualBlock, [3, 4, 6, 3]).to(device)
```

9.1.5 GoogLeNet

[**TODO:**]

9.1.6 U-Net

The U-Net architecture by Ronneberger, Fischer, and Brox (2015) [RFB15] can also be viewed as an encoder-decoder architecture [KW13] with skip connections of `ResNet` (Fig. 9.6). This structure is later adopted in `pix2pix` architecture for image-to-image translation [IZZ+17].

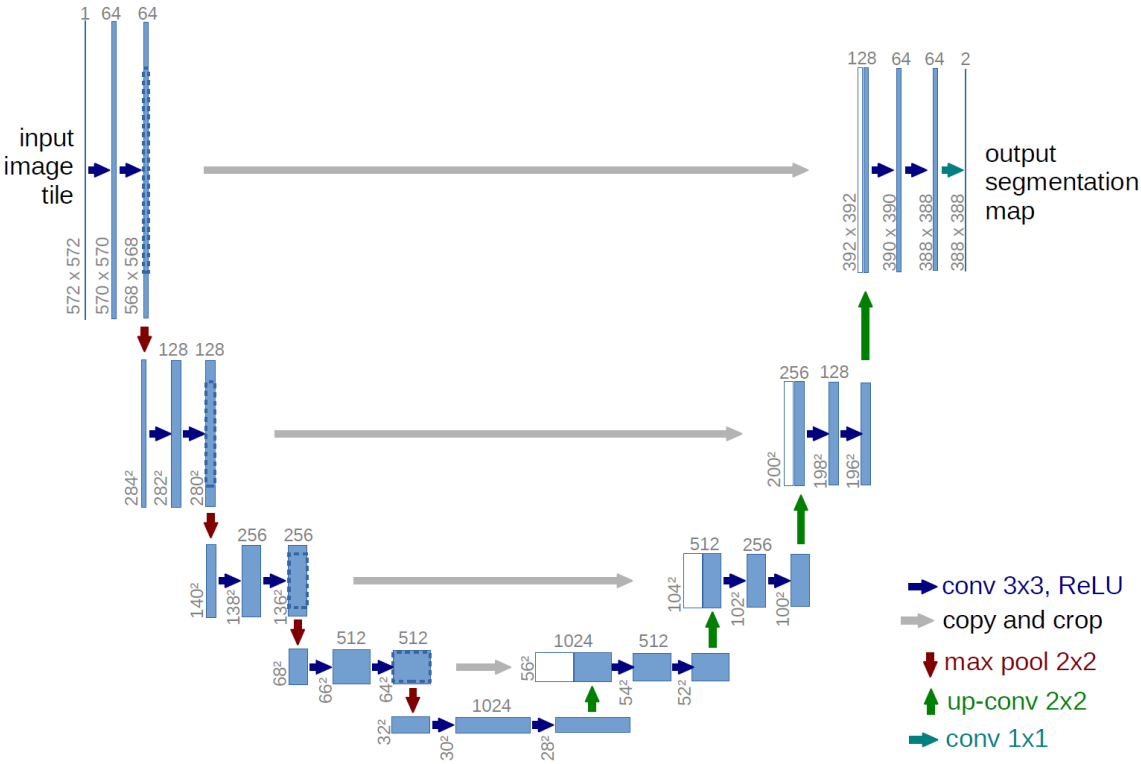


Figure 9.5: The U-Net architecture. [RFB15]

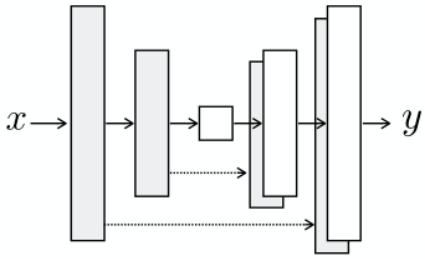


Figure 9.6: The U-Net as a variational auto-encoder with skip connections. [IZZ+17]

[TODO: benefits, comments?]

9.1.7 DenseNet

[TODO:] Huang et al. (2017) [HLVDM+17]. *"Densely connected convolutional networks"*.

9.1.8 Comparison between Networks

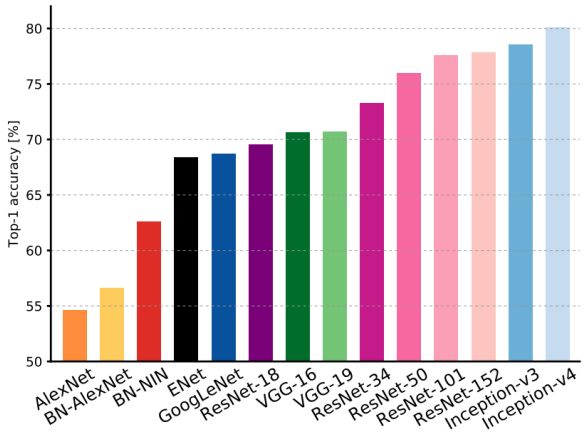


Figure 9.7: Top1 versus (vs.) network.

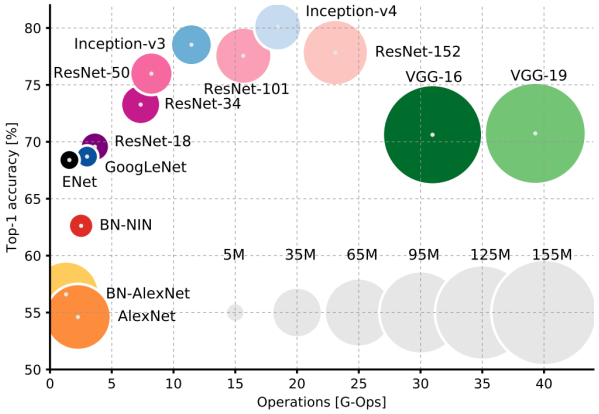


Figure 9.8: Top1 vs. operations, size $\propto$ params..

9.2 Structures for Sequential Data

9.2.1 Recurrent Neural Network

The next three models deal with sequential data. It's a bit uncertain what is the original paper proposing the idea though [Elm90]. Backpropagation through time (BPTT):

[TODO: Add image, content]

$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial w_{ij}} \right) \tag{9.1}$$

$$E = \sum_{1 \leq t \leq T} E_t \tag{9.2}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \geq i \geq k} \frac{\partial h_i}{\partial h_{i-1}} \tag{9.3}$$

9.2.2 LSTM

[SVL14] **[TODO:]**

9.2.3 Transformer

Recurrent Neural Network (RNN) suffers from vanishing/exploding gradient. Long short-term memory (LSTM) suffers from slow training time. Vaswani et al. (2017) [VSP+17] propose a approach based on the idea of attention. This approach can utilize the computation power of

Graphics Processing Unit (GPU) for parallelized matrix computation. This technique originally is meant for NLP problem, but later on, also show meaningful application in CV.

- A easy and detailed explanation from Alammar (2020) [Ala20] This idea is the basis for GPT-3 [BMR+20] and BERT [DCL+18]

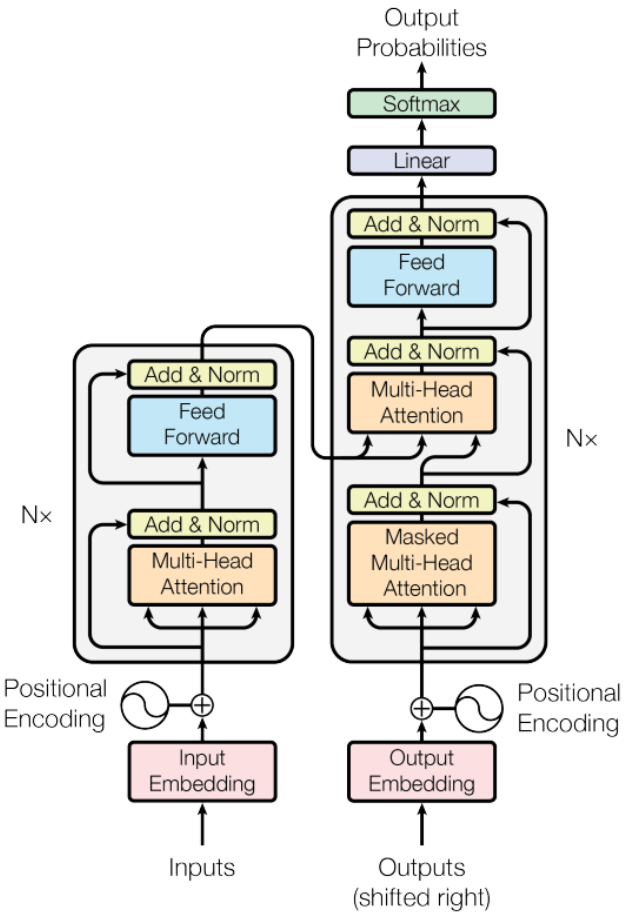


Figure 9.9: The Transformer - model architecture [VSP+17].

9.3 Bayesian Neural Network

Deep neural network's strength is its ability to approximate function. However, as powerful as it is, there is also a threat to overfit to the training data and fail to generalize on the test set. There are various techniques to reduce overfit and improve transfer learning, e.g., weight regularization, dropout, batch norm.

A conventional neural network takes in the same input and produce the same output. A Bayesian neural network different output when it takes in the same inputs twice. The resulting algorithm:

- mitigates overfitting

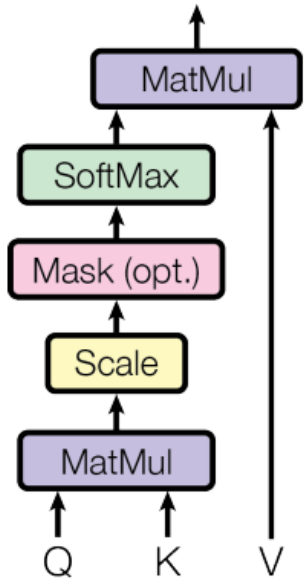


Figure 9.10: Scaled Dot-Product Attention.

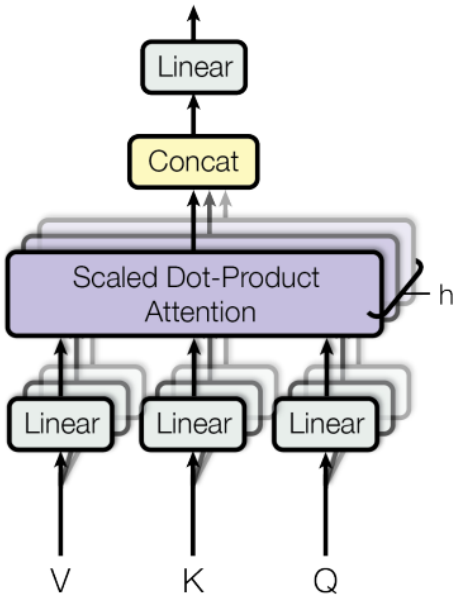


Figure 9.11: Multi-Head Attention.

- enables learning from small datasets
- tells us how uncertain our predictions are.
- adds stochasticity to neural network.
- Instead of weights, it has weights with distribution

$$w_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \tag{9.4}$$

NOTE: There is a thing called Bayesian Network, a.k.a. belief network, which is not the same as Bayesian neural network.

$$\mathcal{D}_{tr} = \{\mathbf{x}_i, y_i\}_{i=1}^n, \quad \text{model } F_\theta, \quad \text{loss } \mathcal{L}$$

Neural Network	Bayesian Neural Network
Training: $\theta^* = \arg\max_\theta \sum_{(\mathbf{x}_i, y_i)} \log[p(y_i \vert \mathbf{x}_i, \theta)]$ $\theta^* = \arg\min_\theta \sum_{(\mathbf{x}_i, y_i)} \mathcal{L}(F_\theta(\mathbf{x}_i), y_i)$	Training: $\mu^*, \Sigma^* = \arg\max_{\mu,\Sigma} \sum_{(\mathbf{x}_i, y_i)} \log[p(y_i \vert \mathbf{x}_i, \theta)] - KL[p(\theta), p(\theta_0)]$ $\theta \sim \mathcal{N}(\mu, \Sigma), \quad \theta_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ $\mu^*, \Sigma^* = \arg\min_{\mu,\Sigma} \sum_{(\mathbf{x}_i, y_i)} \mathcal{L}(F_\theta(\mathbf{x}_i), y_i) + KL[p(\theta), p(\theta_0)]$
Prediction: $p(\hat{y} \vert \hat{\mathbf{x}}, \theta^*)$ $\hat{y} = F_{\theta^*}(\hat{\mathbf{x}})$	Prediction: $p(\hat{y} \vert \hat{\mathbf{x}}, \mathcal{D}_{tr}) = \int p(\hat{y} \vert \hat{\mathbf{x}}, \theta^*) p(\theta^* \vert \mathcal{D}_{tr}) d\theta^*$ $\theta^* \sim \mathcal{N}(\mu^*, \Sigma^*)$ $\hat{y} = \frac{1}{K} \sum_{k=1}^{K} F_{\theta_k^*}(\hat{\mathbf{x}}), \quad \theta_k^* \sim \mathcal{N}(\mu^*, \Sigma^*)$

9.3.1 References

Examples:

- cs.toronto
- Keras's example
- BNN-pytorch
- PyTorch-BayesianCNN

If you want to just get the high level idea, watch these:

- YouTube/PyData

References: **NOTE:** very long and complicate papers, I don't read them

- Jospin et al. (2022) [JLB+22]. *"Hands-on Bayesian neural networks—A tutorial for deep learning users"*.
- Shridhar et al. (2019) [SLL19]. *"A comprehensive guide to bayesian convolutional neural network with variational inference"*.
- Liu et al. (2018) [LLW+18]. *"Adv-bnn: Improved adversarial defense through robust bayesian neural network"*.

9.4 Graph Neural Networks

9.4.1 Introduction

Many systems and interactions can be represented as graphs. Graph Neural Network (GNN) are neural networks that operate on graph-structured data. Examples:

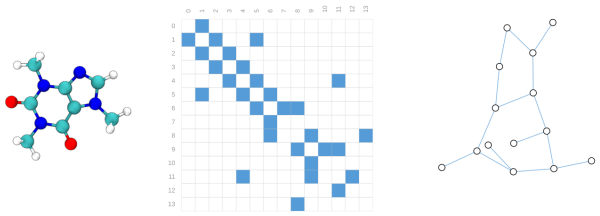


Figure 9.12: Caffeine molecule structure as a graph. [SLRP+21]

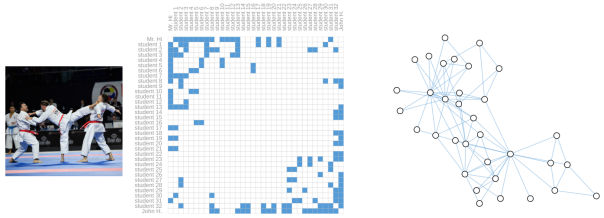


Figure 9.13: A social network as a graph. [SLRP+21]

Image and text can also be modeled as graphs, but it would be redundant, since they have fixed structures. Pixels in image are connected in a grid and words in a sentence are connected only with the prior and next ones.

There are three general types of prediction tasks on graphs: [SLRP+21]

- graph-level: predict a single property for a whole graph. This is analogous to classification / regression problem on MNIST or CIFAR datasets.
  E.g.: Given a graph representing a molecule, predict the smell of it.
- node-level: predict some property for each node in a graph. This is analogous to segmentation problem of CV, in which we predict the class for each pixel.
  E.g.: Given a graph representing the social network, predict the famous score of a person (a node)
- edge-level: predict the property or presence of edges in a graph.
  E.g.: Given a image of different peoples, objects, predict the interactions between them (Fig. 9.14)

There are also

- Node clustering task
- Link Prediction: Predicting missing links.
- Influence Maximization: Identifying influential nodes.

9.4.2 Challenges

NOTE: Batch learning is difficult the setup

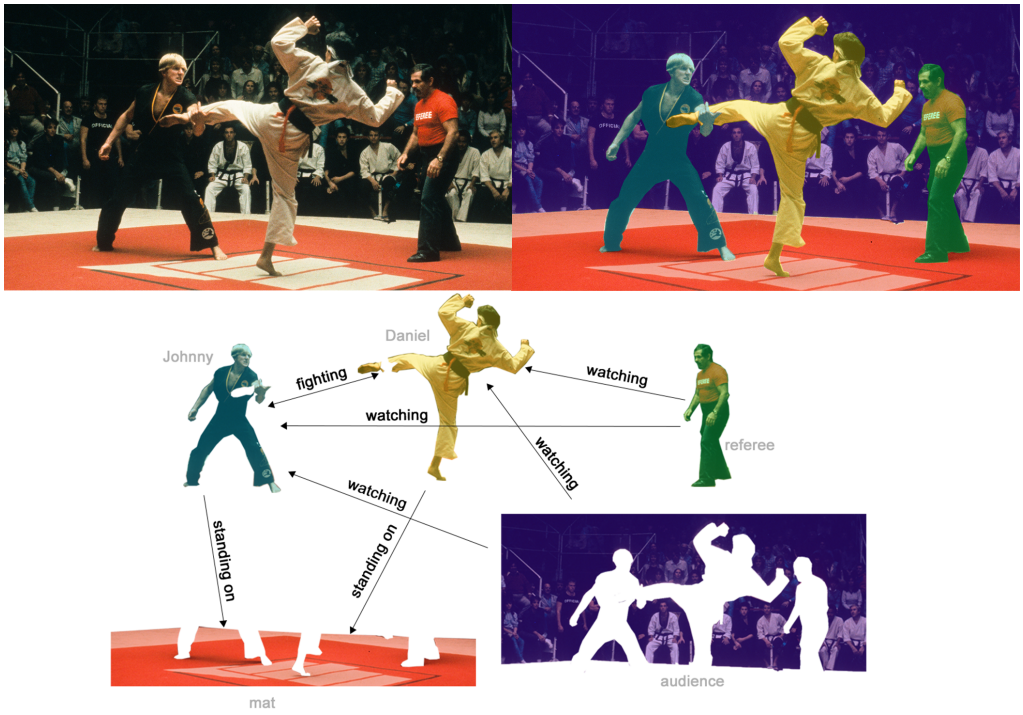


Figure 9.14: Example of edge-level prediction task [SLRP+21].

9.4.3 Graph Representations in Machine Learning

- Using adjacency matrix is not permutation-invariant. (Fig. 9.15)

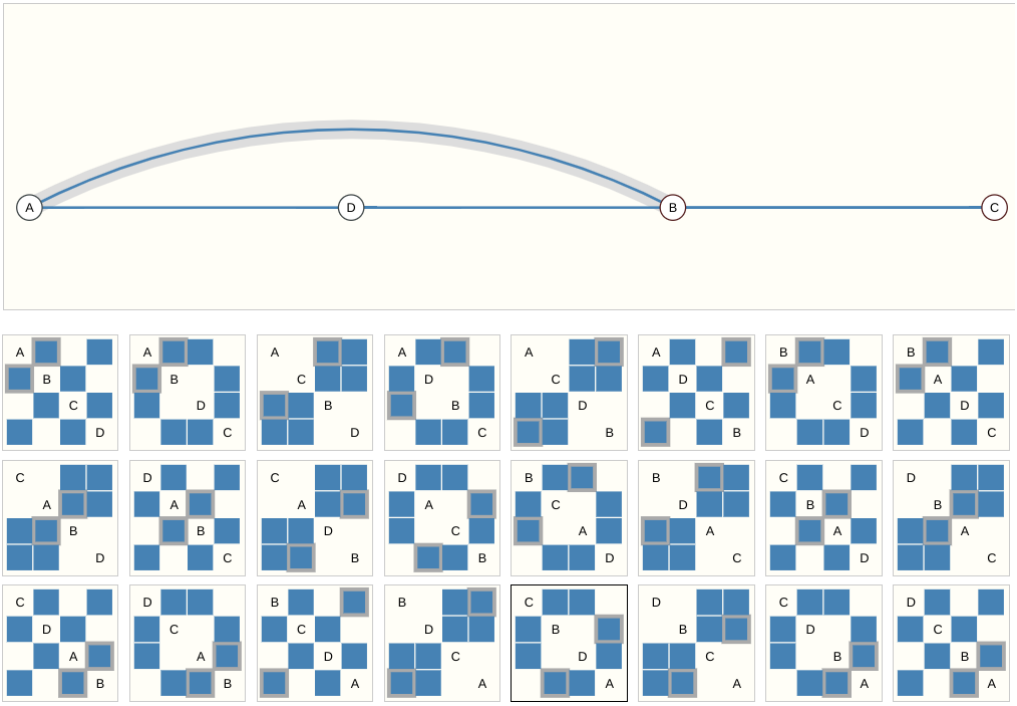


Figure 9.15: Different adjacency matrices represent the same graph [SLRP+21].

- Another option is adjacency list

  - Nodes: list of feature vectors for each node, tensor size $[n_{nodes}, dim_{node}]$
  - Edges: list of feature vectors for each edge, tensor size $[n_{edges}, dim_{edge}]$
  - Adjacency list: list of edges, tensor size $[n_{edges}, 2]$
  - Global: a single feature vectors for global value, tensor size $[1, dim_{global}]$
- Feature vector is also called as representation or embedding E.g.: feature vector for a person node in a social network: $\begin{bmatrix} age \\ job \\ marriage\ status \\ nationality \\ location \end{bmatrix}$

9.4.4 GNN Block

Between layers in GNN, there are 3 Multi-Layer Perceptron (MLP) to update graph attributes: feature vectors of nodes, edges and global value. **NOTE:** The adjacency list stays unchanged.

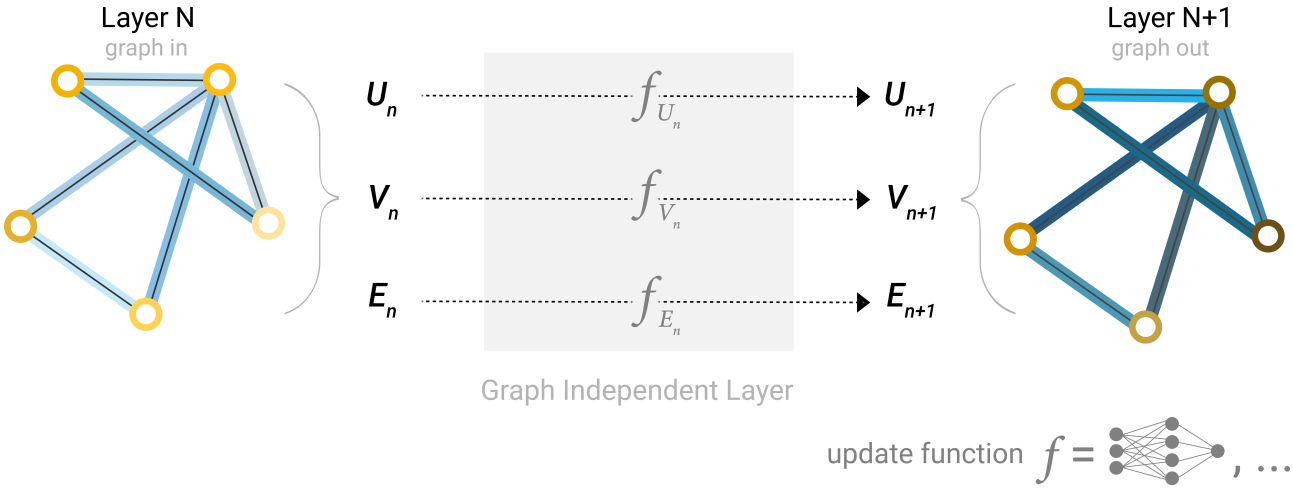


Figure 9.16: A simple GNN block. At layer N, the current graph (V_n, E_n, U_n) is updated to $(V_{n+1}, E_{n+1}, U_{n+1})$, in which V_n is the list of feature vectors for nodes, E_n is the list of feature vectors for edges, and U_n is global value. The three MLP are f_{U_n}, f_{V_n}, f_{E_n} correspond to each mentioned graph attributes.

We of course can have multiple of these blocks, similarly to have multiple hidden layers in a MLP.

9.4.5 GNN Predictions

For specific prediction task, an additional MLP operates with corresponding graph attribute.

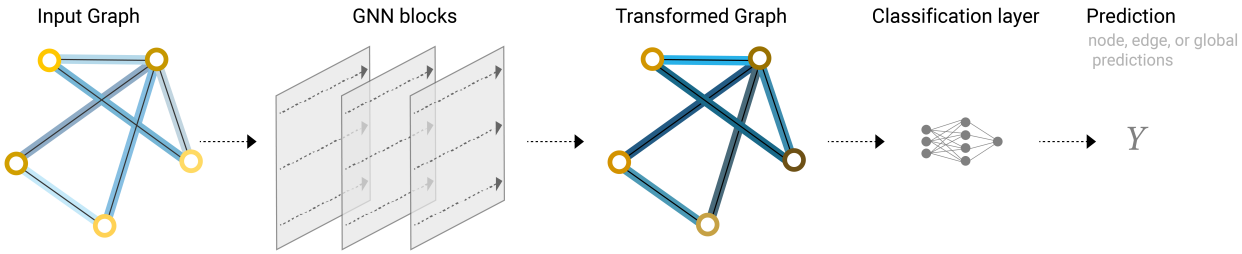


Figure 9.17: Multiple GNN blocks before prediction network.

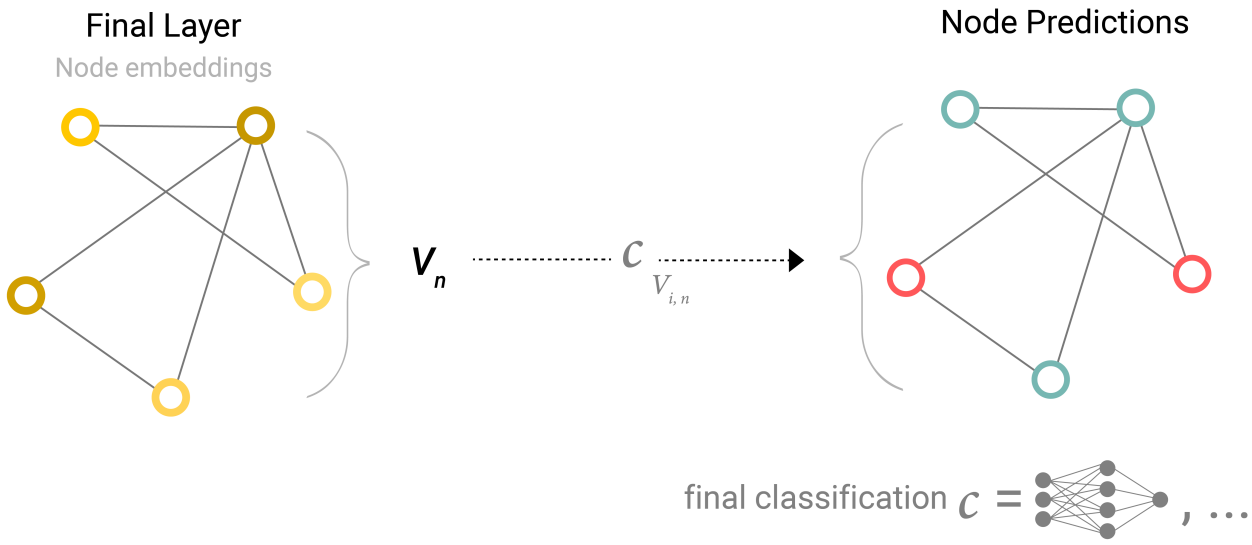


Figure 9.18: Example of node-level prediction task: A MLP $C_{V_{i,n}}$ produces prediction score on each node.

9.4.6 Pooling Information

Information may not be sufficient and complete in graph

- Having only edge-level features, need to make node-level predictions.
- Having only node-level features, need to make edge-level predictions.

In these cases, the information can be transfer via pooling operation:

1. Embeddings are gathered and concatenate into a matrix.
2. The output embedding are then aggregated from the matrix, usually via a sum operation (Fig. 9.19).

9.4.7 Message Passing

In GNN, while pooling implies transferring information from one graph attribute to another (e.g., edge to node), message passing implies exchanging information and influence within the

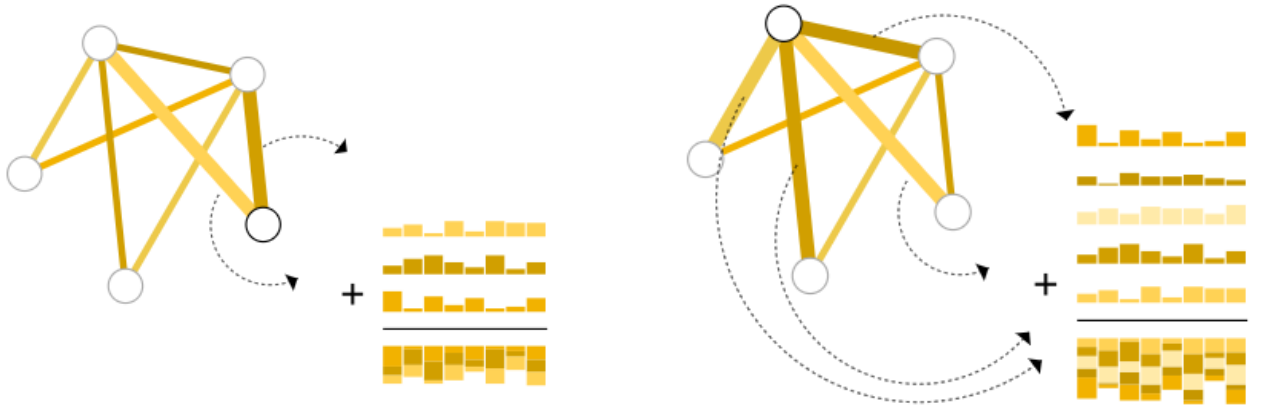


Figure 9.19: Example of pooling operation via sum. The feature embeddings of neighboring edges are aggregated to the node embedding, via a sum operation.

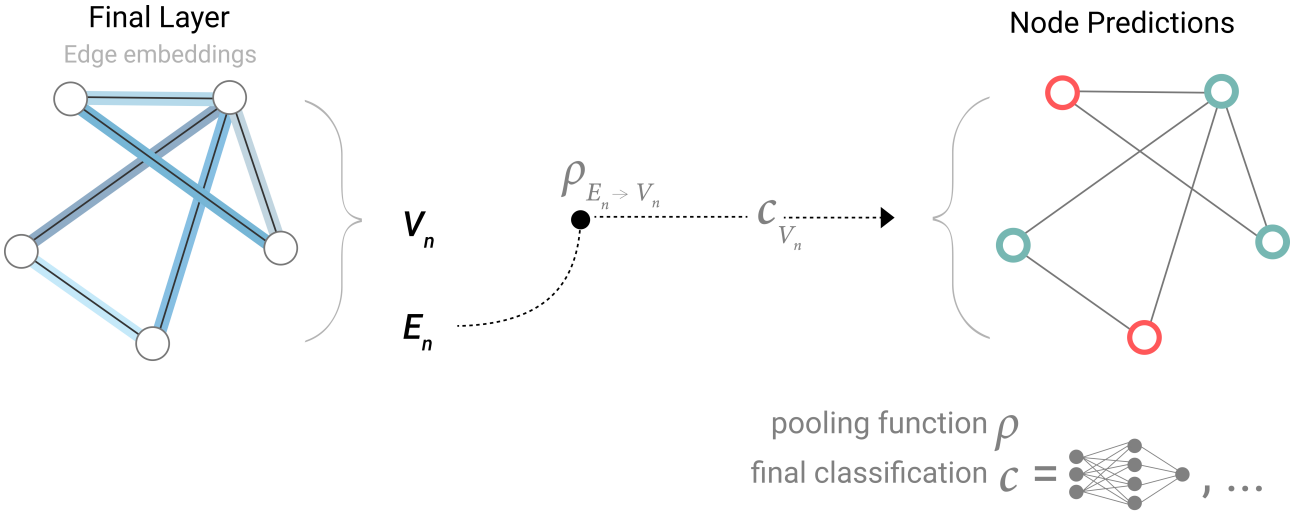


Figure 9.20: Pooling operation from edge to node $\rho_{E_n \to V_n}$ before applying node-level prediction C_{V_n}.

same graph attribute (e.g., node to node). The procedure, however, is similar to pooling.

Combining different embeddings from different graph attributes can all be considered as a conditioning function:

9.4.8 Some Empirical Design Lessons

Source: Sanchez-Lengeling et al. (2021) [SLRP+21].A Gentle Introduction to Graph Neural Networks.

- A higher number of parameters does correlate with higher performance.
- GNNs are a very parameter-efficient model type: for even a small number of parameters
- Models with higher dimensionality or more layers tend to have better mean and lower bound performance, but the same trend is not found for the maximum.

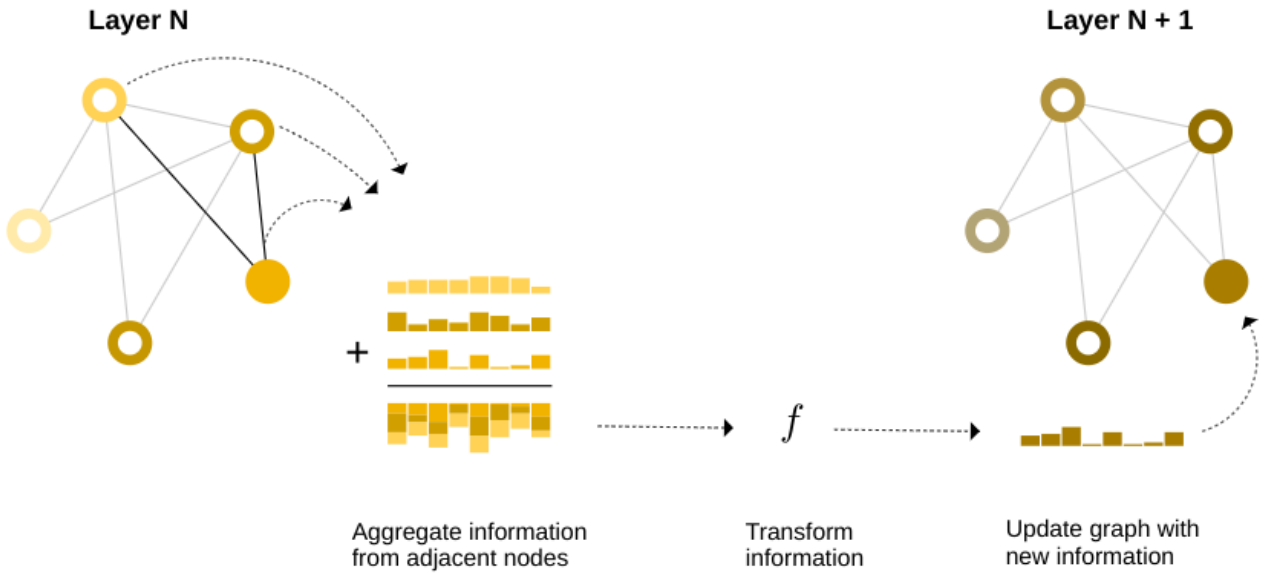


Figure 9.21: Message passing in graph: $\rho_{V_n \to V_{n+1}}$

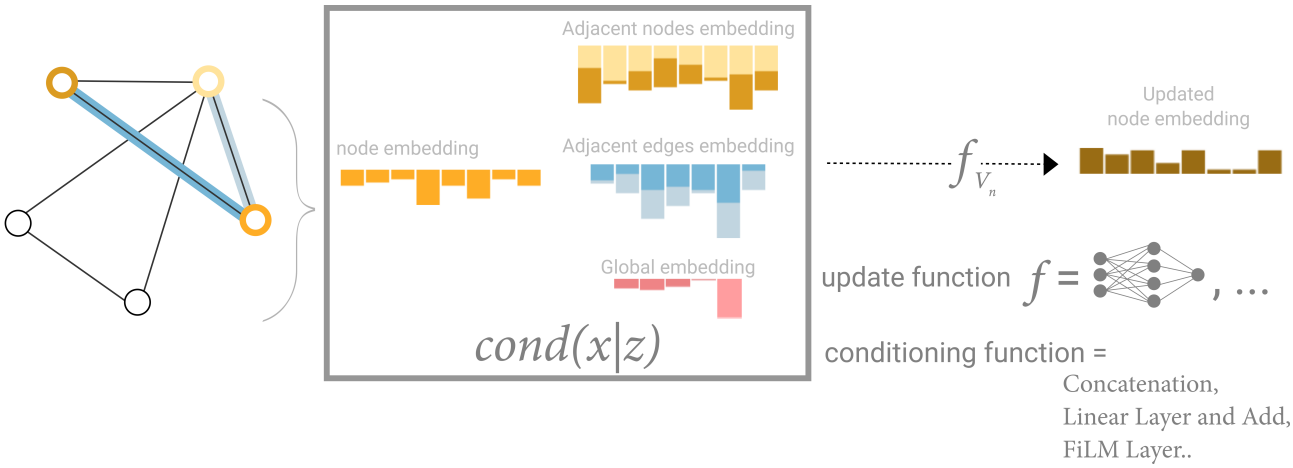


Figure 9.22: Schematic for conditioning one node's embedding based on three other embeddings (adjacent nodes, adjacent edges, global).

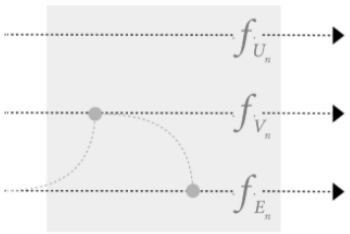


Figure 9.23: Node then edge learning

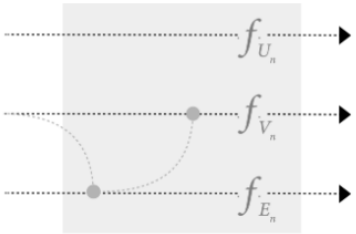


Figure 9.24: Edge then node learning

- GNN with a higher number of layers will broadcast information at a higher distance and can risk having their node representations 'diluted' from many successive iterations
- Sum has a very slight improvement on the mean performance, but max or mean can give equally good models.
- The more type of message passing the model has, the better the performance is.

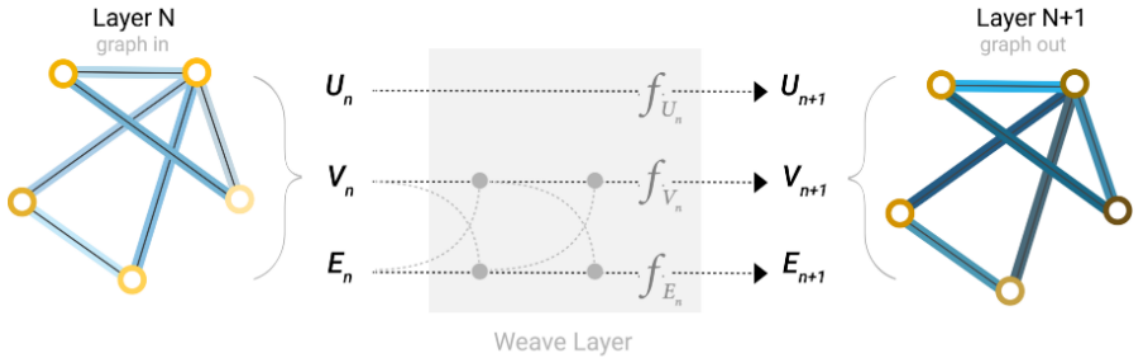


Figure 9.25: Learning in "weave fashion".

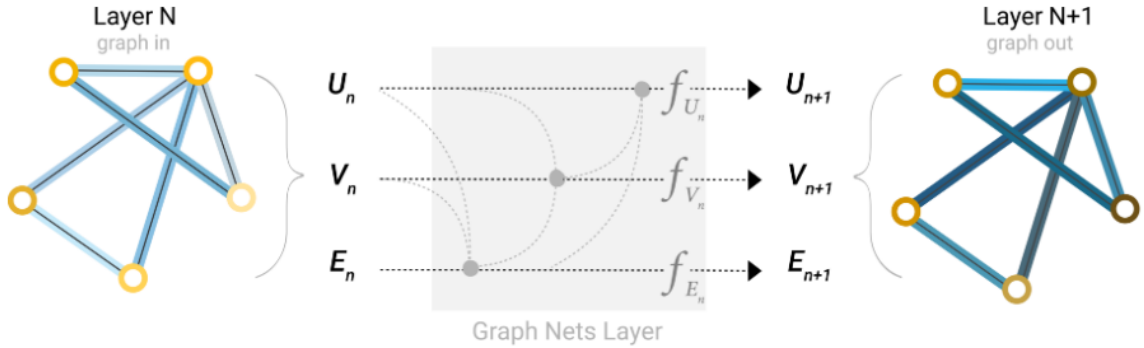


Figure 9.26: Schematic of a Graph Nets architecture leveraging global representations.

NOTE:

- Graph is very scalable. Interaction between edges and nodes in a very large graph can be trained within a smaller graph [PFSG+20]
- Take advantage of inductive biases (DDPS | Learning physical simulation with graph network)

9.4.9 Application

Source: CS224W

- PinSAGE: GNN for Recommendation systems: nodes are users, products. Based on other users' interactions with some products, predict whether one user would interest in a new product.
- Decagon: heterogeneous GNN for drug prediction
- Goal-directed generation: GCPN for molecule generation
- Mesh-based simulation [PFSG+20]

9.4.10 References

- Siraj Raval YouTube
- Sanchez-Lengeling et al. (2021) [SLRP+21].A Gentle Introduction to Graph Neural Networks.
- Daigavane et al. (2021) [DRA21].Understanding Convolutions on Graphs.
- How to get started with Graph Machine Learning - Aleksa Gordić

There are so many current open problems for research (2022)

- Convolutions on Graphs [DRA21]
- Multi-graphs
- Hyper-graphs
- Hyper-nodes
- hierarchical graphs
- Graph convolutions
- Graph Attention Networks
- Generative modeling

10 Generative Models

10.1 Definitions

- *Probabilistic model*: is a model that represents a probability distribution. It could be a marginal prob. distribution $p(x)$ or a conditional prob. distribution $p(y|x)$.
- *Evidence, query, and latent variables*:
  The word ***latent*** implies ***existing, but hidden***. E.g.:
  - In $p(x)$, x is the query variable and there is no evidence variable.
  - In $p(y|x)$, y is the query variable and x is the evidence variable.
  - In MoG $p(x) = \sum_z p(x|z)p(z)$, x is the query variable, z is the latent variable to represent the mixture elements.
  - In $p(y|x) = \sum_z p(y|x,z)p(z)$, z is the latent variable.
  - In a neural network, the inputs are evidence variables, the output are the query variables. In Variational Auto-Encoders (VAE), z is considered as the latent variable with a simple Gaussian distribution.
  - In model-based RL with latent variable models, state x_t is the latent variable.
- *Latent variable models*: is the type of probabilistic model that represent a complex pdf. as the product of multiple simpler pdf.s, one of which belongs to a latent variable.
  E.g.: in VAE, the complex distribution $p(x)$ (of an image) is represented as the product of two simple distribution. $p(z)$ is a Gaussian distribution, and $p(x|z)$ as a learnable neural network
  $$p(x) = \int p(x|z)p(z)dz$$

10.2 Variational Inference

Goal: to model a complex distribution $p_\theta(x)$, given the data $\mathcal{D} = \{x_1, \ldots, x_N\}$

Maximum likelihood fit: $$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \log p_\theta(x_i) \tag{10.1}$$

With latent variable z: $$p(x) = \int p(x|z)p(z)dz \tag{10.2}$$

⇒(sadly, completely ***intractable***) $$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \log\left(\int p(x|z)p(z)dz\right) \tag{10.3}$$

⇒alternative: *expected* log-likelihood $$\theta \leftarrow \arg\max_\theta \frac{1}{N}\mathbb{E}_{z\sim p(z|x_i)}[\log p_\theta(x_i, z)] \tag{10.4}$$

This section describes how to calculate $p(z|x_i)$

10.2.1 The Variational Approximation

Idea: Approximating $p(z|x_i)$ with $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$

$$\log p(x_i) = \log \int_z p(x_i|z)p(z)dz \tag{10.5}$$

$$= \log \int_z p(x_i|z)p(z)\frac{q_i(z)}{q_i(z)}dz \tag{10.6}$$

$$= \log \mathbb{E}_{z\sim q_i(z)}\left[\frac{p(x_i|z)p(z)}{q_i(z)}\right] \tag{10.7}$$

$$\geq \mathbb{E}_{z\sim q_i(z)}\left[\log \frac{p(x_i|z)p(z)}{q_i(z)}\right] \qquad \text{(Jensen's inequality: } \log \mathbb{E}[y] \geq \mathbb{E}[\log y]) \tag{10.8}$$

$$= \mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathbb{E}_{z\sim q_i(z)}[\log q_i(z)] \tag{10.9}$$

$$= \underbrace{\mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)}_{\mathcal{L}_i(p,q_i)} \tag{10.10}$$

- $\mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)$ is the lower bound of $\log p(x_i)$.
  Thus, maximize $\mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathcal{H}(q_i)$ will indirectly maximize $\log p(x_i)$.
- Maximize the 1st term $\mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)]$ will find the place with high $p(x_i, z)$
  Maximize the 2nd term $\mathcal{H}(q_i)$ will increase the randomness of z

In another point of view, as we approximate $p(z|x_i)$ with $q_i(z)$, how good is the approximation can be measured with the KL-divergence $D_{KL}(q_i(z)||p(z|x_i))$:

$$D_{KL}(q_i(z)||p(z|x_i)) = \mathbb{E}_{z\sim q_i(z)}\left[\log \frac{q_i(z)}{p(z|x_i)}\right] = \mathbb{E}_{z\sim q_i(z)}\left[\log \frac{q_i(z)p(x_i)}{p(x_i,z)}\right] \tag{10.11}$$

$$= -\mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] + \mathbb{E}_{z\sim q_i(z)}[\log q_i(z)] + \mathbb{E}_{z\sim q_i(z)}[\log p(x_i)] \tag{10.12}$$

$$= -\underbrace{\mathbb{E}_{z\sim q_i(z)}[\log p(x_i|z) + \log p(z)] - \mathcal{H}(q_i)}_{\mathcal{L}_i(p,q_i)} + \log p(x_i) \tag{10.13}$$

$$\Rightarrow \log p(x_i) = D_{KL}(q_i(z)||p(z|x_i)) + \mathcal{L}_i(p, q_i) \tag{10.14}$$

$$\Rightarrow \log p(x_i) \geq \mathcal{L}_i(p, q_i), \qquad \text{since } D_{KL}(q||p) \geq 0 \tag{10.15}$$

Since $\log p(x_i)$ is independent of q_i, maximizing $\mathcal{L}_i(p, q_i)$ with regard to (w.r.t.) q_i will minimize the KL-divergence, thus, improving the approximation.

10.2.2 Algorithm

Again, the goal is to model the complex distribution $p(x)$. Since the common maximum likelihood is intractable, we maximize its lower bound.

$$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \log p_\theta(x_i) \qquad \text{– intractable maximum likelihood}$$

$$\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \mathcal{L}_i(p, q_i) \qquad \text{– tractable lower bound}$$

The algorithm:

1. for each x_i (or mini-batch):
2. calculate $\nabla_\theta \mathcal{L}_i(p, q_i)$:
3. sample $z \sim q_i(z)$
4. $\nabla_\theta \mathcal{L}_i(p, q_i) \approx \nabla_\theta \log p_\theta(x_i|z)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i(p, q_i)$
6. update q_i to maximize $\mathcal{L}_i(p, q_i)$

In the above 6^{th} step, assuming $q_i(z) = \mathcal{N}(\mu_i, \sigma_i)$:

- calculate the gradient $\nabla_{\mu_i} \mathcal{L}_i(p, q_i)$ and $\nabla_{\sigma_i} \mathcal{L}_i(p, q_i)$
- apply gradient ascent on μ_i and σ_i to maximize $\mathcal{L}_i(p, q_i)$.

Problem: for each x_i, there will be one μ_i and one σ_i, with the appropriate dimension. Thus, the total number of params. would be huge: $|\theta| + (|\mu_i| + |\sigma_i|) \times N$

10.3 Amortized Variational Inference

As mentioned above, the problem with using a Gaussian for $q_i(z)$ is the huge number of params.. The word *amortize* means to reduce or pay off (a debt) with regular payments. *Amortized Variational Inference* alleviates the params. problem by using a neural network (Fig. 10.1)

$$q_i(z) = q(z|x_i) \approx q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)) \tag{10.16}$$

$$\Rightarrow \log p(x_i) \geq \underbrace{\mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z) + \log p(z)] + \mathcal{H}(q_\phi(z|x_i))}_{\mathcal{L}_i(p_\theta(x_i|z), q_\phi(z|x_i))} \tag{10.17}$$

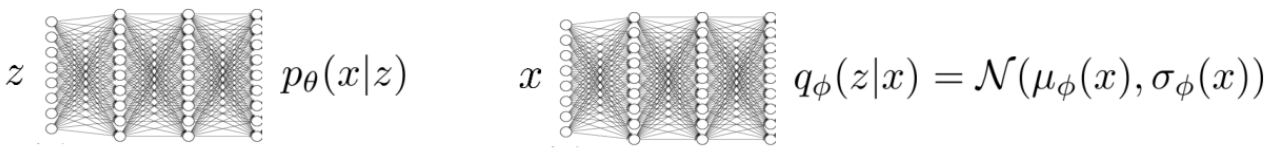


Figure 10.1: Two neural networks: $p_\theta(x|z)$ and $q_\phi(z|x)$.

10.3.1 Algorithm

1. for each x_i (or mini-batch):
2. calculate $\nabla_\theta \mathcal{L}_i(p_\theta(x_i|z), q_\phi(z|x_i))$:
3. sample $z \sim q_\phi(z|x_i)$
4. $\nabla_\theta \mathcal{L} \approx \nabla_\theta \log p_\theta(x_i|z)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$
6. $\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}$

Let's consider the gradient $\nabla_\phi \mathcal{L}$ in the above 6^{th} step:

$$\mathcal{L}_i = \underbrace{\mathbb{E}_{z\sim q_\phi(z|x_i)}\Big[\log p_\theta(x_i|z) + \log p(z)\Big]}_{J(\phi) = \mathbb{E}_{z\sim q_\phi(z|x_i)}\big[r(x_i,z)\big]} + \mathcal{H}(q_\phi(z|x_i)) \tag{10.18}$$

$$= \nabla_\phi J(\phi) + \nabla_\phi \mathcal{H}(q_\phi(z|x_i)) \tag{10.19}$$

- There is a formula to find the gradient of the entropy of a Gaussian

$$\nabla_\phi \mathcal{H}(q_\phi(z|x_i)) = \nabla_\phi \mathcal{H}\big(\mathcal{N}(\mu_\phi, \sigma_\phi)\big) \tag{10.20}$$

- To find the gradient $\nabla_\phi J(\phi)$, we could use similar approach as in policy gradient, however, with the problem of high variance. Instead, we could use the reparameterization trick:

$$J(\phi) = \mathbb{E}_{z\sim q_\phi(z|x_i)}\big[r(x_i,z)\big] \qquad q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x)) \tag{10.21}$$

$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\big[r(x_i, \mu_\phi(x) + \epsilon\sigma_\phi(x))\big] \qquad z = \mu_\phi(x) + \epsilon\sigma_\phi(x) \tag{10.22}$$

  with $\epsilon \sim \mathcal{N}(0,1)$ is independent of ϕ!
  $\Rightarrow$ estimate $\nabla_\phi J(\phi)$:
  - sample $\epsilon_1, \ldots, \epsilon_M$ from $\mathcal{N}(0,1)$ (a single sample works just well!)
  - $\nabla_\phi J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi r(x_i, \mu_\phi(x) + \epsilon\sigma_\phi(x))$

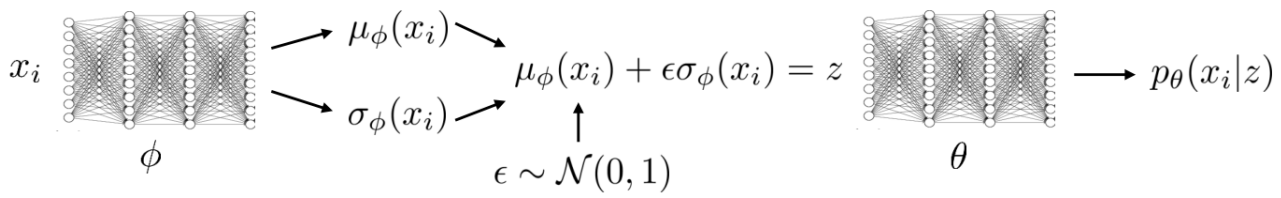


Figure 10.2: The complete structure for amortized variational inference.

10.3.2 Reparameterization Trick

Another way to look:

$$\mathcal{L}_i = \mathbb{E}_{z\sim q_\phi(z|x_i)}\Big[\log p_\theta(x_i|z) + \log p(z)\Big] + \mathcal{H}(q_\phi(z|x_i)) \tag{10.23}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] + \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p(z)] + \mathcal{H}(q_\phi(z|x_i)) \tag{10.24}$$

$$= \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{KL}(q_\phi(z|x_i)||p(z)) \tag{10.25}$$

$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\Big[\log p_\theta(x_i|\mu_\phi(x) + \epsilon\sigma_\phi(x))\Big] - D_{KL}(q_\phi(z|x_i)||p(z)) \tag{10.26}$$

$$\approx \log p_\theta(x_i|\mu_\phi(x) + \epsilon\sigma_\phi(x)) - D_{KL}(q_\phi(z|x_i)||p(z)) \quad \text{(single sample estimate)} \tag{10.27}$$

NOTE: Reparameterization trick vs. policy gradient

- Policy Gradient

$$\nabla_\phi J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi \log q_\phi(z_j|x_i) r(x_i, z_j)$$

  +can handle both discrete and continuous latent variables

  −high variance, requires multiple samples & small learning rates
- Reparameterization trick

$$\nabla_\phi J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi r(x_i, \mu_\phi(x) + \epsilon\sigma_\phi(x))$$

  −only continuous latent variables

  +very simple to implement

  +low variance

10.4 Conditional Variational Inference Models

$$\mathcal{L}_i = \mathbb{E}_{z\sim q_\phi(z|x_i,y_i)}\Big[\log p_\theta(y_i|x_i, z) + \log p(z|x_i)\Big] + \mathcal{H}(q_\phi(z|x_i, y_i)) \tag{10.28}$$

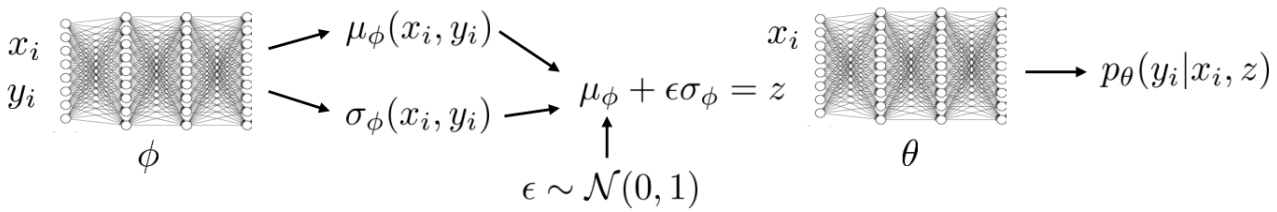


Figure 10.3: The conditional model.

10.5 Generative Models

Generative models are models that generate data x. E.g.: $p(x)$ is a generative model, because knowing $p(x)$, we can sample x.

NOTE: Not all generative models are not necessary latent variable models, and not all latent variable models are generative models. But it's common for a generative model to be a latent variable model, because to generate data, we usually want to know the prob. distribution of it. When that prob. distribution is complex, we would represent it as a product of multiple simple prob. distribution, using some latent variables.

10.5.1 Generative Adversarial Network

Generative Adversarial Network (GAN) by Goodfellow et al. (2014) [GPAM+14]:

$$D(x) \quad \text{– prob. that } x \text{ is real (from training data)} \tag{10.29}$$

$$1 - D(x) \quad \text{– prob. that } x \text{ is fake (from the generator)} \tag{10.30}$$

$$\max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] \quad \text{– the discriminator goal} \tag{10.31}$$

$$\min_G \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad \text{– the generator goal} \tag{10.32}$$

- The discriminator goal is to maximize the log prob. of the data from the training set classified as real.
- The generator goal is to minimize the log prob. of the data from the generator classified as fake.

The loss function of GAN:

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{10.33}$$

In `pytorch`, there is BCELOSS (Binary Cross Entropy loss):

$$l(x, y) = L = \{l_1, \dots, l_N\}^T \tag{10.34}$$

$$l_n = -w_n[y_n \log x_n + (1 - y_n) \log(1 - x_n)] \tag{10.35}$$

10.5.2 DCGAN

Deep Convolutional Generative Adversarial Network (DCGAN) by Radford, Metz, and Chintala (2015) [RMC15] is the completely convolutional version of GAN.

- The discriminator has strided convolution, batch norm layers, Leaky ReLU activations.
- The generator has transposed convolution, batch norm layers, ReLU activations.

[**TODO:**]

10.5.3 CGAN

Conditional Generative Adversarial Network (CGAN) (based on Sec. 10.4) is first proposed by Mirza and Osindero (2014) [MO14].

Example works:

- pix2pix: image-to-image translation [IZZ+17]

$$\texttt{bool} = D(x, y) \quad \text{– given } x \text{, the discriminator classify whether } y \text{ is real.} \tag{10.36}$$

$$y = G(x, z) \quad \text{– given } x \text{ and noise } z \text{, the generator generate } y. \tag{10.37}$$

10.5.4 StyleGAN

NOTE: Move to CV notes

10.5.5 Tips And Tricks

Tips and tricks for training GAN from `github/ganhacks`

- Train the Discriminator: maximizing $\log D(x) + \log(1 - D(G(z)))$
  Construct different mini-batches for real and fake, then accumulate the gradients
  - Compute the loss $\log D(x)$ with real samples
  - Compute the loss $\log(1 - D(G(z)))$ with fake samples
- Train the Generator: maximizing $\log D(G(z))$
  Minimizing the $\log(1 - D(G(z)))$ leads to insufficient gradients, especially in early learning process. Thus, we instead maximize $\log D(G(z))$.

10.5.6 Examples

- GAN: `pytorch`, `tensorflow` (MNIST dataset)
- DCGAN:
  - MNIST: `pytorch`
  - CIFAR10: `pytorch`, `tensorflow`
  - CelebA: `Pytorch`'s tutorial
- CGAN
  - The CMP Facade Database: `tensorflow`'s tutorial

10.6 References

- Variational Inference | CS 285, Stanford | YouTube
- Generative models | CS231, Stanford | YouTube
- Variational Autoencoders | Arxiv Insights | YouTube
- Variational Autoencoders | CodeEmporium | YouTube
- Real "deconv" layer | GitHub

11 Hyperparameters Optimization

11.1 Introduction

Hyperparameters are params. that:

- Define model's architecture
- Do NOT change with your model training
- Are NOT learnt from your model training
- Each model has its own set of params.

Hyperparameters tuning affects:

- Speed of convergence
- Generalization of your model
- Find optimal solution space
- Find right capacity for your solution
- Identify ideal architecture

Examples:

- Different types of gradient descents
- ...

11.2 Exhaustive Search

11.2.1 Grid Search

Grid search will suffer to the curse of dimensionality

- Define range of possible values for all params.
- Evaluate model performance for each hyperparams. combination
- Cross validate
- Pick the top N hyperparams.

11.2.2 Random Search

Random search is generally faster and more efficient, though does NOT guarantee good coverage.

- Define random sampling space (explicit values or distribution)
- Sample hyperparams. randomly
- Evaluate model performance for each hyperparams. combination
- Cross validate
- Pick the top N hyperparams.

11.2.3 Example

Python libraries:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

Examples:

- codebasics YouTube

11.3 Sequential Model Based

11.3.1 Bayesian Optimization

Use Acquisition function [**TODO:**]

11.3.2 Genetic Algorithm

[**TODO:**]

```
from skopt import BayesSearchCV
```

11.4 Tools

Current good tools for hyperparams. search:

- Weights & Biases: `import wandb`
- Scikit-learn: `from sklearn.model_selection import *`
- SkOpt: `skopt`

11.5 References

- Siraj Raval YouTube
- Machine Learning Mastery YouTube

12 Neural Architecture Search

[**TODO: Neural Architecture Search (NAS)**]

- NAS is still a novel field, approaches are still expensive and slightly complicated
- NAS currently (2022) search for micro architecture, not macro.

12.1 References

- Microsoft Research video
- Zoph et al. (2016) [ZL16]. *"Neural Architecture Search with Reinforcement Learning"*.
- Pham et al. (2018) [PGZ+18]. *"Efficient Neural Architecture Search via Parameter Sharing"*.
- Liu et al. (2018) [LZN+18]. *"Progressive Neural Architecture Search"*.
- Liu et al. (2018) [LSY18]. *"DARTS: Differentiable Architecture Search"*.

13 Deep Learning Generalization

This chapter presents different techniques and idea to generalize deep learning and make it more applicable to variety of problems.

13.1 Transfer Learning

Transfer learning focuses on storing knowledge gained while solving a problem and applying it to a different but related problem. Simply put:

- The new model comprises mostly of layers from a pretrained model.
- We take the structure and corresponding params. of a model that is pretrained on a large dataset for the feature extraction.
- We add a few (1-2) new layers in the end, which are designated to our current specific problem.

NOTE:

- Lower the learning rate

Example with `TensorFlow` (src):

```
import tensorflow as tf
import tensorflow_hub as hub

model_path = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
pretrained_model = hub.KerasLayer(
    model_path, input_shape=(224, 224, 3), trainable=False)

num_of_new_classes = 5
model = tf.keras.Sequential([
    pretrained_model,
    tf.keras.layers.Dense(num_of_new_classes)])
model.summary()

model.compile(
    optimizer="adam",
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['acc'])

model.fit(X_train_scaled, y_train, epochs=5)
```

13.2 Fine-Tuning

In fine-tuning, a pretrained model is used, just like in transfer learning.

- In fine-tuning, we retrain the whole model with the new dataset, while in transfer learning, we freeze the params. of pretrained model.
- Can incrementally adapt the pretrained features to the new dataset

13.3 Distillation

[**TODO: Knowledge distillation, model distillation, dataset distillation**]

13.4 Meta-Learning

13.5 Multi-task Learning

13.6 References

14 Technical Tools

This chapter talks about or at least lists out helpful tools, platforms for using/working with DL and ML in general.

14.1 Supporting Platforms & Tools

- fastai: simplifies training fast and accurate neural nets using modern best practices
- Weights and Biases: builds better models faster with experiment tracking, dataset versioning, and model management

14.2 Coding Libraries

Libraries for generic ML:

- scikit-learn is a free software ML library for the Python programming language
- XGBoost: Scalable and Flexible Gradient Boosting

Libraries for building DL models:

- Keras: is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

```
from tensorflow import keras
from tensorflow.keras import layers
```

- TensorFlow: is an end-to-end open source platform for ML

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

- PyTorch: is an open source ML framework that accelerates the path from research prototyping to production deployment.

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
```

- Sonnet: DeepMind's library for constructing neural networks in TensorFlow. Check this YouTube video. It can be seen as a library to make TensorFlow more suitable for research purposes at DeepMind.

```
$ pip install dm-sonnet
import sonnet as snt
import tensorflow as tf
```

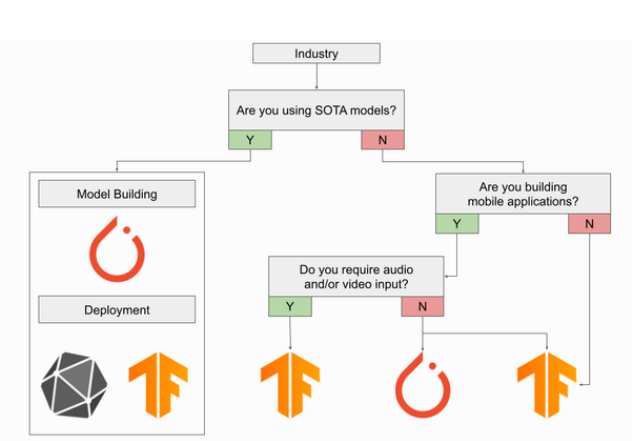
Comparisons on assemblyai.com, towardsdatascience.com, builtin.com: Tab. 14.1

TensorFlow (Google)	PyTorch (Facebook)
Static graph definition	Dynamic graph definition: you can define, change and execute nodes as you go, no special session interfaces or placeholders.
Awesome Tensorboard's visualization	
Better deployment with TensorFlow Serving	
Distributed model training	
Data parallelism require more manual work and careful thought	Easy Declarative data parallelism
	more clear and developer-friendly

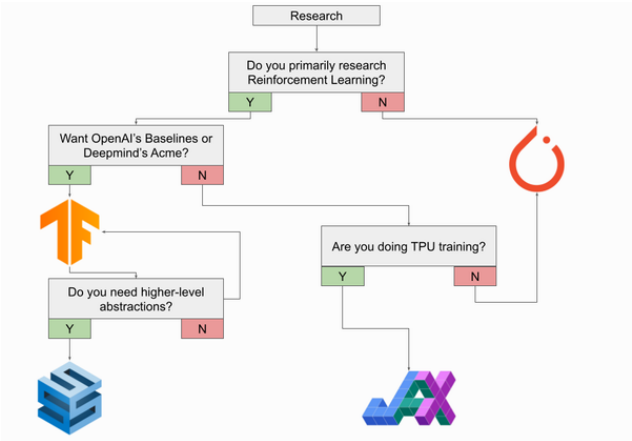
Table 14.1: Comparison between TensorFlow and PyTorch.

Guide on choosing which library to work with:

- If you are in the industry: Fig. 14.1a
- If you are a researcher: Fig. 14.1b



(a) If I'm in Industry



(b) If I'm a Researcher

Figure 14.1: Choosing DL libraries to work with [Cor].

14.3 Cloud GPU Platforms

DL takes an extensive amount of time for training and computation tasks. GPUs are designed to solve this problem. They offer high efficiency to perform heavy computations and faster training for your AI models in parallel. According to Indigo research, GPUs can offer ***250 times faster*** performance than Central Processing Unit (CPU)s while training neural networks associated with deep learning. [Pat]

Benefits of using Cloud GPUs are:

- Highly scalable
- Cost minimization
- Clearance of local resources
- Time saving

Cloud GPU Platforms for AI and Massive Workload:

- Google Cloud GPUs
- IBM Cloud
- AWS and NVIDIA
- Microsoft Azure and their Deep Learning Virtual Machine (DLVM). Check this guide
- Lambda GPU
- Paperspace CORE
- Others worth mentioned: Linode, Elastic GPU service, OVHcloud, Genesis Cloud

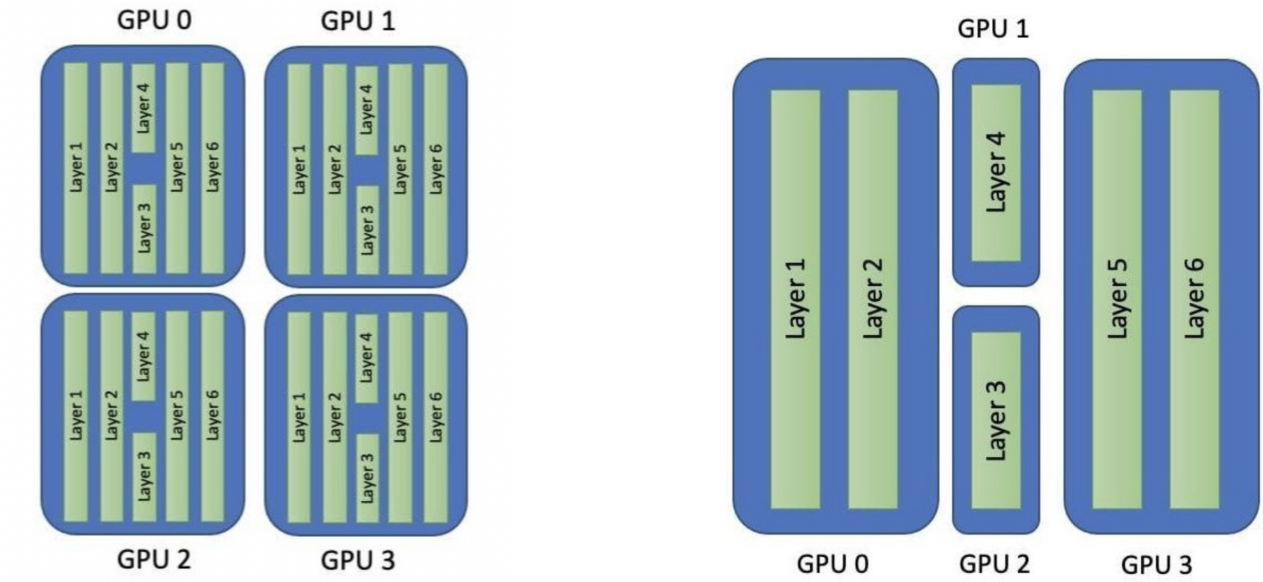
For comparisons between services:

- https://cloud-gpus.com/
- https://www.paperspace.com/gpu-cloud-comparison
- https://www.dataversity.net/cloud-gpu-instances-what-are-the-options/

14.4 Distributed Learning

Distributed Learning, a.k.a. *distributed training*, is the problem of how you couple the training process given multiple GPUs.

- Data Parallelism: Fig. 14.2a. The training could be synchronous or asynchronous
- Model Parallelism: Fig. 14.2b



(a) Data parallelism. **(b)** Model parallelism.

Figure 14.2: Deep Learning on Supercomputers (src).

14.4.1 Example

`TensorFlow` example for data parallelism (src):

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="4"

import tensorflow as tf
from tensorflow import keras

tf.config.experimental.list_physical_devices()
tf.test.is_built_with_cuda()

# Create model
...

strategy = tf.distribute.MirroredStrategy()
strategy.num_replicas_in_sync

# Training with CPU
with tf.device('/CPU:0'):
  cpu_model = get_model()
  cpu_model.fit(train_dataset, epochs=50)

# Training with GPU
with strategy.scope():
  gpu_model = get_model()
```

```
24   gpu_model.fit(train_dataset, epochs=50)
```

PyTorch example for data parallelism (src):

```
1 model = Model(input_size, output_size)
2 if torch.cuda.device_count() > 1:
3   print("Using", torch.cuda.device_count(), "GPUs!")
4   model = nn.DataParallel(model)
5
6 model.to(device)
```

Bibliography

[Ala20] J. Alammar. *The Illustrated Transformer*. https://jalammar.github.io/illustrated-transformer/. Accessed: 2022/07/22. 2020.

[BMR+20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language Models are Few-Shot Learners". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 1877–1901.

[Cor] *PyTorch vs TensorFlow in 2022*. https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/. Accessed: 2022/07/20. 2021.

[DCL+18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[DRA21] A. Daigavane, B. Ravindran, and G. Aggarwal. *Understanding Convolutions on Graphs*. https://distill.pub/2021/understanding-gnns/. Accessed: 2022/08/24. 2021.

[DSK16] V. Dumoulin, J. Shlens, and M. Kudlur. "A Learned Representation for Artistic Style". In: *arXiv preprint arXiv:1610.07629* (2016).

[DV16] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[Elm90] J. L. Elman. "Finding Structure in Time". In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[GPAM+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 27 (2014).

[HB17] X. Huang and S. Belongie. "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1501–1510.

[HLVDM+17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[HVD+15] G. Hinton, O. Vinyals, J. Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[HZR+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *Proc. of the IEEE/CVF Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[IS15] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[IZZ+17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-image translation with conditional adversarial networks". In: *Proc. of the IEEE/CVF Int. Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1125–1134.

[JLB+22] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun. "Hands-on Bayesian neural networks—A tutorial for deep learning users". In: *IEEE Computational Intelligence Magazine* 17.2 (2022), pp. 29–48.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 25 (2012).

[KW13] D. P. Kingma and M. Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[LBB+98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[LLW+18] X. Liu, Y. Li, C. Wu, and C.-J. Hsieh. "Adv-bnn: Improved adversarial defense through robust bayesian neural network". In: *arXiv preprint arXiv:1810.01279* (2018).

[LSY18] H. Liu, K. Simonyan, and Y. Yang. "DARTS: Differentiable Architecture Search". In: *arXiv preprint arXiv:1806.09055* (2018).

[LZN+18] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. "Progressive Neural Architecture Search". In: *Proc. of the European Conference on Computer Vision (ECCV)*. 2018, pp. 19–34.

[MO14] M. Mirza and S. Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

[Pat] *10 Best Cloud GPU Platforms for AI and Massive Workload*. https://geekflare.com/best-cloud-gpu-platforms/. Accessed: 2022/07/20. 2022.

[PFSG+20] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. "Learning Mesh-Based Simulation with Graph Networks". In: *arXiv preprint arXiv:2010.03409* (2020).

[PGZ+18] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. "Efficient Neural Architecture Search via Parameter Sharing". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2018, pp. 4095–4104.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer. 2015, pp. 234–241.

[RMC15] A. Radford, L. Metz, and S. Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[SLL19] K. Shridhar, F. Laumann, and M. Liwicki. "A comprehensive guide to bayesian convolutional neural network with variational inference". In: *arXiv preprint arXiv:1901.02731* (2019).

[SLRP+21] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko. *A Gentle Introduction to Graph Neural Networks*. https://distill.pub/2021/gnn-intro/. Accessed: 2022/08/24. 2021.

[SVL14] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

[SZ14] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[ULV+16] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. "Texture Networks: Feed-forward Synthesis of Textures and Stylized Images". In: *arXiv preprint arXiv:1603.03417* (2016).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention Is All You Need". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 30 (2017).

[Vu18] H. T. Vu. *Deep learning*. 2018.

[ZKT+10] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2528–2535.

[ZL16] B. Zoph and Q. V. Le. "Neural Architecture Search with Reinforcement Learning". In: *arXiv preprint arXiv:1611.01578* (2016).