

# Deep Learning Cheat Sheet

## Evaluation Metrics

$$\begin{aligned} \text{Accuracy} &= \frac{TP + TN}{TP + TN + FP + FN} \\ \text{Error Rate} &= 1 - \text{accuracy} \\ \text{Precision} &= \frac{TP}{TP + FP} \\ \text{TPR} &= \frac{TP}{TP + FN} & \text{FPR} &= \frac{FP}{FP + TN} \\ \text{TNR} &= \frac{TN}{TN + FP} & \text{FNR} &= \frac{FN}{FN + TP} \\ \text{F1-score} &= \frac{2 \cdot \text{Precision} \cdot \text{TPR}}{\text{Precision} + \text{TPR}} \\ \text{Specificity} &= \frac{TP}{TN + FP} \\ \text{AUC} &= \int_0^1 \text{TPR} \cdot d\text{FPR} \\ \text{Macro Average} &= \frac{1}{n} \sum_{i=1}^n \text{avg}_i \\ \text{Micro Average} &= \frac{\sum_{i=1}^n TP_i}{\sum_{i=1}^n TP_i + \sum_{i=1}^n FP_i} \end{aligned}$$

## Bias & Variance

$$\begin{aligned} \text{Bias}(h_\theta) &= \mathbb{E}[h_\theta, D] - f \\ \text{Var}(h_\theta) &= \mathbb{E}[(h_\theta, D - \mathbb{E}[h_\theta, D])^2] \\ \text{MSE} &= \text{Bias}(h_\theta)^2 + \text{Var}(h_\theta) + \sigma^2 \end{aligned}$$

Underfitting
Overfitting

high bias, low variance
low bias, high variance

## Data Preparation

**Min-max [0,1]** :  $x' = \frac{(x - x_{\min})}{(x_{\max} - x_{\min})}$

**Min-max [-1,1]** :  $x' = 2 \cdot \min\_max(x) - 1$   
min-max doesn't handle outliers.

**Z-norm** :  $x' = \frac{(x - \mu)}{\sigma}$

**Scaling & Centering**  
Scaling improves the numerical stability, the convergence speed and accuracy of the learning algorithms. Centering improves the robustness of the learning algorithms

## Activation Functions

—— Sigmoid ——

$\sigma(z) = \frac{1}{1+e^{-z}}$  — Smooth and differentiable. Used in output layers for binary classification.

—— Hyperbolic Tangent (tanh) ——

$f(z) = \tanh(z)$  — Smooth, differentiable, output centered around 0. Used in LSTM.

—— Rectified Linear Unit (ReLU) ——

$f(z) = \max(0, z)$  — Non-linear, used as a standard, but has dying units problem for  $z < 0$ .

—— Leaky ReLU ——

$f(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha z & \text{if } z < 0 \end{cases}$  — Addresses dying units problem with a small  $\alpha$  (typical  $\alpha = 0.01$ ).

—— Exponential Linear Unit (ELU) ——

$f(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha(e^z - 1) & \text{if } z < 0 \end{cases}$  — Similar to Leaky ReLU but more computationally expensive.

—— Softmax ——

$f(z_i) = \frac{e^{z_i}}{\sum_{j=0}^{K-1} e^{z_j}}$  — Used in the last layer for multi-class classification, outputs a probability distribution.

## Universal Approximation Theorem

A feedforward network with a linear output layer and at least one hidden layer with a non-linear activation function (e.g. sigmoid) can approximate a large class of functions  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  with arbitrary accuracy, provided that the network is given enough hidden units.

## Curse of Dimensionality

when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality

## Gradient Descent

- 1: Initialize parameter vector  $\theta_0$
- 2: **repeat**
- 3:   Compute the gradient of the cost function at current position  $\theta_t$  :  $\nabla_{\theta} J(\theta_t)$
- 4:   Update the parameter vector by moving against the gradient :  $\theta_{t+1} = \theta_t - \alpha \cdot \nabla_{\theta} J(\theta_t)$
- 5:   where  $\alpha$  is the learning rate.
- 6: **until** change in  $\theta$  is small

### MSE

$$J_{MSE}(\theta) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}(i) - y(i))^2$$

where :

- $\hat{y}(i) = h_{\theta}(x(i))$  is the prediction of the model,
- $y(i)$  is the true outcome,
- $m$  is the number of training examples.

$$\nabla_w J_{MSE}(w, b) =$$

$$\frac{1}{m} \sum_{i=1}^m \hat{y}(i) \cdot (1 - \hat{y}(i)) \cdot (\hat{y}(i) - y(i)) \cdot x(i)$$

$$\nabla_b J_{MSE}(w, b) =$$

$$\frac{1}{m} \sum_{i=1}^m \hat{y}(i) \cdot (1 - \hat{y}(i)) \cdot (\hat{y}(i) - y(i))$$

### Cross Entropy

$$J_{CE}(\theta) = - \sum_{i=1}^m y(i) \cdot \log h_{\theta}(x(i)) + (1 - y(i)) \cdot \log(1 - h_{\theta}(x(i)))$$

where :

- $p_{\theta}(y(i) | x(i))$  is the probability model parameterized by  $\theta$ , predicting the probability of the true class  $y(i)$  given the input  $x(i)$ ,
- $m$  is the number of observations or data points in the dataset.

$$\begin{aligned} \nabla_w J_{CE}(w, b) &= \frac{1}{m} \sum_{i=1}^m (\hat{y}(i) - y(i)) \cdot x(i) \\ \nabla_b J_{CE}(w, b) &= \frac{1}{m} \sum_{i=1}^m (\hat{y}(i) - y(i)) \end{aligned}$$

## Gradient Descent Variants

### BGD

Smooth, not wiggling, strictly decreasing cost, many epochs needed, choose larger learning rate, no out-of-core support - all data in RAM ( m), easy to parallelise.

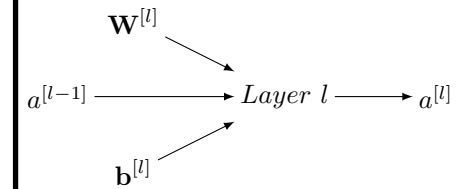
### SGD

Wiggling, needs smoothing, wiggles around minimum, not necessarily decreasing cost, few epochs needed, choose smaller learning rate, out-of-core support - not all data to be kept in RAM of a single machine, not easy to parallelise.

### MBGD

Slightly wiggling, wiggles around minimum, typically decreasing cost, less epochs than BGD, more than SGD needed, choose medium learning rate (dependent on model), out-of-core support - not all data to be kept in RAM of a single machine, easy to parallelise.

## Compute Graph



$$\begin{aligned} \mathbf{W}^{[l]} &= \begin{pmatrix} w_{11} & \cdots & w_{1n^{[l-1]}} \\ \vdots & \ddots & \vdots \\ w_{n^{[l]}1} & \cdots & w_{n^{[l]}n^{[l-1]}} \end{pmatrix} \\ \mathbf{a}^{[l]} &= \begin{pmatrix} a_1 \\ \vdots \\ a_{n^{[l]}} \end{pmatrix} \\ \mathbf{b}^{[l]} &= \begin{pmatrix} b_1 \\ \vdots \\ b_{n^{[l]}} \end{pmatrix} \end{aligned}$$

$$\begin{aligned} \mathbf{a}^{[l]} &= \sigma^{[l]}(\mathbf{z}^{[l]}) \\ \mathbf{z}^{[l]} &= \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \quad \text{with } \mathbf{a}^{[0]} = \mathbf{x} \end{aligned}$$

## Backpropagation

### Matrix Notation

$$\frac{\partial L}{\partial \mathbf{z}^{[l]}} = \frac{\partial L}{\partial \mathbf{a}^{[l]}} * \frac{d\sigma^{[l]}(\mathbf{z}^{[l]})}{dz}$$
$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \frac{\partial L}{\partial \mathbf{z}^{[l]}} \cdot \left(\mathbf{a}^{[l-1]}\right)^T$$
$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$
$$\frac{\partial L}{\partial \mathbf{a}^{[l-1]}} = \left(\mathbf{W}^{[l]}\right)^T \cdot \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$

### Full Batch

$$\frac{\partial L}{\partial \mathbf{Z}^{[l]}} = \frac{\partial L}{\partial \mathbf{A}^{[l]}} * \frac{d\sigma^{[l]}(\mathbf{Z}^{[l]})}{dz}$$
$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \frac{\partial L}{\partial \mathbf{Z}^{[l]}} \cdot \left(\mathbf{A}^{[l-1]}\right)^T$$
$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \frac{1}{m} \cdot \frac{\partial L}{\partial \mathbf{Z}^{[l]}} \cdot \begin{pmatrix} \vdots \\ 1 \\ \vdots \end{pmatrix}$$
$$\frac{\partial L}{\partial \mathbf{A}^{[l-1]}} = \left(\mathbf{W}^{[l]}\right)^T \cdot \frac{\partial L}{\partial \mathbf{Z}^{[l]}}$$

### Batch Normalization

$$\frac{\partial L}{\partial \gamma} = \frac{1}{m} \cdot \sum_{i=1}^m \frac{\partial L}{\partial \hat{a}^{(i)}} \cdot \frac{\partial \hat{a}^{(i)}}{\partial \gamma}$$
$$= \frac{1}{m} \cdot \sum_{i=1}^m \frac{\partial L}{\partial \hat{a}^{(i)}} \cdot \hat{a}^{(i)}$$
$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^m \frac{\partial L}{\partial \hat{a}^{(i)}} \cdot \frac{\partial \hat{a}^{(i)}}{\partial \beta}$$
$$= \sum_{i=1}^m \frac{\partial L}{\partial \hat{a}^{(i)}}$$

## Vanishing Exploding Gradient

### Xavier & Heu Initialization

Sets the initial weights of a layer to values drawn from a uniform distribution with a range that depends on the number of input and output units in the layer. Specifically, the range is set to  $[-r, r]$ , where  $r = \sqrt{\frac{6}{n_{in} + n_{out}}}$ , and  $n_{in}$  is the number of input units and  $n_{out}$  is the number of output units. This range was chosen because it ensures that the variance of the outputs of each layer remains constant, which helps to prevent the vanishing or exploding gradient problem.

### Batch Normalization

We calculate the average  $\mu_r$  and standard deviation  $\sigma_r$  over the  $m$  column vectors  $z_r$  of the mini-batch according to :

$$\mu_r = \frac{1}{m} \sum_{i=1}^m z_r^{[l](i)}$$
$$\sigma_r = \sqrt{\frac{1}{m} \sum_{i=1}^m (z_r^{[l](i)} - \mu_r)^2}$$

Now, the actual normalization of the logit matrix is as follows :

$$\hat{Z}_r^{[l]} = \frac{Z_r^{[l]} - \mu_r}{\sigma_r + \epsilon}$$

Finally, two addition parameter vectors are introduced that rescale the logits according to :

$$\tilde{Z}_r^{[l]} = \gamma^{[l]} \cdot \hat{Z}_r^{[l]} + \beta^{[l]}$$

### Non Saturating Activation Function

To alleviate the saturation of sigmoid and tanh, we can use the ReLU activation function. It still suffer from dying units problem (when the input is negative, the gradient is 0).

### Gradient Clipping

If gradients values exceed a certain threshold, they are "clipped" or rescaled to a smaller value. This prevents the gradients from becoming too large and helps to stabilize the training process.

## Optimizers

Momentum  
AdaGrad  
RMS Prop  
Adam

## Regularization

Weight Penalty  
Dropout  
Early Stopping

## CNN

Convolutional Layer  
Pooling Layer

## Unbalanced Dataset

Bayesian Approach  
Discrete  
Continuous  
Medical Test

## DeepCNN

Conf2D Params  
MaxPooling  
LeNet5  
AlexNet  
VGGnet  
GoogleNet  
ResNet  
Pattern

## Feature Visualization

Data Preparation  
Network  
Compile  
Evaluate  
Activation Map

## Data Augmentation

Principle  
Types  
Strategies  
Keras

## Functional API

Sequential vs Functionals  
Architecture 1  
Architecture 2  
Architecture 3

## Transfer Learning

Principle  
Keras Code  
MobileNet  
Strategies

## RNN

Use Case  
Model Category  
Recurrence Net  
Single Layer  
Many to Many  
Un exemple par catégorie  
Stacked RNN

## LSTM

Long Term Memory Unit Cell  
Gates  
Backprop  
Keras  
GRU

## Word Embedding

Word  
Training

## Sentiment Classification

Strategy  
Architecture

## Autoencoder

Definition  
Use Case

## GenRNN

Many to Many  
Many to One

## Attention

Sequence to Sequence  
Attention

## Transformer

High-Level Architecture  
Self-Attention  
Full Architecture