Deep Learning Cheat Sheet

Evaluation Metrics

Accuracy $= \dfrac{TP+TN}{TP+TN+FP+FN}$

Error Rate $= 1 - accuracy$

Precision $= \dfrac{TP}{TP+FP}$

TPR = Recall $= \dfrac{TP}{TP+FN}$

F1-score $= \dfrac{2 \cdot Precision \cdot Recall}{Precision + Recall}$

Specificity $= \dfrac{TP}{TN+FP}$

TNR $= \dfrac{TN}{TN+FP}$

FPR $= \dfrac{FP}{FP+TN}$

FNR $= \dfrac{FN}{FN+TP}$

AUC $= \displaystyle\int_0^1 TPR \cdot dFPR$

Macro Average $= \dfrac{1}{n} \sum_{i=1}^{n} avg_i$

Micro Average $= \dfrac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FP_i}$

Activation Functions

Sigmoid : $f(z) = \frac{1}{1+e^{-z}}$ — 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 α (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.

Data Preparation

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

Min-max [-1,1] : $x' = 2 \cdot min\_max(x) - 1$

min-max doesn't handle outliers.

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

Scaling & Centering

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

Gradient Descent

Initialize parameter vector θ_0

repeat

Compute the gradient of the cost function at current position θ_t : $\nabla_\theta J(\theta_t)$

Update the parameter vector by moving against the gradient : $\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta J(\theta_t)$

where α is the learning rate.

until change in θ 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 :

- $p_\theta(y(i) \mid x(i))$ is the probability model parameterized by θ, 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.

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

$\nabla_b J_{CE}(w, b) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}(i) - y(i))$

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.

Bias & Variance

Bias$(h_\theta) = \mathbb{E}[h_\theta, D] - f$

Var$(h_\theta) = \mathbb{E}[(h_\theta, D - \mathbb{E}[h_\theta, D])^2]$

MSE $= \text{Bias}(h_\theta)^2 + \text{Var}(h_\theta) + \sigma^2$

$\underbrace{\text{Underfitting}}_{\text{high bias, low variance}}$ $\underbrace{\text{Overfitting}}_{\text{low bias, high variance}}$

Theory

Compute Graph

Universal Approximation Theorem

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 sta- tistical 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

Backpropagation

MLP Layer

$\mathbf{a}^{[l]} = \sigma^{[l]}(\mathbf{z}^{[l]})$

$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$ with $\mathbf{a}^{[0]} = \mathbf{x}$

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

Vanishing Exploding Gradient

Saturation
Variance Change
Xavier & Heu Initialization
Batch Normalization
Non Saturating Activation Function
Gradient Clipping

Optimizers

Momentum
AdaGrad
RMS Prop
Adam
Scheduler

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
GRE

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