Deep Learning Cheat Sheet

Evaluation Metrics

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

Error Rate $= 1 - accuracy$

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

TPR $= \dfrac{TP}{TP+FN}$ FPR $= \dfrac{FP}{FP+TN}$

TNR $= \dfrac{TN}{TN+FP}$ FNR $= \dfrac{FN}{FN+TP}$

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

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

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

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

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

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

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

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

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 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.

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

Initialize parameter vector θ_0
repeat
    Compute the gradient of the cost function at current position $\theta_t : \nabla_\theta J(\theta_t)$
    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.

Compute Graph

$$\begin{tikzcd} \mathbf{W}^{[l]} \arrow[dr] & & \\ a^{[l-1]} \arrow[r] & \mathit{Layer}\ l \arrow[r] & a^{[l]} \\ \mathbf{b}^{[l]} \arrow[ur] & & \end{tikzcd}$$

$$\mathbf{W}^{[l]} = \begin{pmatrix} w_{11} & \cdots & w_{1n^{[l-1]}} \\ \vdots & \ddots & \vdots \\ w_{n^{[l]}1} & \cdots & w_{n^{[l]}n^{[l-1]}} \end{pmatrix}$$

$$\mathbf{a}^{[l]} = \begin{pmatrix} a_1 \\ \vdots \\ a_{n^{[l]}} \end{pmatrix}$$

$$\mathbf{b}^{[l]} = \begin{pmatrix} b_1 \\ \vdots \\ b_{n^{[l]}} \end{pmatrix}$$

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

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

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 μ_r and standard deviation σ_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

Uses a running average of gradients to smooth the optimization path.

$$m_t \leftarrow \beta m_{t-1} + \alpha \nabla_\theta J(\theta)$$
$$\theta_t \leftarrow \theta_{t-1} - m_t$$

Nesterov variant :

$$m_t \leftarrow \beta m_{t-1} + \alpha \nabla_\theta J(\theta_{t-1} - \beta m_{t-1})$$
$$\theta_t \leftarrow \theta_{t-1} - m_t$$

AdaGrad

Scaling down the gradient vector along the steepest dimensions.

$$s_t \leftarrow s_{t-1} + \nabla_\theta J(\theta_{t-1}) \cdot \nabla_\theta J(\theta_{t-1})$$
$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{s_t} + \epsilon} \cdot \nabla_\theta J(\theta_{t-1})$$

RMS Prop

A variant of AdaGrad using an exponentially decaying average of squared gradients.

$$s_t \leftarrow \beta s_{t-1} + (1-\beta) \nabla_\theta J(\theta_{t-1}) \cdot \nabla_\theta J(\theta_{t-1})$$
$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{s_t} + \epsilon} \cdot \nabla_\theta J(\theta_{t-1})$$

Adam

Combines momentum and RMSProp to adapt the learning rate for each parameter.

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) \nabla_\theta J(\theta_{t-1})$$
$$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) \nabla_\theta J(\theta_{t-1}) \cdot \nabla_\theta J(\theta_{t-1})$$
$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$$
$$\hat{s}_t \leftarrow \frac{s_t}{1-\beta_2^t}$$
$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{s}_t} + \epsilon} \cdot \hat{m}_t$$

Scheduler

Strategies to adjust the learning rate during training : **Performance scheduling**
Exponential scheduling : $\alpha(t) = \alpha_0 \cdot 10^{-t/T}$
Power scheduling : $\alpha(t) = \alpha_0 \cdot \left(1 + \frac{t}{T}\right)^{-c}$, where c is typically set to 1.

Regularization

Weight Penalty

Modify the loss function to include a penalty to big weights :

$$J(\theta) = J_0(\theta) + \lambda \cdot \Omega(W)$$

where $\Omega(W)$ is the penalty term :

- **L1 :** $\Omega(W) = \|W\|_1 = \sum |W_{ji}|$
- **L2 :** $\Omega(W) = \frac{1}{2}\|W\|_2^2 = \frac{1}{2}\sum(W_{ji}^2)$

Dropout

Randomly drop neurons during training :

- Each neuron has probability p of being dropped.
- Typical p : 50
- During testing, scale weights by $1-p$.

Early Stopping

Stop training when validation error increases :

- Track validation error during training.
- Save model parameters when validation error improves.
- Stop if no improvement for a set number of steps.

Convolutional Neural Networks

Features

colors, terrain texture, size, presence of straight lines, border

1. Extracting localized low-level features
2. Incrementally allowing the system to appropriately bind together features and their relationships
3. Gradually building up overall spatial invariance

Pooling layer

Maxpooling after a convolution layer eliminates non-maximal values : it is a form of non-linear down-sampling that reduces computation for upper layers and provides a "summary" of the statistics of features in lower layers.

Convolution layer

Different kernel sizes (3x3, 5x5, 7x7, etc.) allow the identification of features at different scales. Multiple layers of 3x3 kernels can implement other kernel sizes.

The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height) but extends through the full depth of the input volume.

We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border.

Input Volume

Size : $W_1 \times H_1 \times D_1$

Hyperparameters

K : Number of conv filters
F : Filter size (spatial extent)
S : Stride
P : The amount of zero padding

Output Volume

W2 : $\lfloor \frac{W_1 - F + 2P}{S} \rfloor + 1$
H2 : $\lfloor \frac{H_1 - F + 2P}{S} \rfloor + 1$
Size : $W_2 \times H_2 \times K$

Parameters

It introduces $(F \times F \times D_1) \times K$ weights plus K biases.

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