1. Techniques de régularisation d'un algo d'apprentissage d'1 NN

2. Techniques d'optimisation

- 'vanilla' Stochastic Gradient Descent (SGD)
- algorithmes plus performants.

① Techniques de regularisaº

Regularisation : Any modification to the learning algo that its intended to reduce the error generalization but its not training error.

↳ cost fonction that we minimiz during training

↳ ex: regression case

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(f_\theta(x_i) - y_i \right)^2$$

$\boxed{\text{Test Set}}$ → erreur de generalizaº

MSE sur le dataset de test.

Overfitting = "Sur apprentissage"

→ train error small

→ generaliza° error bigger.

→ trop de paramètres dans le modèle

→ dataset d'entrainement ⇒ représente toute votre donnée
dataset de test

→ hyperparamètres de votre algorithme d'entrainement.

→ $\boxed{\text{EPOCH}}$ → 1 mise à jour des paramètres sur tout le dataset d'entrainement.

Regularisaº

↳ Finding the trade-off for the errors between:

→ high bias ⇒ underfitting

↘ high variance

↳ overfitting

↳ generalizaº error the smallest.

↳ w/ a small gap between the training error and the generalization error.

In the context of NN

Deep Neural Networks → Billions of Parameters! → High Model Capacity

Keep large models but regularized them to avoid overfitting.

IA/ Parameter Norm Penalties

Cost function $J(\theta, X, y)$

= loss function

$$\tilde{J}(\theta, X, y) = J(\theta, X, y) + \underbrace{\alpha\, \Omega(\theta)}_{\text{terme de régularisation de la loss}}$$

terme de regularisation de la loss.

$\alpha > 0$.

$\Omega(\theta)$ penalized the $\boxed{\text{weights}}$

L^2 regularisation

$$\Omega(\theta) = \frac{1}{2} \|w\|^2 \rightarrow \text{Ridge regression.}$$

ℓ^1 parameter regularization

$$\Omega(\theta) = \|w\|_1 = \sum_i |w_i|$$

norm L^2 :

$$\tilde{J}(\theta, X, y) = \frac{\alpha}{2} w^T w + J(\theta, X, y)$$

$$\nabla_w \tilde{J}(\theta, X, y) = \alpha w + \nabla_w J(w, X, y)$$

mise à jour des weights w

ε : learning rate.

$$w_{k+1} \leftarrow w_k - \varepsilon\left(\alpha w_k + \nabla_{w_k} J(w_k, X, y)\right)$$

$$\boxed{w_{k+1} \leftarrow \left(1 - \underbrace{\varepsilon\alpha}_{\text{additional term}}\right) w_k - \varepsilon \nabla_{w_k} J(w_k, X, y)}$$

shrink the weight vector at each time step.

ℓ_1 regularisation → solution au niveau de la contrainte sur les poids qui est plus 'sparse'

↳ w tends to be more equal to 0.

↳ mécanisme de selection de variables.

IB/ Dropout

Technique computationally inexpensive but powerful to regularize a large family of models

↳ Any kind of Neural Network!

technique ⟶ 'Bagging' Methods for ensemble of very large Neural Networks.

$\boxed{\text{Dropout}}$

↳ trains sub-Networks that can be formed by removing hidden units from the underlying base Network.

@ Dropout algo

Each time we update the Network's parameters, we randomly sample a <u>binary mask</u> to all hidden units of the Neural Network

↓

selects randomly a percentage P of all hidden neurons dataset

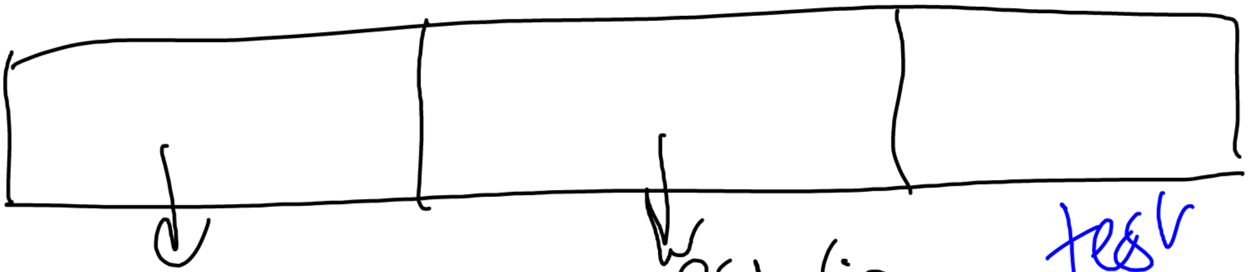
training set	test set

↓ training the NN → parameters θ.

↓ hold-out set to evaluate the capacity of the NN to generalize to unseen data

- hyperparamètres de votre NN
  - ↳ modèle → nbre de couches / nbre de neurones
  - ↳ algorithme d'entraînement
    - @ learning rate.
  - ↳ multiple training by varying its hyper-parameters.

⇒ DATA LEAKAGE



training set

↳ utilisé dans f° coût $J(\theta)$ to find θ^*

↳ modèle (NN) spécifique

↳ série d'hyperparamètres spécifiques.

validation set

evaluation of ≠ hyperparamètres

test set

Compute 'generalization error' on the test set for ('Best hyperparameters' $NN(\theta^*)$)

II/ Optimization strategies for Deep Neural Networks

Empirical-risk Minimization

Goal of an algorithm of ML: reducing the error generalization:

$$\boxed{J^*(\theta) = E_{(x,y)\sim p_{data}} \mathcal{L}(f(x,\theta), y)}$$

"vraie" distribution de la donnée

$\hat{p}_{data}$: training dataset
↳ distribution $\hat{p}_{data}$ pour le training dataset

$$\boxed{J(\theta) = E_{x,y\sim \hat{p}_{data}} \mathcal{L}(f(x,\theta), y)}$$

$\mathcal{D} = ((x_i, y_i)\ i \in \{1 \dots N\})$

$$\boxed{J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i, \theta), y_i)}$$

↳ In pratice, we minimize the negative log-likelihood:

$$J(\theta) = E_{x,y \sim \hat{p}data} \log P_{model}(x, y, \theta)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \log P_{model}(x_i, y_i, \theta)$$

→ Mini-batch gradient descent.

Batch $B \subset \mathcal{D}_{entrainement}$

↳ Most optimization algorithms converges faster if they are allowed to rapidly compute approximate

estimation of $\nabla J(\theta)$ ($\boxed{\nabla J(\theta)}$)
than slowly computing exactly
$J(\theta)$ ($\boxed{\nabla J(\theta)}$)

training algorithms based on
gradient-based learning for
machine learning problem

Batch gradient descent	Mini-batch gradient descent	stochastic online gradient descent
compute $\nabla J(\theta)$ on the whole training dataset & then we update θ.	compute $\nabla J(\theta)$ on the mini-batch & then we update θ	compute $\nabla J(\theta)$ per training sample & then we update θ.

Most efficient method

II B/ Optimization algorithms (beyond SGD)

$\boxed{SGD}$ Mise à jour de θ^k de cette manière :

$$\boxed{\theta_k \leftarrow \theta_{k-1} - \alpha \nabla_{\theta_{k-1}} J(\theta_{k-1})}$$

hyperparamètre α. : learning rate (taux d'apprentissage)

⊂ big sensivity of the performance of the learning algorithm to α.

(a) SGD with decreasing α_k.

$$\alpha_k = (1-\varepsilon)\alpha_0 + \varepsilon \alpha_{\tau}$$

$$\sum_{k=1}^{\infty} \alpha_k = \infty \Big] \text{ serie divergente}$$

$$\sum_{k=1}^{\infty} \alpha_k^2 < \infty \Big] \text{ serie convergente}$$

↳ In theory, convergence guarantees.

$\boxed{\text{Momentum}}$

$\boxed{\text{Goal}}$ ↳ Accelerate learning

↳ Faster Convergence.

→ Small but consistent gradients

↳ noisy gradients

v = velocity → Control direction & speed at which parameters move through the parameter space

Iteration k of the training algo:

$$v_k \leftarrow \boxed{\alpha} v_{k-1} - \boxed{\varepsilon} \nabla_{\theta_{k-1}} J(\theta_{k-1})$$

$$\theta_k \leftarrow \theta_{k-1} + v$$

New hyperparam ε.

IIc/ Algorithms with adaptative learning rates

learning rate α: 1 des hyperparamètres les + durs à tuner.

momentum add another hyperparameter.

learning rate α vary depending the $\neq$ parameters. $\theta \in \mathbb{R}^D \longrightarrow$ Billion of parameters.

Adagrad optimisation algo w/ adaptative learning rate.

↳ ⊗ parameters w/ the larger partial derivative of the loss

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial w_i} \\ \frac{\partial J(\theta)}{\partial b_i} \end{bmatrix}$$

↳ rapid decrease in the learning rate.

↳ ⊗ parameters w/ the smallest partial derivatives : small ↓ of the learning rate.

→ **RMS prop**

↳ **Adam**