AS – Cours 1

Encadrant : B. Piwowarski, Nicolas Baskiotis et Patrick Galinard.

Type d’apprentissages

Il existe 4 paradigmes pour l’apprentissage :

(Les premiers problèmes traités)

1. Supervisé :

Reconnaissance vocale : un domaine qui a trainé dû à la collecte difficile de données. La collecte du signale requiert un expert pour transcrir le signale. De plus, les machines de l’époque étaient incapables d’exécuter les calculs nécessaires.

2. Non supervisé.

(Puis)

3. Semi supervisé

Problème de base supervisé, mais faute de temps pour étiqueter les exemples (temps = argent). L’idée c’est d’exploiter les deux types de données (étiquetées et non étiquetées). La tâche est guidée par les données étiquetées.

Exemple :

* Le modèle doit trouver la structure - page 8.
* Applications liées au web.
* Apprentissage par transfert : On a fait un certain travail et on nous demande de le reproduire ~ transférer la connaissance ~ apprentissage semi supervisé

4. Renforcement (apparaît entre 1950 et 1960 – curiosité) demande :

Énormément d’essais.

Information qualitative (bien/mal).

Exemple :

* Backgammon (1992 – IBM) premier champion informatique
* Alpha Go (2015)
* Alpha Zéro (2017)

Le robot développe une connaissance de l’environnement. On fait jouer deux machines (from scratch- f*rom the* [***beginning***](https://dictionary.cambridge.org/fr/dictionnaire/anglais/beginning)*,* without using anything that already [***exists***](https://dictionary.cambridge.org/fr/dictionnaire/anglais/exist)), celle qui gagne revoit un signal et moi je renforce ce signale à la prochaine partie…

Page 10

1. Data z generated from distribution p(z) (distribution en pratique on ne sait pas ce que c’est !)

2. Learning model fonction paramétrique

F = {} /θ qui minimise/ maximise le coût.

Parmi la famille de fonction on veut l’élément qui minimise/maximise le nombre d’erreur

3. Loss (mesure à quel point on est loin de l’optimal)

Réponse calculée – réponse désirée le coût = distance euclidienne/ norme l2 (exemple)

4. risque (n’est pas simple)

Résoudre le problème de risque empirique la solution est une bonne approximation de la solution qui résout le problème (mais pas satisfaisant si RN dépasse le linéaire faut trouver autre chose)

Apprentissage inductif : distribution => donne une information synthétique sur les données.

Réseau de neurones :

**Neurone :**

**Corps Cellulaire possède des dendrites un long axone au bout on a des synapses pour le relié à d’autre neurone**

1943 par le neurophysiologiste McCulloch et le mathématicien Walter Pitts (calcule logique)

1957 Perceptron

1960 financements se sont barrés

1980 revoit le jour

1990 SVM, Linear Regression and logistic regression RNN (non pas marchant depuis 30 ans)

1990-2000 CNN RNN

**Pourquoi les RNN ne verront pas l’ombre ?**

1. Il y’a beaucoup de données.
2. Grace à l’industrie du jeu (Carte Graphique GPU) le temps ded calcule est raisonnable.
3. Algorithme d’entrainement ont été améliorés ~1990
4. En pratique, il ne reste pas bloqué dans des minimums locaux, généralement assez proches du minima global.
5. Attire l’actualité et donc les financements.

Utilisation des RN :

* RN quasi utilisé en visualisation depuis 5 ans
* Segmenter les différents individus en sur une image.
* RN à convolution + RNR => génération d’étiquettes et de phrases sur une image.
* Décodage et traduction de langue.
* Principalement utiliser sur des données de type image (vidéo c’est plus cher).
* Apprentissage non supervisé, tel que générer des images (hiver => été).

Modèle **McCulloch**

Les années 40, on a modélisé - penser aux neurones (avant les ordinateurs et l’apprentissage par renforcement 60)

Le nombre total de neurones du [cerveau](https://fr.wikipedia.org/wiki/Cerveau) humain est estimé de 86[1](https://fr.wikipedia.org/wiki/Neurone#cite_note-1) à 100 [milliards](https://fr.wikipedia.org/wiki/Milliard)

Unité = neurone simple.

Force = connectivité entre les neurones.

Calcule distribuer sur tous les neurones.

Dendrites (l’information est transmise par son via au cell body du neurone sous forme chimique)

Synapse qui amplifie/ affaiblie l’information.

w module l’information

L’information x0 = 1, x1, x2 est acheminer vers le neurone par des poids synaptiques

le neurone intègre l’information ( Somme wi \* xi ) si cette somme est suffisamment grande > -w0 cela excite le neurone on envoie une information binaire 1si Somme wi \* xi + w0 > 0 sinon -1/0

Modèle perceptron 1958 Rosenblatt

Initialisation aléatoire après itération converge vers une solution (différent à chaque fois)

Méthode stochastique

=> playground.tensorflfow

Équation de l’hyperplan F(x) = 0

Séparation linéaire de l’espace

Problème soluble = linéairement séparable.

2 classes une sortie 1/-1

3 classes - 3 perceptrons

C1 vs C2, C3

C2 vs C1, C3

C3 vs C2, C1

P classes P sorties

|  |  |  |  |
| --- | --- | --- | --- |
|  | Y1 | Y2 | Y3 |
| C1 | 1 | -1 | - 1 |
| C2 | -1 | 1 | -1 |
| C3 | -1 | -1 | 1 |

Décision : j’attribue mon x a la classe dont la sortie est le plus élevée

**Propriétés du perceptron :**

1. théorème de convergence (Novikof-1962) :

Donnée séparable

la distance du point à la courbe = F(x)/ ||x||

yi (\* w.xi la distance algébrique (±)) forcé le tout à être +

min => la plus petite distance de mon ensemble de point a la droite

sup w … w étant fixé je permets a w de varier (occupe différente position )

sup … la distance de cette droite a mon ensemble de w doit être maximale => on cherche l’hyperplan qui est au milieu de mon ensemble de données

plus la marge est petite plus il est difficile de séparer les données.

l’algorithme converge

ro c’est pour borner

la garantie pratique de notre système : on l’entraîne sur un ensemble de donnée sur le test (étiquetées) et on mesure les performances

2. generalisation bound Aizerman 1964

Après la k ième correction (exemple mal classé) mk = … données bien classées

mk \* plus haut => raffinement de la borne

Garantie théorique : l’erreur de test est plus petite qu’epsilon

Il faut donner des garanties théoriques ou empiriques

Perceptron Adaline : comme un perceptron sans fonction à seuil (que la somme)

=> meme avec la retro propagation

F(x) = W. x

C(x,y) = || F(x) – y || \*\*2

D ci / Wr = -2( yi – w.x) x i ,k

D cw = 2(yi -w.xi).xi

y sortie du filtre adaptatif page 34

min E[z\*\*2] = sans min (on ne minimise pas le son de l’enfant )

minimiser la sortie du système c minimiser E[(n0- y )\*\*2]

les algorithmes de gradients sont des algorithmes d’optimisation

définir des direction pour descendre pour plus rapidement

algorithme myope

on connaît le point de part et l’expression de la fonction, le reste on le voit pas

C(w) = Somme || yi – F(xi)w ||\*\*2

(xi, yi) appartient D

Attention fonction de coût non quadratique => on trouve un minimum local

1. méthode de gradient de la plus grande pente (on suit le gradient) deltaw(t) = - gradient C(t),

C(t) fonction de coût

W appartient Rn w1, w2 … wn

Dw appartient Rn (vit dans le meme espace que w) Dc/Dw1, Dc/Dw2 … Dc/Dwn

Dérivée seconde donne des informations sur la courbure + espace de petite taille + optimiser finement (=> optimisation)

Nous on veut on système qui ne généralise pas qui optimise donc dérivé première suffit

Par défaut c des vecteurs colonnes donc T pour avoir un vecteur ligne.

Si données >>> + dimension >>> stochastique bien, c’est bruyant, permet d’échapper a des minimas locaux.

j’optimise un exemple pour l’échantillon suivant en bénéficie stochastique

j’optimise tout le monde pour en bénéficier a l’étape suivante batch

page 40 oscillations gauche droite pas haut bas attention

Perceptron :

Règle d’apprentissage du perceptron

La frontière de décision de chaque neurone de sortie est linéaire, les perceptrons sont incapables d’apprendre des motifs complexes. Cependant, si les instances d’entrainement peuvent etre séparées de façon linéaire, Rosenblatt a montré que l’algorithme converge forcément vers une solution => théorème de convergence du perceptron.

En régression, le perceptron ne produit pas en sortie une probabilité de classe. Ils se contente d’une prédiction basée sur un seul figé. Voila l’une des bonnes raisons de préférer la régression logistique aux perceptrons.

Pour les PMC ils ont remplacé la fonction échelon segment plat => pas de gradient à exploiter) par la fonction logistique (continue et dérivable en tout point =>permet à la DG de progresser à chaque étapes).

|  |  |  |  |
| --- | --- | --- | --- |
| Forme de S :   * [0 :1] | * [-1 :1] => ce qui rend la sortie de chaque couche plus au moins centrée en zéro au debut de l’entrainement. * La convergence s’en trouve souvent accélérée. | * Continue mais non dérivable en 0. * Rapide à calculer. * Convient mieux en pratique (RNA). * Ne reste pas bloquer sur des faux plats car il n’y a pas de saturation pour les grandes valeurs d’entrée (!= logistique et tanh). * +Elle ne sature pas pour les valeurs positive. * +Rapide à calculer. * -souffre d’un problème de mort des ReLU, au court de l’entrainement certains neurones meurent (arretent de produire autre que 0, en particulier si le taux d’apprentissage est >>>. Si la somme pondérée des entrées d’un neurone est négative => le neurone produira 0 en sortie (meurt). Une variante existe leaky Relu avec un hyperparamètre alpha .02 ‘niveau de fuite’ ou randomized leaky ReLU ou +++ELU | * Tache de classification |

Les taches de régression aucune fonction en couche de sortie.

Elle prend en compte les valeurs négatives🡺 permet au neurone d’avoir une sortie moyenne plus proche de zéro (atténue le problème de disparition des gradients).

L’hyperparamètre alpha =1 par défaut.

Elle prend un gradient différent de zéro si z < 0 🡺évite le problème de mort des neurones.

Elle est continue en tout point.

Lente à calculer (utilisation de la fonction exponentielle) mais c’est compensé par la vitesse de convergence.

En test la fonction ELU est plus lente que la fonction RELU.

🡺Variante 2017 la fonction SELU qui s’auto-normalise règle tous les problèmes d’explosion/disparition des gradients.

Quelle fonction choisir ?

Si les performances d’execution importent beaucoup => leanki relu a la place d’elu

Prelu si l’entrainement est immense et de sur-ajustement.

Réseau de neurone profond (DNN) :

Problème :

* Ils sont confrontés aux problèmes de disparition des gradient ou d’explosion des gradients, qui affecte les réseaux de NP et complique l’entrainement des couches inférieures.
* Avec un réseau aussi vaste, l’entrainement sera extrêmement lent.
* Un modèle comprenant des millions de paramètres risquera fort de conduire au sur-ajustement du jeu d’entrainement.

1.1 Explosion des gradients : L’algorithme de rétropropagation procède en deux phases Une passe avant => de la couche d’entrée vers la couche de sortie, suivie d’une passe arrière qui propage le gradient d’erreur de la couche de sortie vers la couche d’entrée. Lorsqu’il a déterminé le gradient de la fonction de cout par rapport à chaque paramètre du réseau, il utilise ces gradients pour actualiser chaque paramètre au cours d’une étape de descente du gradient.

Alors que l’algorithme progresse vers les couches inférieures, les gradients deviennent de plus en plus petits 🡺 La mise à jour modifie très peu les poids des connexions de la couche inférieure et l’entrainement ne converge jamais vers une bonne solution.

1

0

Moy 0.5

C’est une des raisons de l’abandon des RNP

Xavier Clorot et Yoshua Bengio en 2010 identifie le problème 🡺 la fonction d’activation SIGMOIDE et l’initialisation aléatoire avec une loi normale de moyenne 0 et d’écart-type 1.

Ils ont montré que la variance des couches de sorties de chaque couche est largement supérieure à celle de ses entrées. Lors de l’avancée dans un réseau, la variance ne cesse d’augmenter après chaque couche jusqu’à la saturation de la fonction d’activation dans les couches supérieures. Ce comportement est aggravé par le fait que la moyenne de la fonction logistique non pas de 0 mais de 0.5. De plus, si les entrées sont grandes (+/-) la fonction sature en 0 ou 1 🡺 pas de gradient à transmettre en arrière dans le réseau.

Gradient de plus en plus dilué, il ne reste quasi plus rien aux couches inférieures.

1.2 Explosion des gradients : Dans les réseaux récurrents il se peut que le gradient devienne de plus en plus gros, les couches reçoivent des poids extrêmement important 🡺 ce qui diverger l’algorithme.

Analogie : voie qui traverse plusieurs amplificateurs, chaque amplificateur doit faire ressortir la voix avec la meme amplitude.

Solution selon Xavier et Ho

‘Initialisation de Xavier/Glorot – fan-in et fan-out’

Ils soutiennent que la variance des sorties de chaque couche doit etre égale à la variance de ses entrées et les gradients doivent également avoir une meme variance avant et apres le passage au travers d’une couche en sens inverse. Il est impossible de garantir ces deux points, sauf si la couche possède un nombre égal de connexion d’entrée et de sortie.

Préférer la TANH

Une autre technique répandue ‘écrêtage de gradient – gradient clipping’ consiste à simplement les écrêter pendant l’entrainement afin qu’ils ne dépassent jamais un certain seuil (RN Récurrent) + une normalisation par lot + une fonction d’activation SELU

Réutiliser des couches pré entrainées :

Il est déconseillé d’entrainer un très grand RNP à partir de zéro. Il est préférable de toujours essayer de trouver un réseau de neurones existant qui accomplit une tache comparable à celle visé. Cela accéléré l’entrainement et donne de bonne performance avec des jeux de données d’entrainement assez petits.

Technique de régularisation pour les RN :

1. Arrêt précoce
2. Régularisation l1 et l2
3. Dropout 2012 – alpha drop out.
4. Régularisation max-norm
5. Augmentation des données : consiste à utiliser les instances d’entrainement existantes pour en générer de nouvelles, grossissant ainsi artificiellement la taille du jeu d’apprentissage. Cette méthode permet de réduire le sur-apprentissage et donc d’en faire une technique de régularisation. -l’ajout d’un simple bruit blanc (mauvaise idée car les modifications doivent contribuer à l’apprentissage, ce qui n’est pas le cas du bruit blanc).

On peut décaler, pivoter, redimensionner et varier le contraste 🡺 le modèle devient plus tolérant à la position, l’orientation, la taille et l’éclairage …

Il est préférable de générer les instances à la volée pendant l’entrainement plutôt que de gaspiller de l’espace de stockage et de la bande passante réseau.

Neurone à convolution

1996 Deep Blue bat le champion du monde des échecs Garry Kasparov.

CNN :

1998 Yann LeCun : Le-Net 5 (numéro de chèque ecrit à la main) gagnant de Image Net.

2012 Alex-Net : (ressemble à Le-Net, en étant plus large et plus profond, a été le premier à empiler les couches de convolution, sans intercaler des couches de Pooling.

2014 GoogleLeNet : réseau plus profond.

2015 ResNet.

Le perceptron n’est pas trivial et, pour le comprendre => examiner le fonctionnement des modules sensoriels.

Les réseaux de neurones convolutifs sont apparus suite à l’étude du cortex visuel du cerveau. Ils ont vu le jour en 1980, mais ont été fortement utilisés que depuis c’est dernières années grâce à l’augmentation de la puissance de calcul, la quantité de données les astuces sur les réseaux de neurones. Ils sont utilisés dans :

* Reconnaissance d’image.
* Effectuer des taches visuelles complexes.
* Reconnaissance vocale.
* Traitement automatique du langage naturel.

Pourquoi pas utilisé de RNP ?

Les RNP ont des couches intégralement connectées, un tel réseau convient parfaitement aux petites images, mais non adapté aux grandes images en raison de l’énorme quantité de paramètre qu’il exige. Les CNN résolve ce problème en utilisant des couches partiellement connectées. Donc moins de paramètres (entrainement plus rapide) réduit le risque de sur-ajustement et nécessite moins de données d’entrainement.

Le CNN a appris à reconnaitre un motif en un endroit, il peut le reconnaitre partout ailleurs (contrairement à un RNP, il reconnait un motif qu’en cet endroit précis), CNN généralise mieux.

Le RNP n’a pas de connaissance préalable de l’organisation des pixels (flatten), il ne connait pas les pixel voisin, contrairement au CNN qui intègre cette connaissance.

Architecture d’un CNN

Convolution : est une opération mathémarique qui fait glisser une fonction par-dessus une autre et mesure l’intégrale de leur mutltiplication ponctuelle. Ses liens avec les transformées de Fourier et de Laplaces sont étroits (souvent enmployée dans le traitement de signal).

Les couches de convolutions sont des corrélations croisées,

Couche de convolution 1 :

Le bloc le plus important.

Les neurones ne sont pas connectés à chaque pixel de l’image (uniquement aux pixels dans leurs champs récepteurs. A leur tour, les neurones de la deuxième couche de convolution sont chacun connecté uniquement aux neurones situés à l’intérieur d’un petit rectangle de la première couche.

Cette architecture permet au réseau de se focaliser sur des caractéristiques de bas, niveau dans la première couche cachée, puis de les assembler en caractéristiques de plus haut niveau dans la couche cachée suivante, etc.

Attention ajout de zéro pour avoir la meme hauteur- remplissage par zéro – zero padding – ajout d’une marge (seul la marge est remplie par des zéros).

Filtre/masque/noyau de convolution :

Les poids d’un neurone peuvent etre représentés sou la forme d’une petite image de la taille du champ récepteur.

Appliquer le filtre H/V à une image => elle fera ressortir les zones d’une image qui se rapprochent le plus du filtre => feature map.

Au cours de l’entrainement, un CNN détermine les filtres qui seront les plus utiles à sa tache et apprend à les combiner dans des motifs plus complexes. Exemple une croix est une zone de l’image ou le filtre V et H sont tous deux actifs.

Une couche de convolution applique simultanément plusieurs filtres à ses entrées, ce qui lui permet de détecter plusieurs caractéristiques n’importe où dans ses entrées.

Une couche convolution est constituée de plusieurs cartes de caractéristiques de taille égale => 3D.

Tous les neurones partagent les memes paramètres, poids et terme constant, le nombre de paramètre s’en trouve réduit, mais les différentes cartes de caractéristiques peuvent avoir des paramètres distincts. Le CNN a appris à reconnaitre un motif en un endroit, il peut le reconnaitre partout ailleurs (contrairement à un RNP, il reconnait un motif qu’en cet endroit précis).

Les images d’entrées sont constituées de multiples sous-couches (canal de couleur RGB -3, les images en gris en possède qu’un seul).

La couche de convolution fait intervenir les hyperparamètres suivants :

Nombre de filtre leur hauteur et largeur.

Le pas et le type de remplissage (c.-à-d. avec ou sans marge de zéros).

On peut utiliser la validation croisée mais cela prend beaucoup de temps, il existe des architectures de CNN afin d’avoir une idée des valeurs des paramètres.

Les couches de convolution ont besoin d’une grande quantité de RAM, notamment au cours de l’entrainement (conserver les calculs effectués pendant la passe en avant pour la passe en arrière). Exemple : filtre 5\*5, 200 cartes de caractéristiques de taille 150\*100, un pas de 1 avec une marge de zéro et les images d’entrées sont en RGB de taille 150\*100.

Le nombre de paramètres = 5\*5(filtre)\*3(RGB)\*1(intercept)\*200(feature map) = 15200 paramètres.

Chacune des 200 features map contient 150\*100 neurones, chaque neurone calcule une somme pondérée de 5\*5\*3 = 75 entrées => 225 millions d’opérations, si flottant = 32 bites => 200\*150\*100\*32 =96 millions de bits~11,4 Mo cela ne concerne qu’une seule instance ! si on a 100 instances ~1GO de RAM (page 189).

Si pas assez de Ram :

* Supprimer une ou plusieurs couches.
* Distribuer le CNN sur plusieurs processeurs.
* Utiliser des nombres à virgule flottante sur 16 bits à la place de 32 bits.
* Réduire la taille du mini-lot.
* Diminuer la dimensionnalité en utilisant un pas plus grand dans une ou plusieurs couches.

Toutes les couches de convolution sont suivies par des fonctions d’activation (exemple Relu).

Erreur : utiliser des noyaux de convolution très grand, exemple 9\*9, = deux couches de convolution avec deux noyaux de convolution 3\*3 (avec moins de calcul et de paramètres).

Couche Pooling :

A pour objectif de sous-échantillonner (rétrécir) l’image d’entrée afin de réduire la charge des calculs (réduire l’utilisation de la RAM des paramètres (limitant le risque de sur-ajustement)).

Comme pour la couche de convolution chaque neurone est connecté aux sorties d’un nombre limité de neurones de la couche précédente. On doit à nouveau définir sa taille, le pas et le type de remplissage. En revanche, un neurone de pooling ne possède aucun poids et se contente d’agréger les entrées en utilisant une fonction d’agrégation (max (seuls les pixels> seuil max passe ignorer sinon, moyenne …)

C’est une couche destructrice (image plus petite, perte d’information, peu de sortie)

En raison des pas de couches de convolution et de pooling, les cartes de caractéristiques rétrécissent au fur et à mesure que l’on traverse le réseau.

La couche finale produit la prédiction (une couche soft max qui génère des probabilités de classe estimés).

Avantage des couches de pooling par rapport aux couches de convolution : pas de paramètres.

Auto encodeur

Les auto-encodeurs sont capables **d’apprendre des représentations efficaces des données d’entrée**, appelées codages, sans aucune supervision. Ces codages ont généralement une dimensionnalité plus faible que les données d’entrée (c.à.d. mois de caractéristiques).

* Ils sont des détecteurs de caractéristiques puissants et peuvent etre utilisés pour le pré-entrainement non supervisé des RNP.
* Ils sont capables de générer de nouvelles données originales qui ressemblent énormément aux données d’entrainement ; 🡺 modèle génératif.

**Ils opèrent** en apprenant simplement à **copier leurs entrées vers leurs sorties**, cette tache est non triviale car des **contraintes sont imposées** ce qui **les obligent à découvrir des méthodes efficaces de représentation des données (découvrir et à exploiter les motifs présents dans les données).**

Analogie : Un joueur d’échecs professionnel, mémorise l’échiquiers en 5s car il reconnait des motifs de placement plus facilement en raison de leur expérience, et donc stocke l’information plus efficacement.

**Codage** : sous-produit issu des tentatives de l’auto-encodeur d’apprendre la fonction identité sous certaines contraintes.

Un auto-encodeur examine les entrées, les convertit en une représentation interne efficace produit en sortie quelque chose qui l’on espère ressemble à l’entrée.

Il est composé de deux étapes :

Suivit

* Il possède la meme architecture que le PMC, mais le nombre de neurones de ma couche de sorties doit etre égal au nombre d’entrées.
* Les sorties sont appelées reconstructions, car l’encodeur tente de reconstruire les entrées.
* La fonction de cout inclut une perte de reconstruction qui pénalise le modèle lorsque les reconstructions diffèrent des entrées.
* Puisque la représentation interne est inférieure à celle des données d’entrée, on qualifie l’auto-encodeur de sous complet. Il ne peut pas copier simplement les entrées, il est obligé d’apprendre les caractéristiques les plus importantes des données d’entrée (et d’ignorer les moins importantes).
* Un auto-encodeur empilé/profond, possède plusieurs couches cachées, ce qui lui permet d’apprendre des codages plus complexes.

ACP avec un auto-encodeur linéaire :

Si l’auto-encodeur utilise uniquement des activations linéaires (pas de fonctions du tout) et si la fonction de cout est MSE, alors on peut montrer qu’il réalise une analyse en composante principale.

Visualiser des caractéristiques :

Méthode simple : Trouver les instances d’entrainement qui activent le plus le neurone. Cette méthode fonctionne bien pour les couches cachées supérieurs (car elle capture des caractéristiques relativement larges). En revanche, elle ne convient pas au couches inférieures (car les caractéristiques sont plus petites et plus abstraites, donc difficile de comprendre ce qui excite le neurone).

Méthode 2 : afficher les poids de chaque neurone par couche.

Méthode 3 : donner en entrée une image aléatoire puis mesurer l’activation du neurone qui nous intéresse, puis à effectuer une rétropropagation afin d’ajuster l’image de sorte que l’activation du neurone soit encore plus importante (montée de gradient). Méthode utilisée pour visualiser les types d’entrées recherchés par un neurone.

Si l’auto-encodeur sert à une classification, une solution simple de vérifier l’intérêt des caractéristiques apprises, consiste à mesurer la performance du classifieur.

Auto-encodeurs débruteurs :

Pour obliger un auto-encodeur à apprendre des caractéristiques utiles, une autre approche consiste à rajouter du bruit (bruit gaussien) sur ses entrées et à l’entrainer pour qu’il retrouve les entrées d’origine, sans le bruit (l’idée date des années 1990 – mentionner dans l’article de Yan LeCun en 1987).

Auto-encodeur épais :

La dispersion est un autre type de contrainte qui conduit souvent à une bonne extraction de caractéristiques. En ajoutant le terme approprié à la fonction de cout, cela le pousse à réduire le nombre de neurones actifs dans la couche de codage (=> l’incite à representer chaque entrée comme une combinaison d’un petit nombre d’activations, chaque neurone finit par représenter une caractéristique utile).

On calcule l’activation moyenne de chaque neurone de la couche de codage, puis on pénalise ceux qui sont trop actifs en ajoutant à la fonction de cout une perte de dispersion (sparsy Loss).

Approche 1 : ajouter l’erreur quadratique (moy neurone – dispersion cible) \*\*2.

Approche 2 : divergence de Kullback-Leibler, en effet, elle a des gradients plus forts que l’erreur quadratique. On veut mesurer la divergence entre la probabilité cible p qu’un neurone de la couche de codage s’activera et la probabilité réelle q (c.à.d. l’activation moyenne sur le lot d’entrainement).

Le dilemme biais/variance exprime l’équilibre à trouver entre la taille de l’ensemble d’apprentissage, la complexité du modèle et de la barre d’erreur sur l’estimation de l’erreur théorique obtenue par l’erreur empirique.