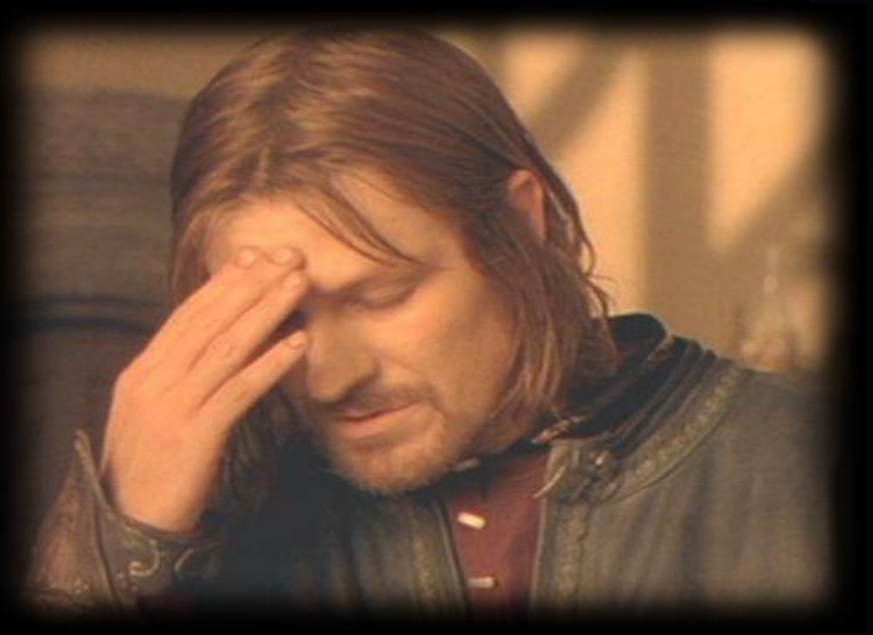
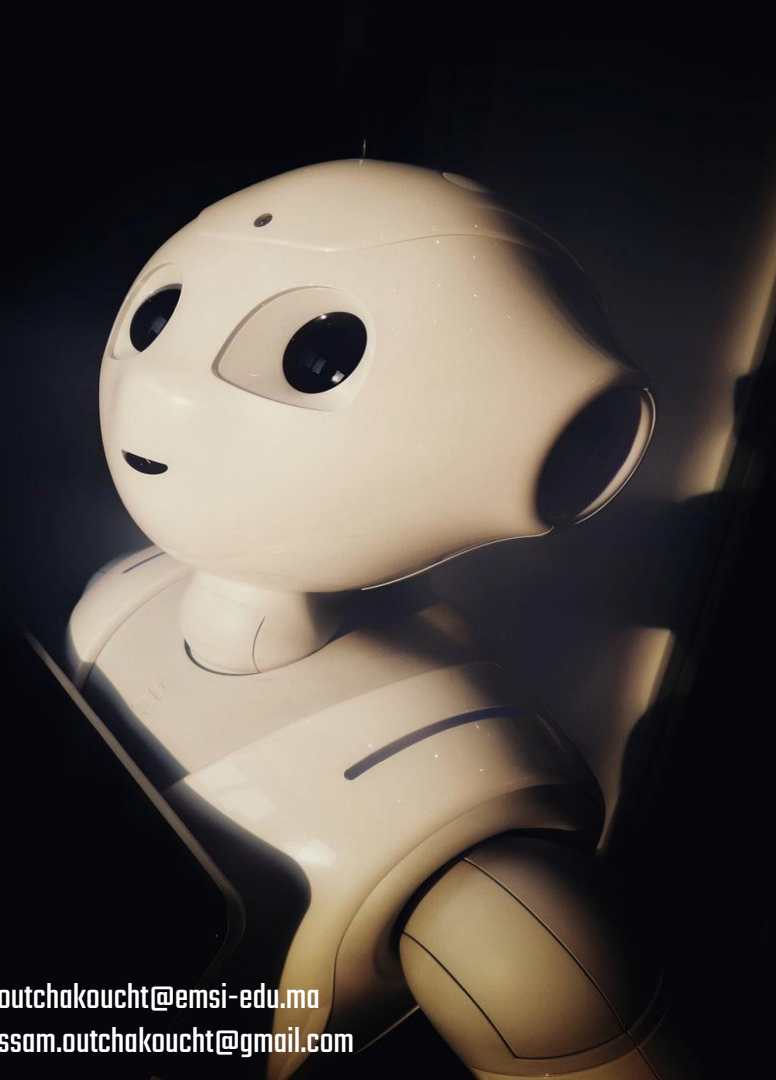
ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

DEEP LEARNING: DÉFIS

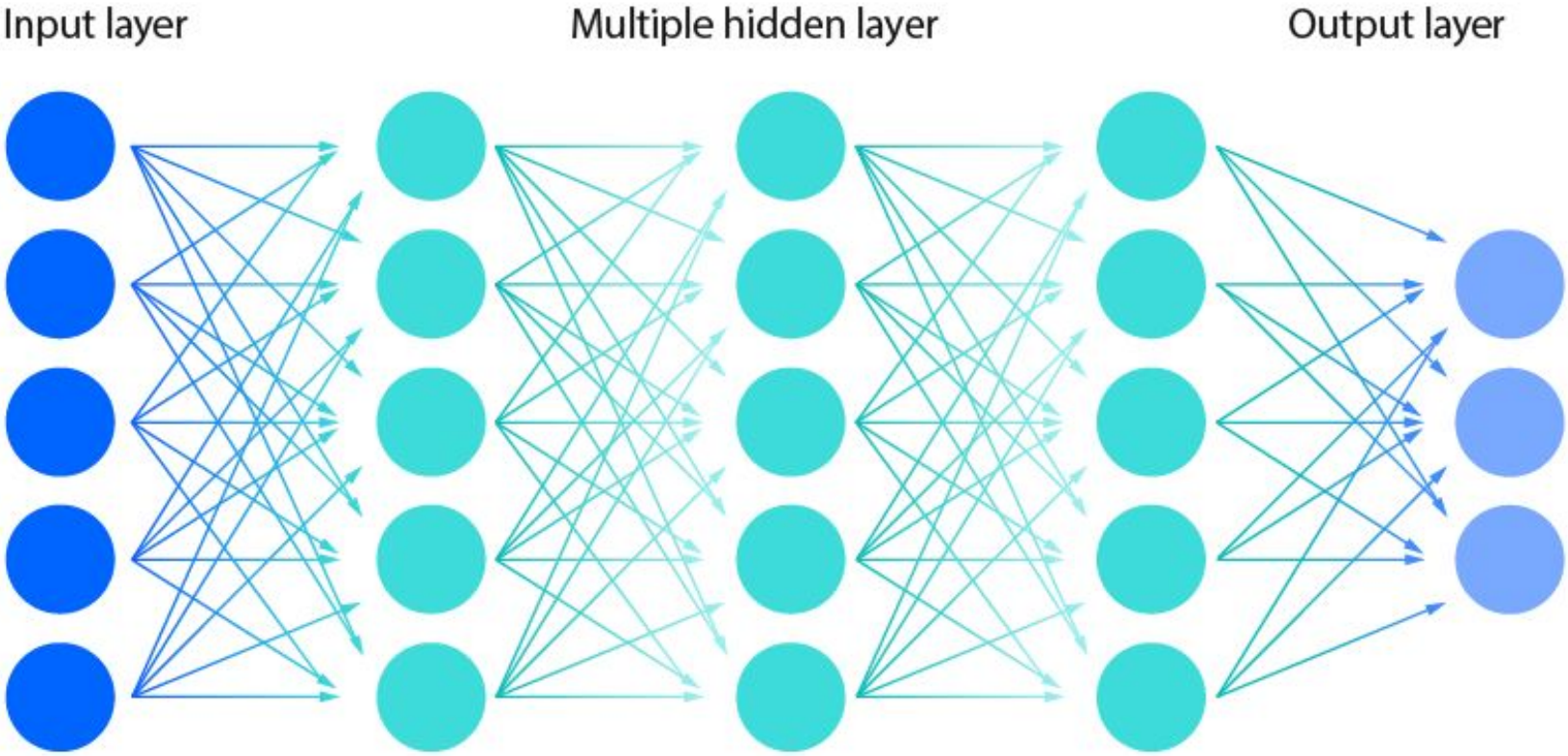




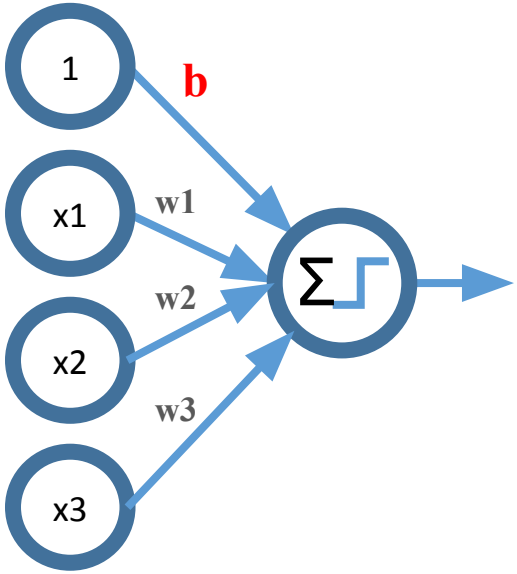
Aissam Outchakoucht

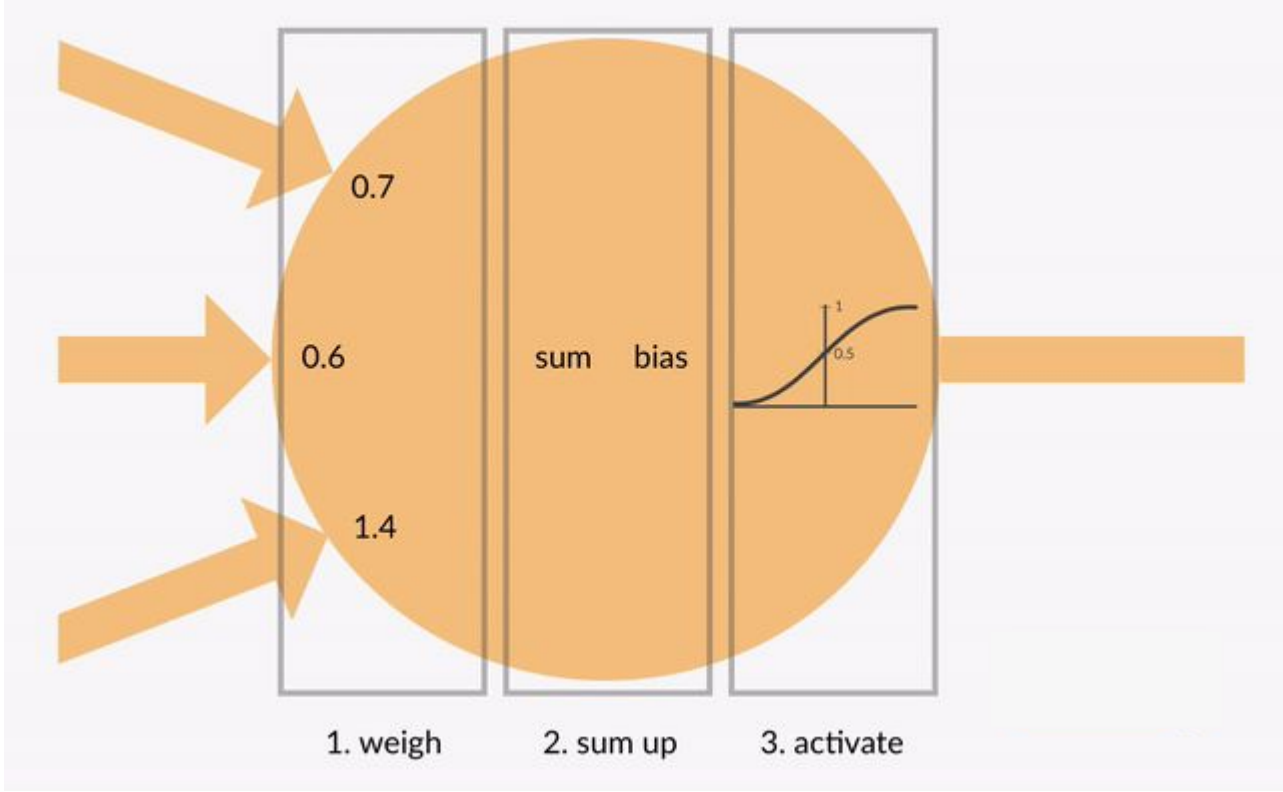
a.outchakoucht@emsi-edu.ma
aissam.outchakoucht@gmail.com

RECAP



RECAP

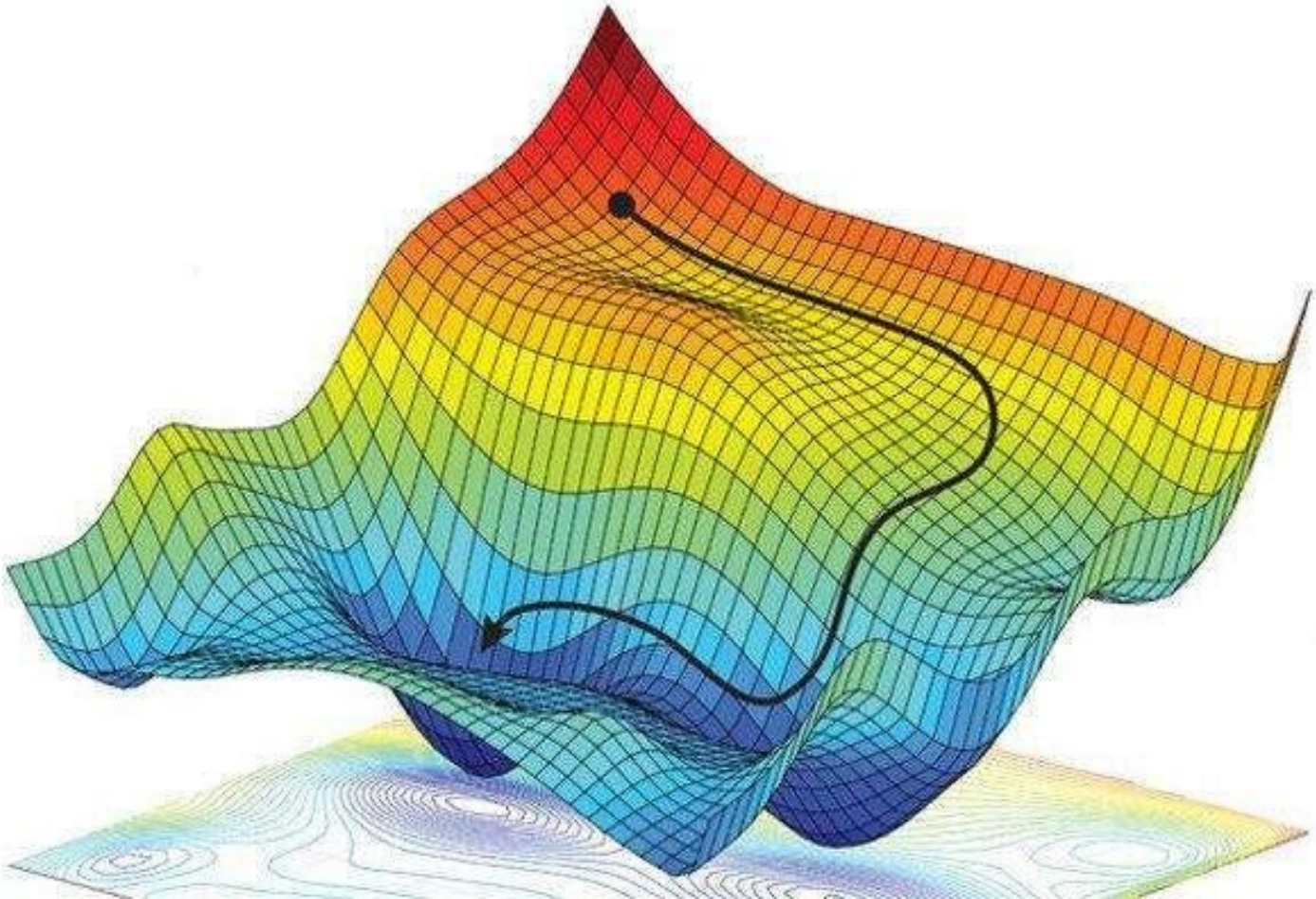




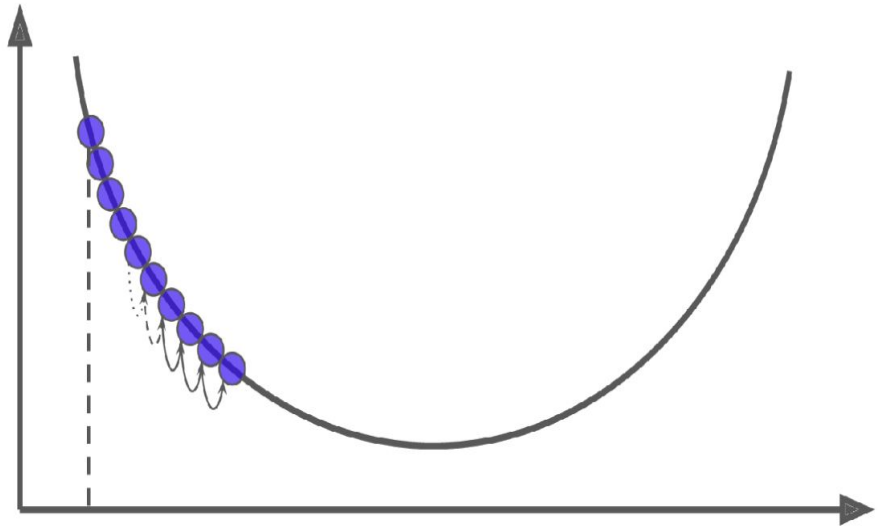
$F(x) = \sigma(\ W\ .X + b\ )$

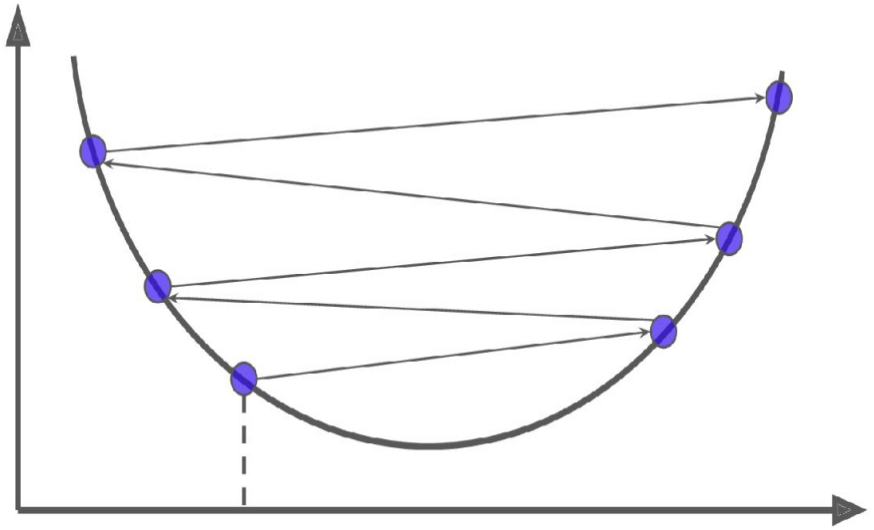
RECAP, GRADIENT DESCENT

$$w = w - \alpha \nabla_w C(w)$$



RECAP, LEARNING RATE





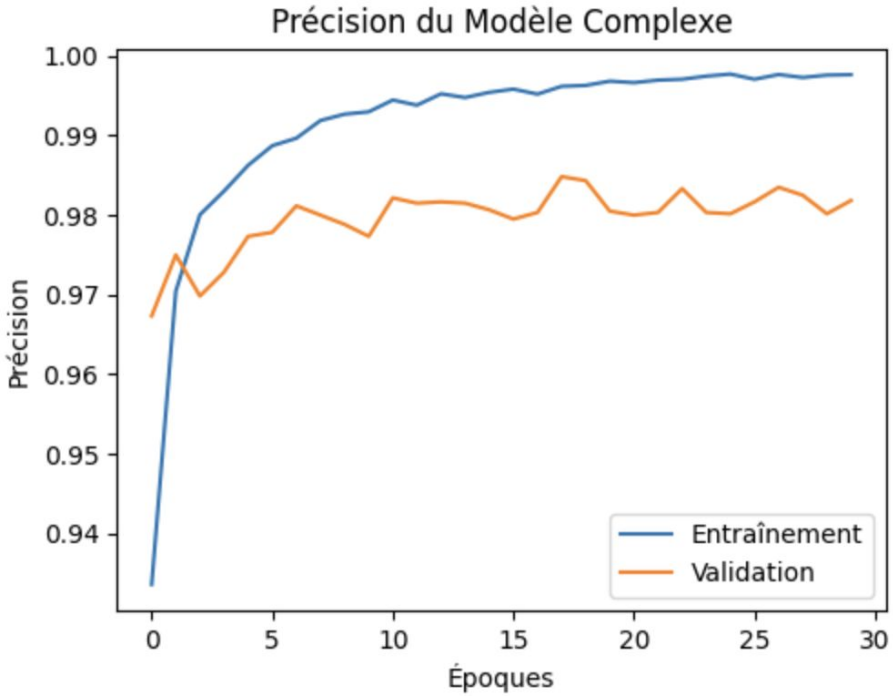
RECAP.py

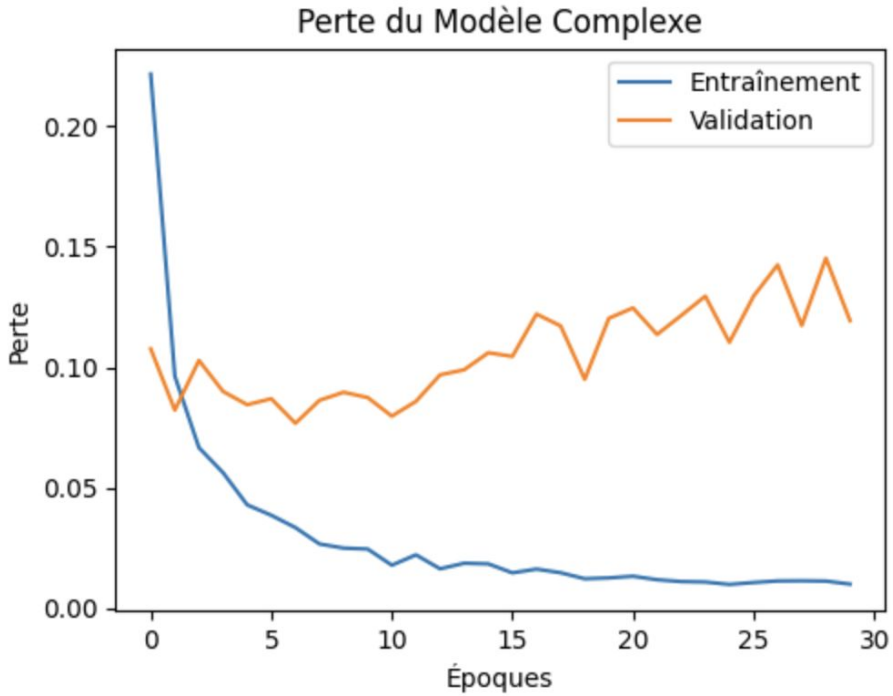
```
[ ] # création du model
    model = Sequential([
        Flatten(input_shape=(28,28)),
        Dense(512, activation='relu'),
        Dense(256, activation='relu'),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])

    # Compilation du modèle
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[ ] history = model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.1)
```

```
Epoch 1/30
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.8839 - loss: 0.3845 - val_accuracy: 0.9673 - val_loss: 0.1076
Epoch 2/30
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 20s 10ms/step - accuracy: 0.9699 - loss: 0.0958 - val_accuracy: 0.9750 - val_loss: 0.0822
Epoch 3/30
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.9802 - loss: 0.0663 - val_accuracy: 0.9698 - val_loss: 0.1028
Epoch 4/30
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.9836 - loss: 0.0528 - val_accuracy: 0.9728 - val_loss: 0.0899
Epoch 5/30
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.9873 - loss: 0.0396 - val_accuracy: 0.9773 - val_loss: 0.0844
Epoch 6/30
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - accuracy: 0.9882 - loss: 0.0400 - val_accuracy: 0.9778 - val_loss: 0.0869
```

RECAP.py


Précision du Modèle Complexe
Précision
Époques
Entraînement
Validation
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0
5
10
15
20
25
30


Perte du Modèle Complexe
Perte
Époques
Entraînement
Validation
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30

HYPER PARAMETRES

Nom	Définition
Learning Rate (α)	Contrôle la taille des pas pour mettre à jour les poids du modèle.
Batch Size	Nombre d'exemples utilisés dans chaque mise à jour du gradient.
Number of Epochs	Nombre de passages complets sur l'ensemble des données d'entraînement.
# couches/neurones	Nombre de couches cachées et de neurones par couche dans un réseau neuronal
Dropout Rate	Fraction des neurones désactivés aléatoirement pour chaque itération (dans les réseaux neuronaux).
Momentum	Accélère le gradient descent en dirigeant le pas selon les gradients précédents.
Weight Initialization	Méthode pour initialiser les poids (e.g., Xavier, He) pour améliorer la convergence.
Activation Function	Fonction utilisée dans les neurones pour introduire la non-linéarité (e.g., ReLU, Sigmoid).
Optimizer	Algorithme pour mettre à jour les poids (e.g., SGD, Adam, RMSprop).
Learning Rate Decay	Réduction progressive du taux d'apprentissage pour stabiliser la convergence.

Nombre de couches cachées -Hidden layers-

Pour de nombreux problèmes, vous pouvez commencer avec une seule couche cachée et obtenir des résultats raisonnables. Un NN avec une seule couche cachée peut théoriquement modéliser même les fonctions les plus complexes, à condition qu'il ait suffisamment de neurones. Mais pour les problèmes complexes, les réseaux profonds ont une efficacité de paramétrage beaucoup plus élevée que les réseaux peu profonds : ils peuvent modéliser des fonctions complexes en utilisant exponentiellement moins de neurones que les réseaux peu profonds, ce qui leur permet d'atteindre de meilleures performances avec la même quantité de données d'entraînement.

Nombre de couches cachées -Hidden layers-

En résumé, pour de nombreux problèmes, vous pouvez commencer avec seulement une ou deux couches cachées et le réseau neuronal fonctionnera parfaitement.

Pour les problèmes plus complexes, vous pouvez augmenter le nombre de couches jusqu'à ce que vous commenciez le surapprentissage (overfitting) des données d'entraînement.

Les tâches très complexes (grandes images, reconnaissance vocale) nécessitent généralement des réseaux comportant des dizaines de couches (voire des centaines, mais pas entièrement connectées -CNN-).

Rarement on forme de tels réseaux à partir de zéro : il est beaucoup plus courant de réutiliser des parties d'un réseau pré-entraîné qui effectuent une tâche similaire.

Nombre de neurones par couche

Le nombre de neurones dans les couches d'entrée et de sortie est déterminé par le type d'entrée et de sortie de la tâche en question.

Par exemple, le classement des images MNIST nécessite 28 × 28 = 784 neurones d'entrée et 10 neurones de sortie.

En ce qui concerne les couches cachées, il était courant de les dimensionner pour former une pyramide, avec de moins en moins de neurones à chaque couche - la raison étant que de nombreuses caractéristiques de bas niveau peuvent se fondre en beaucoup moins de caractéristiques de haut niveau. Cependant, cette pratique a été largement abandonnée car il semble que l'utilisation du même nombre de neurones dans toutes les couches cachées donne des résultats tout aussi bons dans la plupart des cas, voire meilleurs ; De plus (et c'est important), il n'y a qu'un seul hyperparamètre à régler, au lieu d'un par couche.

Nombre de neurones par couche

Tout comme le nombre de couches, on pourrait essayer d'augmenter progressivement le nombre de neurones jusqu'à ce que le réseau commence à surinterpréter (overfitting).

Mais en pratique, il est souvent plus simple et plus efficace de choisir un modèle comportant plus de couches et de neurones que ce dont on a réellement besoin, puis d'utiliser l'arrêt précoce (early stopping), l'abandon des neurones (dropout) et d'autres techniques de régularisation pour éviter la surinterprétation.

Taille des lots/ensembles (Batch size)

Yann LeCun
@ylecun

Training with large minibatches is bad for your health.
More importantly, it's bad for your test error.
Friends dont let friends use minibatches larger than 32.
arxiv.org/abs/1804.07612

2:00 PM · Apr 26, 2018 · Facebook

Le principal avantage de l'utilisation de lots de grande taille est que les accélérateurs matériels comme les GPUs peuvent les traiter efficacement, de sorte que l'algorithme d'entraînement verra plus d'instances par seconde.

En pratique, les grandes tailles de lot entraînent souvent des instabilités, surtout au début de l'entraînement, et le modèle qui en résulte peut ne pas se généraliser aussi bien qu'un modèle formé avec une petite taille de lot.

Une stratégie consiste donc à essayer d'utiliser une grande taille de lot, et si l'entraînement est instable ou si la performance finale est décevante, alors essayez d'utiliser une petite taille de lot à la place.

sklearn.model_selection.GridSearchCV

GridSearch est utilisée pour trouver les hyper paramètres optimaux d'un modèle.

```
param_grid = {
        'epochs' :          [100, 150, 200],
        'batch_size' :      [32, 128],
        'optimizer' :       ['Adam', 'Nadam'],
        'dropout_rate' :    [0.2, 0.3],
        'activation' :      ['relu', 'elu']
}

gs = GridSearchCV(
          estimator = model,
          param_grid = param_grid
)
```

VANISHING/EXPLODING GRADIENTS

Vanishing Gradients : Les gradients deviennent très petits, ce qui bloque l'apprentissage des couches profondes.

- Résultat : Les poids ne changent presque pas, les couches proches de l'entrée sont sous-entraînées.

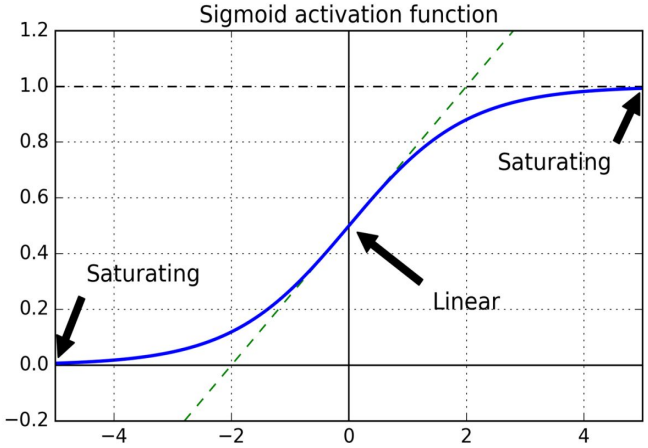
Exploding Gradients : Les gradients deviennent très grands, causant des mises à jour instables.

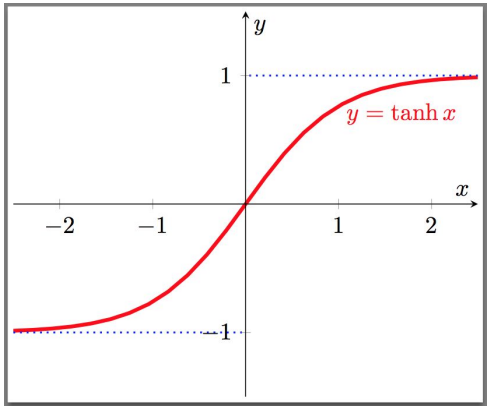
- Résultat : La perte diverge, et les poids deviennent incohérents.

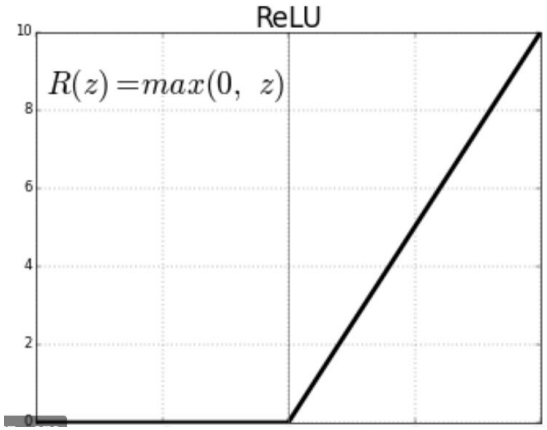
Pourquoi cela arrive-t-il ?

- Propagation des gradients : Lors de la rétropropagation, le gradient est multiplié à chaque couche.
- Profondeur du réseau : Plus le réseau est profond, plus ces effets sont amplifiés.
- Fonctions d'activation non adaptées ou initialisation des poids inadéquate.

ACTIVATION







ReLU:

- Avantage : Pas de saturation pour les entrées positives.
- Inconvénient : Problème des neurones morts (sortie toujours 0 pour certains).

Leaky ReLU :

- $f(x)=x$ si $x>0$ $f(x)=\alpha x$ sinon (α petit).
- Résout le problème des neurones morts.

INITIALISATION

Il est important d'initialiser les poids de connexion de toutes les couches cachées de façon aléatoire, sinon l'entraînement échouera. Par exemple, si on initialise tous les poids et biais à zéro, tous les neurones d'une couche donnée seront parfaitement identiques, et la rétropropagation les affectera donc exactement de la même manière, de sorte qu'ils resteront identiques. En d'autres termes, bien qu'il ait des centaines de neurones par couche, le modèle agira comme s'il n'avait qu'un seul neurone par couche.

Si, au contraire, on initialise les poids de manière aléatoire, on brise la symétrie et permet à la rétropropropagation de former une collection diversifiée de neurones

Une mauvaise initialisation peut amplifier ou réduire les activations des couches successives, provoquant des vanishing ou exploding gradients, rendant l'apprentissage difficile.

INITIALISATION

Xavier Initialization:

- Initialise les poids pour que les activations soient équilibrées entre les couches.
- Évite l'amplification ou l'atténuation des gradients dans les réseaux utilisant des activations symétriques.
- Fonctionne bien avec *tanh* et *sigmoid*.

He Initialization:

- Ajuste les poids pour maintenir la propagation stable dans les réseaux profonds.
- Conçu pour gérer la non-linéarité des activations de type ReLU et éviter les neurones morts.
- Idéal pour ReLU, Leaky ReLU, et variantes.

BATCH NORMALIZATION

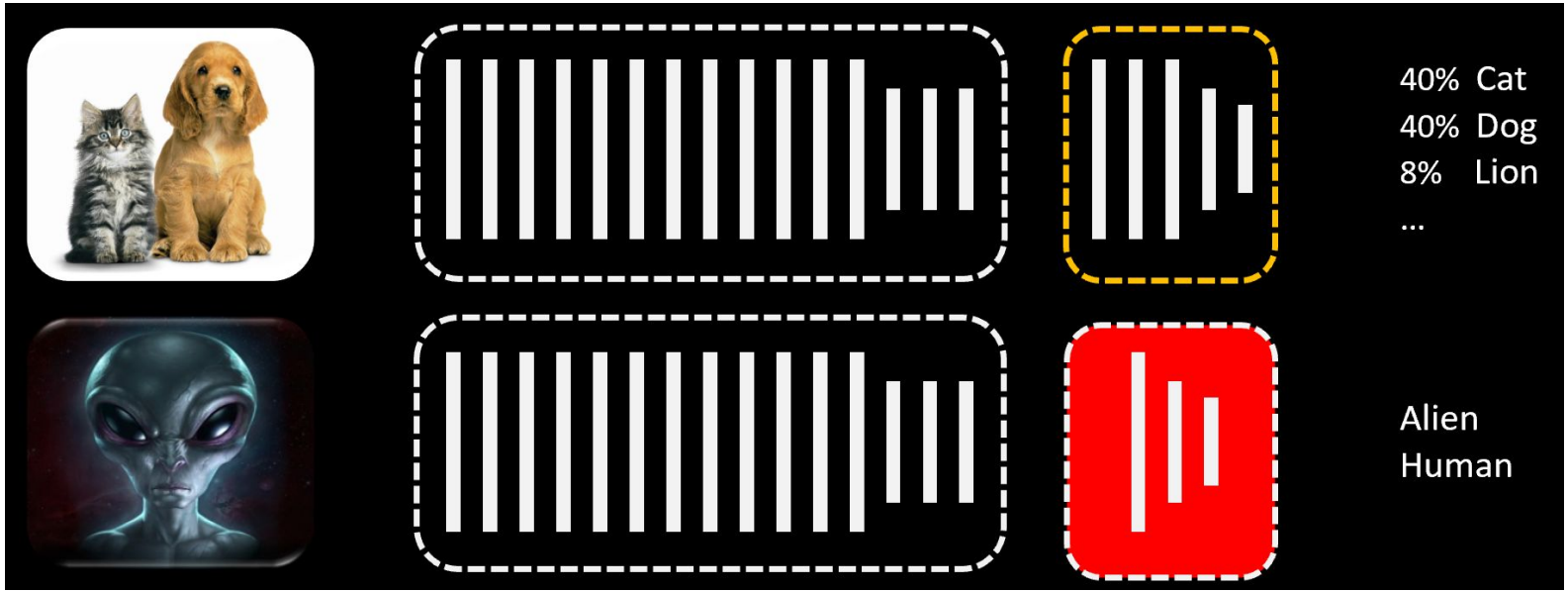
Bien que l'utilisation de l'initialisation He avec ELU (ou toute variante de ReLU) puisse réduire considérablement le danger de disparition/explosion des gradients au début de l'entraînement, elle ne garantit pas qu'ils ne reviendront pas pendant l'entraînement.

Si la couche d'entrée bénéficie de l'ajustement des entrées (features scaling), pourquoi ne pas faire la même chose pour les valeurs des couches cachées, qui changent tout le temps.

On peut considérer la normalisation par lots (batch normalization) comme un prétraitement à chaque couche du réseau.

Cependant chaque époque prendra maintenant beaucoup plus de temps. Ceci est généralement contrebalancé par le fait que la convergence est beaucoup plus rapide avec la BN.

TRANSFER LEARNING



TRANSFER LEARNING

```python
# Charger le modèle pré-entraîné VGG16 sans la dernière couche (include_top=False)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

# Geler les couches du modèle (elles ne seront pas réentraînées)
for layer in base_model.layers:
    layer.trainable = False

# Ajouter des couches personnalisées pour notre dataset
x = Flatten()(base_model.output)                    # Aplatir les caractéristiques extraites par VGG16
x = Dense(128, activation='relu')(x)                # Ajouter une couche dense avec 128 neurones
x = Dropout(0.5)(x)                                 # Ajouter un Dropout pour éviter le surapprentissage
output = Dense(3, activation='softmax')(x)          # Sortie avec 3 classes (softmax)

# Créer le modèle final
model = Model(inputs=base_model.input, outputs=output)

# Compiler le modèle
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```