Détection de mélanome de la peau

Présentation TIPE réalisée par : ORKHIS Walid

Numéro d'inscription SCEI : 21072

Sommaire

1. Introduction
2. Réseaux de neurones artificiels.
   - Principe
   - Perceptron
   - Sigmoid
3. Apprentissage du réseaux de neurones.
   - Principe
   - Descente du gradient
   - Retropropagation
4. Réseaux de neurones convolutifs
   - architecture
   - Convolution
   - Pooling
   - Test du réseau pour mon etude de melanome
5. Conclusion

Introduction

Mélanome

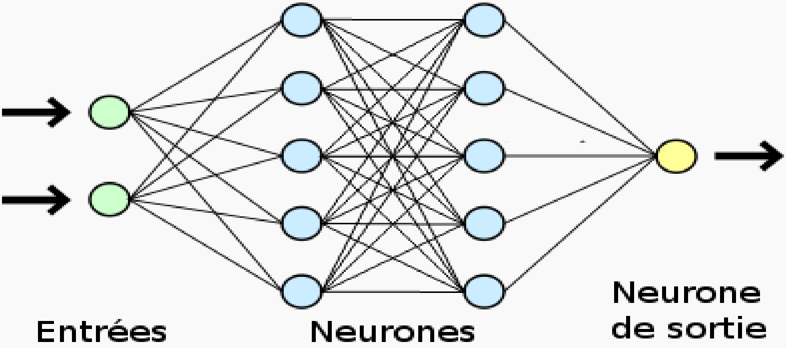
Figure 1 : **grain de beauté**

Figure 2 : **mélanome de la peau**

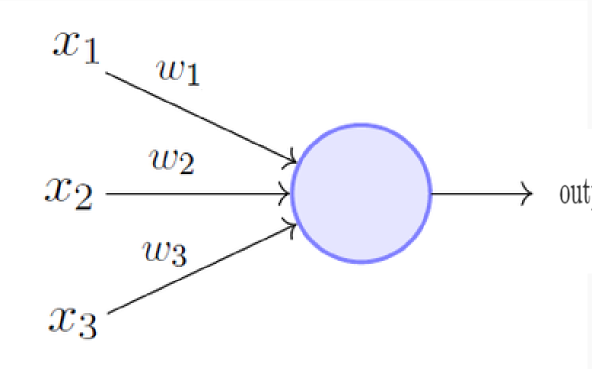
Réseaux de neurones artificiels

Qu'est ce qu'un réseau de neurones artificiel

- Le Deep learning ou apprentissage profond est l'une des technologies principales du Machine learning. Avec le Deep Learning, nous parlons d'algorithmes capables de mimer les actions du cerveau humain grâce à des **réseaux de neurones artificielles**.



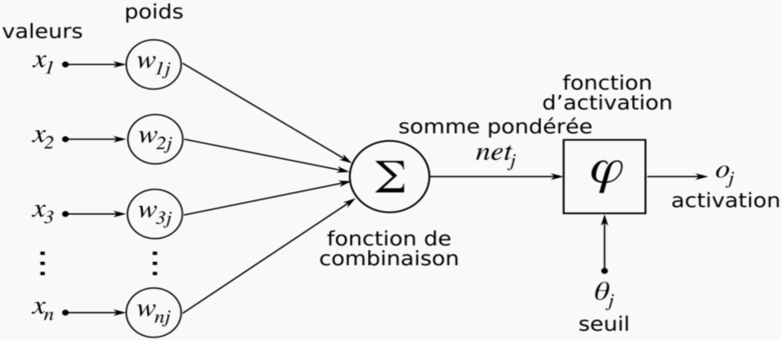
Perceptron

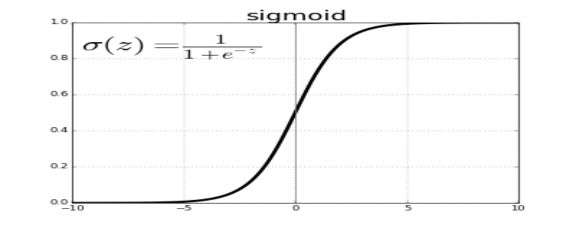


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Figure 3 : **Perceptron**

Sigmoid


poids
valeurs
x_1
w_{1j}
x_2
w_{2j}
x_3
w_{3j}
x_n
w_{nj}
Σ
fonction de
combinaison
somme pondérée
net_j
fonction
d'activation
φ
o_j
activation
θ_j
seuil


sigmoid
$\sigma(z) = \frac{1}{1+e^{-z}}$
1.0
0.8
0.6
0.4
0.2
0.0
−10
−5
0
5
10

Apprentissage du réseaux de neurones.

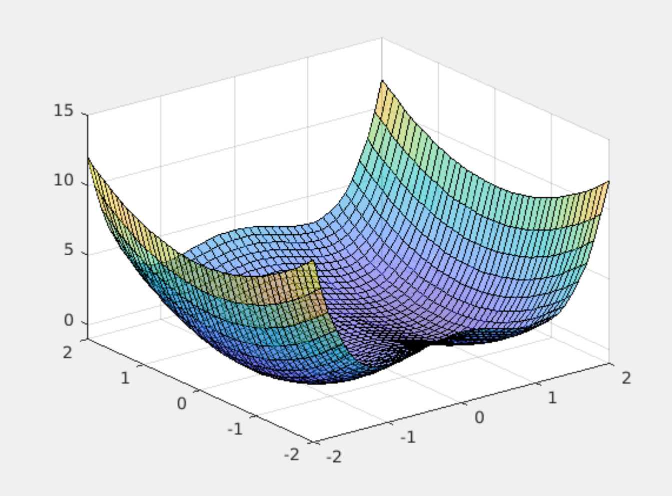
Principe

- Pour quantifier dans quelle mesure nous atteignons l'apprentissage du réseaux, nous définissons une fonction de coût:

$$\boldsymbol{C(w,b) \equiv (1/2n)\sum_x \parallel y(x) - a \parallel^2}$$

- Notre algorithme d'entraînement a fait du bon travail s'il peut trouver des poids et des biais tels que : $\boldsymbol{C(w,b) \approx 0}$

Descente du gradient



Descente du gradient

- Réfléchissons à ce qui se passe lorsque nous déplaçons d'une petite quantité Δv1 dans la direction v1, et d'une petite quantité Δv2 dans la direction v2. le Calcul nous dit que C change comme suit :

$$\Delta \boldsymbol{C} \approx \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v1}} \Delta \boldsymbol{v1} + \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v2}} \Delta \boldsymbol{v2}$$

- Nous définirons alors le gradient de C: $\boldsymbol{\nabla C} = (\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v1}}, \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{v2}})$
- On suppose on choisit : $\Delta \boldsymbol{v} = -\boldsymbol{\eta \nabla C}$
- Donc : $\Delta \boldsymbol{C} \approx -\boldsymbol{\eta \nabla C . \nabla C} = -\boldsymbol{\eta} \|\boldsymbol{C}\|^{2}$

  qui est toujours negative car $\|\boldsymbol{C}\| \geq \boldsymbol{0}\ \boldsymbol{et}\ \boldsymbol{\eta} > \boldsymbol{0}$
- Et on utilise à nouveau cette règle de mise à jour pour effectuer un autre mouvement : $\boldsymbol{v} \rightarrow \boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{\eta \nabla C}$

Retropropagation

- La rétropropagation consiste à comprendre comment la modification des pondérations et des biais dans un réseau modifie la fonction de coût.
- La rétropropagation est basée sur quatre équations fondamentales :
  - $\boldsymbol{\delta^L = \Delta_a C \odot \sigma'\left(z^L\right)}$
  - $\boldsymbol{\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)},$
  - $\boldsymbol{\frac{\partial C}{\partial b_j^l} = \delta_j^l},$
  - $\boldsymbol{\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l}.$ (preuve en annexe)

Retropropagation

- Les équations de rétropropagation nous fournissent un moyen de calculer le gradient de la fonction de coût. Et on peut Écrire cela explicitement sous la forme d'un algorithme :
  - Entrée x : définissez l'activation correspondante a_1 pour la couche d'entrée.
  - Anticipation : Pour chaque l=2,3,…,L calculer
    $$\boldsymbol{z^l = w^l a^{l-1} + b^l \; et \; a^l = \sigma(z^l)}.$$
  - Erreur de sortie δ_L : Calculer le vecteur
    $$\boldsymbol{\delta^L = \Delta_a C \odot \sigma'(z^L)}$$
  - Rétropropager l'erreur : Pour chaque l=L−1,L−2,…,2 calculer : $\boldsymbol{\delta^l = \left((w^{l+1})^T \delta^{l+1}\right) \odot \sigma'(z^l)}$,
  - Sortie :Le gradient de la fonction de coût est donné par
    $$\boldsymbol{\frac{\partial C}{\partial b_j^l} = \delta_j^l, \;\; \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l}.$$

Réseaux de neurones convolutifs

Architecture du réseaux de neurones convolutifs

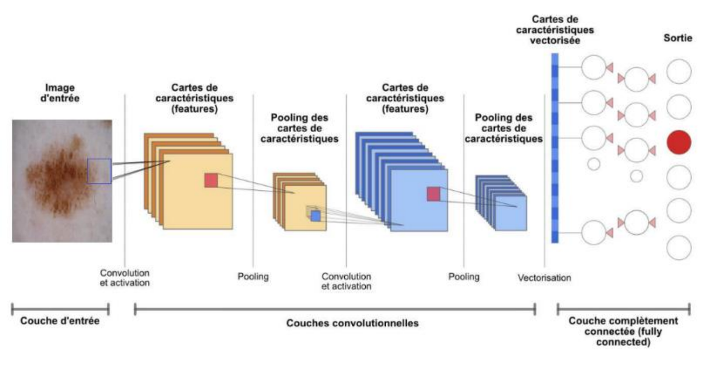


Figure : **architecture des couches du RNC**

Y aller **Implémentation en Python**

Convolution

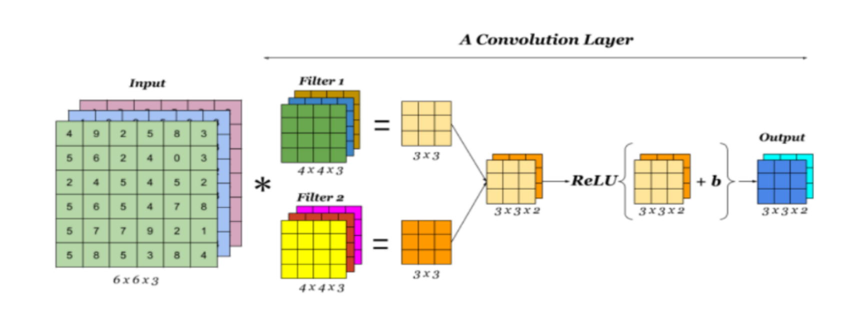


Figure 9 : **convolution**

Y aller **Implémentation en Python**

Pooling

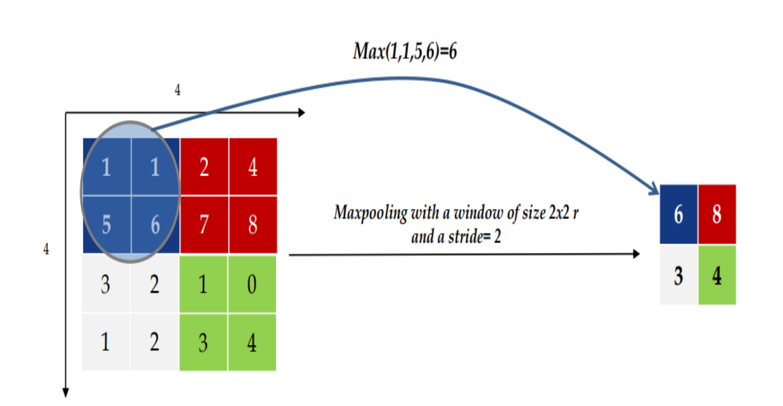


Figure 10 : **pooling**

Y aller **Implémentation en Python**

Apprentissage du Réseaux de neurones convolutifs

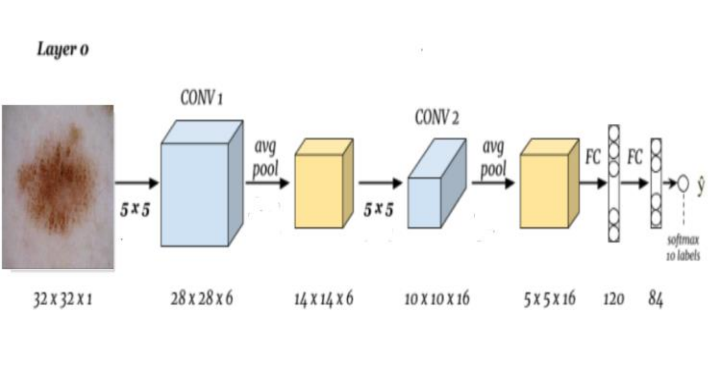
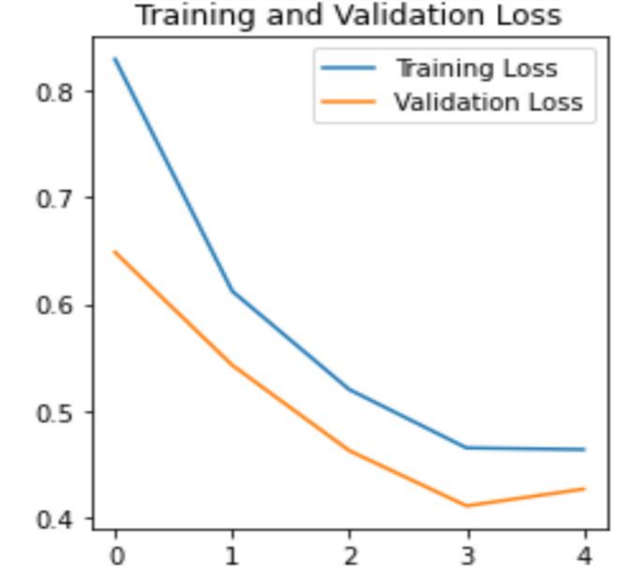


Figure 11 : **Apprentissage du Réseaux de neuronnes convolutifs**

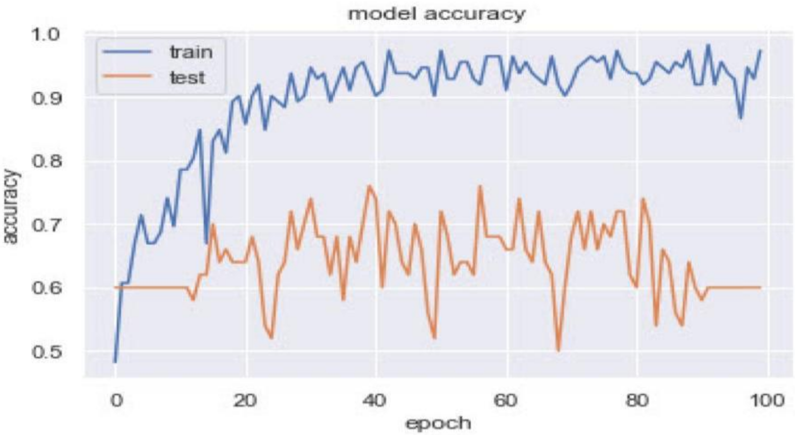
Y aller **Implémentation en Python**

Changement la fonction de coût pendant l'apprentissage



Évolution de la précision pendant l'apprentissage

```
Validation loss: 67.61959979408665
Validation accuracy: 0.31578946
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



Prédire la catégorie des images connues:

Après avoir demandé au dermatologue KHALLAA youness mehdi de m'envoyer un exemple d'un de ses patients atteints de mélanome, il m'a envoyé l'exemple.

Demande d'aide dans une présentation Boîte de réception ×

walid orkhis <orkhiswalid@gmail.com> mar. 24 mai 17:26
À khallaayounemehdi

Bonjour docteur,
Je m'appelle walid orkhis je suis un étudiant en classes préparatoires aux grandes écoles. A la fin de nos 2 années au CPGE nous avons un travail à faire sous le nom de tipe (TRAVAUX D'INITIATIVE PERSONNELLE ENCADRES) en un thème précis, le thème de cette année est santé, prévention. Alors, la détection de mélanome a l'aide d'un programme informatique avec l'apprentissage profond (deep learning) va être mon travail. Je veux votre aide si possible, pour tester ce programme je veux que vous me fournissiez des photos réelles de ce cancer de quelqu'un de vos patients qui a eu ou probablement eu ce type de cancer pour donner à mon travail un peu de réalisme.
Cordialement.

Mehdi Khallaayoune sam. 4 juin 17:30 (il y a 3 jours)
À moi

IMG-20200708-WA0011.jpg
IMG-20200708-WA0012.jpg
Bonjour Walid,

Toutes mes excuses pour le retard, je viens de me rendre compte que mon email n'a pas été envoyé. Tout d'abord, je te félicite pour la pertinence du choix de ton sujet. L'intelligence artificielle se met de plus en plus au service du dépistage des cancers cutanés notamment en dermoscopie. Les mélanomes restent toutefois de tumeurs rares qui ne sont qu'exceptionnellement vues en consultation libérale (incidence faible au Maroc). Je t'envoie tout de même des photos d'un mélanome nodulaire à un stade avancé (malheureusement souvent le cas au Maroc) que j'ai retrouvé dans mes archives.

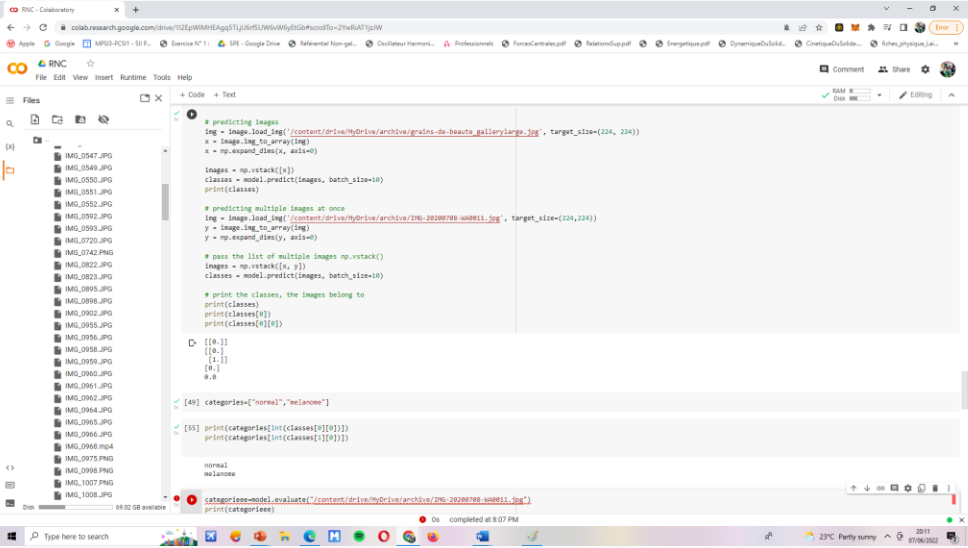
En te souhaitant bonne chance.
Bien cordialement.

photo à tester

Grain de beauté

Mélanome de la peau envoyé par le médecin flouté

Test du réseaux de neurones de convolution



Conclusion

Merci pour votre attention

Annexe

Les Scripts Python

- Importer les bibliothèques nécessaires
- Importation des données
- Visiualiser les quelques données
- Implementation de l'architecture des couches de Convolution et de pooling
- Visualisation du réseaux
- Apprentissage du RNC
- Générer des prédictions

Preuve des équations de retropropagation.

importer bibliothèque nécessaire

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers, losses
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Importer les données

```python
PATH = "/content/drive/MyDrive/archive"
train_dir = os.path.join(PATH, 'train')
test_dir = os.path.join(PATH, 'test')

train_benign_dir = os.path.join(train_dir, 'benign')
train_malignant_dir = os.path.join(train_dir, 'malignant')
test_benign_dir = os.path.join(test_dir, 'benign')
test_malignant_dir = os.path.join(test_dir, 'malignant’)
train_image_generator = ImageDataGenerator(rescale=1./255)
test_image_generator = ImageDataGenerator(rescale=1./255)

train_data_gen = train_image_generator.flow_from_directory( batch_size=128,directory=train_dir,shuffle=True,target_size=(224,224),class_mode='binary')

test_data_gen = test_image_generator.flow_from_directory(batch_size=128, directory=test_dir,target_size=(224, 224), class_mode='binary')
```

Visiualiser les quelques données

```
def plot_images(img_arr):
    fig, axes = plt.subplots(1, 5, figsize=(25, 25))
    axes = axes.flatten()
    for img, ax in zip(img_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
sample_training_images, _ = next(train_data_gen)

plot_images(sample_training_images[:5])
```

Exécution

Implementation de l’architecture des couches de Convolution et de pooling

```
model = Sequential([
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(100, activation='relu'),
    layers.Dropout(.2),
    layers.Dense(1, activation='sigmoid')
])
```

On y retourne

Visualisation du réseaux

```
model.summary()
```

Exécution:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 222, 222, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 111, 111, 16)     0
 2D)

 conv2d_4 (Conv2D)           (None, 109, 109, 64)      9280

 max_pooling2d_4 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 52, 52, 64)        36928

 max_pooling2d_5 (MaxPooling  (None, 26, 26, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 43264)             0

 dense_2 (Dense)             (None, 100)               4326500

 dropout_1 (Dropout)         (None, 100)               0

 dense_3 (Dense)             (None, 1)                 101

=================================================================
Total params: 4,373,257
Trainable params: 4,373,257
Non-trainable params: 0
_________________________________________________________________
```

Apprentissage du RNC

```
model.compile(optimizer='adam', loss=losses.BinaryCrossentropy(),
 metrics=['accuracy’])

history = model.fit(train_data_gen,
         batch_size=128,
         epochs=5,
         steps_per_epoch=total_train_size // 128,
         validation_data=test_data_gen,
         validation_steps=total_test_size // 128)
```

Générer des prédictions

```
# predicting images
img = image.load_img('/content/drive/MyDrive/archive/grains-de-
beaute_gallerylarge.jpg', target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

images = np.vstack([x])
classes = model.predict(images, batch_size=10)
# predicting multiple images at once
img = image.load_img('/content/drive/MyDrive/archive/IMG-20200708-
WA0011.jpg', target_size=(224,224))
y = image.img_to_array(img)
y = np.expand_dims(y, axis=0)
# pass the list of multiple images np.vstack()
images = np.vstack([x, y])
classes = model.predict(images, batch_size=10)
# print the classes, the images belong to
print(classes)
categories=["normal","melanome"]
print(categories[int(classes[0][0])])
print(categories[int(classes[1][0])])
```

Preuve

Nous prouverons la première équation

on sait que : $\boldsymbol{\delta_j^L = \frac{\partial C}{\partial z_j^l}}$

En appliquant la règle de la chaine : $\boldsymbol{\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}}$

où la somme est sur tous les neurones k de la couche de sortie. Bien entendu, l'activation de sortie a_k^L du kème neurone ne dépend que de l'entrée pondérée z_j^L pour le jème neurone lorsque k=j. Et donc $\frac{\partial a_k^L}{\partial z_j^L}$ s'annule quand k≠j. En conséquence, nous pouvons simplifier l'équation précédente pour : $\boldsymbol{\delta_j^L = \frac{\partial C}{\partial a_J^L} \frac{\partial a_J^L}{\partial z_j^L}}$

Et puisque $\boldsymbol{a_J^L = \sigma(z_j^L)}$

Donc

$$\boldsymbol{\delta_j^L = \frac{\partial C}{\partial a_J^L} \sigma'(z_j^L)}$$

Preuve

Ensuite, nous prouverons la deuxième equation , qui donne une équation pour l'erreur δ^{l} en termes d'erreur dans la couche suivante, δ^{l+1}. Pour ce faire, on veut réécrire $\boldsymbol{\delta_j^l} = \frac{\partial C}{\partial z_j^l}$

en fonction de $\boldsymbol{\delta_k^{l+1}} = \frac{\partial C}{\partial z_k^{l+1}}$

. Nous pouvons le faire en utilisant la règle de la chaîne,

$$\boldsymbol{\delta_j^l} = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^L} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^L} \boldsymbol{\delta_k^{l+1}}$$

On sait que $\boldsymbol{z_k^{l+1}} = \sum_j \boldsymbol{w_{kj}^{l+1} a_j^l} + \boldsymbol{b_k^{l+1}} = \sum_j \boldsymbol{w_{kj}^{l+1}} \boldsymbol{\sigma}(\boldsymbol{z_j^L}) + \boldsymbol{b_k^{l+1}}$

En differenciant $\frac{\partial z_k^{l+1}}{\partial z_j^L} = \boldsymbol{w_{kj}^{l+1}} \boldsymbol{\sigma'}(\boldsymbol{z_j^L})$

En remplacant

$$\boldsymbol{\delta_j^l} = \sum_k \boldsymbol{w_{kj}^{l+1}} \, \boldsymbol{\delta_k^{l+1}} \boldsymbol{\sigma'}(\boldsymbol{z_j^L})$$

Références

- ann LeCun, Yoshua Bengio, Geoffrey Hinton : Deep learning
- https://faculty.uca.edu/ecelebi/documents/JAAD_2018.pdf
- https://towardsdatascience.com/backpropagation-made-easy-e90a4d5ede55
- https://www.actuia.com/contribution/cyrille-kone/a-travers-les-reseaux-de-neurones-aconvolution-en-deep-learning/
- https://www.mayoclinic.org/diseases-conditions/melanoma/symptoms-causes/syc-20374884
- https://www.imaios.com/fr/Societe/blog/Classification-des-images-medicales-comprendre-lereseau-de-neurones-convolutifs-CNN