Deep Learning par la pratique

Jour 3, matin : Auto-encoders

Plan

- Sauver les modèles et les réutiliser - production
  - TFX
- Comment inventer des grands modèles ?
- Retour sur les mécanismes internes aux RNNs
  - Demo simple
  - RNN sur série temporelle et comparaison avec CNN
  - RNN sur série multi variée
- travailler sur du texte
  - RNN sur du texte
- Auto encoders
  - Débruitage d'image
  - Détection d'anomalie

Sauvegarder le modèle dans un fichier

pour sauvegarder le modèle complet (archi et poids), on a 2 formats.
HDF5 (Keras legacy) et **SavedModel** (architecture, poids, config d'entraînement)

- `SavedModel : model.save('path/to/save_directory')`
- `HDF5 : model.save('path/to/save_model.h5')`

Sauvegarder **l'architecture** au format JSON, yaml

- `json_config = model.to_json()`
- `json_config = model.to_yaml()`

Sauvegarder les poids

- `model.save_weights('path/to/checkpoints')`
- `model.save_weights('path/to/weights.h5')`

Charger un modèle, son archi ou ses poids

Modèle complet

- `model = tf.keras.models.load_model('path/to/save_directory')`
- `model = tf.keras.models.load_model('path/to/save_model.h5')`

L'architecture

```
with open('model_config.json', 'r') as json_file:
    json_config = json_file.read()

model = tf.keras.models.model_from_json(json_config)
```

Les poids

- `model.load_weights('path/to/weights.h5')`
- `model.load_weights('path/to/checkpoints')`

formats HDF5 ou SavedModel

SavedModel

- parfaitement intégré dans l'écosystème tensorflow : lite, js et TFX
- Flexible : peut contenir d'autres éléments : graphs, …
- Optimisé pour la performance tensorflow

HDF5 est historiquement le format de sauvegarde de Keras utilisé dans de nombreux domaines au dela du deep learning : calcul scientifique, astronomy, physique, bioinformatique etc

- compression, I/O rapide, file structure

Exercice

- entraîner un modèle simple sur Colab
- sauver en local format SavedModel puis sur github (~10M)
- Dans un autre notebook
- Loader les data et le model et faire des predictions

Tensorflow en production
TFX

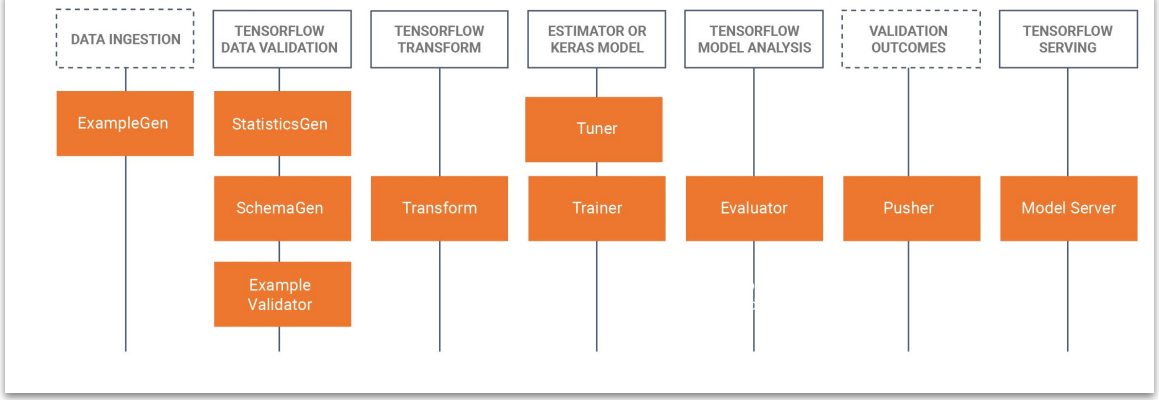
https://www.tensorflow.org/tfx

TFX is an end-to-end platform for deploying production ML pipelines

When you're ready to move your models from research to production, use TFX to create and manage a production pipeline.

Run Colab	View tutorials	View the guide	Explore addons
Get started by exploring each built-in component of TFX.	Learn how to use TFX with end-to-end examples.	Guides explain the concepts and components of TFX.	Additional TFX components contributed by the community.

TFX



Un **pipeline** TFX comprend généralement les composants suivants :

- **ExampleGen** est le composant d'entrée initial d'un pipeline qui ingère et éventuellement divise l'ensemble de données d'entrée.
- **StatisticsGen** calcule des statistiques pour l'ensemble de données.
- **SchemaGen** examine les statistiques et crée un schéma de données.
- **ExampleValidator** recherche les anomalies et les valeurs manquantes dans l'ensemble de données.
- **Transform** effectue l'ingénierie des fonctionnalités sur le jeu de données.
- Le **formateur** entraîne le modèle.
- **Tuner** règle les hyperparamètres du modèle.
- **Evaluator** effectue une analyse approfondie des résultats de la formation et vous aide à valider vos modèles exportés, en veillant à ce qu'ils soient "assez bons" pour être mis en production.
- **InfraValidator** vérifie que le modèle est réellement utilisable depuis l'infrastructure et empêche le mauvais modèle d'être poussé.
- **Pusher** déploie le modèle sur une infrastructure de service.
- **BulkInferrer** effectue un traitement par lots sur un modèle avec des demandes d'inférence sans étiquette.

https://www.tensorflow.org/tfx/guide?hl=fr

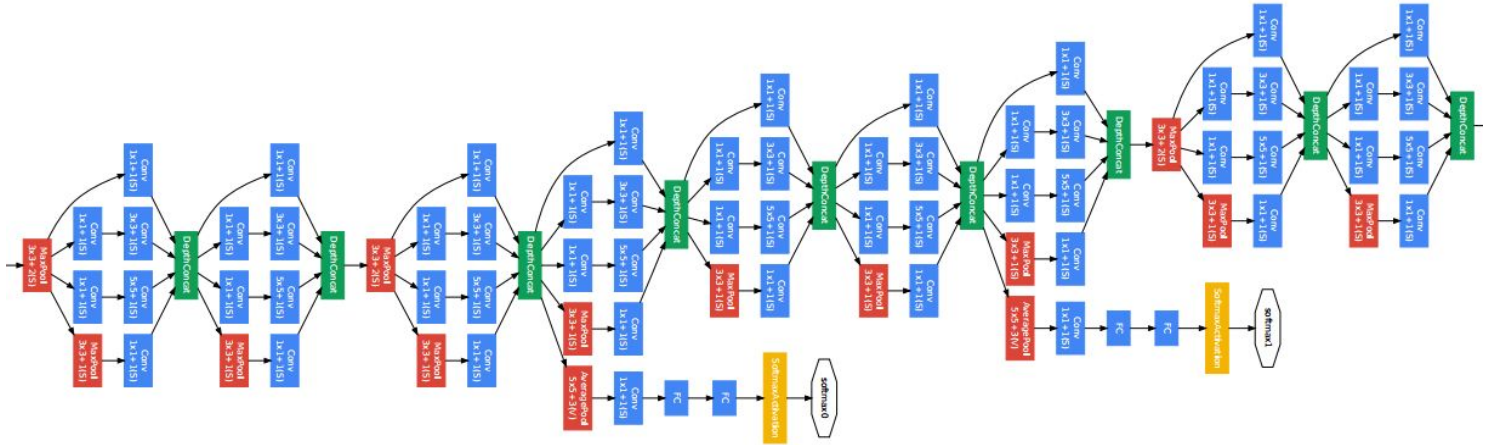
Exemple TFX Transform

Définir une série de transformation

```
def preprocessing_fn(inputs):
    # Extract features from the input
    x = inputs['x']
    y = inputs['y']

    # Scale the features
    x_scaled = tft.scale_to_0_1(x)
    y_scaled = tft.scale_to_0_1(y)

    # Combine the scaled features
    features = {
        'x_scaled': x_scaled,
        'y_scaled': y_scaled
    }

    return features
```

```
transform = tft.Transform(
    preprocessing_fn=preprocessing_fn,
    input_metadata=input_metadata
)
```

Inception V3
Mais comment font-ils ?



Methode de création des modeles complexes

- construction modulaire et hiérarchique
- Partage des poids entre modules dans différentes architecture

- Transfer learning
- **Neural Architecture Search (NAS)**: Automated methods to search for optimal network architectures using reinforcement learning or evolutionary algorithms
  - search space : espace des architectures possibles
  - search strategy : reinforcement learning, optimisation bayesienne
  - définir les modalités d'évaluation
  - NASnet, EfficientNet, DARTS

Keras Tuner Optimisation : grid search++

Keras Tuner is an easy-to-use hyperparameter optimization framework that solves the pain points of performing a hyperparameter search. It helps to find optimal hyperparameters for an ML model.

Keras Tuner makes it easy to define a search space and work with algorithms to find the best hyperparameter values. Keras Tuner comes with built-in Bayesian Optimization, Hyperband, and Random Search algorithms and is easily extendable to experiment with other algorithms.

- https://github.com/keras-team/keras-tuner
- https://keras.io/keras_tuner/

Keras Tuner - Example simple

1

```
def build_model(hp):
  model = keras.Sequential()
  model.add(keras.layers.Dense(
      hp.Choice('units', [8, 16, 32]),
      activation='relu'))
  model.add(keras.layers.Dense(1, activation='relu'))
  model.compile(loss='mse')
  return model
```

2

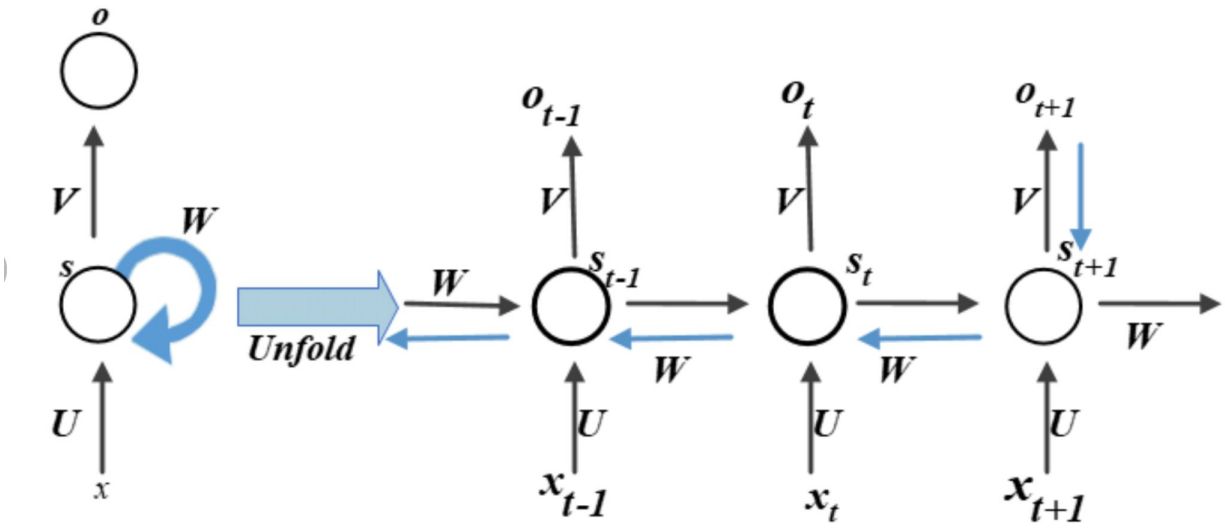
```
tuner =
keras_tuner.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=5)
```

3

```
tuner.search(x_train, y_train, epochs=5, validation_data=(x_val, y_val))
best_model = tuner.get_best_models()[0]
```

RNN

RNN - mecanismes internes



RNN - mecanismes internes

Forward pass

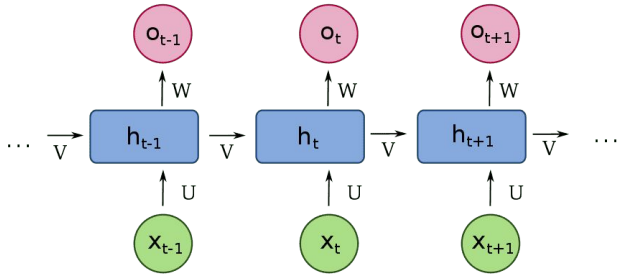
- x_t: Input vector at time step t
- h_t: Hidden state at time step t
- o_t: Output at time step t
- W: Input weight matrix
- U: Recurrent weight matrix
- V: Output weight matrix
- b_h: Bias for the hidden state
- b_o: Bias for the output
- σ: Activation function (e.g., tanh)

Hidden state calculation

$$h_t = \sigma(W \cdot x_t + U \cdot h_{t-1} + b_h)$$

Output calculation

$$o_t = \sigma(V \cdot h_t + b_o)$$



Les 3 matrices de poids

Purpose and Role:

- **W**: Transforms the input x_t to the hidden state space.
- **U**: Transforms the previous hidden state h_{t-1} to the current hidden state h_t
- **V**: Transforms the hidden state h_t to the output space.

Dimensions:

- **W**: Typically H×I (hidden state size by input size).
- **U**: Typically H×H (hidden state size by hidden state size).
- **V**: Typically O×H (output size by hidden state size).

Data Transformation:

- **W**: Deals with the direct transformation of input data into the hidden state.
- **U**: Deals with the transformation of the hidden state over time, capturing temporal dependencies.
- **V**: Deals with the transformation of the hidden state into the final output.

backpropagation through time (BPTT). This process is an extension of the standard backpropagation algorithm used in feed-forward neural networks, adapted to handle the temporal dependencies in sequential data. H

Backward : Backpropagation Through Time (BPTT)

1. **Computing Gradients for W:**
   - For each time step t, compute the partial derivative of the loss with respect to W:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

   - The gradient involves the sum of contributions from all time steps, reflecting how the input weights influence the entire sequence.

2. **Computing Gradients for U:**
   - For each time step t, compute the partial derivative of the loss with respect to U:

$$\frac{\partial L}{\partial U} = \sum_t \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial U}$$

   - This gradient involves the sum of contributions from all time steps and all previous time steps k, reflecting how the recurrent weights influence the hidden states over the entire sequence.

RNN simple Sunspot

Notebook

RNN_sunspot_simple.ipynb

https://github.com/SkatAI/deeplearning/blob/master/notebooks/RNN_sunspot_simple.ipynb

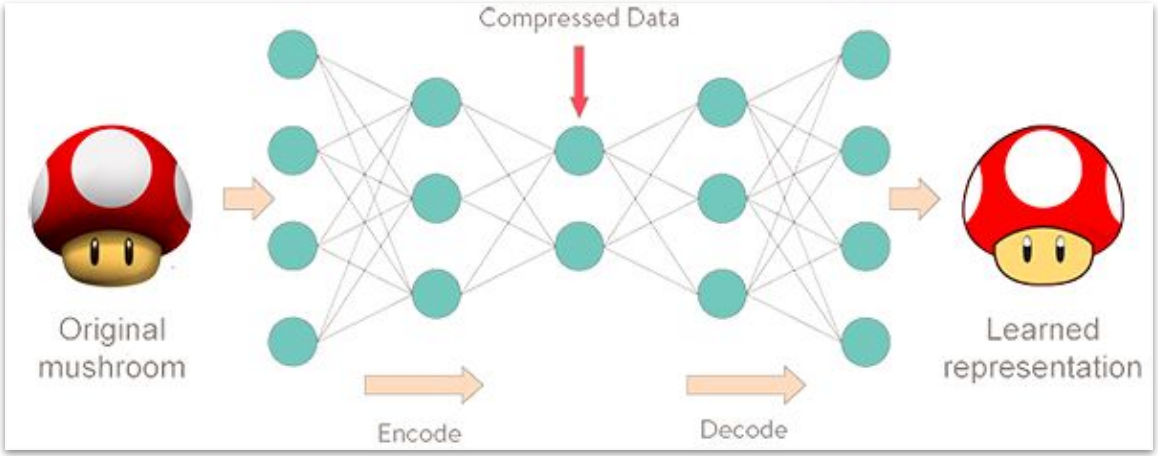
Auto-encoders

Autoencoder

utilisé pour le non supervisé : reduction de dimension et feature extraction.
Le but d'un autoencoder est d'obtenir une représentation "efficace" des données

2 étapes :

- encoding : Compression des données dans un espace de dimensions plus petit
- decoding : reconstruction des données a partir de la représentation interne



https://www.bpesquet.fr/mlhandbook/algorithms/autoencoders.html

Architecture de base

```
encoding_dim = 32

# This is our input image

input_img = keras.Input(shape=(784,))

# "encoded" is the encoded representation of the input

encoded = layers.Dense(encoding_dim, activation='relu')(input_img)

# "decoded" is the lossy reconstruction of the input

decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction

autoencoder = keras.Model(input_img, decoded)
```

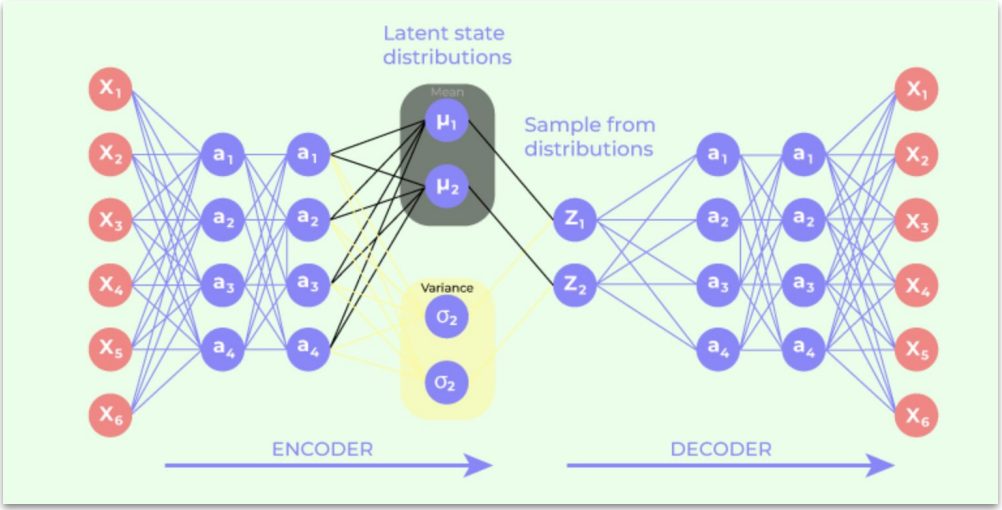
Debruiter les images avec un autoencoder convolutionel

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 32)          9248

 up_sampling2d (UpSampling2  (None, 14, 14, 32)        0
 D)

 conv2d_3 (Conv2D)           (None, 14, 14, 32)        9248

 up_sampling2d_1 (UpSamplin  (None, 28, 28, 32)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 28, 28, 1)         289

=================================================================
Total params: 28353 (110.75 KB)
Trainable params: 28353 (110.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Variational autoencoders

Variational autoencoders

- modèles génératifs
- apprennent les distributions de probabilités de l'espace latent
- la fonction de coût de reconstruction pousse le modèle a reconstruire les données d'entrée
- la regularisation (Kullback-divergence) contraint l'espace latent a respecter la distribution
- The probabilistic nature of the latent space also enables the generation of novel samples by drawing random points from the learned distribution.



https://www.geeksforgeeks.org/variational-autoencoders/

Pour aller plus loin

Machine Learning Notebooks, 3rd edition

Autoencoders, GANs, and Diffusion Models

https://github.com/ageron/handson-ml3/blob/main/17_autoencoders_gans_and_diffusion_models.ipynb

PCA avec autoencoder
https://medium.com/xebia-engineering/principal-component-analysis-autoencoder-257e90d08a4e