# Auto Encodeur de débruitage pour la détection d'anomalies vibratoires

May 5, 2023

**Antonin Lefevre**
antoninlefevre45@icloud.com

## Auto-Encodeur

Les auto-encodeurs (AE) sont des réseaux de neurones, entraînés de manière non supervisée, dont le but est de reconstruire les données d'entrées avec un maximum de précision en ayant codé ces dernières. Ils sont composés de deux parties, un encodeur et un décodeur. L'encodeur transforme les données dans un espace latent et le décodeur reconstruit les données à partir de cet espace latent. Dans le cas où cet espace latent est de dimension inférieure aux données, on parle d'architecture "sous-complète" (undercomplete). Quand la dimension de l'espace latent est supérieure à celle des données, elle est appelée "sur-complète" (overcomplete).

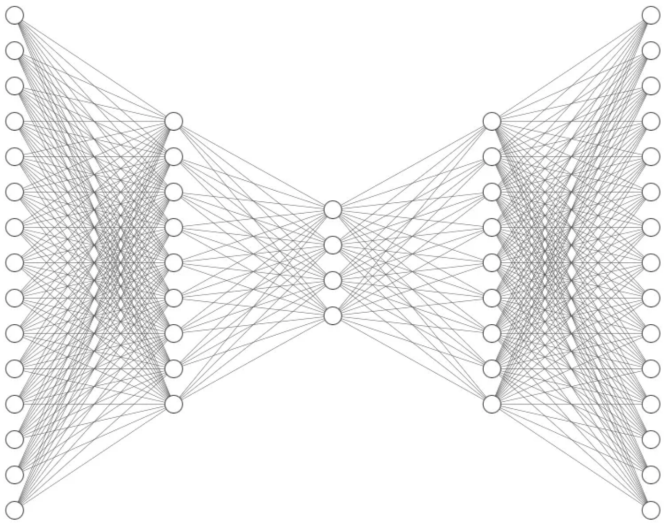

Figure 1: Architecture classique d'auto encodeur (undercomplete) utilisant des couches denses.

### Fonction de perte des Auto-Encodeurs

L'objectif de l'entraînement des AE est de minimiser l'écart entre l'entrée et sa version reconstruite. Ainsi, on peut exprimer la perte globale du réseau comme étant la perte moyenne par échantillon, ainsi:

$$L_{\text{total}} = \frac{1}{m}\sum_{\text{i}=1}^{m} l(x^{(\text{i})}, \hat{x}^{(\text{i})})$$

Si l'entrée est catégorielle alors on utilise la cross-entropy pour la perte de chaque échantillon, donc:

$$l(x^{(\text{i})}, \hat{x}^{(\text{i})}) = -\sum_{\text{j}=1}^{n}[x_i \log(\widehat{x_i}) + (1 - x_i)\log(1 - \widehat{x_i})]$$

Sinon, nous utiliserons la Mean Squared Error (MSE), donc:

$$l(x^{(\text{i})}, \hat{x}^{(\text{i})}) = \frac{1}{2}\|x - \hat{x}\|^2$$

Les auto-encodeurs de débruitage utilisent ces même fonctions de perte. Il n'y a rien n'a ajouter.

## Auto-encodeur de débruitage

Les auto-encodeurs de débruitage (DAE) corrompent intentionnellement les données d'entrée $x$ de manière aléatoire pour générer un nouveau vecteur $\tilde{x}$, qui est ensuite transmis à travers le réseau. La sortie du DAE, notée $\hat{x}$, est comparée à la version originale des données $x$, poussant ainsi le réseau à débruiter automatiquement les données corrompues $\tilde{x}$ afin qu'elles correspondent aux données originales $x$.

La corruption appliquée au vecteur d'entrée $x$ est contrôlée par un mappage stochastique $\tilde{x} \rightsquigarrow \mathbb{M}_{\mathbb{D}}(\tilde{x}|x)$. Historiquement [4], le processus de corruption des entrées consiste à sélectionner aléatoirement certaines valeurs d'entrée et à les remplacer par la valeur 0, similaire à un « dropout ». Cependant, en pratique, on suppose généralement que nous injectons la même distribution de bruit que celle présente dans la réalité, afin que notre modèle fonctionne de manière optimale dans des conditions réelles.

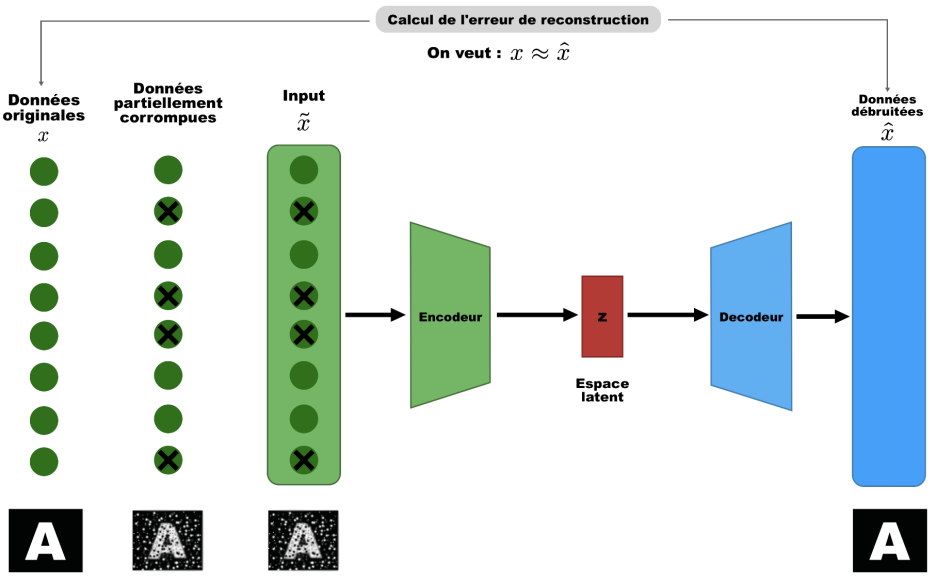


Figure 2: Fonctionnement du DAE.

Cette approche est inspirée par la capacité humaine à identifier un objet ou une scène, même lorsque la vision est partiellement masquée ou altérée. Afin de « restaurer » l'entrée partiellement endommagée, l'auto-encodeur de débruitage doit explorer et comprendre la relation entre les différentes dimensions de l'entrée pour déduire les éléments manquants.

## Intuition

Si l'on considère que nos données sont représentées par la courbe en rouge sur la Figure 3, la corruption va alors transformer chaque point $x$ en un point $\tilde{x}$. L'objectif de l'auto-encodeur de débruitage est de débruiter $\tilde{x}$ afin de se rapprocher le plus possible de la valeur initiale du point, c'est-à-dire $x$.

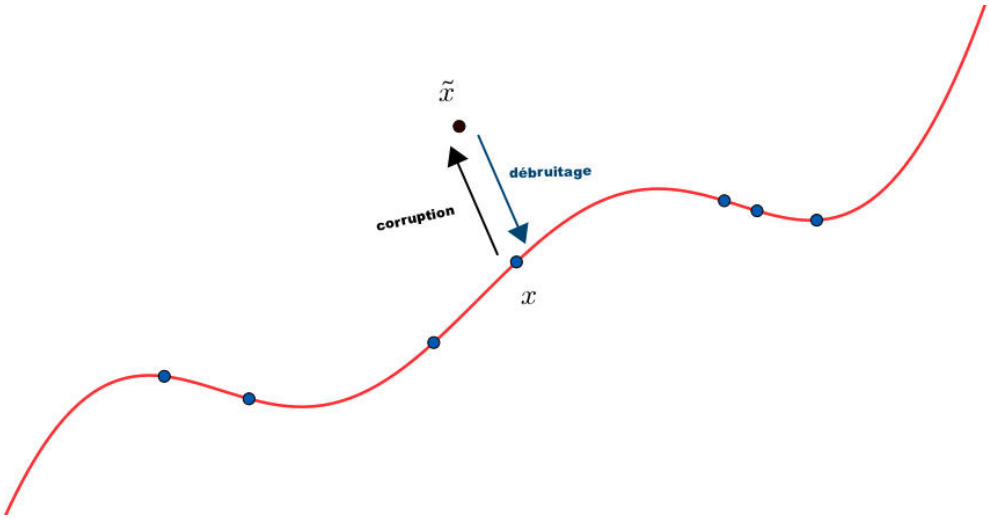


Figure 3: Intuition du DAE.

## Cas d'utilisation

**Détection d'anomalies** : En apprenant à reconstruire des données normales, les débruiteurs autoencodeurs peuvent être utilisés pour détecter des anomalies dans les données en mesurant la différence entre la reconstruction et les données d'entrée. Les anomalies peuvent inclure des erreurs, des fraudes ou d'autres événements inhabituels.

**Suppression de bruit dans les images** : Les débruiteurs autoencodeurs sont fréquemment utilisés pour supprimer le bruit dans les images, en particulier dans les cas où les données d'entraînement sont limitées. Ils peuvent apprendre à reconstruire des images bruitées en préservant les détails et les caractéristiques importantes.

**Traitement du signal audio** : Les débruiteurs autoencodeurs peuvent être appliqués pour réduire le bruit de fond et les interférences dans les enregistrements audio, améliorant ainsi la qualité et la clarté du signal audio d'origine.

# Implémentation d'un DAE pour la détection d'anomalies vibratoires

## Data

Dans cet article, nous présentons une application de détection d'anomalies vibratoires à l'aide d'un autoencodeur de débruitage et de données d'accélération obtenues à partir d'un capteur MPU-6050 (GY-521). Le MPU-6050 est un capteur 6 axes composé d'un accéléromètre et d'un gyroscope, communément utilisé dans les projets Arduino pour mesurer l'accélération et la rotation.

### Circuit

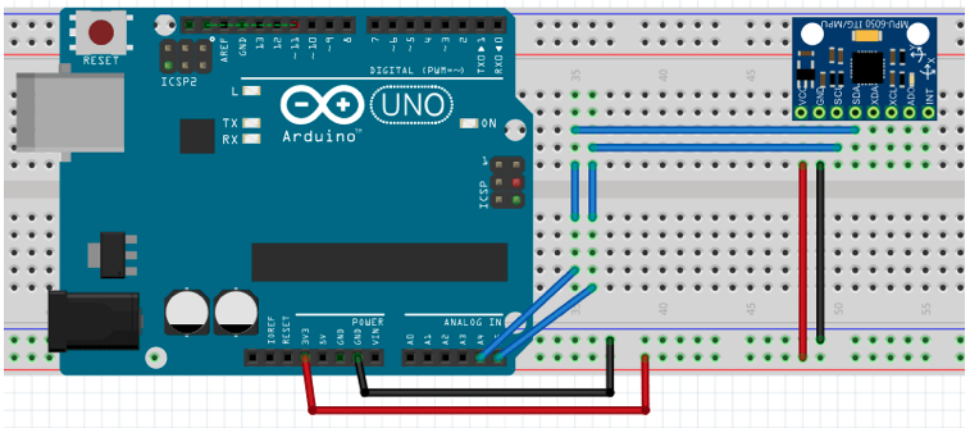


Figure 4: Schéma du circuit.

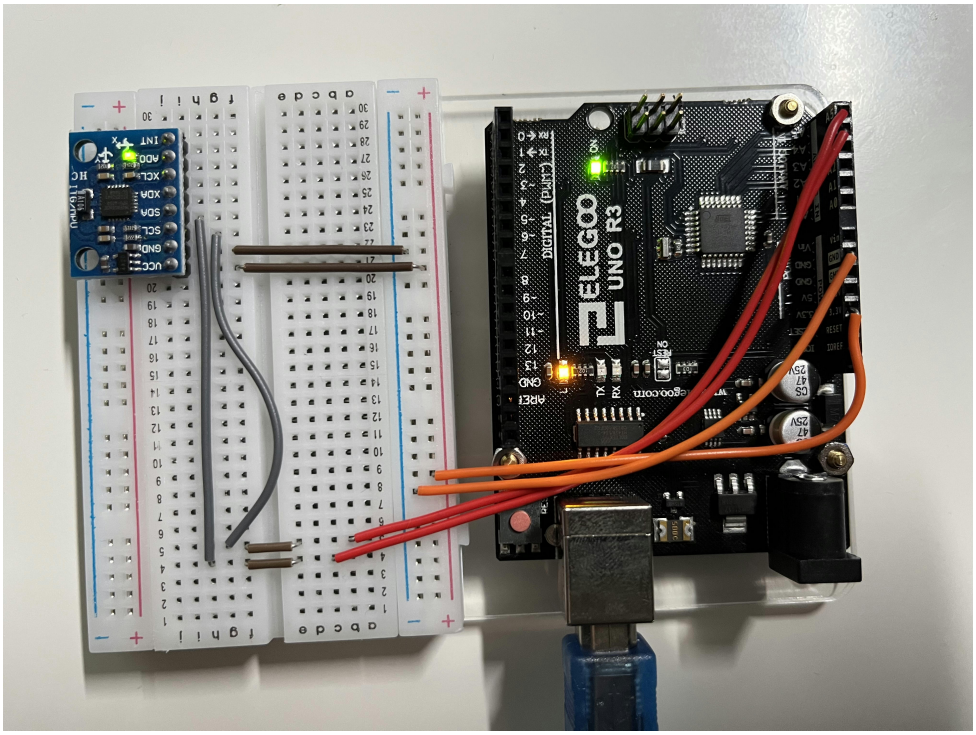


Figure 5: Photo du montage.

### Code d'acquisition des données

Le code utilisé est écrit en C++ pour l'environnement Arduino et en python pour le stockage des données. Nous utilisons la bibliothèque `Wire.h` pour établir une connexion I2C avec le capteur GY-521. Le code lit les données d'accélération brutes sur les axes X, Y et Z, puis les convertit en unités de m/s². Cette conversion est détaillée après.

Code d'acquisition des données du capteur :

```
#include <Wire.h>

const int MPU_ADDR = 0x68;
int16_t ax, ay, az;

void setup() {
  Wire.begin();
  Serial.begin(9600);
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
}

void loop() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 6, true);
  ax = Wire.read() << 8 | Wire.read();
  ay = Wire.read() << 8 | Wire.read();
  az = Wire.read() << 8 | Wire.read();
  float ax_mps2 = ax * (2.0 * 9.81) / 32768.0;
  float ay_mps2 = ay * (2.0 * 9.81) / 32768.0;
  float az_mps2 = az * (2.0 * 9.81) / 32768.0;
  float magnitude = sqrt(ax_mps2 * ax_mps2 + ay_mps2 * ay_mps2 + az_mps2 * az_mps2);
  Serial.println(magnitude);
  sample_count++;
  delay(100);
}
```

Puis, on utilise un script python pour stocker les données dans un fichier txt:

```python
import serial
import time
import os

ser = serial.Serial('/dev/tty.usbmodem00001', 9600)
time.sleep(2)

max_samples = 300  # nombre max de données dans le fichier
sample_count = 0
filename = "shocks_file.txt"

with open(filename, "w") as f:
  while sample_count < max_samples:
    if ser.inWaiting():
      data = ser.readline().decode("utf-8").strip()
      f.write(data + "\n")
      sample_count += 1
      print(f"Sample {sample_count}: {data}")

ser.close()
```

**Traitement des données brutes**

Les données d'accélération brutes sont converties en unités de m/s² en utilisant la formule suivante pour chaque axe :

$$a_{\text{mps2}} = a_{\text{raw}} \times \frac{2 \times 9.81}{32768}$$

Où $a_{\text{mps2}}$ est l'accélération en m/s², $a_{\text{raw}}$ est la valeur brute d'accélération et 9,81 est la gravité terrestre en m/s². Les données sont ensuite utilisées pour calculer la magnitude de l'accélération vectorielle en utilisant la formule suivante :

$$\text{magnitude} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Où $a_x$, $a_y$ et $a_z$ sont les accélérations en m/s² sur les axes X, Y et Z, respectivement.

La magnitude de l'accélération vectorielle est une mesure de l'amplitude totale de l'accélération, indépendamment de la direction. Cette mesure peut être utilisée comme entrée pour l'autoencodeur de débruitage afin de détecter des anomalies dans les données d'accélération.

Exemple (Affichage dans le « Serial Monitor » de Arduino IDE) :

```
Accel magnitude (m/s²): 8.31
Accel magnitude (m/s²): 8.31
Accel magnitude (m/s²): 8.37
Accel magnitude (m/s²): 8.34
Accel magnitude (m/s²): 8.36
Accel magnitude (m/s²): 8.37
Accel magnitude (m/s²): 8.36
Accel magnitude (m/s²): 8.38
Accel magnitude (m/s²): 8.37
Accel magnitude (m/s²): 9.53
Accel magnitude (m/s²): 8.35
Accel magnitude (m/s²): 8.28
Accel magnitude (m/s²): 8.41
Accel magnitude (m/s²): 8.32
Accel magnitude (m/s²): 8.37
Accel magnitude (m/s²): 8.34
Accel magnitude (m/s²): 8.38
Accel magnitude (m/s²): 8.38
Accel magnitude (m/s²): 8.42
```

## Modèle

Dans cet article, nous avons développé un modèle d'auto-encodeur de débruitage pour la détection d'anomalies dans des données de vibrations. L'architecture du réseau est basée sur des couches de convolution 1D pour traiter les séquences de données unidimensionnelles. Voici une description détaillée de l'architecture du réseau :

L'encodeur est composé de deux couches de convolution 1D, chacune suivie d'une fonction d'activation ReLU et d'une couche de max-pooling. La première couche de convolution prend en entrée un signal à une dimension (un canal) et produit 16 canaux de sortie avec des noyaux de taille 3 et un padding de 1. La seconde couche de convolution prend en entrée les 16 canaux produits par la première couche et produit 8 canaux de sortie.

```
self.encoder = nn.Sequential(
    nn.Conv1d(1, 16, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool1d(2),
    nn.Conv1d(16, 8, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool1d(2)
)
```

Le décodeur est composé de deux couches de convolution transposée 1D, chacune suivie d'une fonction d'activation ReLU. La première couche de convolution transposée prend en entrée les 8 canaux produits par l'encodeur et produit 16 canaux de sortie avec un noyau de taille 2 et un stride de 2. La seconde couche de convolution transposée prend en entrée les 16 canaux produits par la première couche et produit un canal de sortie (le signal débruité).

```
self.decoder = nn.Sequential(
nn.ConvTranspose1d(8, 16, kernel_size=2, stride=2),
nn.ReLU(),
nn.ConvTranspose1d(16, 1, kernel_size=2, stride=2)
)
```

Le fonctionnement général du réseau est le suivant : les données bruitées sont passées à travers l'encodeur, qui extrait des caractéristiques de bas niveau et réduit la dimensionnalité des données. Ensuite, le décodeur utilise ces caractéristiques pour reconstruire le signal d'origine sans bruit. L'objectif de l'entraînement est de minimiser la différence entre le signal d'origine et le signal reconstruit en ajustant les paramètres du modèle. Le bruit est ajouté de cette façon : on met entre 10 et 20 valeurs aléatoires entre 18 et 30 dans chaque fenêtre.

Voici un diagramme de l'architecture du réseau:

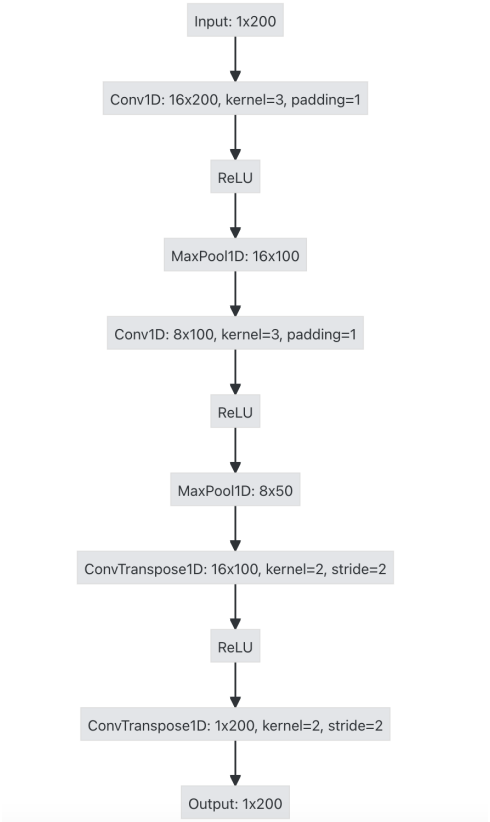


Figure 6: Architecture du réseau.

### Exemple de résultats de train (sur une fenêtre)


Figure 7: Exemple de résultats.

On voit que le modèle a supprimé les anomalies duent aux chocs, en conservant la forme initiale du signal.

## Détection d'anomalies en temps réel

Le modèle développé dans cet article est utilisé pour détecter des chocs avec les données du capteur transmises via une connexion série en temps réel.

Le processus de détection commence par la lecture des données du capteur en continu à travers la connexion série. Les valeurs d'accélération sont ajoutées à un tampon de taille égale à la fenêtre temporelle utilisée pour entraîner le modèle. Ensuite, les données du tampon sont normalisées en utilisant la moyenne et l'écart-type des données d'entraînement, assurant ainsi que les données en temps réel sont mises à l'échelle de manière similaire aux données utilisées pour l'entraînement.

Ces données normalisées sont ensuite transmises au modèle, qui tente de reconstruire les données d'entrée tout en éliminant le bruit anormal, y compris les chocs potentiels. La méthode `detect_shock` est utilisée pour comparer les données d'entrée et de sortie du modèle afin de déterminer si un choc a été détecté. Cette détection repose sur la distance euclidienne entre les données d'entrée et de sortie. Si cette distance dépasse un seuil prédéterminé, un choc est considéré comme détecté et un message d'alerte est affiché.

La bibliothèque utilisée pour visualiser en temps réel les données du capteur et recevoir les notifications en cas de détection d'anomalies est Streamlit. Voici l'interface du dashboard :

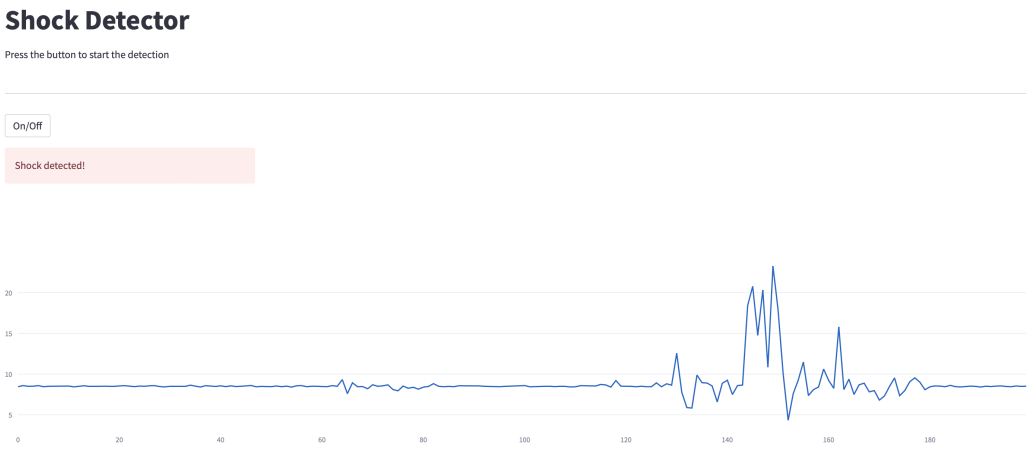


Figure 8: Dashboard streamlit.

Pour éviter de générer des notifications répétées lors de la détection continue d'anomalies, un délai d'attente est ajouté entre les alertes. Le programme attend un nombre spécifié de données reçues (par exemple, 50) avant d'émettre un nouveau message d'anomalie.

En utilisant cette approche, les chocs peuvent être détectés en temps réel en surveillant et en analysant en permanence les données du capteur d'accélération. Le modèle d'autoencodeur de débruitage permet de distinguer les chocs des autres variations de l'accélération, offrant ainsi une détection précise et fiable des anomalies.

# Références

[1] NYU Spring 2020 courses from Yann LeCun Y. LeCun, A. Canziani

[2] Credit Card Fraud Detection Using Autoencoder Neural Network P. Jiang, J. Zhang, J. Zou

[3] From Autoencoder to Beta-VAE Weng, Lilian

[4] Extracting and Composing Robust Features with Denoising Autoencoders 2008, P. Vincent, H. Larochelle, Y. Bengio, P-A. Manzagol