# BÀI THỰC HÀNH 2: CÁC KỸ THUẬT TỐI ƯU HOÁ MÔ HÌNH

**Hướng dẫn nộp bài**: Nộp file jupyter notebook (đuôi là .jpynb), đặt tên là MSSV_BaiThucHanh2.jpynb + File báo cáo PDF: MSSV_BaiThucHanh2.pdf

Nộp qua course, giảng viên sẽ tạo submission sau mỗi buổi học.

## 1. Bộ dữ liệu

Bộ dữ liệu: MNIST (bộ dữ liệu nhận dạng chữ số viết tay)

PS: Bộ dữ liệu này được hỗ trợ mặc định trong thư viện keras

```
from keras.datasets.mnist import load_data
(X_train, y_train), (X_test, y_test) = load_data()
```

## 2. Chuẩn bị dữ liệu.

Chia tập train ra làm 2 phần: 90% cho huấn luyện và 10% cho validation (development).

Sử dụng hàm train_test_split trong thư viện sklearn.

```
from sklearn.model_selection import train_test_split

X_train, X_dev, y_train, y_dev = train_test_split(X_train,
y_train, test_size=0.1)
```

Reshape data:

```
X_train_reshaped = X_train.reshape(-1, 784)
X_dev_reshaped = X_dev.reshape(-1, 784)
X_test_reshaped = X_test.reshape(-1, 784)
```

Chuẩn bị y_train và y_dev:

```
from tensorflow.keras.utils import to_categorical

y_train_new = to_categorical(y_train,num_classes=10)
y_dev_new = to_categorical(y_dev, num_classes = 10)
```

**3. Huấn luyện mô hình.**

Xây dựng mạng neural đơn giản gồm 2 lớp

+ Lớp 1: Dense với 784 units.

+ Lớp 2: Dense với 10 units.

```
model = Sequential()
model.add(Dense(784, input_shape=(784, ),
activation='relu'))
model.add(Dense(10, input_shape=(10, ),
activation='sigmoid'))
```

Hàm loss sử dụng: BinaryCrossentropy

```
loss = BinaryCrossentropy()
```

Thuật toán tối ưu: SGD với learning_rate là 0.01

```
optimizer = SGD(learning_rate=0.01)
```

Khi huấn luyện mô hình, sử dụng validation_data để tính toán **val_loss**

```
history = model.fit(X_train_reshaped, y_train_new,
validation_data=(X_dev_reshaped,y_dev_new) ,batch_size=12
8, epochs=30)
```

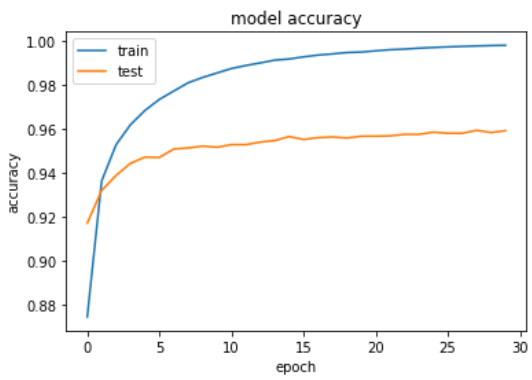
Hiển thị đồ thị học với **accuracy:**

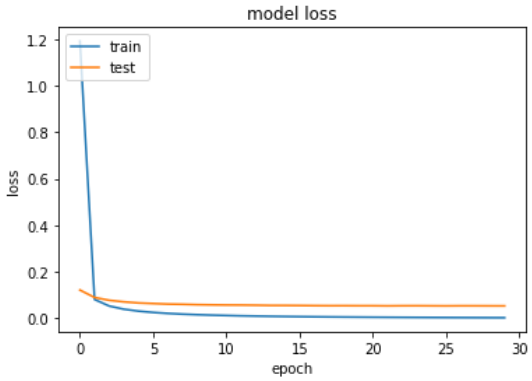
```
import matplotlib.pyplot as plt


plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Hiển thị đồ thị học với **loss**:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

**Kết quả:**




**4. Regularization cho mô hình**

Mục tiêu: Giảm overfiting (high variance).

Để chuẩn hoá, đối với Dense, ta thêm vào tham số **kernel_regularizer** để chuẩn hoá cho tham số W và **bias_regularizer** để chuẩn hoá cho tham số b

Các dạng chuẩn hoá: **l1 và l2**.

Chuẩn hoá cho lớp Dense với hệ số lamda cho W và b là 0.01

```
model2.add(Dense(784, input_shape=(784, ),
kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01),
activation='relu'))
model2.add(Dense(10, input_shape=(10, ),
kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01),
activation='sigmoid'))
```

Thực hiện huấn luyện lại mô hình, và xem kết quả.

Xem thêm: https://keras.io/api/layers/regularizers/

**5. Khởi tạo tham số**

Đối với lớp Dense, ta sử dụng tham số kernel_initializer và bias_initializer để khởi tạo tham số ban đầu lần lượt cho các tham số W và b trong mô hình.

Các phương pháp khởi tạo:

+ Khởi tạo số 0: Zeros().
+ Khởi tạp số 1: Ones().
+ Khởi tạo ngẫu nhiên theo phân phối chuẩn: RandomNormal().
+ Khởi tạo ngẫu nhiên theo Xavier uniform: GlorotUniform().

Mặc định thì đối với layer Dense sẽ khởi tạo W theo **GlorotUniform** và bias là 0.

Khởi tạo tham số cho lớp Dense:

```
model.add(Dense(784, input_shape=(784, ),
          kernel_initializer=GlorotUniform(),
          bias_initializer=Zeros,
          activation='relu')
)
```

Hãy khởi tạo tham số 0 và tham số 1 cho mô hình, sau đó huấn luyện và xem sự ảnh hưởng của khởi tạo tham số đối với mô hình như thế nào?

**6. Các thuật toán tối ưu**

Keras cung cấp một số thuật toán tối ưu nổi tiếng gồm:

+ SGD: Stochastic Gradient Descent.
+ RMSProp.
+ Adam.
+ Adagrad.

Sử dụng thuật toán tối ưu SGD:

```
from tensorflow.keras.optimizers import SGD
optimizer = SGD(learning_rate=0.01)
```

Sử dụng Momentum với giá trị là 0.9

```
from tensorflow.keras.optimizers import SGD
optimizer = SGD(learning_rate=0.01, momentum=0.9)
```

Xem thêm: https://keras.io/api/optimizers/

**7. Hàm loss (hàm mất mát).**

Keras hỗ trợ nhiều hàm loss khác nhau, gồm các loại sau:

+ ***Hàm loss tính dựa theo phân phối xác suất (dùng cho phân lớp)***:

BinaryCrossentropy, CategoricalCrossentropy, SparseCategoricalCrossentropy, ....

+ ***Hàm loss tính cho hồi quy***: MeanSquaredError, MeanAbsoluteError, CosineSimilarity, ....

+ ***Hàm loss tính cho maximum-margin (Hinge-loss).***

Lưu ý: Việc chọn hàm loss cũng rất quan trọng, có ảnh hưởng đến khả năng học của mô hình. Quá trình huấn luyện và điều chỉnh mô hình đều dựa trên cơ sở giá trị của hàm loss.

Sử dụng hàm loss trong keras:

```
from tensorflow.keras.losses import BinaryCrossentropy
loss = BinaryCrossentropy()
```

Xem thêm: https://keras.io/api/losses/

**8. Early Stopping.**

Là kỹ thuật giúp giảm được overfiting bằng cách dừng thuật toán huấn luyện nếu như độ chính xác không thể cải thiện thêm.

Để xác định Early Stopping, có thể dựa vào các thông số (monitor) như: loss, val_loss, acc, val_acc.

```
from keras.callbacks import EarlyStopping
callback = EarlyStopping(monitor="loss", patience=2)
```

Ghi chú: patience là số epoch mà kết quả không cải thiện so với trước đó.

Xem thêm: https://keras.io/api/callbacks/early_stopping/

***BÀI TẬP THỰC HÀNH:***

**Bài 1**: Thực hiện huấn luyện mô hình mạng neural ở Mục 3 trên bộ dữ liệu MNIST. Vẽ đồ thị học của mô hình với thông số accuracy và loss.

**Bài 2**: Thực hiện huấn luyện mô hình mạng neural ở bài 1 với kỹ thuật regularization cho tham số W và b với lamda là 0.01. Cho biết kết quả huấn luyện mô hình trước và sau khi áp dụng kỹ thuật regularization (Dựa vào đồ thị học).

**Bài 3**: Thực hiện huấn luyện mô hình mạng neural ở bài 1 với kỹ thuật khởi tạo tham số cho tham số W. Nếu sử dụng kỹ thuật khởi tạo tham số là 1 cho tham số W thì kết quả độ chính xác mô hình như thế nào?

**Bài 4**: Thực hiện huấn luyện mô hình mạng neural ở bài 2 (đã áp dụng regularization) với thuật toán optimizer RMSProp và Adam. So sánh kết quả giữa 2 thuật toán.

**Bài 5**: Hãy giảm batch size xuống 8 và huấn luyện mô hình mạng neural ở bài 2 (sử dụng optimizer SGD). Khi batch_size nhỏ thì có chuyện gì xảy ra (dựa vào đồ thị học).

**Bài 6**: Hãy tăng epoch lên 100 và huấn luyện mô hình. Cho biết kết quả.

**Bài 7**: *Thực hiện các yêu cầu từ Bài 1, Bài 2, Bài 6 với bộ dữ liệu CIFAR10.