Content today: Tut 1 Q3, Tut 2, Q1 - 4.

Tutorial 1: (MLP/NN/TensorFlow API)

– Train MLP with tf.keras in six steps:

1. Import → libraries
2. train, test → data prep. { x_train/test, y_train/test }
3. model = tf.keras.models.Sequential → Build layers like LEGO!
4. model.compile → Define How to train (Optimizer, loss fun. metrics...)
5. model.fit → Train with data, define batch size & itr.
6. model.summary → Report MLP summary

EE4802/IE4213 - Part II - Tutorial 1, Question 3

This question is on the Multi-Layer Perceptron (MLP) and using it to do classification. The aim is to find the best number of hidden nodes in the 3 hidden layers, assuming the same number of hidden nodes in each hidden layer. Cross-validation needs to be done on the training set. The MLP classifier with the best network size is then used for testing.

We shall use the Tensorflow Keras package to implement the MLP classifier. Obtain the data set “from sklearn.datasets import load_iris”. Import the necessary packages.

①

```
## load data from scikit
import numpy as np
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
```

```
pandas version: 1.1.5
```

(a) Load the data and split the database into two sets: 80% of samples for training, and 20% of samples for testing.

②

```
## load data
iris_dataset = load_iris()
## split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'],
                                                    iris_dataset['target'],
                                                    test_size=0.20,
                                                    random_state=0)
```

Change y_train and y_test to categorical values (required for classification).

```
y_train = keras.utils.to_categorical(y_train, num_classes = 3)
y_test = keras.utils.to_categorical(y_test, num_classes = 3)
```

(b) Perform a 5-fold Cross-validation using only the training set to determine the best 3-layer MLP classifier with hidden_layer_sizes=(Nhidd,Nhidd,Nhidd) for Nhidd in range(1,11))^ for prediction. In other words, partition the training set into two sets, 4/5 for training and 1/5 for validation; and repeat this process until each of the 1/5 has been validated. ^ The assumption of hidden_layer_sizes=(Nhidd,Nhidd,Nhidd) is to reduce the search space in this exercise. In field applications, the search needs to consider different sizes for each hidden layer.

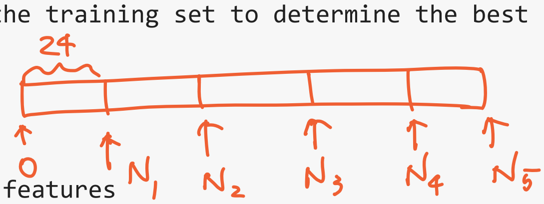
3

4

```
def MLP_model(Nhidd):
  model = Sequential()
  model.add(Dense(Nhidd, activation='relu', input_shape=(4,)))
  model.add(Dense(Nhidd, activation='relu'))
  model.add(Dense(Nhidd, activation='relu'))
  model.add(Dense(3, activation='softmax'))
  model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model
```

```
acc_train_array = []
acc_valid_array = []
for Nhidd in range(1,11):
    acc_train_array_fold = []
    acc_valid_array_fold = []
    ## Random permutation of data
    Idx = np.random.RandomState(seed=8).permutation(len(y_train))
    ## Tuning: perform 5-fold cross-validation on the training set to determine the best network size
    clf = MLP_model(Nhidd)
    for k in range(0,5):
```

→ 5 folds.



Slice the data for k-folds.

```
        N = np.around((k+1)*len(y_train)/5)
        N = N.astype(int)
        Xvalid = X_train[Idx[N-24:N]] # validation features
        Yvalid = y_train[Idx[N-24:N]] # validation targets
        Idxtrn = np.setdiff1d(Idx, Idx[N-24:N])
        Xtrain = X_train[Idxtrn] # training features in tuning loop
        Ytrain = y_train[Idxtrn] # training targets in tuning loop
        ## MLP Classification with same size for each hidden-layer (specified in question)
        clf.fit(Xtrain, Ytrain, epochs = 100, verbose = 0)
```

5

→ 0: Silent mode
1: Animated progress bar
2: Numerical output 1/50

Make prediction & Evaluate Performance

```
        ## trained output
        y_est_p = clf.predict(Xtrain)
        Ytrain_class = np.argmax(Ytrain, axis=1)
        y_est_p_class = np.argmax(y_est_p, axis=1)
        acc_train_array_fold += [metrics.accuracy_score(y_est_p_class,Ytrain_class)]
        ## validation output
        yt_est_p = clf.predict(Xvalid)
        Yvalid_class = np.argmax(Yvalid, axis=1)
        yt_est_p_class = np.argmax(yt_est_p, axis=1)
        acc_valid_array_fold += [metrics.accuracy_score(yt_est_p_class,Yvalid_class)]
    acc_train_array += [np.mean(acc_train_array_fold)]
    acc_valid_array += [np.mean(acc_valid_array_fold)]
    clf.summary()
```

6

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1)                 5
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 2
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 6
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
```

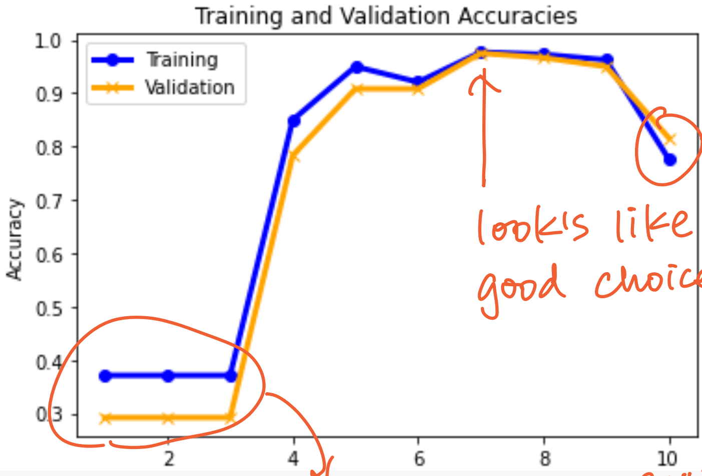
Observation:

- No. of parameters grows with No. of neurons.
- (None, N) placeholder. It does not affect size of the layer, but able to take in arbitrary length of data.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 2)                 10        
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 6         
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 6         
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 9         
=================================================================
Total params: 31
Trainable params: 31
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 3)                 15        
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 12        
_________________________________________________________________
dense_10 (Dense)             (None, 3)                 12        
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 12        
=================================================================
Total params: 51
Trainable params: 51
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_12 (Dense)             (None, 4)                 20        
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 20        
_________________________________________________________________
dense_14 (Dense)             (None, 4)                 20        
_________________________________________________________________
dense_15 (Dense)             (None, 3)                 15        
```

(c) Provide a plot of the average 5-fold training and validation accuracies over the different network sizes, i.e. different number of nodes in the hidden layer. Determine the hidden layer size Nhidd that gives the best validation accuracy for the training set.

```
## plotting
import matplotlib.pyplot as plt
hiddensize = [x for x in range(1,11)]
plt.plot(hiddensize, acc_train_array, color='blue', marker='o', linewidth=3, label='Training')
plt.plot(hiddensize, acc_valid_array, color='orange', marker='x', linewidth=3, label='Validation')
plt.xlabel('Number of hidden nodes in each layer')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracies')
plt.legend()
plt.show()
## find the best hidden layer size that gives the best validation accuracy using only the training se
Nhidden = np.argmax(acc_valid_array,axis=0)+1
print('best hidden layer size =', Nhidden, 'based on 5-fold cross-validation on training set')
```



looks like a good choice!

!! 100 epochs might not be enough to train a 4-10-10-10-3 MLP!

sign of underfitting

```
print(acc_train_array)
print(acc_valid_array)
```

```
[0.37083333333333335, 0.37083333333333335, 0.37083333333333335, 0.85, 0.95, 0.9208333333333332, 0.
[0.29166666666666663, 0.29166666666666663, 0.29166666666666663, 0.7833333333333333, 0.908333333
```

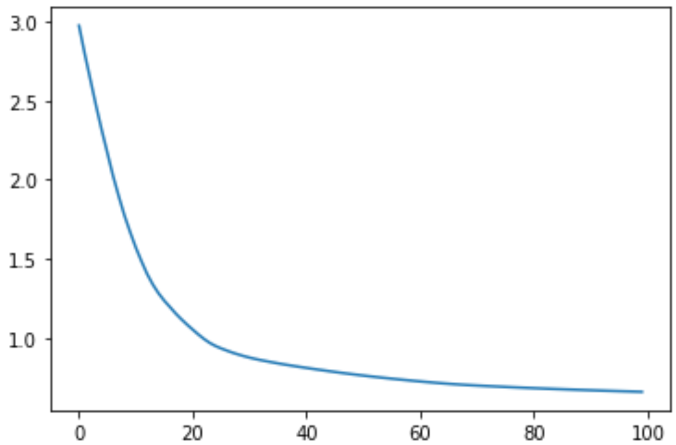
(d) Using the best hidden layer size Nhidd in the MLP classifier with hidden_layer_sizes=(Nhidd,Nhidd,Nhidd), evaluate the performance of the MLP by computing the prediction accuracy based on the 20% of samples for testing in part (a).

```
## perform evaluation
clf = MLP_model(Nhidden)
history=clf.fit(X_train, y_train, epochs = 100, batch_size = 32, verbose = 0)
## trained output
y_test_predict = clf.predict(X_test)
y_test_class = np.argmax(y_test, axis=1)
y_test_predict_class = np.argmax(y_test_predict, axis=1)
test_accuracy = metrics.accuracy_score(y_test_predict_class,y_test_class)
print('test accuracy =', test_accuracy)
```

```
test accuracy = 0.5666666666666667
```

```
plt.plot(history.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7f3cfdddd410>]
```



Tutorial 2 : (RNN)

Q1. What is the unstable gradients problem in deep learning and how can it be overcome?

Recall: the training of NNs are done through BP, from what we derived last time:

For the output layer: $\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial net} \cdot \frac{\partial net}{\partial w_{hj}}$

For the last hidden layer: $\frac{\partial E}{\partial w_{ij}} = \sum_k \left(\frac{\partial E}{\partial net_{ok}} \cdot \frac{\partial net_{ok}}{\partial y_{hj}} \right) \cdot \frac{\partial y_{hj}}{\partial net_{hj}} \cdot \frac{\partial net_j}{\partial w_{ij}}$

These highlighted terms are (or contains) the partial derivation of activation function (g').

As we back-prop to more layers, due to the chain rule, these terms will multiply with each other.

If: All the g' are very small $\rightarrow$ Vanishing Gradient

e.g. $(0.001)^5 = 1 \times 10^{-15}$

All the g' are very large $\rightarrow$ Exploding Gradient

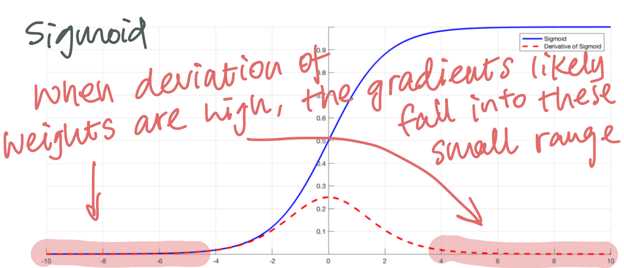
e.g. $(100)^5 = 1 \times 10^{10}$

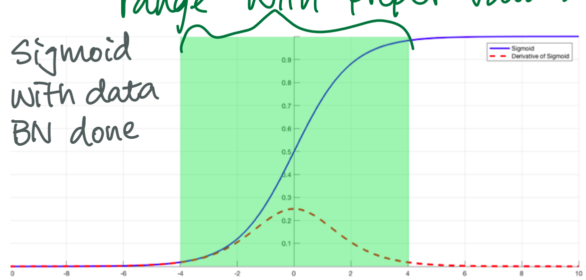
Solutions: 1. Proper weight initialization { Xavier, Normalized Xavier, He, ...

2. Avoid sigmoid, use ReLU.

3. Batch Normalization

BN reduces deviation, gradients likely fall in the range with proper values.





Q2. Add on batch normalization to Tutorial 1 Question 3 and comment on the differences observed in the performance and number of parameters. Why are there some non-trainable parameters?

→ Add BN: model.add(BatchNormalization())

```python
# Tut1 Q3
def MLP_model(Nhidd):
    model = Sequential()
    model.add(Dense(Nhidd, activation='relu', input_shape=(4,)))
    model.add(Dense(Nhidd, activation='relu'))
    model.add(Dense(Nhidd, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

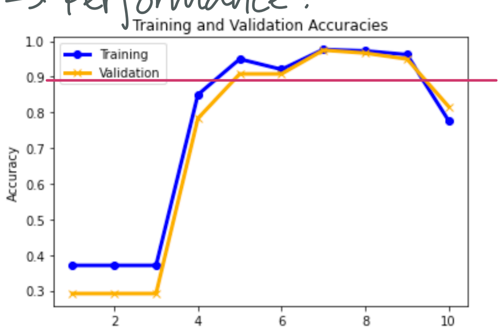
```python
# Tut2 Q2
from keras.layers import BatchNormalization
def MLP_model(Nhidd):
    model = Sequential()
    model.add(Dense(Nhidd, activation='relu', input_shape=(4,)))
    model.add(BatchNormalization())
    model.add(Dense(Nhidd, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(Nhidd, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

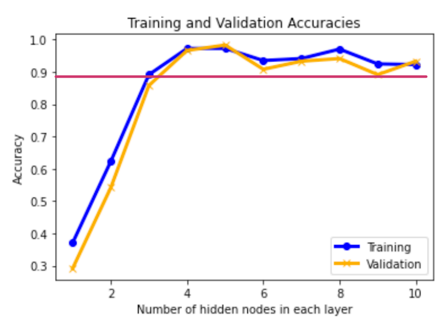
```
_________________________________________________________________
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 3)                 15
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 12
_________________________________________________________________
dense_10 (Dense)             (None, 3)                 12
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 12
=================================================================
Total params: 51
Trainable params: 51
Non-trainable params: 0
```

```
_________________________________________________________________
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 10)                50
_________________________________________________________________
batch_normalization_27 (Batc (None, 10)                40
_________________________________________________________________
dense_37 (Dense)             (None, 10)                110
_________________________________________________________________
batch_normalization_28 (Batc (None, 10)                40
_________________________________________________________________
dense_38 (Dense)             (None, 10)                110
_________________________________________________________________
batch_normalization_29 (Batc (None, 10)                40
_________________________________________________________________
dense_39 (Dense)             (None, 3)                 33
=================================================================
Total params: 423
Trainable params: 363
Non-trainable params: 60
```

→ Non-trianable params →
From the BN layers.
i.e. mean & std of each batch,
not to be trained.

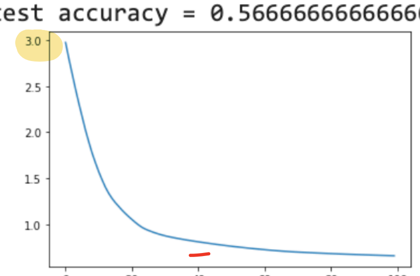
→ Performance:

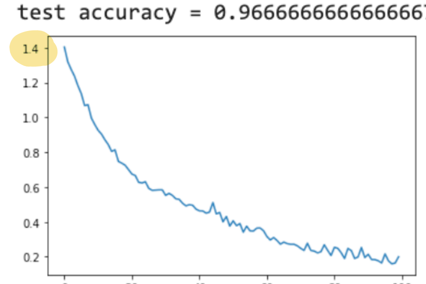




test accuracy = 0.5666666666666667

test accuracy = 0.9666666666666667





Q3. What enables a recurrent neural network (RNN) to remember input signals or patterns that occur over several time steps

From the defination of RNN:

The output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, it has a form of memory.

- Since the output of a recurrent neuron at time step *t* is a function of all the inputs from previous time steps, it has a form of *memory*. A part of a neural network that preserves some **state** across time steps is called a *memory cell* (or simply a *cell*).

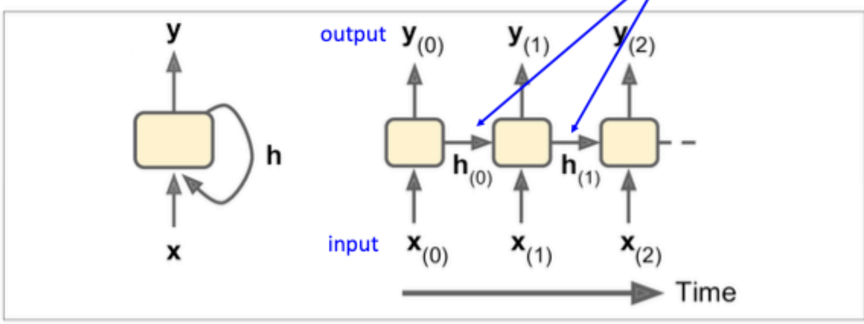


Figure 15-3. A cell's hidden state and its output may be different

Q4. What input and output sequence type is useful for time series prediction? What should the output be in this application?

Input and Output Sequences

- *Seq-to-seq* ✓
  - *useful for time series prediction, e.g. output is predicted future values*
- *Seq-to-vector*
  - *useful for e.g. sentiment analysis*
- *Vector-to-seq*
  - *feed same input over a few time steps, e.g. trigger output sequence*
- *Encoder–Decoder networks*
  - *seq-to-vector, followed by vector-to-seq*
  - *useful for, e.g. language translation*

note: Y can be a vector or a sequence

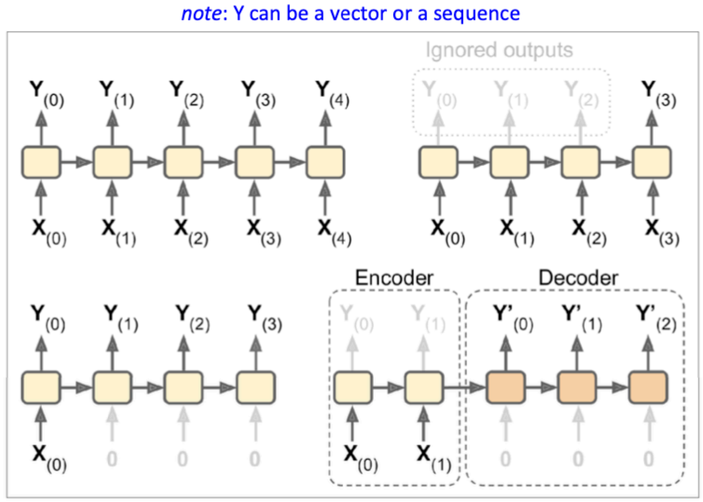


Figure 15-4. Seq-to-seq (top left), seq-to-vector (top right), vector-to-seq (bottom left), and Encoder–Decoder (bottom right) networks

Output of each time step should be a value shifted by one or more steps into the future.