0907779 Machine Learning (Fall 2018)
<u>Final Exam</u>

رقم الشعبة: 1 رقم التسجيل: .... الاسم: ....

===
<u>Instructions</u>: Time **80** min. Open book and notes exam. No electronics. Please answer all problems in the space provided and limit your answer to the space provided. No questions are allowed. There are eight problems. Each problem has 5 points.
===

P1. Complete the following code to find and print the model's RMSE on the test set.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

train_set, test_set = train_test_split(housing, test_size=0.3)
X_train = train_set.drop("y", axis=1)
y_train = train_set["y"].copy()
X_test = test_set.drop("y", axis=1)
y_test = test_set["y"].copy()

…   # some code is omitted

X_train_prepared = full_pipeline.fit_transform(X_train)

forest_reg = RandomForestRegressor(random_state=42)
forest_reg.fit(X_train_prepared, y_train)
```

<u>The solution is:</u>

```
X_test_prepared = full_pipeline.transform(X_test)
y_predictions = forest_reg.predict(X_test_prepared)

mse = mean_squared_error(y_test, y_predictions)
print("RMSE = ", np.sqrt(mse))
```

P2. Take a list, say for example this one: a = [1, 3, 5, 8, 13, 21], and write a Python program that prints out the sublist of all elements that are less than 5.

<u>The solution is:</u>

```
a = [1, 3, 5, 8, 13, 21]

new_list = []

for i in a:
    if i < 5:
        new_list.append(i)

print(new_list)
```

P3. The following code is used in the MNIST classification problem. What is the main purpose of this code and how many training jobs it performs?

```
from sklearn.model_selection import GridSearchCV

param_grid = [{'weights':     ["uniform", "distance"],
               'algorithm':   ["auto", "ball_tree"],
               'n_neighbors': [3, 5, 7]}]

knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(knn_clf, param_grid, cv=10)
grid_search.fit(X_train, y_train)
print(grid_search.best_params_)
```

The solution is:
This is grid search to find best hyper-parameters for the KNN classifier.

It fits 10 folds for each of the 2 × 2 × 3 = 12 candidates, totaling 120 fits.

P4. For each of the following five problems, specify how many neurons you need in the output layer and the output activation function type.

Problem	Output Layer Size	Activation Function
Reinforcement learning with actions in the four directions and shoot.	***5 neurons***	***Softmax***
Classify tumor into benign and malignant	***One neuron***	***Logistic***
English letter classification	***26 neurons***	***Softmax***
Predict gold prices	***One neuron***	***None***
Decimal digits classification	***10 neurons***	***Softmax***

P5. For the following two-layer convolutional network with one pooling layer, what is the number of output feature maps of this network and what is the size of each output map?

```
height = 28
width = 28
channels = 1
n_inputs = height * width

with tf.name_scope("inputs"):
    X = tf.placeholder(tf.float32, shape=[None, n_inputs], name="X")
    X_reshaped = tf.reshape(X, shape=[-1, height, width, channels])

conv1 = tf.layers.conv2d(X_reshaped, filters=10, kernel_size=3,
                         strides=1, padding="SAME",
                         activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(conv1, pool_size=2, strides=2)
conv2 = tf.layers.conv2d(pool1, filters=20, kernel_size=3,
                         strides=2, padding="SAME",
                         activation=tf.nn.relu)
```

The solution is:
There are 20 output feature maps.

Each map is 7 by 7.

P6. Complete the following code to implement an RNN that has three GRU layers and one fully connected layer. The first hidden layer should have 200 neurons, the second 100 neurons, and the third 50 neurons. Use TensorFlow's dynamic unrolling through time function. Note that it is not required to train this network; you only need to write the code to model the neural network and to generate the output logits.

```
import tensorflow as tf
from tensorflow.contrib.layers import fully_connected
n_steps = 28
n_inputs = 28
n_outputs = 10

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])

# your code goes here
```

The solution is:

```
layers = [tf.contrib.rnn. GRUCell(num_units=200),
          tf.contrib.rnn. GRUCell(num_units=100),
          tf.contrib.rnn. GRUCell(num_units=50)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)

outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X,
          dtype=tf.float32)
logits = fully_connected(states, n_outputs, activation_fn=None)
```

P7. Complete the following code to create a neural network policy for reinforcement learning. Your model should have one fully-connected hidden layer. Note that it is not required to train this policy; you only need to write the code to model the neural network and to generate the action.

```
import tensorflow as tf
from tensorflow.contrib.layers import fully_connected

n_inputs = 5
n_hidden = 10
n_outputs = 3
```

The solution is:

```
X = tf.placeholder(tf.float32, shape=[None, n_inputs])
hidden = fully_connected(X, n_hidden, activation_fn=tf.nn.elu,
                         weights_initializer=initializer)
logits = fully_connected(hidden, n_outputs, activation_fn=None,
                         weights_initializer=initializer)
outputs = tf.nn.softmax(logits)

action = tf.multinomial(tf.log(outputs), num_samples=1)
```

P8. Complete the following code to create and train a Keras regressor. Your model should have two fully-connected hidden layers. The first layer should have 30 neurons and the second layer should have 15 neurons.

```
import pandas
import tensorflow as tf
from tensorflow.keras.models import layers

# load dataset
dataframe = pandas.read_csv("data.csv")
dataset = dataframe.values
# split into input (X) and output (Y) variables
X = dataset[:,0:10]
Y = dataset[:,10]
```

The solution is:

```
# create model
model = tf.keras.Sequential()
model.add(layers.Dense(30, input_dim=10, activation='relu'))
model.add(layers.Dense(15, activation='relu'))
model.add(layers.Dense(1))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam',
              metrics['mse'])
model.fit(X, Y, epocs=10, batch_size=32)
```

<Good Luck>