Problem 1

(a) (a) Algorithm 1: Loading, Preprocessing, Reshaping Data

```
import numpy as np
import keras
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(f'Orig train set range: [{np.min(x_train)},{np.max(x_train)}]')
print(f'Orig test set range: [{np.min(x_test)},{np.max(x_test)}]')
"""
Orig train set range: [0,255]
Orig test set range: [0,255]
"""

x_train = np.divide(x_train,255)
x_test  = np.divide(x_test,255)

print(f'Normalized train set range: [{np.min(x_train)},{np.max(x_train)}]')
print(f'Normalized test set range: [{np.min(x_test)},{np.max(x_test)}]')
"""
Normalized train set range: [0.0,1.0]
Normalized test set range: [0.0,1.0]
"""

print('Orig train samples shape: ', x_train.shape )
print('Orig test samples shape: ', x_test.shape)
"""
Orig train samples shape:  (60000, 28, 28)
Orig test samples shape:  (10000, 28, 28)
"""

x_train = np.reshape(x_train,(x_train.shape[0],784))
x_test = np.reshape(x_test,(x_test.shape[0],784))

print('Vectorized train samples shape: ', x_train.shape )
print('Vectorized test samples shape: ', x_test.shape )
"""
Vectorized train samples shape:  (60000, 784)
Vectorized test samples shape:  (10000, 784)
"""

one_hot_train = keras.utils.to_categorical(y_train, num_classes=10)
one_hot_test = keras.utils.to_categorical(y_test, num_classes=10)
```

(b) Algorithm 2: Plot Some Data

```
import numpy as np
```

```python
import matplotlib.pyplot as plt

def plot_5(trn, labs):
    f,axrow = plt.subplots(1,5)
    axrow[0].axis('off')
    axrow[0].imshow(trn[0],cmap='gray')
    axrow[0].set_title(f"class={labs[0]}")
    axrow[1].axis('off')
    axrow[1].imshow(trn[1],cmap='gray')
    axrow[1].set_title(f"class={labs[1]}")
    axrow[2].axis('off')
    axrow[2].imshow(trn[2],cmap='gray')
    axrow[2].set_title(f"class={labs[2]}")
    axrow[3].axis('off')
    axrow[3].imshow(trn[3],cmap='gray')
    axrow[3].set_title(f"class={labs[3]}")
    axrow[4].axis('off')
    axrow[4].imshow(trn[4],cmap='gray')
    axrow[4].set_title(f"class={labs[4]}")
    # plt.show()
    plt.savefig('./first_5.png')
    return

plot_5(x_train,y_train) # See Figure 1
```

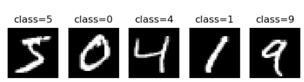

Figure 1: Plot of the first 5 training examples

(c) Algorithm 3: Build A Model

```python
def build_model(dp=False,sigma=0.01):
    units=512
    hiddens=2
    model = Sequential()
    model.add(Dense(units,input_dim=(28*28), activation='relu'))
    model.add(Dense(units, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    if dp: optim = DPSGD(sigma=sigma)
    else:  optim = 'rmsprop'
    model.compile(optimizer=optim, loss='categorical_crossentropy',metrics=
```

```
        print(f"Number of trainable parameters: {hiddens*(units**2)+units}")
        return model

    model = build_model()
    """
    Number of trainable parameters: 524800
    """
```

The model has $k * n^2 + n$ trainable parameters, where k is the number of hidden layers and n is the number of units per hidden layer. As such, the model has 524,800 trainable parameters.

(d) Algorithm 4: Train the Model

```
    """ Model compiled in build_model() function above """
    epochs=20
    history = model.fit(x_train, one_hot_train, epochs=epochs, batch_size=128)
    acc = history.history['acc']
    loss = history.history['loss']

    loss_tst, acc_tst = model.evaluate(x_test,one_hot_test, verbose=1)
    print(f"\nModel Evaluation after {epochs} epochs:\tLoss: {loss_tst}\n\t\t\t
    """
    Model Evaluation after 20 epochs:       Loss: 0.16491800990887517
                                            Accuracy: 0.9801

    """
```

(b) Algorithm 5: Plot Training Accurcy Loss

```
def plot_training(ax,ls,fn,sigs=['']):
    f,axrow = plt.subplots(1,2)

    for acc in range(len(ax)):
        axrow[0].plot(ax[acc],label=sigs[acc])
    axrow[0].set(xlabel='Epochs')
    axrow[0].set_ylim(0,1)
    axrow[0].set_title('Training Accuracy')
    if len(sigs)>1:
        axrow[0].legend()
    for loss in range(len(ls)):
        axrow[1].plot(ls[loss],label=sigs[loss])
    axrow[1].set(xlabel='Epochs')
    # axrow[1].set_ylim(0,1)
    axrow[1].set_title('Training Loss')
    if len(sigs)>1:
        axrow[1].legend()
    # plt.show()
    plt.savefig(f'{fn}.png')
    return
```

```
plot_training([acc],[loss],'non_priv')
```

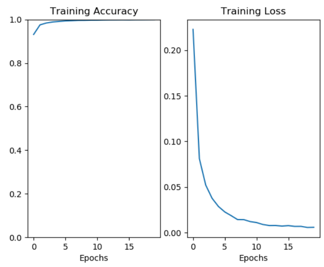

Figure 2: Non-Private Training Accuracy Loss, 20 Epochs

Problem 2

Algorithm 6: Plot Training Accurcy Loss

```
class DPSGD(Optimizer):
    """
    Differentially Private Stochastic gradient descent optimizer.
    # Arguments
    lr:    float >= 0. Learning rate.
    decay: float >= 0. Learning rate decay over each update.
    sigma: float >= 0. scale parameter of (Gaussian) noise distribution
    """
    def __init__(self, lr=0.01, decay=0., sigma=0.0001, **kwargs):
        super(DPSGD, self).__init__(**kwargs)
        with K.name_scope(self.__class__.__name__):
            self.iterations = K.variable(0, dtype='int64',
            name='iterations')
            self.lr = K.variable(lr, name='lr')
        self.initial_decay = decay
        self.sigma = sigma
    @interfaces.legacy_get_updates_support
    def get_updates(self, loss, params):
        """ Noise injected in lines below """
        grads=self.get_gradients(loss,params)
        noisy_grads = [np.add(g,
                            keras.backend.random_normal(g.shape,mean=0.,
                            stddev=self.sigma**2)) for g in grads]
        self.updates = [K.update_add(self.iterations, 1)]
        lr = self.lr
        if self.initial_decay > 0:
            lr = lr * (1. / (1. + self.decay *
           K.cast(self.iterations,
```

```
                        K.dtype(self.decay))))
        shapes = [K.int_shape(p) for p in params]
                # returns array of tensors the size of each param
        self.weights = [self.iterations]
        for p, g in zip(params, noisy_grads):
            v = -lr * g    # velocity
            """ update occurs here """
            new_p = p + v
            if getattr(p, 'constraint', None) is not None:
                new_p = p.constraint(new_p)
            self.updates.append(K.update(p, new_p))
        return self.updates

    def get_config(self):
        config = {'lr': float(K.get_value(self.lr)),
                  'decay': float(K.get_value(self.decay)),
                  'sigma': float(K.get_value(self.sigma))
                 }
        base_config = super(DPSGD, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

(a)

1. Algorithm 7: Plot Training Accuracy Loss

```
dp_ax=[]
dp_ls=[]
sigs=[0.001,0.01,0.1,1.0,2.0]

for sigma in sigs:
    dp_model = build_model(dp=True,sigma=sigma)
    dp_history = dp_model.fit(x_train, one_hot_train,
    epochs=epochs, batch_size=128)
    dp_acc  = dp_history.history['acc']
    dp_loss = dp_history.history['loss']
    dp_ax.append(dp_acc)
    dp_ls.append(dp_loss)
    dp_loss_tst, dp_acc_tst =
    dp_model.evaluate(x_test,one_hot_test, verbose=1)
    print(f"DP Model Evaluation after {epochs} epochs:\t",
    "Noise Variance: {sigma}\n\t\t\t\t\t",
    "Loss: {dp_loss_tst}\n\t\t\t\t\tAccuracy: {dp_acc_tst}")
"""
                ****Results Vary Per Iteration****
DP Model Evaluation after 20 epochs:        Noise Variance: 0.001
                                            Loss: 0.1541813389956951
                                            Accuracy: 0.9555

DP Model Evaluation after 20 epochs:        Noise Variance: 0.01
                                            Loss: 0.15676156164556743
                                            Accuracy: 0.9547
```

```
DP Model Evaluation after 20 epochs:        Noise Variance: 0.1
                                             Loss: 0.15204105867668985
                                             Accuracy: 0.9559

DP Model Evaluation after 20 epochs:        Noise Variance: 1.0
                                             Loss: 3.314135622215271
                                             Accuracy: 0.7934

DP Model Evaluation after 20 epochs:        Noise Variance: 2.0
                                             Loss: 14.241949020385743
                                             Accuracy: 0.1164

"""
```

2.

3. Algorithm 8: Plot Training Accurcy Loss

```
plot_training(dp_ax, dp_ls, 'priv_training', sigs)
```

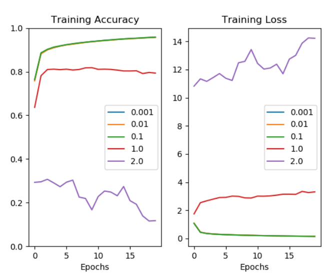

Figure 3: Differentially Private Training Accuracy Loss, 20 Epochs**

** The plots for sigma=0.001,0.01,0.1 overlap, as is evidenced by their accuracies at the end of training, printed above.