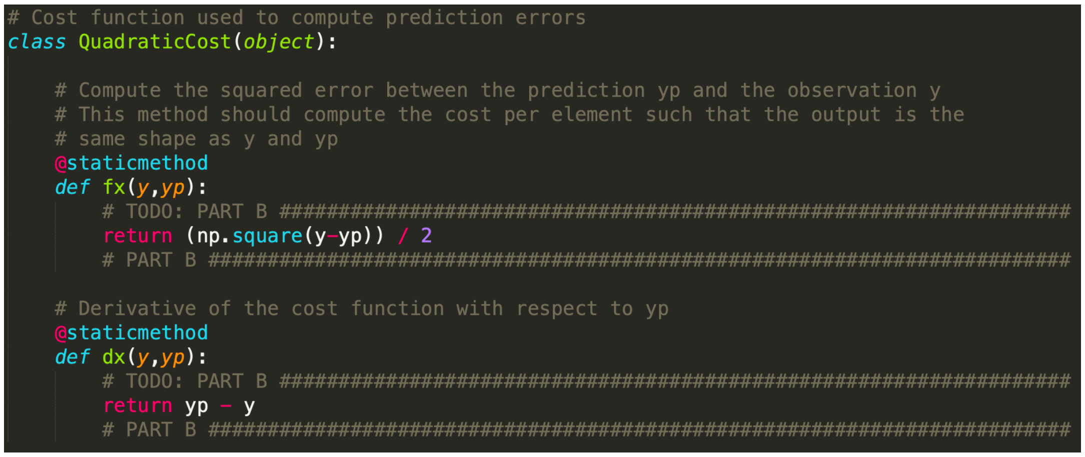
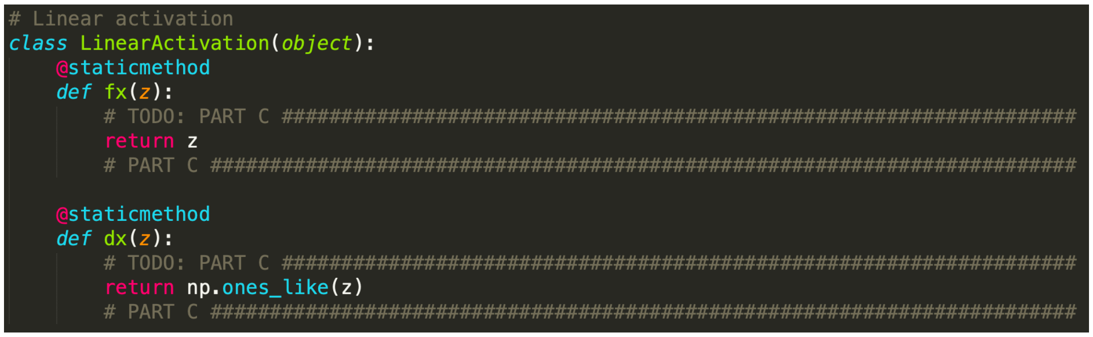
1.

(b)

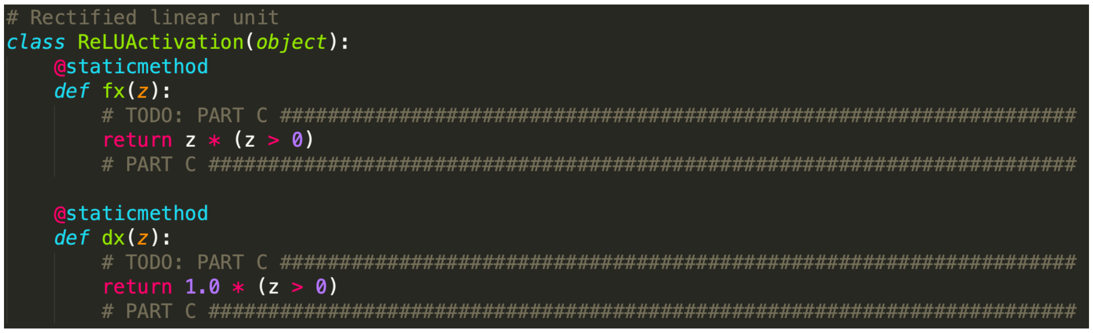


```python
# Cost function used to compute prediction errors
class QuadraticCost(object):

    # Compute the squared error between the prediction yp and the observation y
    # This method should compute the cost per element such that the output is the
    # same shape as y and yp
    @staticmethod
    def fx(y,yp):
        # TODO: PART B ###################################################################
        return (np.square(y-yp)) / 2
        # PART B #########################################################################

    # Derivative of the cost function with respect to yp
    @staticmethod
    def dx(y,yp):
        # TODO: PART B ###################################################################
        return yp - y
        # PART B #########################################################################
```

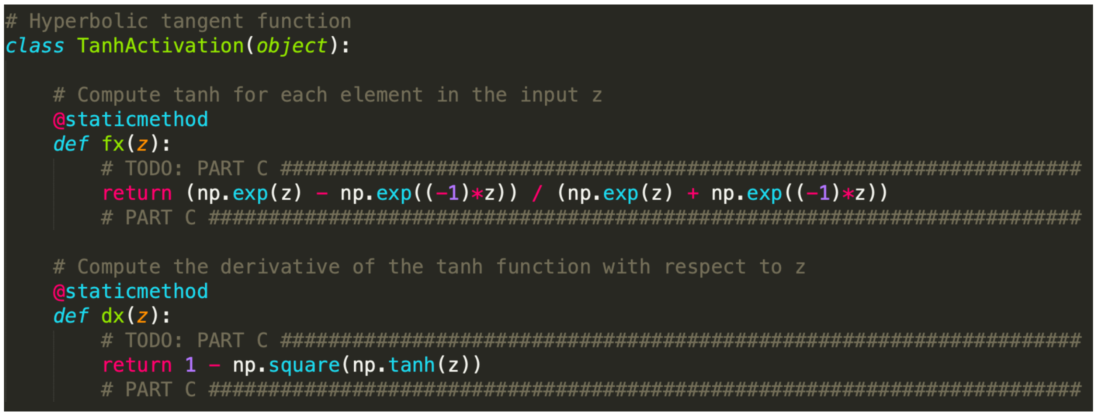
(c)



```python
# Linear activation
class LinearActivation(object):
    @staticmethod
    def fx(z):
        # TODO: PART C ###################################################################
        return z
        # PART C #########################################################################

    @staticmethod
    def dx(z):
        # TODO: PART C ###################################################################
        return np.ones_like(z)
        # PART C #########################################################################
```

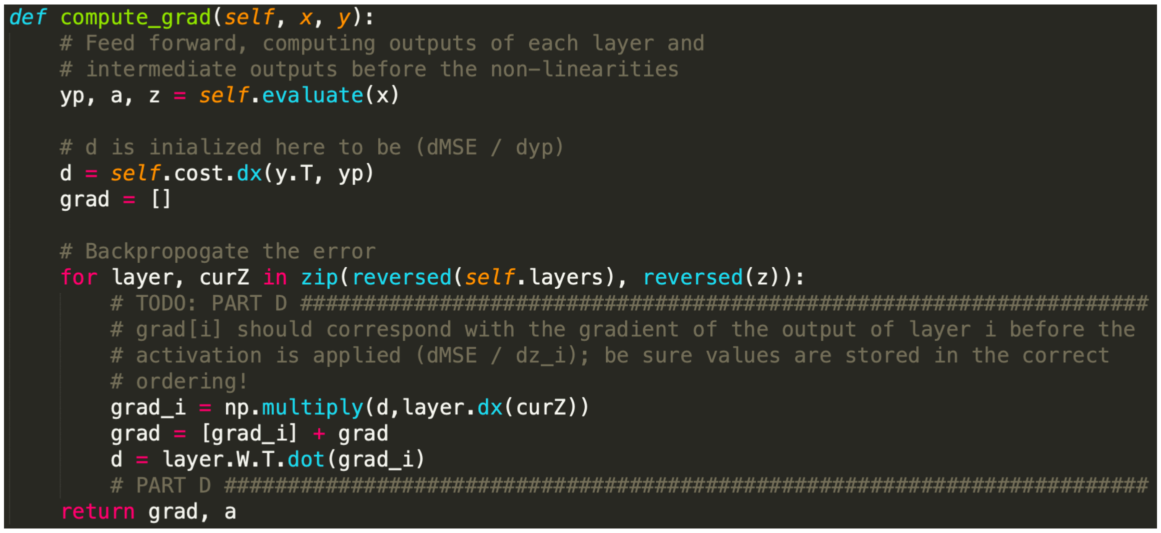


```python
# Rectified linear unit
class ReLUActivation(object):
    @staticmethod
    def fx(z):
        # TODO: PART C ###################################################################
        return z * (z > 0)
        # PART C #########################################################################

    @staticmethod
    def dx(z):
        # TODO: PART C ###################################################################
        return 1.0 * (z > 0)
        # PART C #########################################################################
```



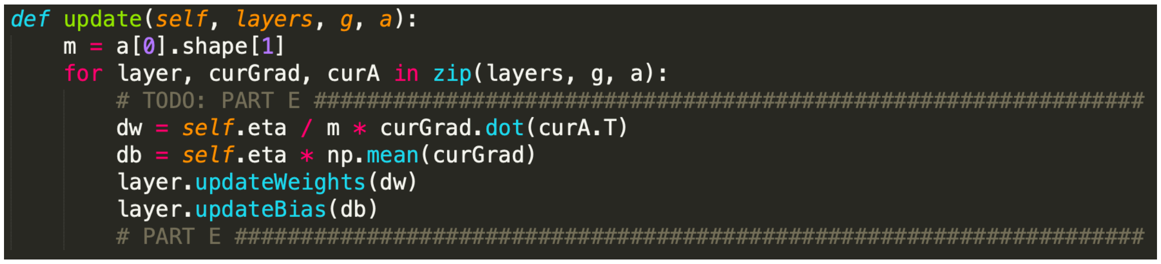
```python
# Hyperbolic tangent function
class TanhActivation(object):

    # Compute tanh for each element in the input z
    @staticmethod
    def fx(z):
        # TODO: PART C ###################################################################
        return (np.exp(z) - np.exp((-1)*z)) / (np.exp(z) + np.exp((-1)*z))
        # PART C #########################################################################

    # Compute the derivative of the tanh function with respect to z
    @staticmethod
    def dx(z):
        # TODO: PART C ###################################################################
        return 1 - np.square(np.tanh(z))
        # PART C #########################################################################
```

(d)



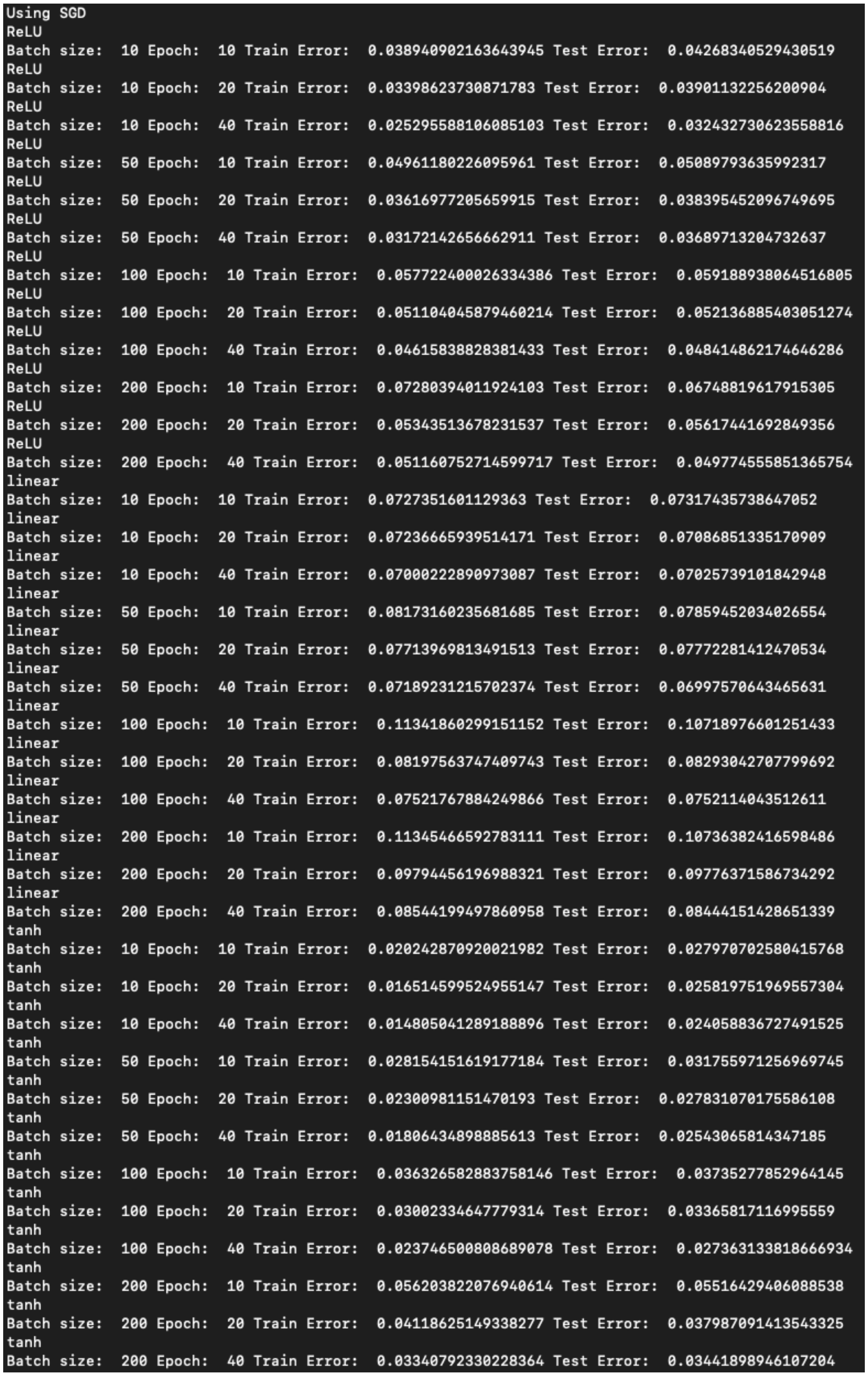
```python
def compute_grad(self, x, y):
    # Feed forward, computing outputs of each layer and
    # intermediate outputs before the non-linearities
    yp, a, z = self.evaluate(x)

    # d is inialized here to be (dMSE / dyp)
    d = self.cost.dx(y.T, yp)
    grad = []

    # Backpropogate the error
    for layer, curZ in zip(reversed(self.layers), reversed(z)):
        # TODO: PART D #################################################################
        # grad[i] should correspond with the gradient of the output of layer i before the
        # activation is applied (dMSE / dz_i); be sure values are stored in the correct
        # ordering!
        grad_i = np.multiply(d,layer.dx(curZ))
        grad = [grad_i] + grad
        d = layer.W.T.dot(grad_i)
        # PART D #######################################################################
    return grad, a
```

(e)



```python
def update(self, layers, g, a):
    m = a[0].shape[1]
    for layer, curGrad, curA in zip(layers, g, a):
        # TODO: PART E #################################################################
        dw = self.eta / m * curGrad.dot(curA.T)
        db = self.eta * np.mean(curGrad)
        layer.updateWeights(dw)
        layer.updateBias(db)
        # PART E #######################################################################
```

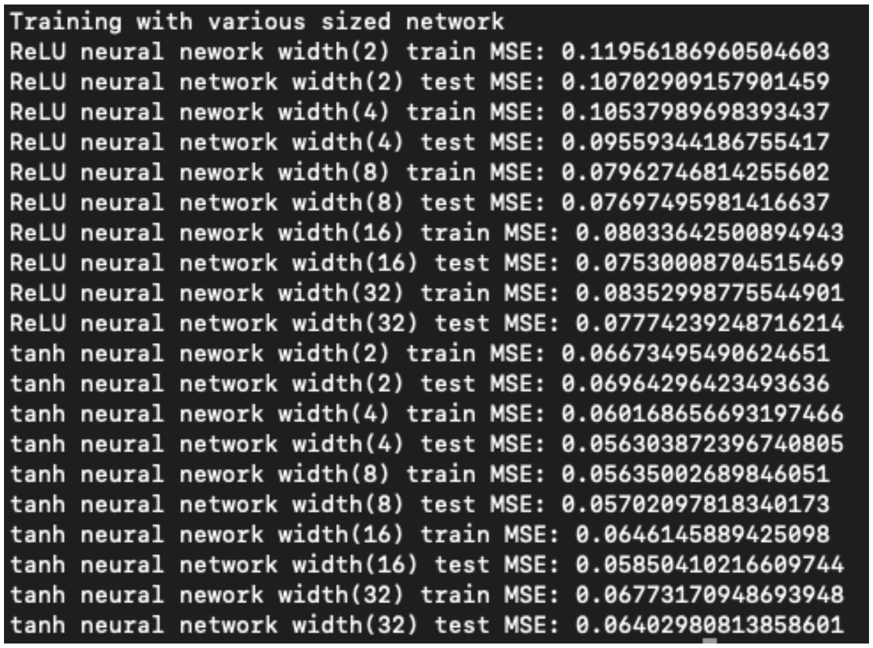
The screenshot of the output is shown on the next page.



```
Using SGD
ReLU
Batch size:  10 Epoch:  10 Train Error:  0.038940902163643945 Test Error:  0.04268340529430519
ReLU
Batch size:  10 Epoch:  20 Train Error:  0.03398623730871783 Test Error:  0.03901132256200904
ReLU
Batch size:  10 Epoch:  40 Train Error:  0.025295588106085103 Test Error:  0.032432730623558816
ReLU
Batch size:  50 Epoch:  10 Train Error:  0.04961180226095961 Test Error:  0.05089793635992317
ReLU
Batch size:  50 Epoch:  20 Train Error:  0.03616977205659915 Test Error:  0.038395452096749695
ReLU
Batch size:  50 Epoch:  40 Train Error:  0.03172142656662911 Test Error:  0.03689713204732637
ReLU
Batch size:  100 Epoch:  10 Train Error:  0.057722400026334386 Test Error:  0.059188938064516805
ReLU
Batch size:  100 Epoch:  20 Train Error:  0.051104045879460214 Test Error:  0.052136885403051274
ReLU
Batch size:  100 Epoch:  40 Train Error:  0.04615838828381433 Test Error:  0.048414862174646286
ReLU
Batch size:  200 Epoch:  10 Train Error:  0.07280394011924103 Test Error:  0.06748819617915305
ReLU
Batch size:  200 Epoch:  20 Train Error:  0.05343513678231537 Test Error:  0.05617441692849356
ReLU
Batch size:  200 Epoch:  40 Train Error:  0.051160752714599717 Test Error:  0.049774555851365754
linear
Batch size:  10 Epoch:  10 Train Error:  0.0727351601129363 Test Error:  0.07317435738647052
linear
Batch size:  10 Epoch:  20 Train Error:  0.07236665939514171 Test Error:  0.07086851335170909
linear
Batch size:  10 Epoch:  40 Train Error:  0.07000222890973087 Test Error:  0.07025739101842948
linear
Batch size:  50 Epoch:  10 Train Error:  0.08173160235681685 Test Error:  0.07859452034026554
linear
Batch size:  50 Epoch:  20 Train Error:  0.07713969813491513 Test Error:  0.07772281412470534
linear
Batch size:  50 Epoch:  40 Train Error:  0.07189231215702374 Test Error:  0.06997570643465631
linear
Batch size:  100 Epoch:  10 Train Error:  0.11341860299151152 Test Error:  0.10718976601251433
linear
Batch size:  100 Epoch:  20 Train Error:  0.08197563747409743 Test Error:  0.08293042707799692
linear
Batch size:  100 Epoch:  40 Train Error:  0.07521767884249866 Test Error:  0.0752114043512611
linear
Batch size:  200 Epoch:  10 Train Error:  0.11345466592783111 Test Error:  0.10736382416598486
linear
Batch size:  200 Epoch:  20 Train Error:  0.09794456196988321 Test Error:  0.09776371586734292
linear
Batch size:  200 Epoch:  40 Train Error:  0.08544199497860958 Test Error:  0.08444151428651339
tanh
Batch size:  10 Epoch:  10 Train Error:  0.020242870920021982 Test Error:  0.027970702580415768
tanh
Batch size:  10 Epoch:  20 Train Error:  0.016514599524955147 Test Error:  0.025819751969557304
tanh
Batch size:  10 Epoch:  40 Train Error:  0.014805041289188896 Test Error:  0.024058836727491525
tanh
Batch size:  50 Epoch:  10 Train Error:  0.028154151619177184 Test Error:  0.031755971256969745
tanh
Batch size:  50 Epoch:  20 Train Error:  0.02300981151470193 Test Error:  0.027831070175586108
tanh
Batch size:  50 Epoch:  40 Train Error:  0.01806434898885613 Test Error:  0.02543065814347185
tanh
Batch size:  100 Epoch:  10 Train Error:  0.036326582883758146 Test Error:  0.03735277852964145
tanh
Batch size:  100 Epoch:  20 Train Error:  0.03002334647779314 Test Error:  0.03365817116995559
tanh
Batch size:  100 Epoch:  40 Train Error:  0.023746500808689078 Test Error:  0.027363133818666934
tanh
Batch size:  200 Epoch:  10 Train Error:  0.056203822076940614 Test Error:  0.05516429406088538
tanh
Batch size:  200 Epoch:  20 Train Error:  0.04118625149338277 Test Error:  0.037987091413543325
tanh
Batch size:  200 Epoch:  40 Train Error:  0.03340792330228364 Test Error:  0.03441898946107204
```

(f)

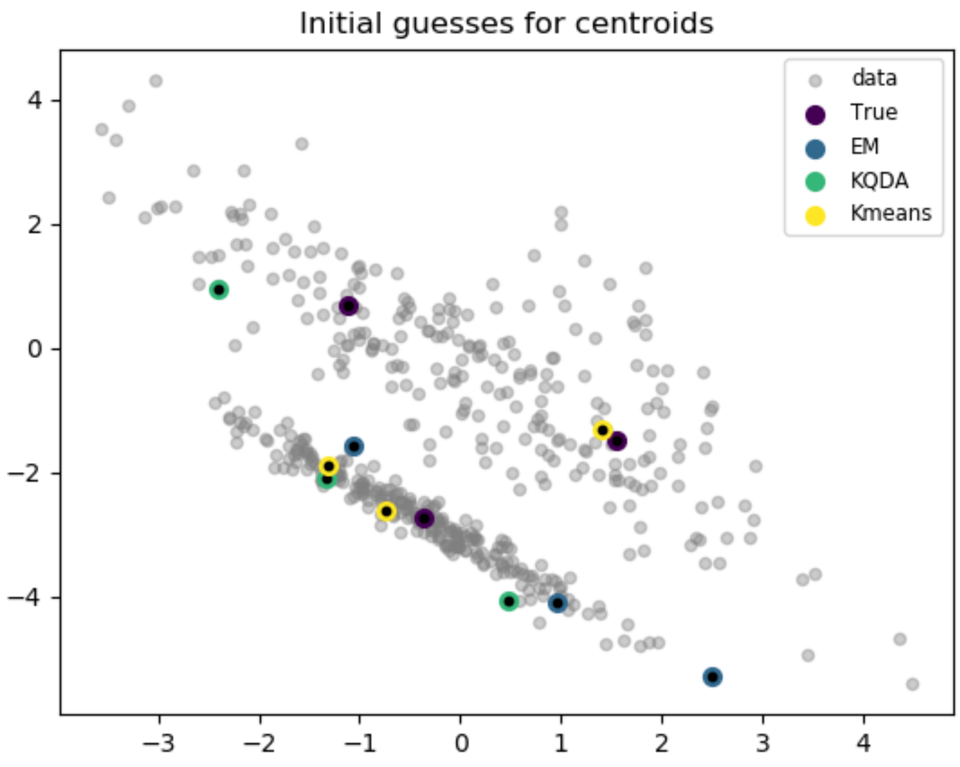
Training error result.

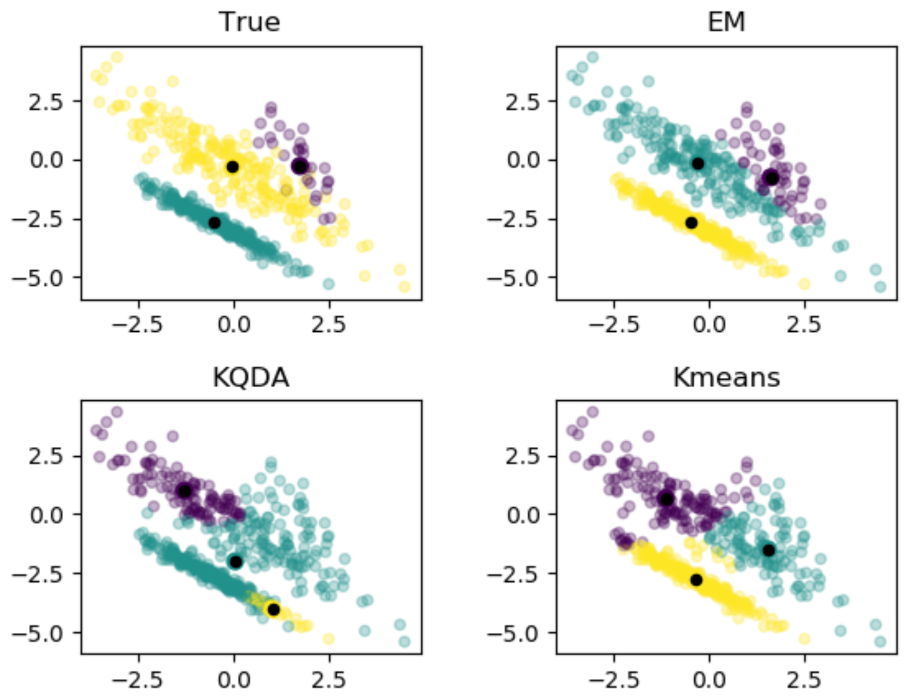


```
Training with various sized network
ReLU neural nework width(2) train MSE: 0.11956186960504603
ReLU neural network width(2) test MSE: 0.10702909157901459
ReLU neural nework width(4) train MSE: 0.10537989698393437
ReLU neural network width(4) test MSE: 0.09559344186755417
ReLU neural nework width(8) train MSE: 0.07962746814255602
ReLU neural network width(8) test MSE: 0.07697495981416637
ReLU neural nework width(16) train MSE: 0.08033642500894943
ReLU neural network width(16) test MSE: 0.07530008704515469
ReLU neural nework width(32) train MSE: 0.08352998775544901
ReLU neural network width(32) test MSE: 0.07774239248716214
tanh neural nework width(2) train MSE: 0.06673495490624651
tanh neural network width(2) test MSE: 0.06964296423493636
tanh neural nework width(4) train MSE: 0.060168656693197466
tanh neural network width(4) test MSE: 0.056303872396740805
tanh neural nework width(8) train MSE: 0.05635002689846051
tanh neural network width(8) test MSE: 0.05702097818340173
tanh neural nework width(16) train MSE: 0.0646145889425098
tanh neural network width(16) test MSE: 0.05850410216609744
tanh neural nework width(32) train MSE: 0.06773170948693948
tanh neural network width(32) test MSE: 0.06402980813858601
```

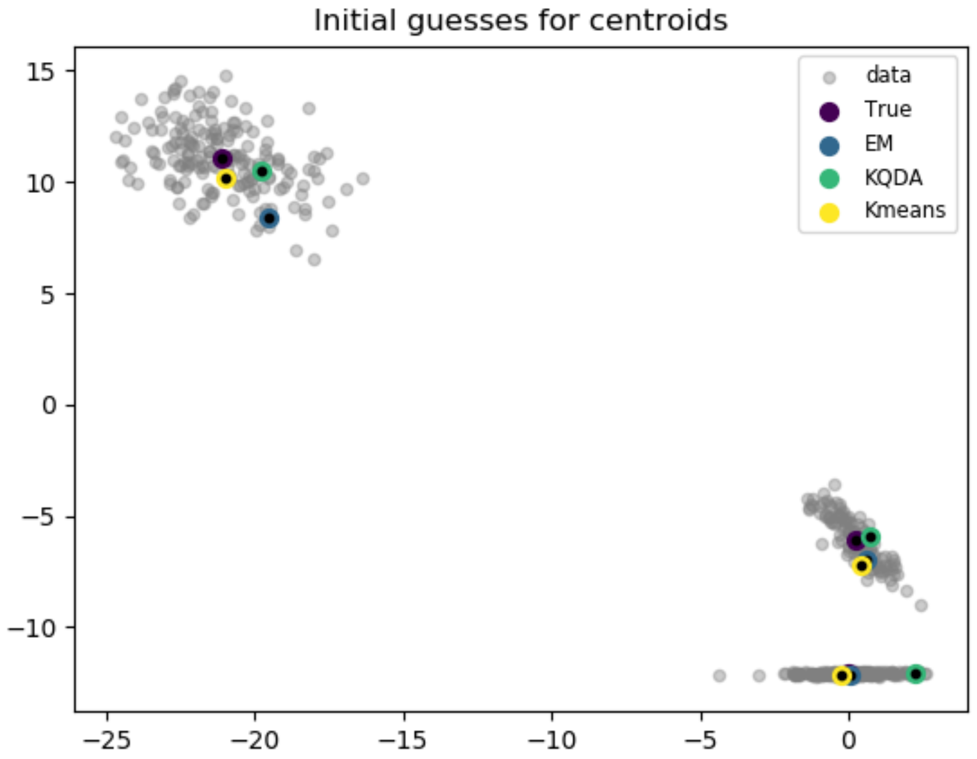
3.

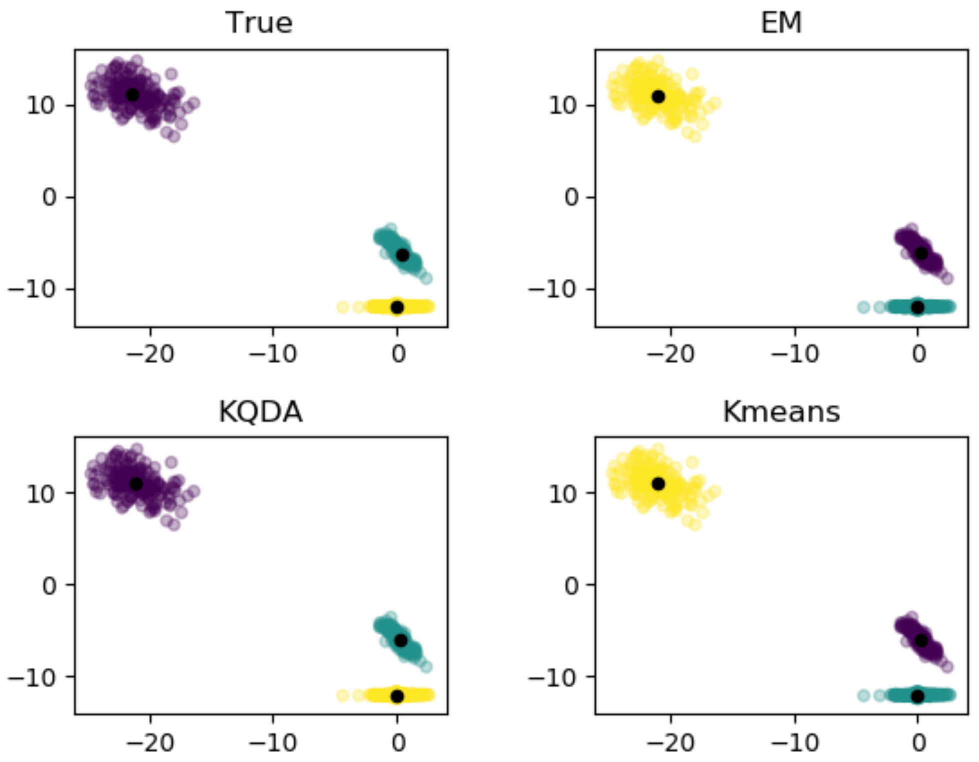
(a)




True
EM
KQDA
Kmeans
2.5
0.0
−2.5
−5.0
−2.5
0.0
2.5

(b)


Initial guesses for centroids
data
True
EM
KQDA
Kmeans
15
10
5
0
−5
−10
−25
−20
−15
−10
−5
0


True
EM
KQDA
Kmeans
10
0
−10
−20
−10
0