Exercise 1

Softmax function

$$X=[x_1,\dots,x_n]^T \mapsto \sigma(X)=[\sigma_1(X),\dots,\sigma(x_n)]$$
$$=\left[\frac{\exp(x_1)}{\sum_{i=1}^{n}\exp(x_i)},\dots,\frac{\exp(x_n)}{\sum_{i=1}^{n}\exp(x_i)}\right]^T$$

Softmax-cross-entropy loss

$$l(y,\hat{y}) := -\sum_{i=1}^{n} 1(y=\text{class } i)\log\sigma_i(\hat{y}) \qquad y,\hat{y}\in\mathbb{R}^n$$
$$= -\sum_{i=1}^{n} 1(y=\text{class } i)\log\left(\frac{e^{\hat{y}_i}}{\sum_{i=1}^{n} e^{\hat{y}_i}}\right)$$
$$= -\sum_{i=1}^{n} 1(y=\text{class } i)\left(\hat{y}_i - \log\left(\sum_{u} e^{\hat{y}_u}\right)\right)$$

i) $$\frac{\partial\sigma_i}{\partial x_j} = \frac{\partial}{\partial x_j}\left[e^{x_i}\frac{1}{\sum_{i=1}^{n} e^{x_i}}\right]$$
$$=\left[\frac{\partial e^{x_i}}{\partial x_j}\frac{1}{\sum_{i=1}^{n} e^{x_i}} + e^{x_i}\frac{\partial}{\partial x_j}\left(\frac{1}{\sum_{i=1}^{n} e^{x_i}}\right)\right]$$
$$=\left[\frac{1(i=j)e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} + e^{x_i}\left[\frac{0-e^{x_j}}{\left(\sum_{i=1}^{n} e^{x_i}\right)^2}\right]\right]$$
$$=\left[\frac{e^{x_i}1(i=j)}{\sum_{i=1}^{n} e^{x_i}} - \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}\cdot\frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}\right]$$
$$=\sigma_i(x)1(i=j)-\sigma_i(x)\sigma_j(x)$$
$$=\sigma_i(x)(1(i=j)-\sigma_j(x)) \quad \blacksquare$$

ii)

$$L(y,\hat{y}) := -\sum_{i=1}^{n} 1(y=\text{class } i)\log \sigma_i(\hat{y})$$

$$= -\sum_{i=1}^{n} 1(y=\text{class } i)\log\left(\frac{e^{\hat{y}_i}}{\sum_{i=1}^{n} e^{\hat{y}_i}}\right)$$

$$= -\sum_{i=1}^{n} 1(y=\text{class } i)\left(\hat{y}_i - \log\left(\sum_{u} e^{\hat{y}_u}\right)\right)$$

$$\frac{\partial L}{\partial \hat{y}_a} = -\sum_{i=1}^{n} y_i\left(\frac{\partial \hat{y}_i}{\partial y_a} - \frac{\frac{\partial}{\partial y_a}\left(\sum_{i=1}^{n} e^{\hat{y}_u}\right)}{\sum_{i=1}^{n} e^{\hat{y}_u}}\right)$$

$$= -\sum_{i=1}^{n} y_i\left(\delta_{ia} - \frac{e^{\hat{y}_a}}{\sum e^{\hat{y}_u}}\right)$$

$$= -y_a + \sum_{i=1}^{n} y_i \frac{e^{\hat{y}_a}}{\sum_{i=1} e^{\hat{y}_u}}$$

$$= -y_a + 1 \cdot \frac{e^{\hat{y}_a}}{\sum_{i=1} e^{\hat{y}_u}}$$

$$= -y_a + \sigma_a(\hat{y}) = \sigma(\hat{y}) - y \quad \blacksquare$$

Exercise 2

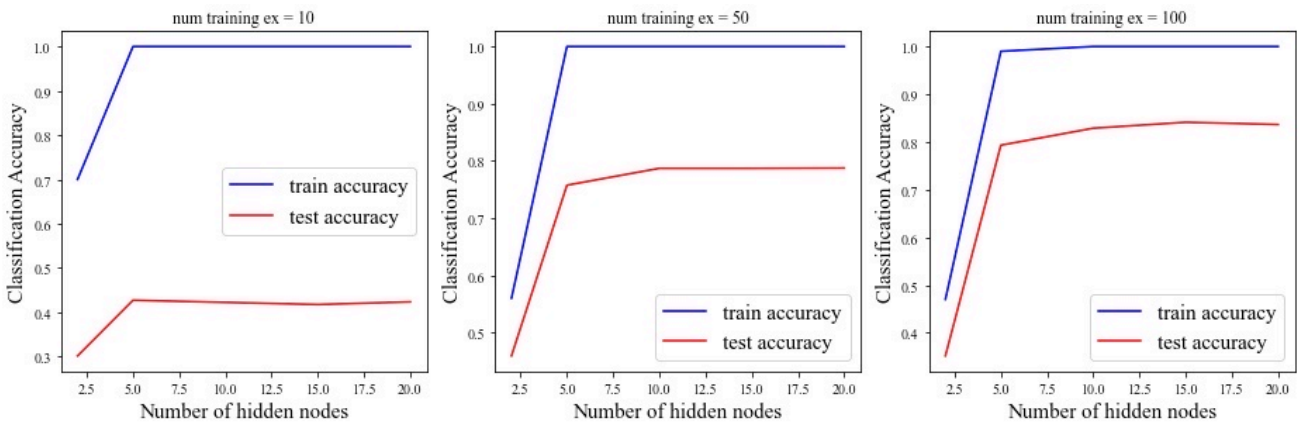
Exercise 3

i) From figure 1, we can see that when we had only 10 training examples, our train accuracy was perfect after 5 hidden nodes, but our test accuracy was less than 78%. no matter how many hidden nodes we used.

Using 50 training examples, we can see that our test accuracy reaches about 85% accuracy after 15 hidden nodes. Anything after 15 hidden nodes does not seem to improve accuracy, and at 20 hidden nodes we can see accuracy decrease a bit which may be due to overfitting.

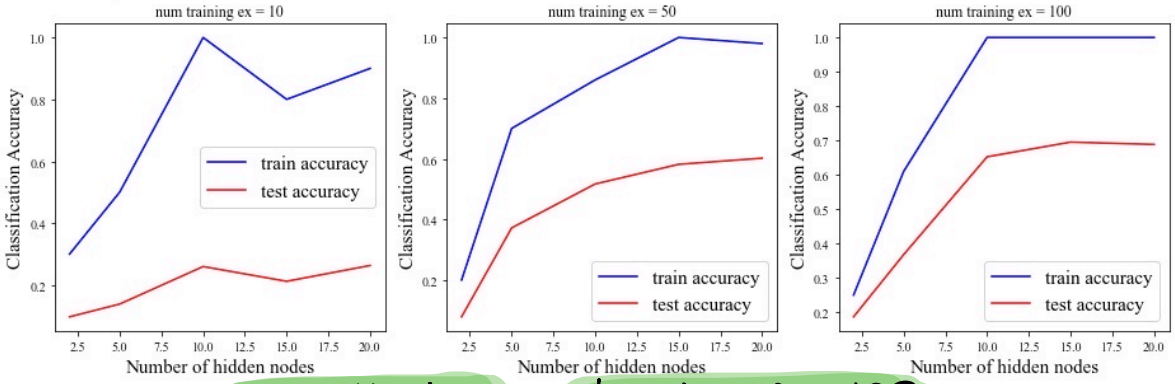
using 100 training examples, we can see about 93% accuracy at 10 hidden nodes. Adding more hidden nodes decreases accuracy, and we can see this as overfitting when: 15 < hidden nodes < 20

ii)

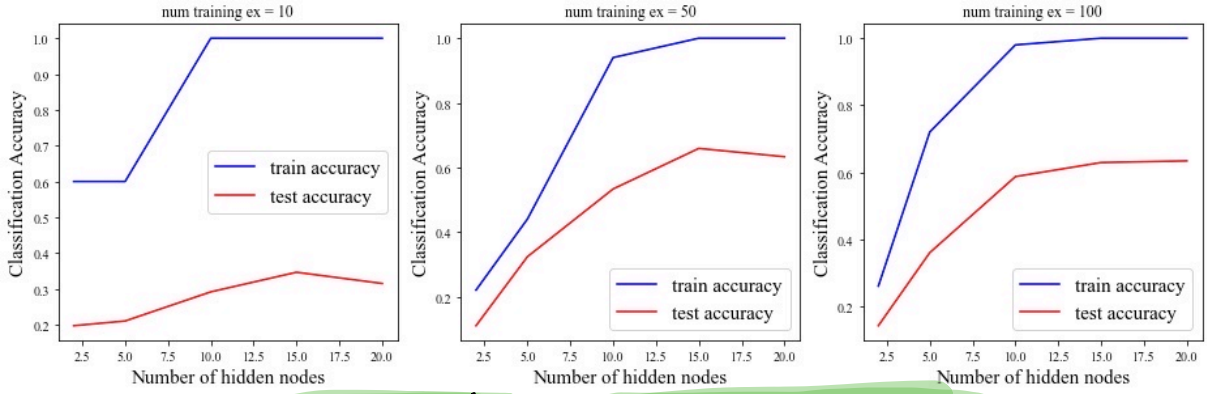


Images [5,6,7,8,9]

Our observations change a bit compared to part i) we can see that accuracy drops accross all three graphs compared to the previous set of images. This may be because this set of numbers contains digits that look much more similar compared to the first set. for 50 training examples we can see accuracy flat line after 10 hidden nodes. For 100 training examples we have that 15 is the optimal hidden nodes, w/ overfitting after that.



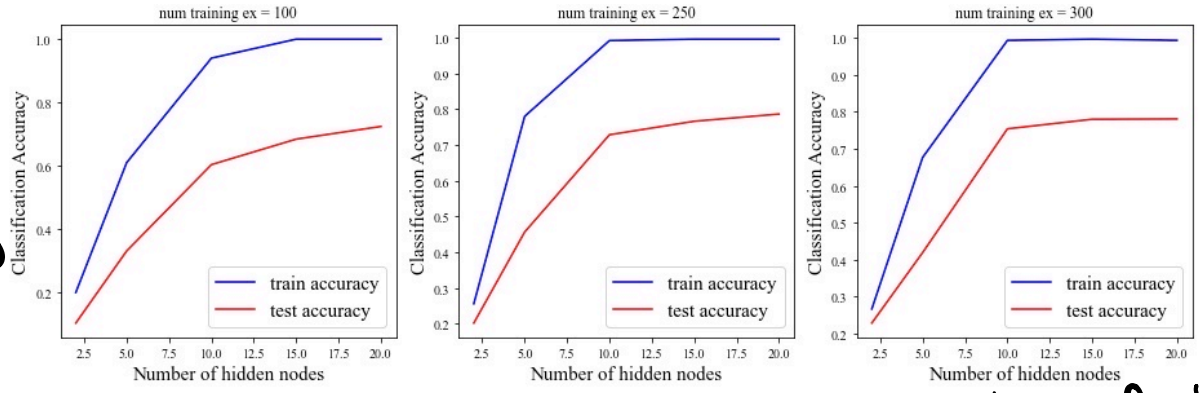
All digits iterations = 100



All digits iterations = 200

we again have very different performance compared to part i), we can also see that we don't actually reach our minimum with only 100 iterations, and re-doing our experiment with 200 iterations helped. for 50 training examples our optimal hidden nodes is 15, and for 100 training examples we see that our accuracy flatlines after 15 hidden nodes. I was curious to see if accuracy improved if we added more training examples so I ran the following:

(200 iterations)



we can see that increasing the traing examples further improves our model. But we still do not reach accuracy the accuracy we had in part i), as our model has a hard time with [5,6,7,8,9] and even harder w/ all digits.

Exercise 4

i)

```
def train(self, iterations=100, learning_rate=0.5, momentum=0.5, rate_decay=0.01, l2_norm = False, lamb = 1):
    # N: learning rate
    self.learning_rate = learning_rate
    self.momentum = momentum
    self.rate_decay = rate_decay
    error = 10
    i=0
    while (i<iterations) and (error>0.001):
        error = 0.0
        random.shuffle(self.training_data)
        for p in self.training_data:
            inputs = p[0]
            targets = p[1]
            self.feedForward(inputs)
            grad0, grad1 =self.backPropagate(targets)
            if(l2_norm == False):
            # update the weights connecting hidden to output
                self.wo -= self.learning_rate * grad1 + self.momentum * self.co
                self.co = grad1 # store current gradient

            # update the weights connecting input to hidden
                self.wi -= self.learning_rate * grad0 + self.momentum * self.ci
                self.ci = grad0 # store current gradient
            else:
                self.wo -= self.learning_rate * grad1 + self.momentum * self.co + (lamb * np.linalg.norm(self.wi))
                #self.wo += lamb*np.linalg.norm(self.wo)
                self.co = grad1 # store current gradient

            # update the weights connecting input to hidden
                self.wi -= self.learning_rate * grad0 + self.momentum * self.ci + (lamb * np.linalg.norm(self.wi))
                #self.wi += lamb*np.linalg.norm(self.wi)
                self.ci = grad0 # store current gradient

            error += (0.5) * np.linalg.norm(np.asarray(targets) - self.ao)**2

        with open('error.txt', 'a') as errorfile:
            errorfile.write(str(error) + '\n')
            errorfile.close()

        if i % 5 == 0:
            print('iteration %i, error %-.5f' % (i, error))
        # learning rate decay
        self.learning_rate = 1/(np.log(i+2) * (i+50)**(0.5))
        # self.learning_rate = self.learning_rate * (self.learning_rate / (self.learning_rate + (self.learning_rate * self.rate_decay)))

        i += 1
```