Report : Question 04

Nilesh Patil

In [109]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


filename = './results.pkl'
filehandler = open(filename, 'r')
cPickle.load(filehandler)
```

01. *Variation of layers*

In [127]:

```
names = ['layers','error_train_byEpoch','validation_error_final','train_time']
results = variation_layer

data_plot=pd.DataFrame(results,columns=names)
data_plot['layers']+=1
data_plot
```

Out[127]:

	layers	error_train_byEpoch	validation_error_final	train_time
0	1	{0: 0.4464, 1: 0.4401, 2: 0.4352, 3: 0.4343, 4...	0.4333	62.427842
1	2	{0: 0.2505, 1: 0.2335, 2: 0.1489, 3: 0.1456, 4...	0.1353	130.168888
2	3	{0: 0.0922, 1: 0.0796, 2: 0.0674, 3: 0.0611, 4...	0.0446	435.317105
3	4	{0: 0.0966, 1: 0.0849, 2: 0.0687, 3: 0.0598, 4...	0.0468	912.588439
4	5	{0: 0.1699, 1: 0.127, 2: 0.1458, 3: 0.0871, 4:...	0.0605	1359.827098

Observations :

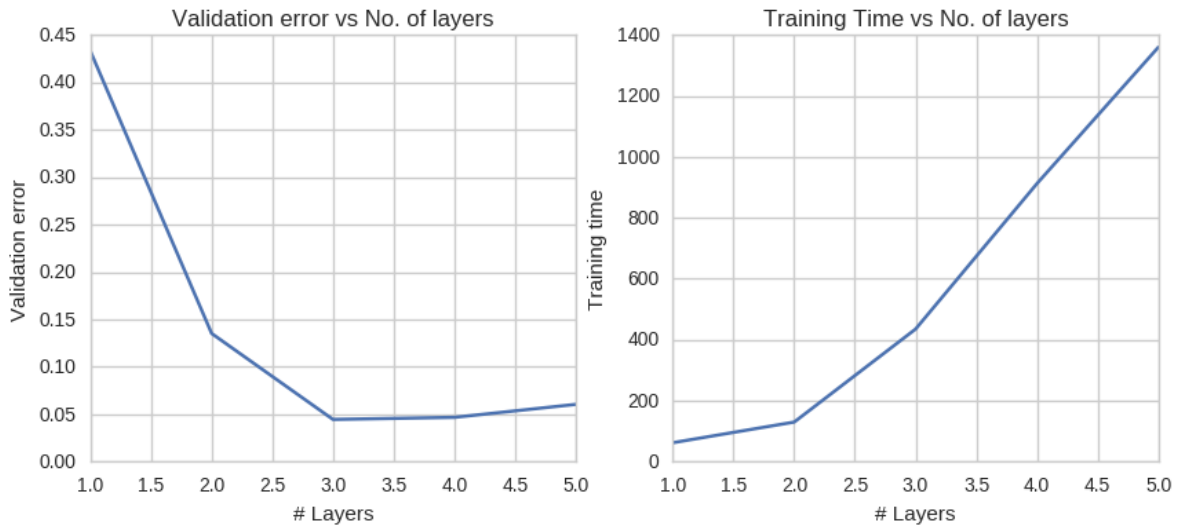
- A trivial observation is that training time increases with increase in number of layers used, because of increase in number of weights to train for.
- Validation error drops dramatically with increasing number of layers (1-3), my reasoning being that the increased complexity of additional layers leads to better approximation of the output function.
- The increase in error might be because of overfitting or because of low number of updates. In the second case, since we have a large number of weights involved, a larger number of training epochs are warranted.
- Given the high accuracy with lower training time & lesser computational complexeity obtained by the networks with just 3 layers, that seems to be a better approach than training a deeper network for larger number epochs.

In [143]:

```
sns.set_style('whitegrid')
plot = sns.plt.figure(figsize=(10,4))

plot.add_subplot(121)
sns.plt.plot(data_plot.layers,
             data_plot.validation_error_final)
sns.plt.xlabel('# Layers')
sns.plt.ylabel('Validation error')
sns.plt.title('Validation error vs No. of layers');

plot.add_subplot(122)
sns.plt.plot(data_plot.layers,
             data_plot.train_time)
sns.plt.xlabel('# Layers')
sns.plt.ylabel('Training time')
sns.plt.title('Training Time vs No. of layers');
```



.

.

02. *Variation of nodes*

In [145]:

```
names = ['nodes','error_train_byEpoch','validation_error_final','train_time']
results = variation_node

data_plot=pd.DataFrame(results,columns=names)
data_plot['nodes'] = [500,300,200,100,30]
data_plot
```

Out[145]:

	nodes	error_train_byEpoch	validation_error_final	train_time
0	500	{0: 0.6952, 1: 0.6638, 2: 0.6707, 3: 0.6445, 4...	0.4769	721.440764
1	300	{0: 0.4905, 1: 0.3571, 2: 0.3426, 3: 0.3401, 4...	0.3173	400.405412
2	200	{0: 0.7058, 1: 0.671, 2: 0.5162, 3: 0.429, 4: ...	0.4090	292.629497
3	100	{0: 0.3354, 1: 0.3269, 2: 0.3234, 3: 0.3249, 4...	0.3146	177.972748
4	30	{0: 0.1007, 1: 0.0808, 2: 0.0711, 3: 0.067, 4:...	0.0576	111.340080

Observations :

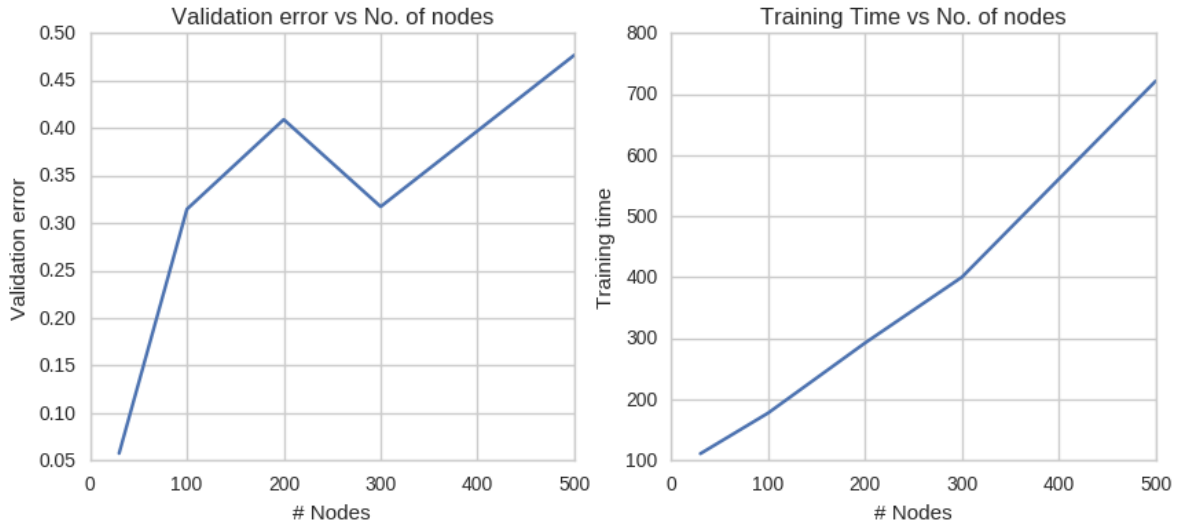
- A trivial observation is that training time increases with increase in number of nodes used, because of increase in number of weights to train for.
- Validation error is extremely low for low number of nodes in our setting, as a result of noisy combination formed when using high number of nodes in the hidden layer without appropriate training epochs. High number of nodes need longer to reach an optimal value and hence, in the current setting, their accuray is low.
- Given the high accuracy with lower training time & lesser computational complexeity obtained by the networks with just 30 neurons in hidden layer, that seems to be a better setting than training a wider network for larger number epochs.

In [147]:

```
sns.set_style('whitegrid')
plot = sns.plt.figure(figsize=(10,4))

plot.add_subplot(121)
sns.plt.plot(data_plot.nodes,
             data_plot.validation_error_final)
sns.plt.xlabel('# Nodes')
sns.plt.ylabel('Validation error')
sns.plt.title('Validation error vs No. of nodes');

plot.add_subplot(122)
sns.plt.plot(data_plot.nodes,
             data_plot.train_time)
sns.plt.xlabel('# Nodes')
sns.plt.ylabel('Training time')
sns.plt.title('Training Time vs No. of nodes');
```



.

.

.

.

.

.

.

.

.

03. *Variation of learning rate*

In [154]:

```
names = ['learning_rate','error_train_byEpoch','validation_error_final','train_time']
results = variation_learning_rate

data_plot=pd.DataFrame(results,columns=names)
data_plot['learning_rate'] = learning_rate_variation
data_plot
```

Out[154]:

	learning_rate	error_train_byEpoch	validation_error_final	train_time
0	0.0001	{0: 0.9126, 1: 0.9131, 2: 0.9131, 3: 0.9113, 4...	0.8809	805.409498
1	0.0010	{0: 0.9177, 1: 0.9076, 2: 0.8955, 3: 0.8942, 4...	0.7869	785.211745
2	0.0100	{0: 0.894, 1: 0.8636, 2: 0.833, 3: 0.8003, 4: ...	0.5037	761.278201
3	0.1000	{0: 0.6369, 1: 0.4692, 2: 0.3887, 3: 0.2898, 4...	0.0830	817.321678
4	1.0000	{0: 0.1345, 1: 0.0981, 2: 0.0882, 3: 0.0818, 4...	0.0541	774.891111
5	10.0000	{0: 0.1006, 1: 0.0811, 2: 0.0746, 3: 0.0688, 4...	0.0536	737.068849

Observations :

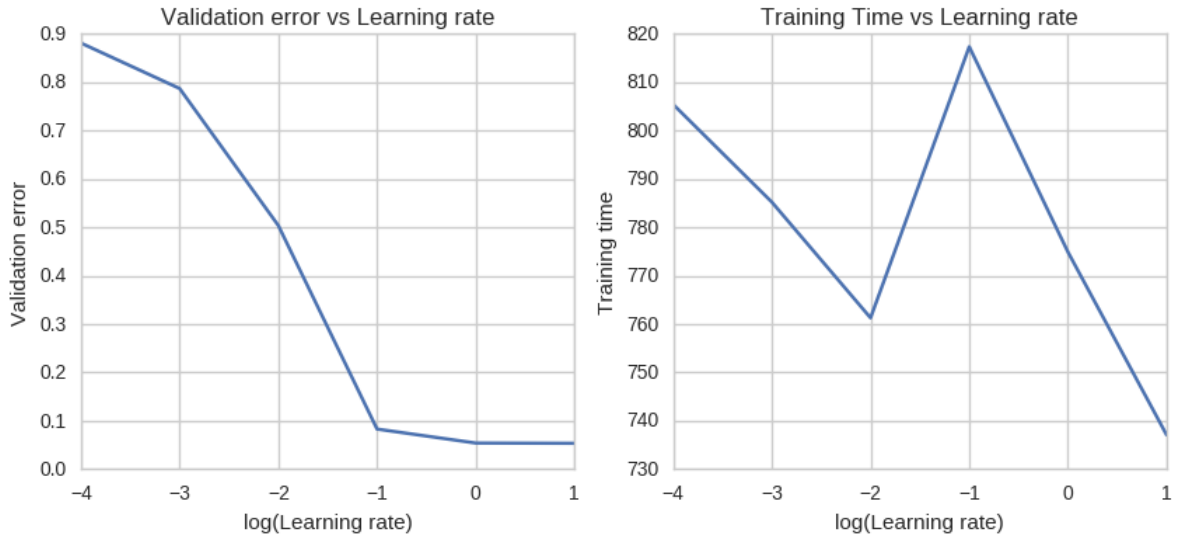
- Training time decreases with increase in learning rate because our optimizer descends faster with larger rates. FOr small rates, the effect of dWs is extremely small & hence the optimizer has to run for a long time to actually make seizable improvements to the weights.
- Validation error is low for high learning rate in the given context because of bigger strides leading to faster updates towards the minima. In geenral, larger weights might overshoot the minima value & smaller weights might take a very long time to reach the minima value & so we have to find a balance between the two.

In [158]:

```
sns.set_style('whitegrid')
plot = sns.plt.figure(figsize=(10,4))

plot.add_subplot(121)
sns.plt.plot(np.log10(data_plot.learning_rate),
             data_plot.validation_error_final)
sns.plt.xlabel('log(Learning rate)')
sns.plt.ylabel('Validation error')
sns.plt.title('Validation error vs Learning rate');

plot.add_subplot(122)
sns.plt.plot(np.log10(data_plot.learning_rate),
             data_plot.train_time)
sns.plt.xlabel('log(Learning rate)')
sns.plt.ylabel('Training time')
sns.plt.title('Training Time vs Learning rate');
```



.

.

.

.

.

.

.

.

.

04. *Variation of batch size*

In [159]:

```
names = ['batch_size','error_train_byEpoch','validation_error_final','train_time']
results = variation_batch_size

data_plot=pd.DataFrame(results,columns=names)
data_plot['batch_size'] = batch_size_variation
data_plot
```

Out[159]:

	batch_size	error_train_byEpoch	validation_error_final	train_time
0	1	{0: 0.2329, 1: 0.2343, 2: 0.1986, 3: 0.1871, 4...	0.1395	191.988162
1	10	{0: 0.088, 1: 0.0746, 2: 0.0691, 3: 0.0657, 4:...	0.0526	111.019440
2	25	{0: 0.13, 1: 0.0951, 2: 0.0832, 3: 0.0765, 4: ...	0.0616	103.521634
3	50	{0: 0.2319, 1: 0.1983, 2: 0.1854, 3: 0.1784, 4...	0.1604	100.663616
4	100	{0: 0.3845, 1: 0.242, 2: 0.2067, 3: 0.1947, 4:...	0.1682	99.761203

Observations :

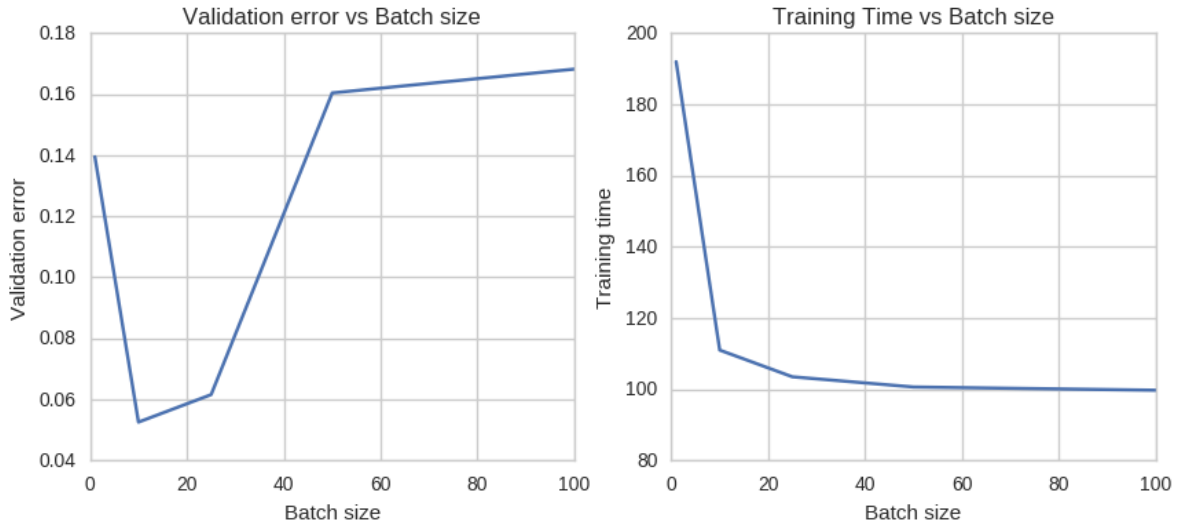
- A trivial observation is that training time increases with increase in batch size, because of decrease in total updates required.
- Error decreases sharply on increasing batch size from 1 to 10 (essentially, batch size 1 = SGD & batch size = 10 implies mini-batch gradient descent) because the gradients calcualted in this way are more stable. On further increase in batch size, error starts increasing because insufficient updates are done as the total size of input is just 10k.
- Given the high accuracy with batch_size =10, it seems to be the ideal choice.

In [160]:

```
sns.set_style('whitegrid')
plot = sns.plt.figure(figsize=(10,4))

plot.add_subplot(121)
sns.plt.plot(data_plot.batch_size,
             data_plot.validation_error_final)
sns.plt.xlabel('Batch size')
sns.plt.ylabel('Validation error')
sns.plt.title('Validation error vs Batch size');

plot.add_subplot(122)
sns.plt.plot(data_plot.batch_size,
             data_plot.train_time)
sns.plt.xlabel('Batch size')
sns.plt.ylabel('Training time')
sns.plt.title('Training Time vs Batch size');
```



.

.

.

.

.

.

.

.

.

05. *Variation of epochs*

In [161]:

```
names = ['epochs','error_train_byEpoch','validation_error_final','train_time']
results = variation_epoch

data_plot=pd.DataFrame(results,columns=names)
data_plot['epochs'] = epoch_variation
data_plot
```

Out[161]:

	epochs	error_train_byEpoch	validation_error_final	train_time
0	1	{0: 0.1766}	0.1746	11.330550
1	10	{0: 0.0948, 1: 0.0801, 2: 0.0701, 3: 0.0661, 4...	0.0522	110.964714
2	25	{0: 0.0906, 1: 0.0764, 2: 0.0727, 3: 0.0698, 4...	0.0524	276.842178
3	50	{0: 0.0912, 1: 0.078, 2: 0.0734, 3: 0.0699, 4:...	0.0477	553.567445
4	100	{0: 0.0895, 1: 0.0807, 2: 0.0753, 3: 0.0707, 4...	0.0439	1113.654386

Observations :

- A trivial observation is that training time increases is a directly proportional to epochs, because the train method is nalled n(epochs) times.
- Large value for epochs generally lead to lower error rate on validation set. This is because of more optimisation runs in the train phase.
- For a given problem, a reasonable amount of epochs can be decided by checking difference in accuracy for each successive epoch & stopping when the accuracy-gain is below a pre-defined threshold.

In [163]:

```
sns.set_style('whitegrid')
plot = sns.plt.figure(figsize=(10,4))

plot.add_subplot(121)
sns.plt.plot(data_plot.epochs,
             data_plot.validation_error_final)
sns.plt.xlabel('# epochs')
sns.plt.ylabel('Validation error')
sns.plt.title('Validation error vs No. of epochs');

plot.add_subplot(122)
sns.plt.plot(data_plot.epochs,
             data_plot.train_time)
sns.plt.xlabel('# epochs')
sns.plt.ylabel('Training time')
sns.plt.title('Training Time vs No. of epochs');
```