Deep Learning (CS 470, CS 570)

Module 3, Lecture 3: MLP Implementation, Hyperparameter Tuning

TensorFlow and Keras

TensorFLow:

- An open source library developed by Google for dataflow programming
    - Use data flow graph for computing, where a node is a mathematical operation and an edge is a Tensor.
    - Tensor: multidimensional array of data
- Mainly used for ML application such as ANN
- One of the most widely used framework for ML
- Fast to execute but little difficult for beginner programmer

Keras:

- A high level neural network library/package/API build on TensorFlow/ CNTK, and Theano
- Easy to code therefore allows fast prototyping of ML models
- Execution is slightly slow compared to TensorFlow but implementation is beginner friendly

MLP for Digit Classification

Import packages:

```
[ ]  # TensorFlow and tf.keras
     import tensorflow as tf
     from tensorflow import keras

     # Helper libraries
     import numpy as np
     import matplotlib.pyplot as plt

     print(tf.__version__)
```

```
2.3.0
```

MLP for Digit Classification

Import dataset and print the data information:

```
# load dataset
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()

print("Shape of the training dataset, number of images and resolution:", train_images.shape)
print("All distinct training labels:", np.unique(train_labels))
```

```
Shape of the training dataset, number of images and resolution: (60000, 28, 28)
All distinct training labels: [0 1 2 3 4 5 6 7 8 9]
```

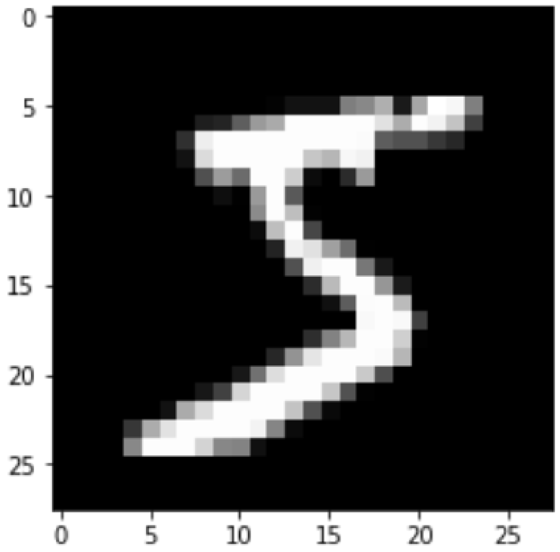
How many nodes in the input and output layers?

MLP for Digit Classification

Sample training image and its label:

```
plt.figure()
plt.imshow(train_images[0], cmap='gray')
#plt.colorbar()
plt.grid(False)
plt.suptitle('First Training Image', fontsize=16)
plt.show()
print("First training label: ", train_labels[0])
```

First Training Image




```
First training label:  5
```

MLP for Digit Classification

MLP architecture:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)
```

Reshape input image as a vector and feed the input node.

Dense hidden layer

Output layer

Adaptive Moment Estimation (Adam) is a gradient descent optimization technique

This is a variation of loss function (E). We will cover it in a future class.

Train the MLP model. '**epochs**' are number of times the training dataset is revisited during the training process. One more important parameter is '**batch_size**' (not used here). This determines the batch size for mini batch gradient descent technique.

How many connections between input and hidden layers?

MLP for Digit Classification

Training accuracies:

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.2922 - accuracy: 0.8494
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3573 - accuracy: 0.9120
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2667 - accuracy: 0.9318
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2373 - accuracy: 0.9395
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2193 - accuracy: 0.9431
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2081 - accuracy: 0.9479
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1994 - accuracy: 0.9509
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1920 - accuracy: 0.9530
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1846 - accuracy: 0.9550
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1856 - accuracy: 0.9558
<tensorflow.python.keras.callbacks.History at 0x7fd6e2ac35f8>
```

MLP for Digit Classification

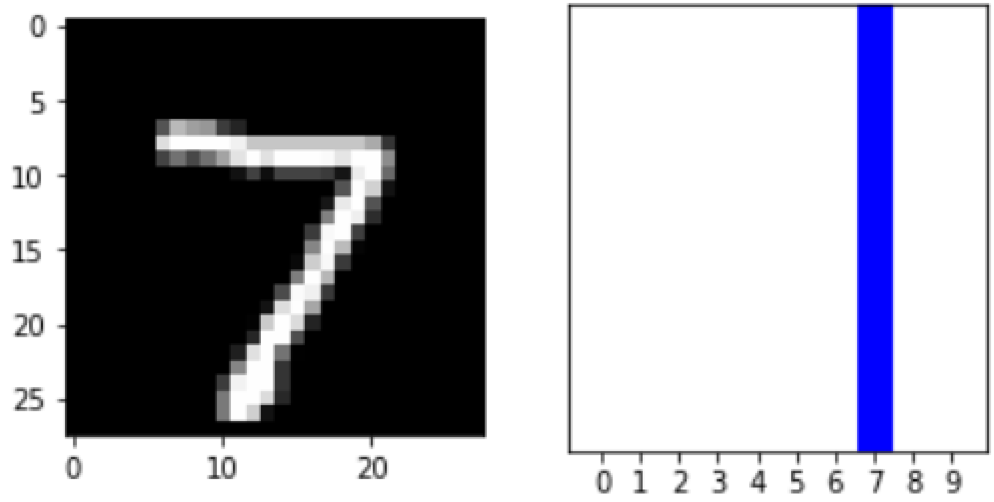
Validation of the test dataset:

```
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  #plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')

probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plt.imshow(test_images[i], cmap='gray')
#plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()

print(predictions[i])
```

Predictions on the test dataset. a softmax layer to converts the model’s output to probabilities

MLP for Digit Classification

A sample prediction on a test image:




```
[7.5201165e-31 3.1707087e-12 3.3092821e-12 4.7458649e-08 2.4188384e-19
 4.0565565e-13 9.1595503e-17 1.0000000e+00 1.5684971e-25 1.6122577e-13]
```

Hyperparameter Tuning

What!

- Hyperparameters are associated with model architecture
- Example hyperparameters for a MLP are
  - Number of hidden layers
  - Number of nodes in each hidden layer
  - Activation functions
  - Learning rate
  - Number of epochs

Why & How!

- Depending on the dataset complexity, number of training data, problem definition, different model architecture provides different results
- The goal is to select the best model architecture for a given problem
- Hyperparameter tuning technique allows to select the best hyperparameters combination based on validation accuracy
- Grid search along with cross-validation are often used for hyperparameter tuning

Hyperparameter Tuning: Grid Search

Grid search: Brute force search for all hyperparameter combinations and select the combinations with best accuracy/lowest error rate.

Consider a possible list of hyperparameters and the possible values each parameter can take as given below. In reality the number of hyperparameters and possible values of each hyperparameter can be large.

Hyperparameters:

- Number of hidden layers: 3, 4
- Number of nodes in each layer: 32, 64
- Activation functions: ‘sigmoid’, ‘relu’

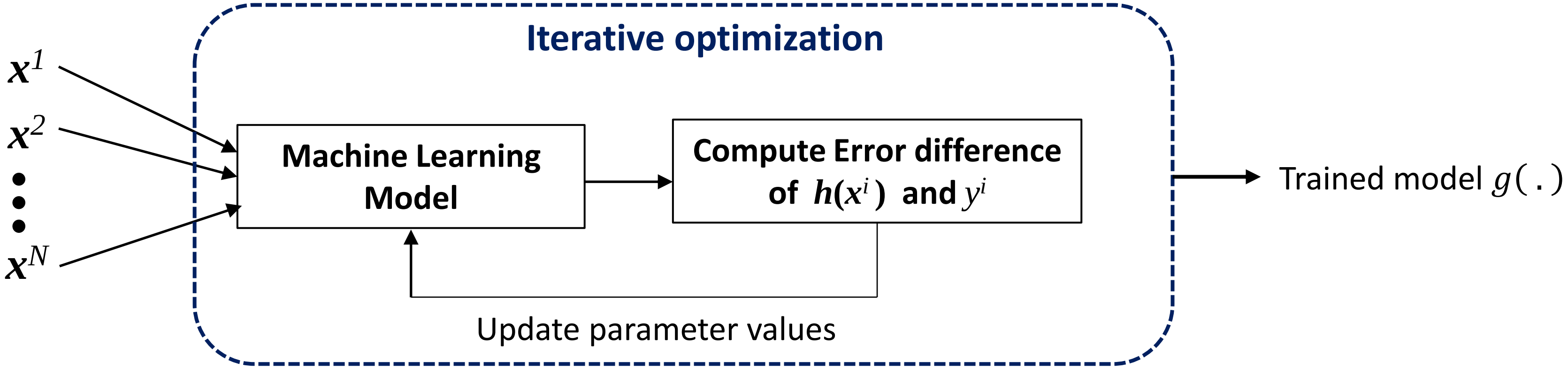
All possible combinations of hyperparameters:

1. [3, 32, ‘sigmoid’]
2. [3, 32, ‘relu’]
3. [3, 64, ‘sigmoid’]
4. [3, 64, ‘relu’]
5. [4, 32, ‘sigmoid’]
6. [4, 32, ‘relu’]
7. [4, 64, ‘sigmoid’]
8. [4, 64, ‘relu’]

Grid search goes through each of the combinations, and compute model accuracy/lowest error rate.
Finally the combination with the best result is selected as the final combination

Training, Validation, and Test Set

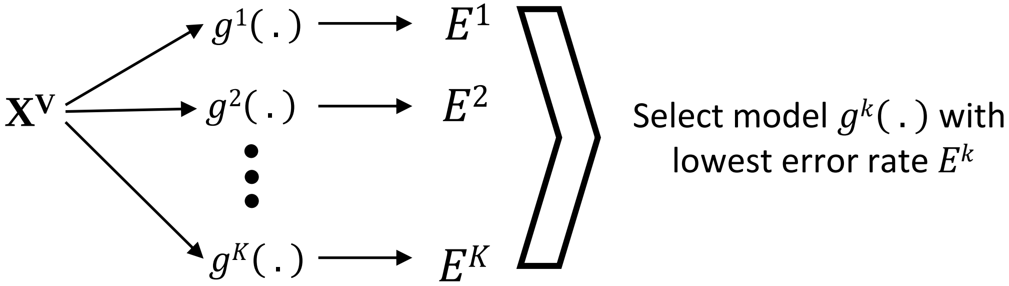
Training set: Model parameters are optimized using training data to produce best possible accuracy.



Validation set: The performance of different machine learning models and their hyper-parameters are compared based on their accuracies on the validation set. The best performing model and hyper-parameter combination are selected as the final classifier.

Validation data

$$\mathbf{X}^{\mathbf{V}} = [\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{N_V}]$$



Test set: An unbiased set for final validation of $g^k(.)$

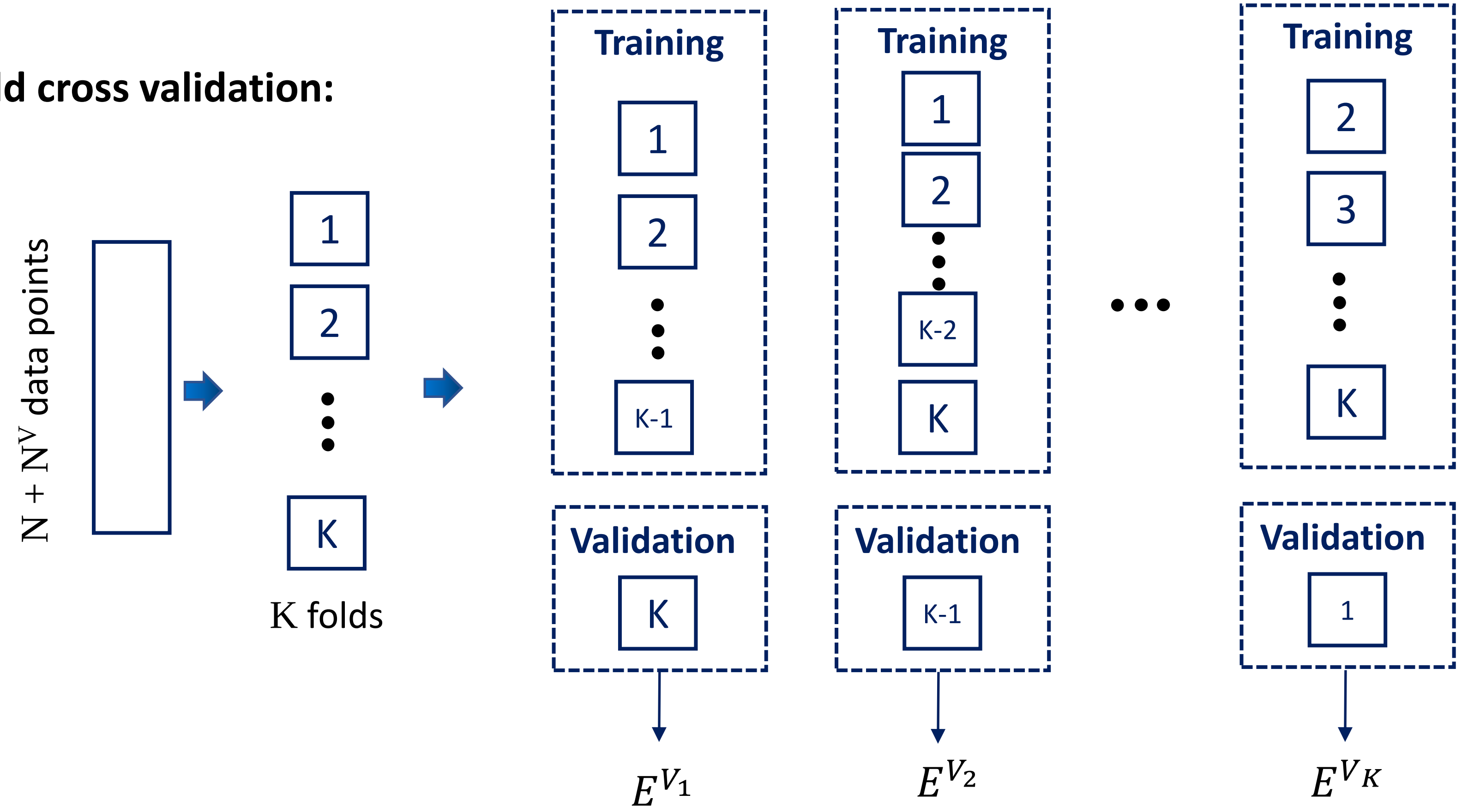
Cross Validation

We have a limited training set with $N + N^V$ data points.

If we increase N + N^V will decrease, and vise versa

We want big N and big N^V

K fold cross validation:



$$E^V = \frac{1}{K}\sum_{k=1}^{K} E^{V_k}$$

Additional Readings

[Cross validation]