Lecture 8

Neural Networks

Neural Networks

➢ Properties?

- Uni-direction
- Convergence and divergence
- Summation
- Excitation or inhibition

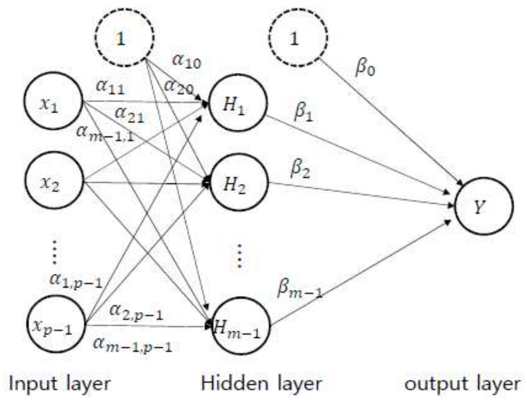
< neuron synapse image: Google >

Neural Networks

- Networks
  - Networks consists of multiple layers
  - Each layer consists of a set of nodes
  - Nodes are linked to other nodes in another layer

Neural Networks

- Structure
  - A neural network model with a single hidden layer



Neural Networks

- Modeling
  - we have:

$$Y = \beta_0 + \beta_1 H_1 + \cdots + \beta_{m-1} H_{m-1}$$

  - and:

$$H_j = \alpha_{j0} + \alpha_{j1} X_1 + \cdots + \alpha_{j,p-1} X_{p-1}$$

Neural Networks

- Activation functions
  - Identity activation function: $\sigma(z) = z$
  - Logistic (sigmoid) activation function: $\sigma(z) = \begin{cases} 1 & , z \to \infty \\ 0 & , z \to -\infty \end{cases}$
  - Softmax activation function: $\sigma(z) = \frac{\exp(z)}{\sum_{j=1}^{n} \exp(z_j)}$

Neural Networks

- Activation functions
  - ReLU activation function: $\sigma(z) = \max(0, z)$
  - Leaky ReLU activation function: $\sigma(z) = \max(\alpha z, z)$
  - Tanh activation function: $\sigma(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \mathrm{ex}\ \ (-z)}$

Neural Networks

- Modeling
  - These expressions can be generalized using an activation function.

$$Y = \sigma(\beta_0 + \sum_{j}^{m-1} \beta_j H_j)$$

$$H_j = \sigma(\alpha_{j0} + \sum_{k}^{p-1} \alpha_{jk} X_k)$$

  - $\sigma(z)$: an activation function.

Neural Networks

- Some useful loss functions
  - SSE (Sum of Squared Error)

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

  - MSE (Mean Squared Error)

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Neural Networks

- Some useful loss functions
  - Categorical Cross Entropy

$$CCE = \sum_{x \in \chi} p(x) \log(q(x))$$

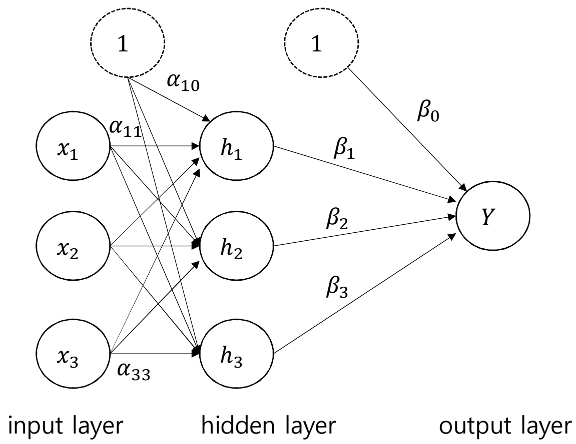
  - Kullback-Leibler (KL) divergence

$$D_{KL}(P||Q) = \sum_{x \in \chi} p(x) \log(\frac{P(x)}{Q(x)})$$

Practice

- Toy example

x1	x2	x3	Y
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0



➢ Dataset

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

```
In [4]: x = np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
        y = np.array([[0],[1],[1],[0]])

        print("The shape of X is ", x.shape)
        print("The shape of Y is ", y.shape)

        The shape of X is  (4, 3)
        The shape of Y is  (4, 1)
```


➢ Define the activation and the loss function

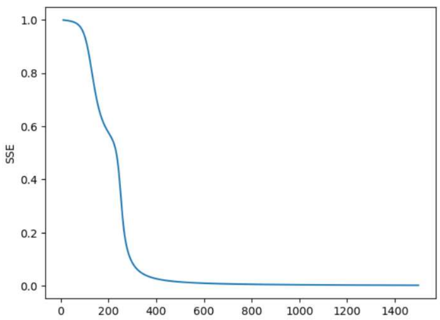
```
In [5]: def act_sigmoid(z):
            return 1/(1+np.exp(-z))

        def grad_sigmoid(z):
            return z*(1-z)

        def sse(y, output):
            return np.sum(np.power(y-output,2))
```

➢ Working with a neural network

```
In [8]: class nn_toy:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.alphas = np.random.rand(self.x.shape[1],4)
        self.betas = np.random.rand(4,1)
        self.output = np.zeros(self.y.shape)

    def forward_process(self):
        self.H = act_sigmoid(np.dot(self.x, self.alphas))
        self.output = act_sigmoid(np.dot(self.H, self.betas))

    def backprop_process(self):
        grad_betas = np.dot(self.H.T, (2*(self.y-self.output)*grad_sigmoid(self.output)))
        grad_alphas = np.dot(self.x.T, (np.dot(2*(self.y-self.output)*grad_sigmoid(self.output),
                                               self.betas.T)*grad_sigmoid(self.H)))
        self.alphas +=grad_alphas
        self.betas +=grad_betas
```


➢ Running...

```
In [20]: my_data = nn_toy(x,y)

for i in range(1500):
    my_data.forward_process()
    my_data.backprop_process()
    print("SSE:", sse(y, my_data.output))
```




```
SSE: 1.3633628732530294
SSE: 1.0620406646638603
SSE: 0.997403335270371
SSE: 0.9968249679842298
SSE: 0.9966096453576148
SSE: 0.9964027653381119
SSE: 0.9961906927675295
SSE: 0.9959725883569573
SSE: 0.995748074932304
SSE: 0.9955167780805947
SSE: 0.995278308405698
SSE: 0.995032259904416
SSE: 0.9947782089482315
SSE: 0.9945157132258386
SSE: 0.994244310633263
```

➢ Predicted probability

```
In [21]: print("_______________________________")
         print("+          predicted          +")
         print("_______________________________")
         print(my_data.output)
```

```
_______________________________
+          predicted          +
_______________________________
[[0.02201119]
 [0.97607162]
 [0.98517954]
 [0.02044553]]
```

Y
0
1
1
0

True values of Y

➢ tensorflow.keras

- API for deep learning consisting of modular units
  - -. .models: a module for models
  - -. .layers: a module for layers
  - -. .optimizers: a module for built-in optimizers
  - -. .activations: a module for built-in activation functions
  - -. .losses: a module for loss functions

Building a neural network

- we need models.Sequential() and layers.Dense().

-. models.Sequential()
: helps to construct a model by stacking layers sequentially

-. .layers.Dense()
: Dense oprates element-wise activation function as:

$$\sigma(\sum_{i=1}^{K} \beta_i \cdot x_i + b_l)$$

Building a neural network

-..fit(x, y, epochs, verbose)
x, y: x and y in arrays
epochs: training epochs
verbose =0 or 1 or 2 : 0 is silent, 1 shows progress bar,
2 as a single line per each epoch

-. .compile(loss, optimizer)
loss: loss functions such as 'mse', 'categorical_crossentropy'
optimizer: training algorithm such as 'sgd', 'adam'

- Hand-written digits Dataset (Kaynak, C. 1995)
  - -. we can use "digits" dataset for classification.
  - -. we can use a dataset from *keras*

In [2]:
```
from keras.datasets import mnist
train_set, test_set = mnist.load_data()
x_train, y_train = train_set
x_test, y_test = test_set
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```


- Observation by the label

In [4]:
```
fig, ((ax1, ax2, ax3, ax4, ax5), (ax6, ax7, ax8, ax9, ax10)) = plt.subplots(2,5, figsize=(10,5))

for idx, ax in enumerate([ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10]):
    for i in range(1000):
        if y_test[i]== idx:
            ax.imshow(x_test[i],cmap='gray')
            ax.grid(False)
            ax.set_xticks([])
            ax.set_yticks([])
            break

plt.tight_layout()
plt.show()
```

➢ Observation

➢ Data reshaping and normalization

```
In [9]: print(x_train.shape, x_test.shape)

(60000, 28, 28) (10000, 28, 28)

In [10]: x_train_vec = x_train.reshape((x_train.shape[0], x_train.shape[1]*x_train.shape[2]))
         x_test_vec = x_test.reshape((x_test.shape[0], x_test.shape[1]*x_test.shape[2]))

         print("The dim of x_train: ", x_train_vec.shape)
         print("The dim of x_test: ", x_test_vec.shape)

The dim of x_train:  (60000, 784)
The dim of x_test:  (10000, 784)

In [6]: x_train_vec = x_train_vec/255.
        x_test_vec = x_test_vec/255.
```

➢ Label?

```
In [3]: y_train

Out[3]: array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
In [12]: from keras.utils.np_utils import to_categorical

         y_train = to_categorical(y_train, 10)
         y_test = to_categorical(y_test, 10)
```

```
In [13]: y_train

Out[13]: array([[0., 0., 0., ..., 0., 0., 0.],
                [1., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```


➢ Constructing a NN using API (1)

```
In [4]: import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense

        num_classes=10
        my_model = Sequential()
        my_model.add(Dense(units = 20, input_shape =(x_train_vec.shape[1],), activation='relu'))
        my_model.add(Dense(units = num_classes, activation='softmax'))
```

➤ Summary()

In [5]:
```
my_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                15700

 dense_1 (Dense)             (None, 10)                210

=================================================================
Total params: 15910 (62.15 KB)
Trainable params: 15910 (62.15 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```


➤ Constructing a NN using API (2)

In [20]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.optimizers import SGD, Adam, RMSprop
```

In [22]:
```
num_classes=10
my_model2 = Sequential()
my_model2.add(Dense(units = 512, input_shape =(x_train_vec.shape[1],)))
my_model2.add(Activation('relu'))
my_model2.add(Dropout(0.2))
my_model2.add(Dense(units = 512))
my_model2.add(Activation('relu'))
my_model2.add(Dropout(0.2))
my_model2.add(Dense(units = num_classes))
my_model2.add(Activation('softmax'))
```

In [23]:
```
my_model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 512)               401920

 activation_3 (Activation)   (None, 512)               0

 dropout_2 (Dropout)         (None, 512)               0

 dense_4 (Dense)             (None, 512)               262656

 activation_4 (Activation)   (None, 512)               0

 dropout_3 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 10)                5130

 activation_5 (Activation)   (None, 10)                0

=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```


➢ Compiling and fitting (my_model 2)

In [24]:
```
my_model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
my_model2.fit(x_train_vec, y_train, epochs =10, verbose=1)
```

```
Epoch 1/10
1875/1875 [==============================] - 23s 12ms/step - loss: 0.2120 - accuracy: 0.9354
Epoch 2/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.1058 - accuracy: 0.9676
Epoch 3/10
1875/1875 [==============================] - 22s 12ms/step - loss: 0.0788 - accuracy: 0.9751
Epoch 4/10
1875/1875 [==============================] - 23s 12ms/step - loss: 0.0635 - accuracy: 0.9798
Epoch 5/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0568 - accuracy: 0.9826
Epoch 6/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0495 - accuracy: 0.9843
Epoch 7/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0467 - accuracy: 0.9855
Epoch 8/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0403 - accuracy: 0.9873
Epoch 9/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0391 - accuracy: 0.9880
Epoch 10/10
1875/1875 [==============================] - 21s 11ms/step - loss: 0.0384 - accuracy: 0.9879
```

Out[24]: <keras.src.callbacks.History at 0x1d9617f7f90>

➢ Prediction

```
In [25]: output = my_model2.predict(x_test_vec)

313/313 [==============================] - 1s 3ms/step

In [26]: output[0,:]

Out[26]: array([4.5980282e-14, 7.1095255e-11, 2.2075770e-10, 7.0837801e-11,
                2.3529572e-11, 1.1822084e-13, 1.8544796e-21, 1.0000000e+00,
                1.8453263e-14, 8.8900443e-09], dtype=float32)
```