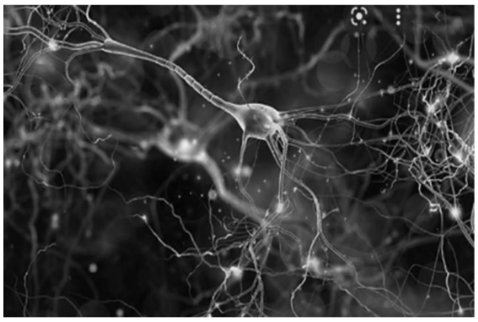
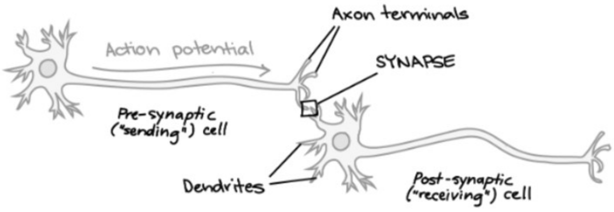
Neural Networks

- Electrical or chemical transmission of Brain cells





Neural Networks

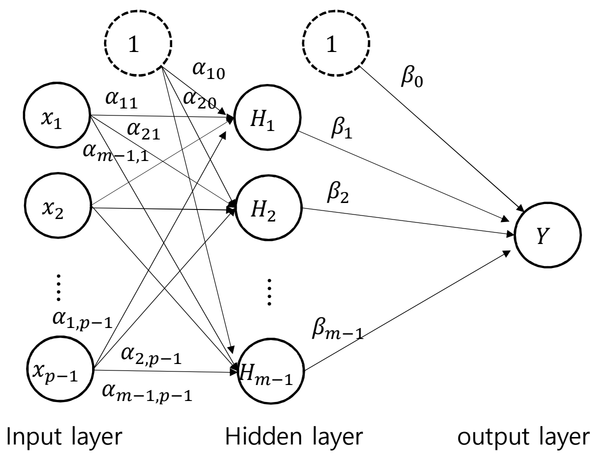
- Properties of Synapse
  - Uni-directional conduction: impulse by neurotransmitter gets conducted from pre to post synaptic region.
  - Convergence and divergence: different number of nerve fibers between pre and post-synaptic region
  - Summation: stimuli can get added up to develop action potential at post-synaptic region
  - Excitation or inhibition: conduction can either stimulate or inhibit activity at postsynaptic region

Neural Networks

- Networks
  - Node: a connection point or a vertex (neurons)
  - Edge: a link of a network (synapse)
  - Layer : a place where a set of nodes are placed

Neural Networks

- Network Structure
  - Single hidden layer neural network model



Neural Networks

- Expression
  - We have:

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

and:

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

  - Can we combine these two? By ‘Feedforward’ process!

한양대학교 HANYANG UNIVERSITY

Neural Networks

- Expression

$$Y = \left[\beta_0 + \sum_{j=1}^{m-1} \beta_j \alpha_{j0}\right] + \left[\sum_{j=1}^{m-1} \beta_j \alpha_{j1}\right] \cdot X_1 + .. + \left[\sum_{j=1}^{m-1} \beta_j \alpha_{j,p-1}\right] \cdot X_{p-1}$$

  - Can it always be expressed as this?

한양대학교 HANYANG UNIVERSITY

Neural Networks

- Expression
  - The expression can be generalized using activation functions.

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

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

  - Here, σ(z) is activation function.

Neural Networks

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

한양대학교 HANYANG UNIVERSITY

Neural Networks

- Some activation functions for NNs

  - ReLU (Rectifier Linear Unit) activation : $\sigma(z) = \max(0, z)$

  - Leaky ReLU activation : $\sigma(z) = \max(\alpha z, z)$

  - Tanh activation : $\sigma(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$

한양대학교 HANYANG UNIVERSITY

Neural Networks

- Computation

  - SSE (sum of squared error)

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

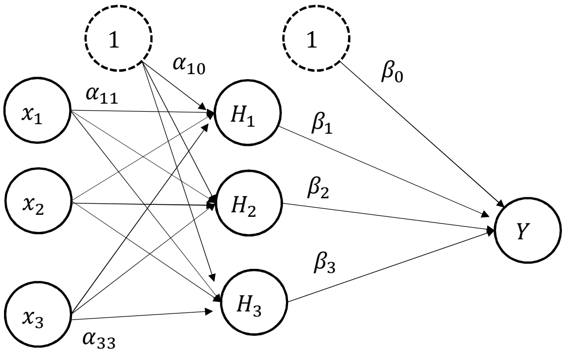
  - Feedforward process
    : Compute $\hat{y}_i$

  - Backpropagation process
    : Update coefficients using chain rule!

한양대학교 HANYANG UNIVERSITY

Neural Networks

- Practice: (Toy)



Neural Networks

- Practice: (Toy)

```
In [24]: x= np.array([[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
         y = np.array([[0],[1],[1],[0]])

         toys = nn_toy(x, y)

         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)
```

Neural Networks

- Practice

```
In [27]: def sig_act(z):
             return 1/(1+np.exp(-z))

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

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

Neural Networks

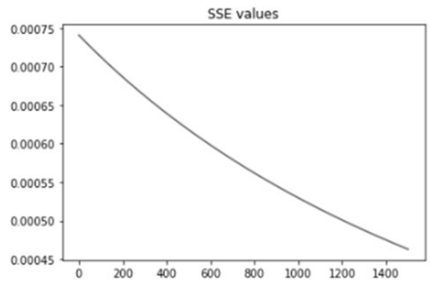
- Practice

```
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 ff_process(self):
        self.H = sig_act(np.dot(self.x, self.alphas))
        self.output = sig_act(np.dot(self.H, self.betas))

    def bp_process(self):
        d_betas = np.dot(self.H.T, (2*(self.y - self.output)*d_sig_act(self.output)))
        d_alphas = np.dot(self.x.T, (np.dot(2*(self.y - self.output)*
                                    d_sig_act(self.output), self.betas.T)*
                               d_sig_act(self.H)))

        self.alphas += (d_alphas)
        self.betas += (d_betas)
```

Neural Networks

- Practice

```
In [33]: for i in range(1500):
             toys.ff_process()
             toys.bp_process()
             print("SSE:", sse(y, toys.output))

         print("++++++++++++++++++++++")
         print(toys.output)
```



Neural Networks

- Practice

```
In [14]: print("======== predicted ================")
         print(toys.output)

======== predicted ================
[[0.00899192]
 [0.97101736]
 [0.97090606]
 [0.03590682]]
```

  - True Y values are [0, 1, 1, 0]

Neural Networks

tensorflow.keras
: modular units, API for deep learning

-. *.models: model API*
-. *.layers: layers API*
-. *.optimizers : built-in optimizer classes*
-. *.activations : built-in activation classes*
-. *.losses : built-in loss classes*

Neural Networks

models.Sequential() *: groups a linear stack of layers into a Model*
layers.Dense() *: fully connected layers*

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

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

Neural Networks

- Practice : tf.keras

In [6]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

toyes = Sequential()
toyes.add(Dense(units= 3, activation = 'sigmoid', input_dim = 3))
toyes.add(Dense(units = 1, activation = 'sigmoid'))
```


Neural Networks

- Practice : tf.keras

In [7]:
```
toyes.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 3)                 12

 dense_1 (Dense)             (None, 1)                 4

=================================================================
Total params: 16
Trainable params: 16
Non-trainable params: 0
_________________________________________________________________
```

Neural Networks

- Practice : tf.keras

```
In [10]: toyes.compile(loss = 'mean_squared_error', optimizer = SGD(lr=1))
         toyes.fit(x,y, epochs =1500, verbose=0)

Out[10]: <keras.callbacks.History at 0x1fbb616e2b0>

In [11]: print(toyes.predict(x))

         1/1 [==============================] - 0s 30ms/step
         [[0.05716112]
          [0.9648627 ]
          [0.9579204 ]
          [0.02031318]]
```


Neural Networks

- Neural Networks
  - Universal approximation theorem (By Homik et al, 1989)
    : when f is a continuous function on a compact set, it can be universally approximated by a single layer neural network $\hat{f}$.

$$\sup\|f-\hat{f}\|<\varepsilon, \qquad \varepsilon>0$$

  - Extendable and flexible structure
    : make it deeper!