

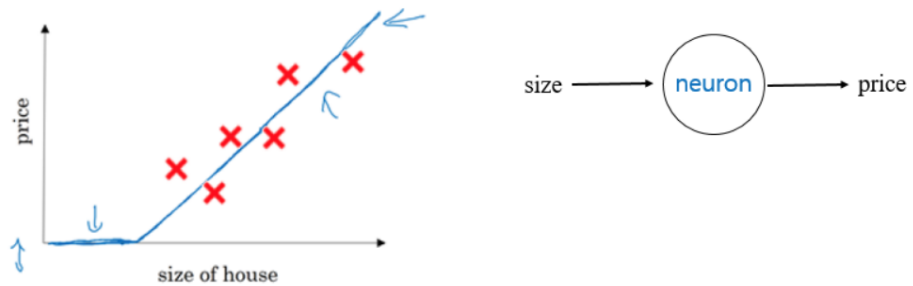
# Lecture 2. Deep Neural Networks

## 2.1 What is neural network

### Example 1 - single neural network

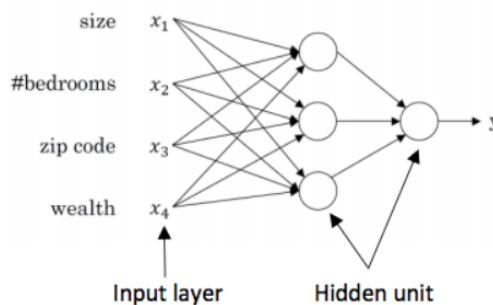
- 집의 크기로 집 가격을 예측하려고 함
- Linear regression:  $price = \beta_0 + \beta_1 size + \epsilon$ 
  - Input: size of house
  - Output: price
  - 회귀분석에서  $\beta_0, \beta_1$ 는 회귀계수라고 불리지만 neural network에서는 weight( $\beta_1$ )와 bias( $\beta_0$ )라고 불림
- 집의 가격은 음수가 될 수 없으므로 Rectified Linear Unit (ReLU) 함수를 사용한다면?
  - The “neuron” implements the function ReLU (blue line)

### Housing Price Prediction



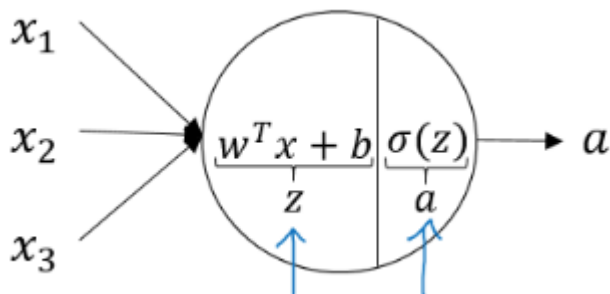
### Example 2 - Multiple neural network

- Input이 집의 크기 외에 화장실 수, 우편번호, 지역 소득수준 등 여러 개인 경우
- 바로 output으로 연결되는 것이 아닌 hidden layer를 거쳐서 output으로 연결 되는 경우



### Neural network representation

- 하나의 hidden node의 구성



- Input vector:  $x = (x_1, x_2, x_3)^T$
- Weights:  $w = (w_1, w_2, w_3)^T$
- Bias:  $b \in \mathbb{R}$

- Activation function:  $\sigma(\cdot)$

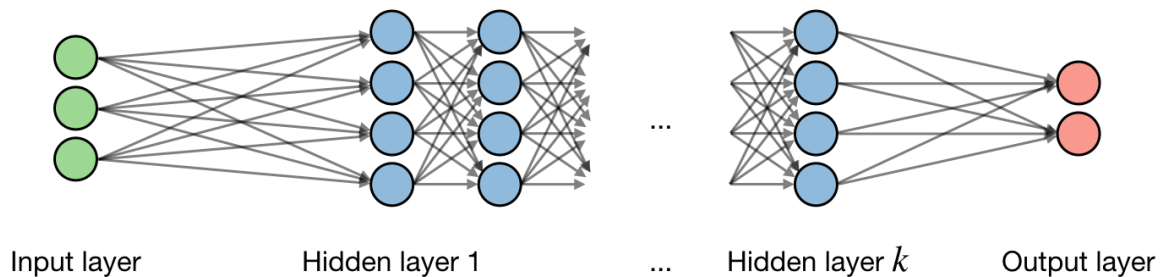
$$z = w_1x_1 + w_2x_2 + w_3x_3 + b \Leftrightarrow z = \mathbf{w}^T \mathbf{x} + b$$

$$a = \sigma(z)$$

- Special case: linear regression
  - $\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + b)$
  - $\sigma(z) = z$
- Special case: logistic regression
  - $\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + b)$
  - $\sigma(z) = \frac{1}{1 + e^{-z}}$ : sigmoid function (inverse-logit function)

## 2.2 Deep neural network

- Input, hidden, output layer로 구성
- 각 layer는 여러 개의 neuron들로 구성
- Deep neural network 구조를 결정하는 요소
  - layer의 개수
  - 한 layer의 neuron(hidden unit)의 개수
  - Activation 함수의 종류

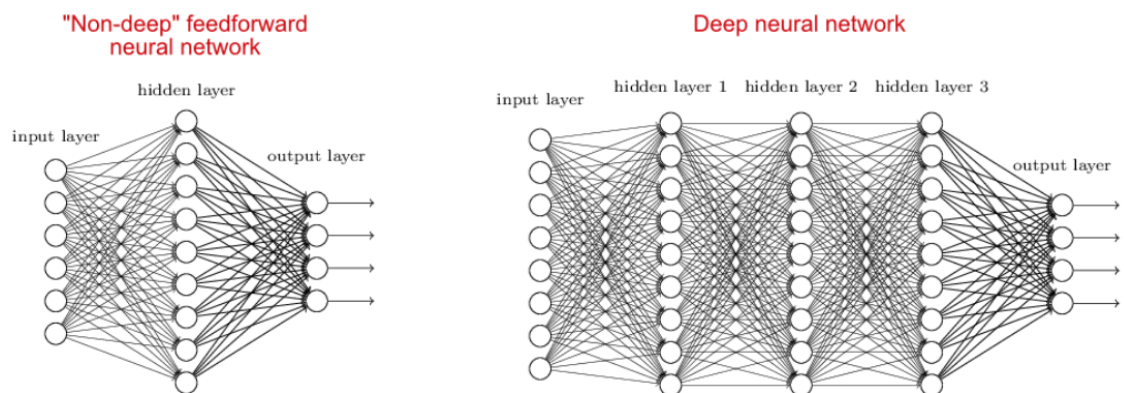


- $i$  번째 layer의  $j$  번째 hidden unit에 대한 output

$$z_j^{[i]} = \mathbf{w}_j^{[i]T} \mathbf{x} + b_j^{[i]}$$

$$a_j^{[i]} = \sigma(z_j^{[i]})$$

### Example



- Non-deep neural network
  - Input layer: input neuron들로 구성. dimension=6
  - Hidden layer: 1개 layer, 9개 unit
  - Output layer: 4개 unit
- Deep neural network

- Input dimension = 8
- Hidden layer: 3개 layer, 각 9개 unit
- Output layer: 4개 unit

```
In [3]: from keras.models import Sequential
        from keras.layers import Dense
        from keras.utils import plot_model
        from IPython.display import Image

        model = Sequential()
        model.add(Dense(9, input_shape=(6,)))
        model.add(Dense(4))
        model.summary()
```

```
-----
Layer (type)                 Output Shape          Param #
-----
dense_9 (Dense)              (None, 9)             63
-----
dense_10 (Dense)             (None, 4)             40
-----
Total params: 103
Trainable params: 103
Non-trainable params: 0
-----
```

- Layer 1: weight 54개(6\*9) + bias 9개 = 63개
- Layer 2: weight 36개(9\*4) + bias 4개 = 40개
- Total parameters = 63+40 = 103개

```
In [2]: from keras.models import Sequential
from keras.layers import Dense
from keras.utils import plot_model
from IPython.display import Image
```

```
model = Sequential()
model.add(Dense(9, input_shape=(8,)))
model.add(Dense(9))
model.add(Dense(9))
model.add(Dense(4))
model.summary()
```

```
-----
Layer (type)                 Output Shape          Param #
-----
dense_5 (Dense)              (None, 9)             81
-----
dense_6 (Dense)              (None, 9)             90
-----
dense_7 (Dense)              (None, 9)             90
-----
dense_8 (Dense)              (None, 4)             40
-----
Total params: 301
Trainable params: 301
Non-trainable params: 0
-----
```

TO DO: 위의 deep neural network 예에서 parameter 개수는 어떻게 계산이 되는가?

## 2.3 Anatomy of a neural network

- Layers: network를 구성하는 요소
- Input and target: 학습과 예측을 위한 데이터
- Loss function: 학습이 잘 되고 있는지 평가
- Optimizer: 어떻게 학습을 진행하는가

### 2.3.1 Layers

- Tensor를 입력받아 tensor를 출력하는 데이터 처리 모듈
- 대부분 가중치를 가짐 (dropout, pooling과 같이 가중치가 없는 layer도 있음)
- 많이 사용되는 Layer의 예
  - Fully connected layer (dense layer)
  - Convolution layer
  - Recurrent layer
  - Embedding layer

Layers in Keras: <https://keras.io/layers/about-keras-layers/> (<https://keras.io/layers/about-keras-layers/>)

### 2.3.2 Model

- Layer를 쌓아 만드는 네트워크
- 일반적으로 하나의 input을 받아 하나의 output을 주는 층을 순서대로 쌓음
- 다양한 형태의 layer가 가능 (Keras Functional API 활용: <https://keras.io/getting-started/functional-api-guide/> (<https://keras.io/getting-started/functional-api-guide/>))
  - Input이 여러 개인 네트워크

- Output이 여러 개인 네트워크
- Inception block
- 적절한 network architecture를 찾는 것은 과학 보다는 예술의 경지! 창의력이 필요함
- 기존의 잘 작동한 architecture를 모방하는 방식으로 접근

### 2.3.3 Loss function

- Model을 통해 나온 prediction  $\hat{y}^{(i)}$ 와 output  $y^{(i)}$ 의 차이를 수량화
- 훈련하는 동안 최소화될 값
- 주어진 문제에 대한 성공 지표

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

- 해결하고자 하는 문제에 따라 표준적인 loss function이 존재함
  - Binary classification
    - 두 개의 클래스를 분류
    - 예) 문장을 입력하여 긍정/부정 구분
    - Binary cross-entropy를 loss로 사용

$$L(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)} \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- Multi-class classification
  - 두 개 이상의 클래스를 분류
  - 예) 이미지를 0,1,2,...,9로 구분
  - Categorical cross-entropy를 loss로 사용

$$L(\hat{y}^{(i)}, y^{(i)}) = - \sum_{c=1}^C y_c^{(i)} \log(\hat{y}_c^{(i)})$$

- Regression
  - 연속형 값을 예측
  - 예) 주가 예측
  - Mean squared error를 loss로 사용

$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

Loss functions in Keras: <https://keras.io/losses/#available-loss-functions>  
[\(https://keras.io/losses/#available-loss-functions\)](https://keras.io/losses/#available-loss-functions)

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

### 2.3.4 Optimizer

- Loss function을 기반으로 네트워크가 어떻게 업데이트 될지를 결정하는 알고리즘
- 특정한 종류의 stochastic gradient descent 알고리즘을 구현

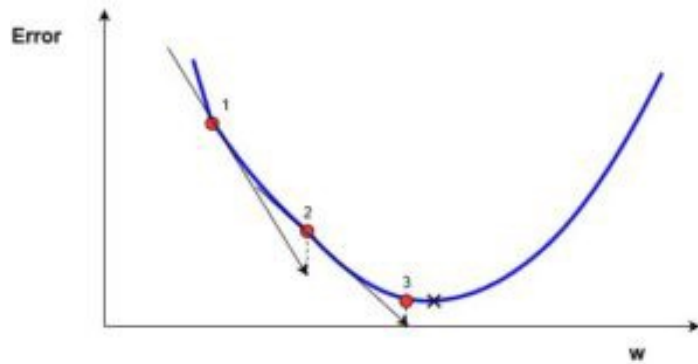
#### Gradient descent

- Gradient: 특정한 점에서 함수  $f$ 의 기울기
- Gradient의 양의 방향 = 함수  $f$ 가 가장 빠르게 증가하는 방향
- Gradient의 음의 방향 = 함수  $f$ 가 가장 빠르게 감소하는 방향
  - 예)

$$f(x) = x^2$$

$$\frac{df(x)}{dx} = 2x$$

- $x = -1$ 에서의 gradient =  $-2$
- $f(x)$ 가 가장 빠르게 감소하는 방향은  $+2$ 의 방향

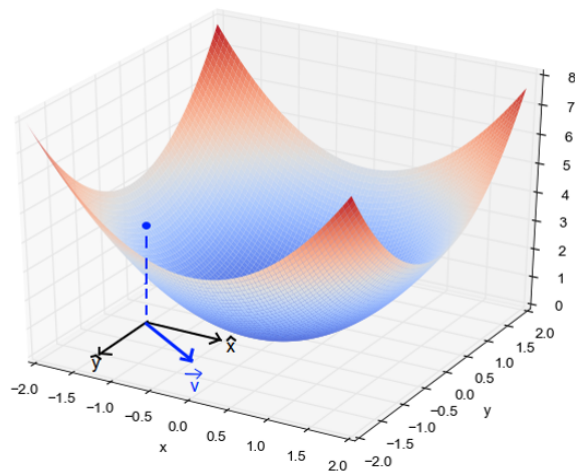


■ 예)

$$f(x, y) = x^2 + y^2$$

$$\nabla f(x, y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

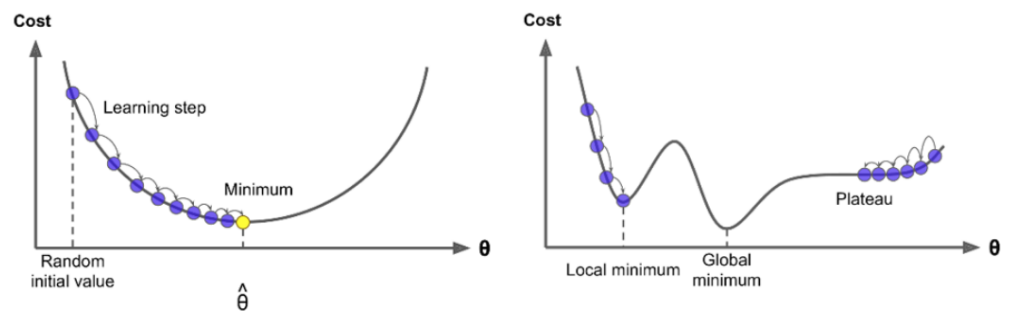
- $(x, y) = (-1, -1)$ 에서의 gradient =  $(-2, -2)^T$
- $f(x)$ 가 가장 빠르게 감소하는 방향은  $(2, 2)^T$



- Gradient descent algorithm

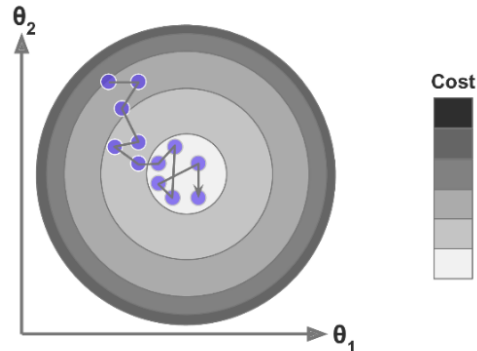
$$x^{(k+1)} = x^{(k)} - \alpha \nabla f(x^{(k)})$$

- 초기값의 위치에서 가장 기울기가 가파른 방향( $-\nabla f$ )으로 조금( $\alpha$ , learning rate, step size) 이동
- 이동한 지점에서 다시 gradient를 계산하고 가장 기울기가 가파른 방향으로 조금 이동
- Local minima로 수렴할 가능성이 있음
- 초기값의 위치에 따라 수렴속도가 매우 느릴 수 있음



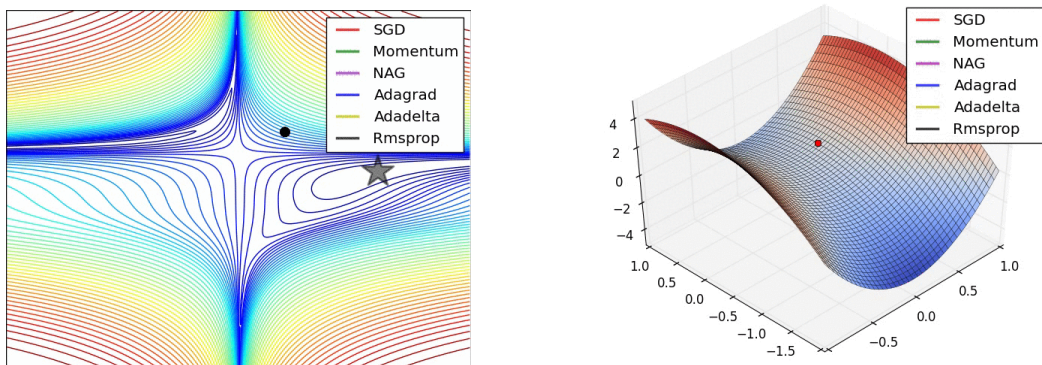
### Stochastic gradient descent (SGD) algorithm

- Gradient 계산 시 전체 데이터를 사용하지 않고 무작위로 선정된 일부 데이터(mini-batch)를 사용
- 계산속도가 빠름
- Local minima에 빠질 가능성을 줄여줌



### Others variants

- weight를 업데이트 할 때 현재 gradient 뿐만 아니라 이전에 업데이트된 weight, 기울기 등을 여러 방식으로 고려하여 SGD를 개선하는 방법들이 있음
- Adagrad, RMSProp, momentum, Adam 등이 많이 사용됨



Images credit: [Alec Radford \(https://twitter.com/alecrad\)](https://twitter.com/alecrad)

Optimizers in Keras: <https://keras.io/optimizers/> (<https://keras.io/optimizers/>)

### Backpropagation

- 신경망은 여러 개의 텐서 연산으로 연결되어 있음

$$z = w^T x + b$$

$$y = \sigma(z)$$

$$L(w) = \sum_i \frac{1}{2} (\hat{y}^{(i)} - y^{(i)})^2$$

- $\frac{dL}{dw_1}$ 을 어떻게 계산?
- Chain rule을 이용 (연쇄법칙)
  - 합성함수의 미분을 차례로 한 단계씩 계산해나가는 과정

$$\blacksquare y = (x^2 + 1)^3, \frac{dy}{dx} ?$$

$$g(x) = u = x^2 + 1$$

$$y = f(u) = u^3$$

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = (3u^2)(2x) = 3(x^2 + 1)^2(2x) = 6x(x^2 + 1)^2$$

- Chain rule을 사용하여 loss function의 gradient를 계산

$$\frac{dL}{dw_1} = \frac{dL}{dy} \frac{dy}{dz} \frac{dz}{dw_1}$$

- Loss 값에 각 parameter가 기여한 정도를 계산하기 위해 최상층 부터 하위 층까지 거꾸로 연쇄법칙을 적용하여 계산 - backpropagation algorithm(reverse-mode automatic differentiation)
- Tensorflow처럼 symbolic differentiation이 가능한 프레임워크를 사용하여 빠르게 계산 가능

```
In [3]: # Do not run
model.compile(optimizer='adam', loss='binary_crossentropy')
```

### 2.3.5 Activation

- Sigmoid (inverse-logit)

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- 0과 1사이의 값을 출력
  - Output layer에서 확률값을 출력하고자 할 때 주로 사용
- Hyperbolic tangent

$$\tanh(z) = 2\sigma(2z) - 1$$

- 1 < tanh(z) < 1
  - Sigmoid의 이동한 형태
  - Output이 0을 중심으로 분포하므로 sigmoid보다 학습에 효율적
- ReLU(Rectified Linear Unit)

$$\text{Relu}(z) = \max(0, z)$$

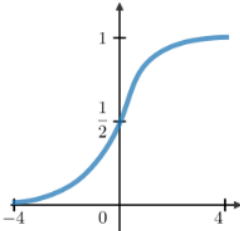
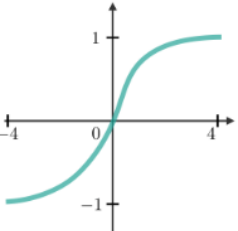
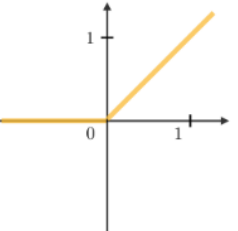
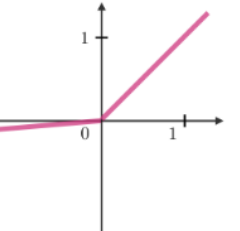
- 최적화 과정에서 gradient가 0과 가까워져 수렴이 느려지는 문제를 해결
  - 음수를 모두 0으로 처리하고 평균이 0이 되지 않는다는 단점
  - Leaky ReLU, ELU 등 변화된 형태도 있음
- Softmax

$$\sigma(z_j) = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}, j = 1, 2, \dots, K$$

- 각 class의 score를 정규화 하여 각 class에 대한 확률값으로 변환(sum=1)
- Multi-class classification 문제의 output layer에서 사용

Activation functions in Keras: <https://keras.io/activations/> (<https://keras.io/activations/>)



Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

```
In [ ]: # Do not run
model = Sequential()
model.add(Dense(9, input_shape=(8,)))
model.add(Dense(9, activation='relu'))
model.add(Dense(9, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

#### References

- <https://www.coursera.org/specializations/deep-learning>  
(<https://www.coursera.org/specializations/deep-learning>)
- [Hands on Machine Learning with Scikit-Learn and Tensorflow, Aurélien Géron](http://www.hanbit.co.kr/store/books/look.php?p_code=B9267655530)  
([http://www.hanbit.co.kr/store/books/look.php?p\\_code=B9267655530](http://www.hanbit.co.kr/store/books/look.php?p_code=B9267655530))
- [Deep Learning with Python, François Chollet](https://www.manning.com/books/deep-learning-with-python), (<https://www.manning.com/books/deep-learning-with-python>)
- <https://stanford.edu/~shervine/teaching/cs-229/>  
(<https://stanford.edu/~shervine/teaching/cs-229/>)