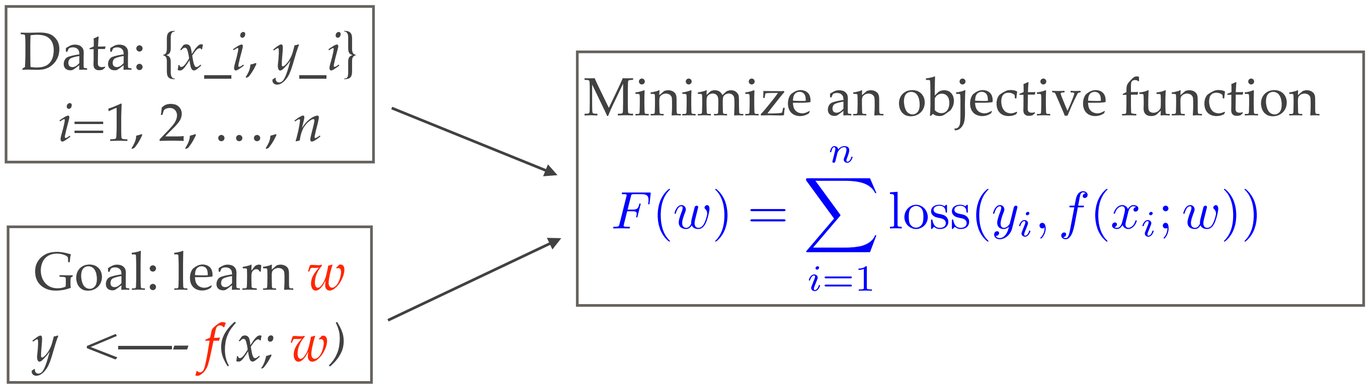
How Does Machine Learning Work



- Collect Data
- Determine a **function space**
- Pick a **loss function**
- Pick an **optimizer**

Optimizer

1. The minimizer w^* may be in closed form.
2. Try optimization algorithms that can guarantee to converge to the global minimizer.
3. In the worst case, try *gradient descent*.

Gradient Descent and Stochastic Gradient Descent

Derivative

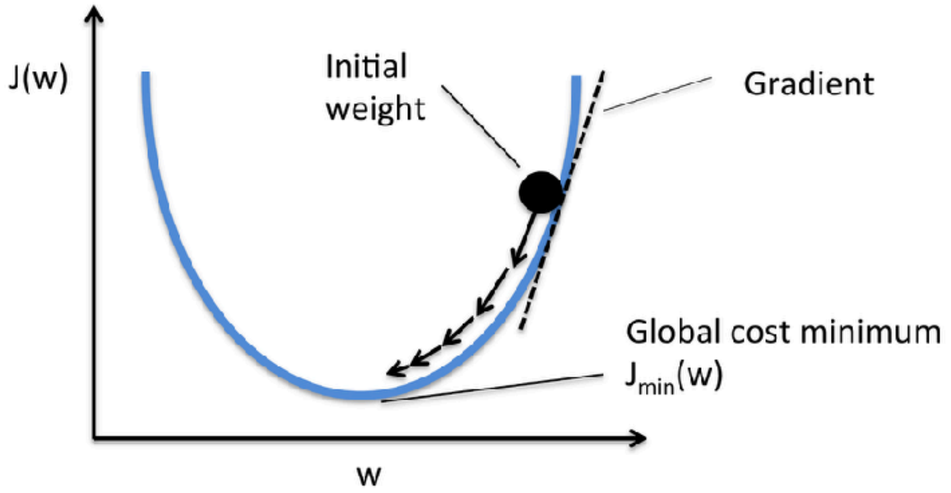
$$J'(w_0) = \lim_{\delta \to 0} \frac{J(w_0 + \delta) - J(w_0)}{\delta}$$

$$J(w_0 + \delta) \approx J(w_0) + \delta \cdot J'(w_0)$$

Gradient Descent

$$w_{n+1} = w_n - \gamma_n J'(w_n)$$

Learning Rate



Gradient Descent and Stochastic Gradient Descent

Derivative

$$J'(w_0) = \lim_{\delta \to 0} \frac{J(w_0 + \delta) - J(w_0)}{\delta}$$

$$J(w_0 + \delta) \approx J(w_0) + \delta \cdot J'(w_0)$$

Gradient Descent

$$w_{n+1} = w_n - \gamma_n J'(w_n)$$

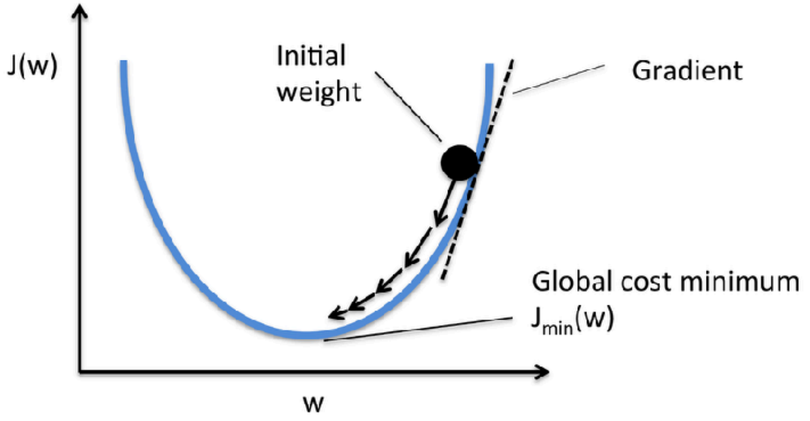
Learning Rate

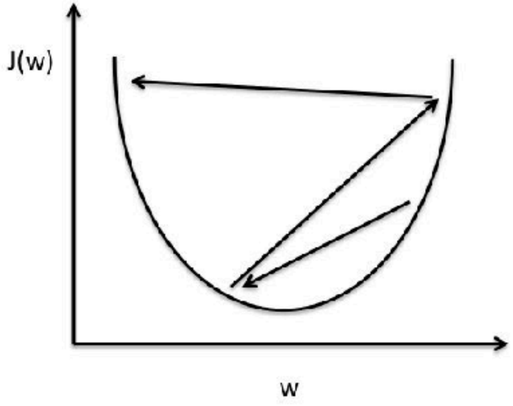
SGD

$$F(w) = \sum_{i=1}^{n} f_i(w)$$

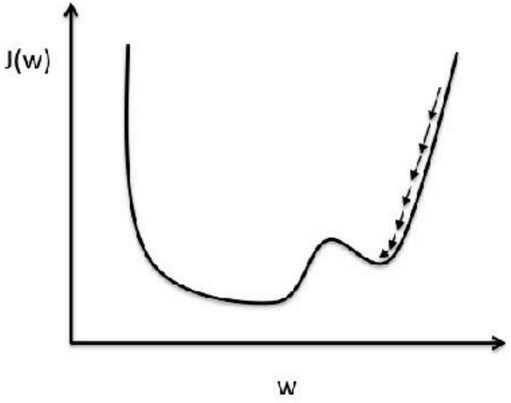
$$F'(w) = \sum_{i=1}^{n} f_i'(w)$$

$$F'_{\mathrm{SG}}(w) = \sum_{i \in I_t} f_i'(w)$$

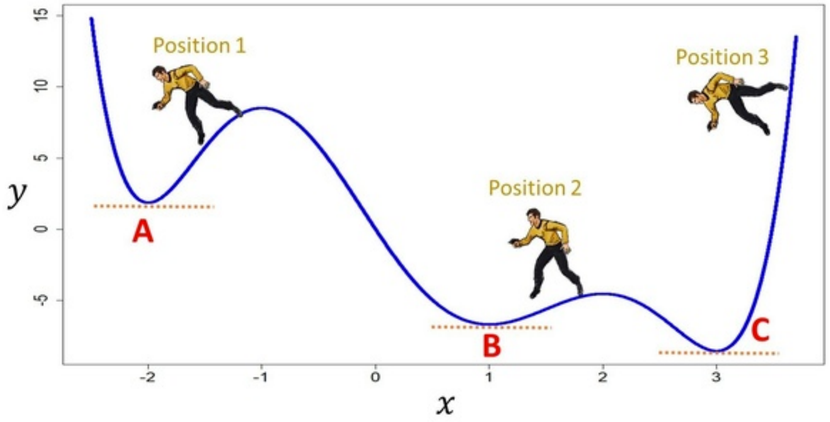




Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.



Gradient Descent and Stochastic Gradient Descent

Partial Derivative

$$\nabla F(\mathbf{w}) = \begin{pmatrix} \frac{\partial F(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial F(\mathbf{w})}{\partial w_m} \end{pmatrix}$$

Gradient Descent

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \cdot \nabla F(\mathbf{w}_n)$$

How to Improve Learning Rate?

1. **Reduce Oscillation**
2. **Adaptive to different dims**

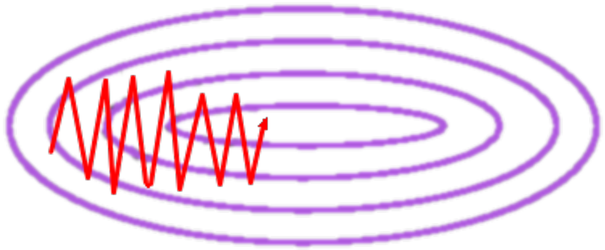
https://keras.io/optimizers/

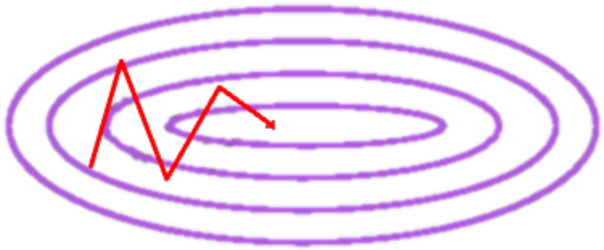
SGD

$$F(w) = \sum_{i=1}^{n} f_i(w)$$

$$F'(w) = \sum_{i=1}^{n} f_i'(w)$$

$$F'_{\mathrm{SG}}(w) = \sum_{i \in I_t} f_i'(w)$$



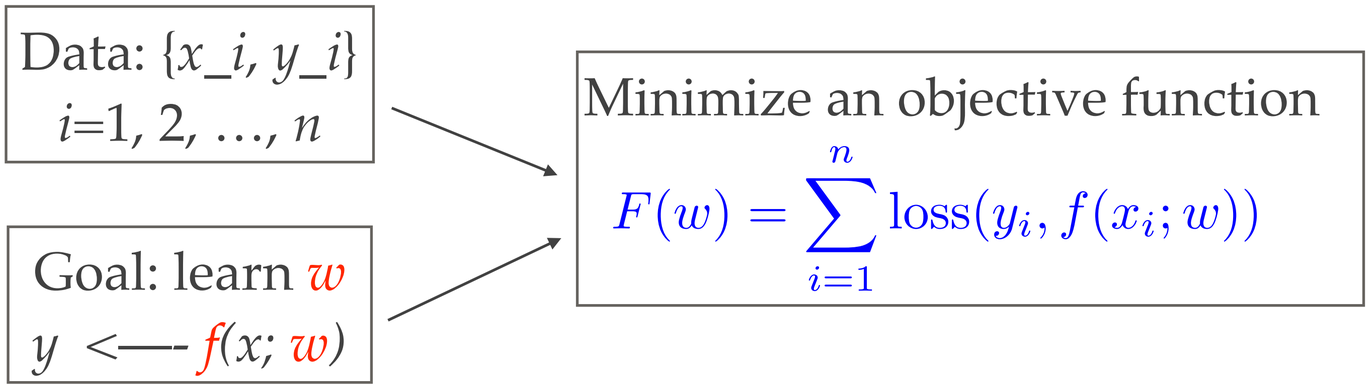


Chain Rule

$$F(w) = g(z), \quad z = h(w)$$

$$\frac{dF(w)}{dw} = \frac{dg(z)}{dz} \cdot \frac{dh(w)}{dw}$$

How Does Machine Learning Work



- Collect Data
- Determine a **function space**
- Pick a **loss function**
- Pick an **optimizer**

Optimizer

1. The minimizer w^* may be in closed form.
2. Try optimization algorithms that can guarantee to converge to the global minimizer.
3. In the worst case, try ***gradient descent***.

Loss Function/Metric

1. Regression; Classification
2. https://keras.io/losses/
3. https://keras.io/metrics/

Function Space: How to Go Beyond Linear Functions

Suppose $x \in \mathbf{R}^1$ (univariate case).

- Polynomial regression

$$f(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_d x^d.$$

- Spline models: regression splines or smoothing splines.
- Local polynomial regression: Loess

Suppose $\mathbf{x} \in \mathbf{R}^p$ (multivariate)

- For a polynomial model with degree 3, how many terms in the model?

$$\{X_j\}_{j=1}^p, \{X_j^2\}, \{X_j^3\}, \{X_j X_l\}, \{X_j^2 X_l\}, \{X_j X_k X_l\}.$$

COD (Curse Of Dimensionality): num of parameters explodes.

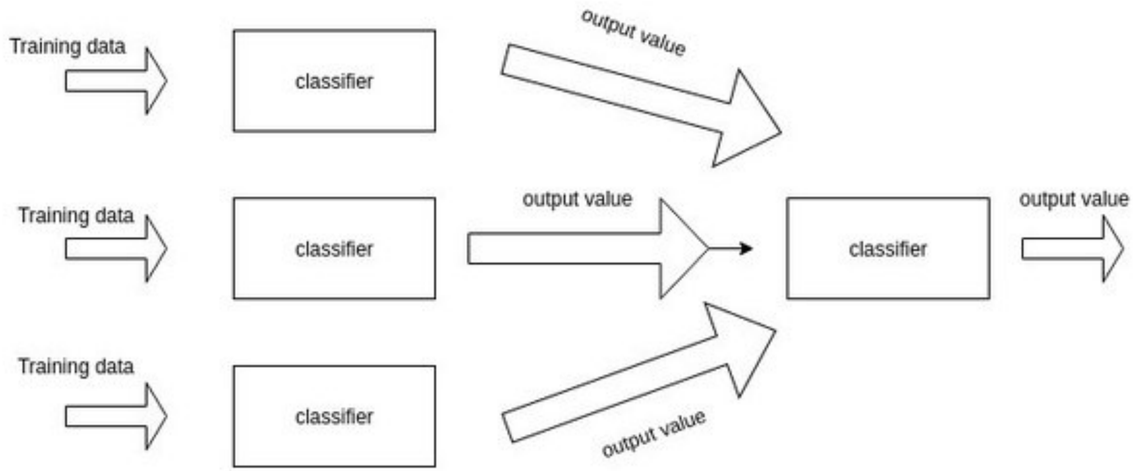
- A Compromise: Additive model

$$f(\mathbf{X}) = \mu + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$

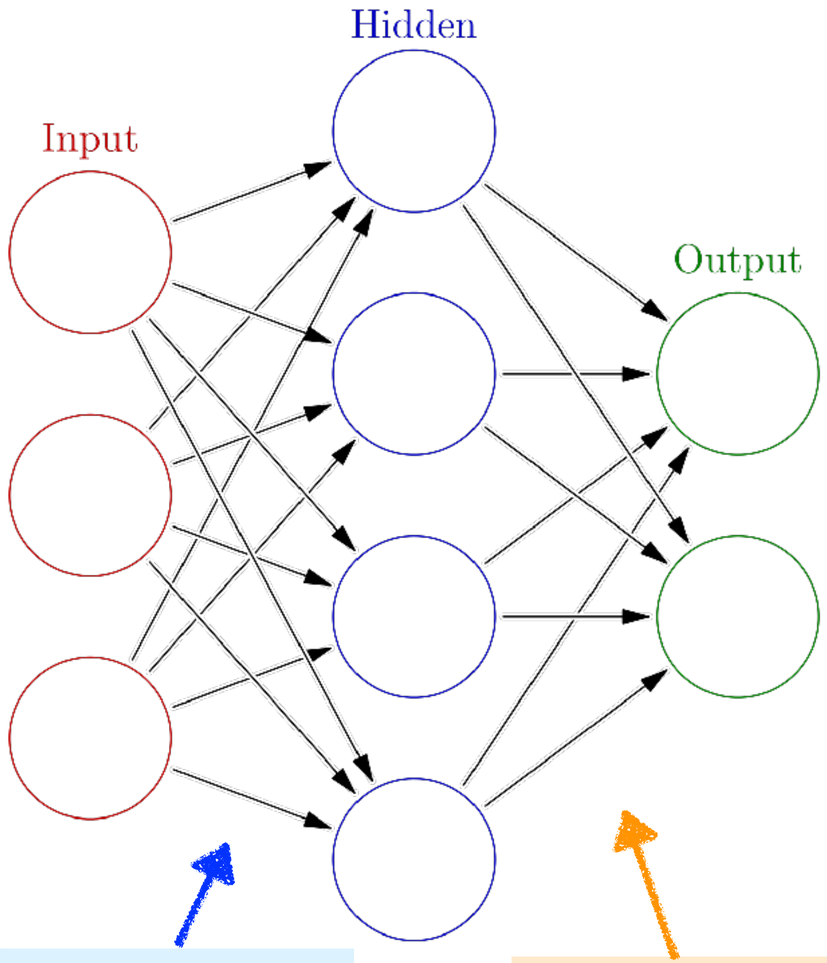
Drawback: totally ignore interactions among the p covariates.

What else ?

SVM,
Tree Models,
Model Ensemble,
Model Stacking →



Neutral Networks



(W_4x3, c_4x1)

(*w*_2x4, b_2x1)

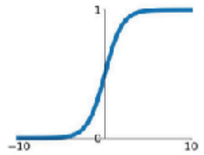
$$h_1 = \sigma(x_1 \cdot W_{1,1} + x_2 \cdot W_{1,2} + x_3 \cdot W_{1,3} + c_1)$$
$$h_2 = \sigma(x_1 \cdot W_{2,1} + x_2 \cdot W_{2,2} + x_3 \cdot W_{2,3} + c_2)$$
$$h_3 = \sigma(x_1 \cdot W_{3,1} + x_2 \cdot W_{3,2} + x_3 \cdot W_{3,3} + c_3)$$
$$h_4 = \sigma(x_1 \cdot W_{4,1} + x_2 \cdot W_{4,2} + x_3 \cdot W_{4,3} + c_4)$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{c}) \qquad \mathbf{y} = \mathbf{w}\mathbf{h} + \mathbf{b}$$

Activation Functions

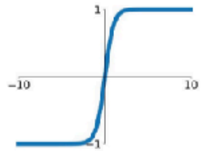
Sigmoid

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



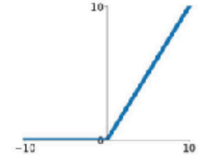
tanh

$\tanh(x)$



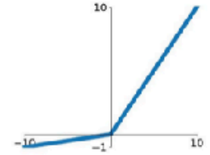
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

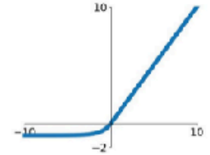


Maxout

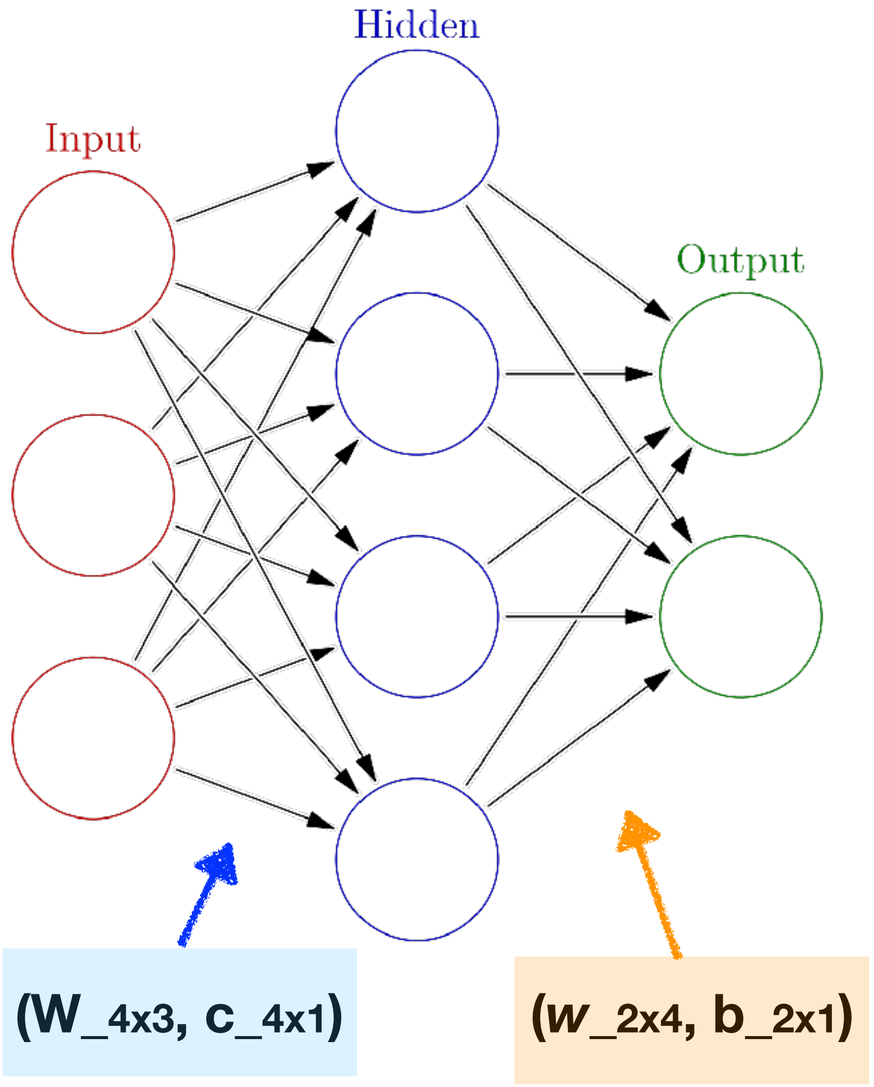
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neutral Networks



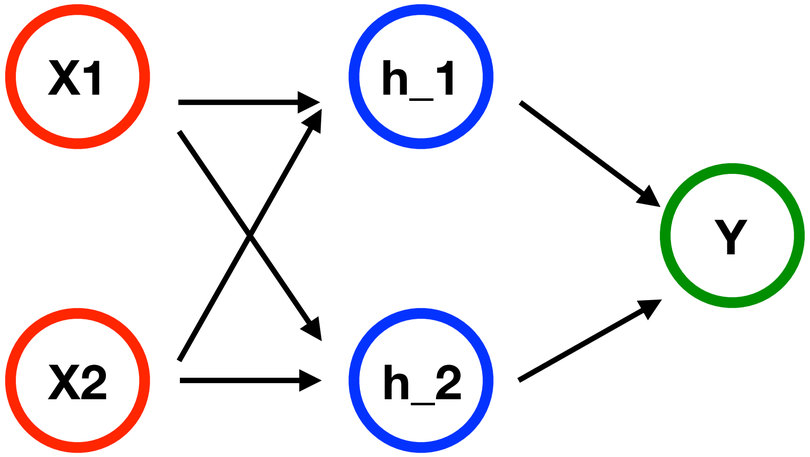
$$h_1 = \sigma(x_1 \cdot W_{1,1} + x_2 \cdot W_{1,2} + x_3 \cdot W_{1,3} + c_1)$$
$$h_2 = \sigma(x_1 \cdot W_{2,1} + x_2 \cdot W_{2,2} + x_3 \cdot W_{2,3} + c_2)$$
$$h_3 = \sigma(x_1 \cdot W_{3,1} + x_2 \cdot W_{3,2} + x_3 \cdot W_{3,3} + c_3)$$
$$h_4 = \sigma(x_1 \cdot W_{4,1} + x_2 \cdot W_{4,2} + x_3 \cdot W_{4,3} + c_4)$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{c}) \qquad \mathbf{y} = \mathbf{w}\mathbf{h} + \mathbf{b}$$

# of parameters from layer(j) to layer(j+1)

= [length_of_layer(j) + 1] x length_of_layer(j+1)

A Simple Example of NN



$$\mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\mathbf{c} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

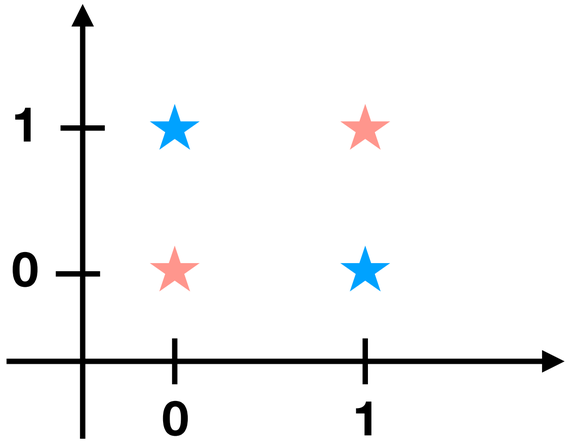
$$\mathbf{w} = (1, -2)$$

$$b = 0$$

$$y = \mathbf{w}\sigma(\mathbf{W}\mathbf{x} + \mathbf{c}) + b$$

ReLU

XOR



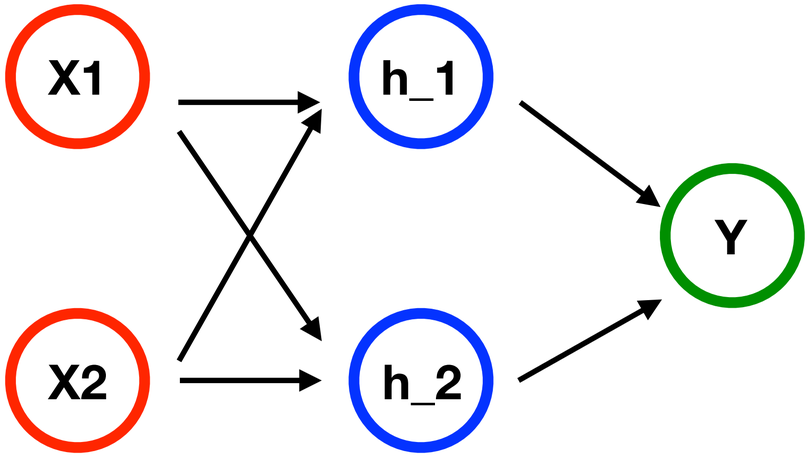
Input

(0, 0)
(1, 1)
(0, 1)
(1, 0)

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

$$\sigma \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

A Simple Example of NN



$$\mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\mathbf{c} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

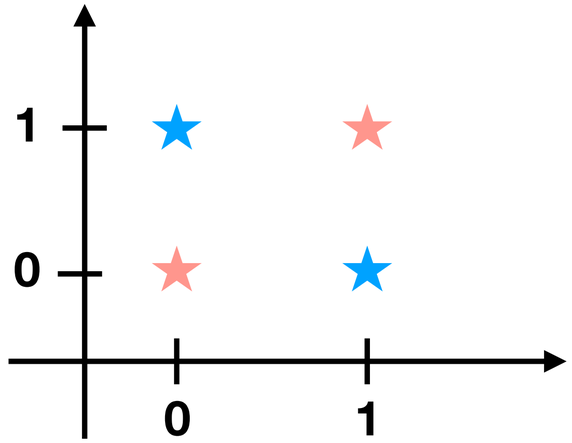
$$\mathbf{w} = (1, -2)$$

$$b = 0$$

$$y = \mathbf{w}\sigma(\mathbf{W}\mathbf{x} + \mathbf{c}) + b$$

ReLU

XOR



Input

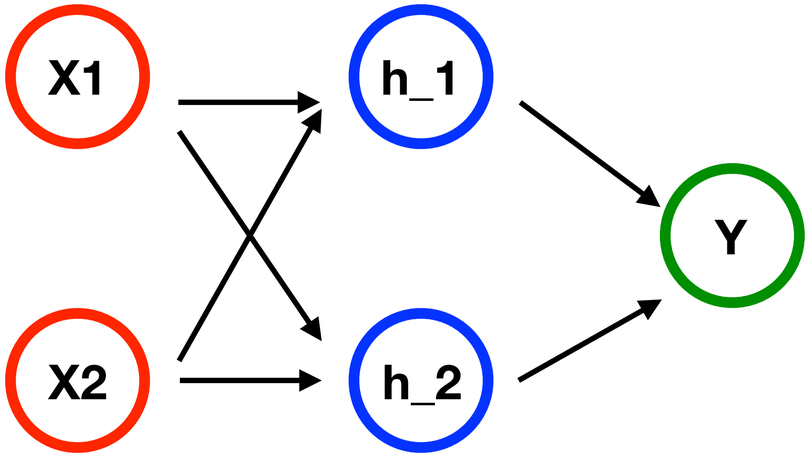
(0, 0)
(1, 1)
(0, 1)
(1, 0)

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$\sigma \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$(1 - 2) \begin{pmatrix} 2 \\ 1 \end{pmatrix} = 0$$

A Simple Example of NN



$$\mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\mathbf{c} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

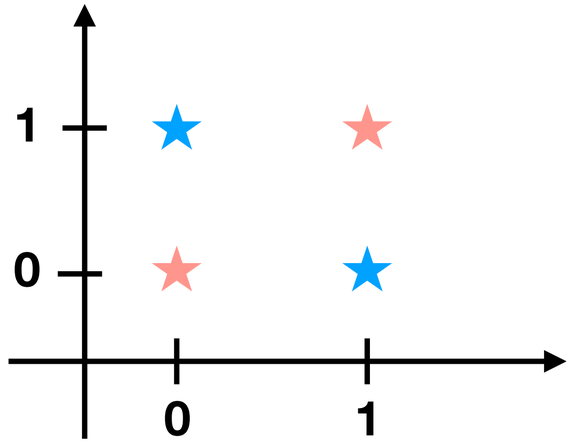
$$\mathbf{w} = (1, -2)$$

$$b = 0$$

$$y = \mathbf{w}\sigma(\mathbf{W}\mathbf{x} + \mathbf{c}) + b$$

ReLU

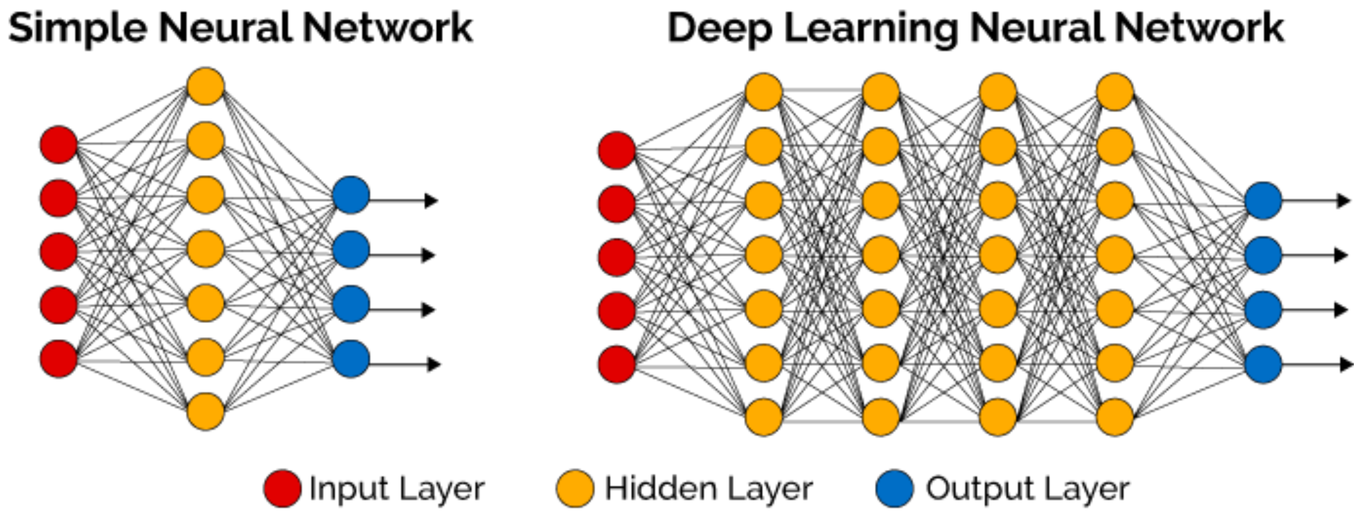
XOR



Input	Hidden	Output
(0, 0)		0
(1, 1)		0
(0, 1)		-1
(1, 0)		-1

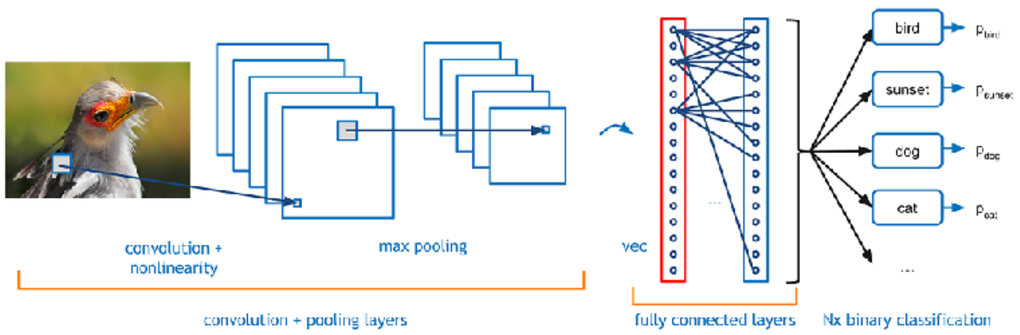
Deep Neutral Networks

Fully Connected

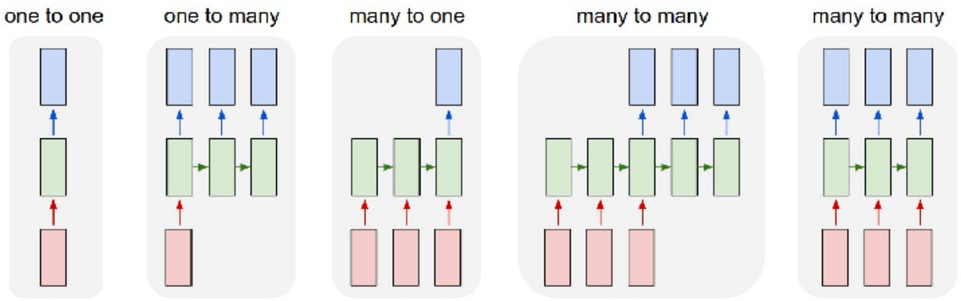


- Collect Data
- Pick a Network Architecture
- Pick a **loss function**
- Pick an **optimizer**

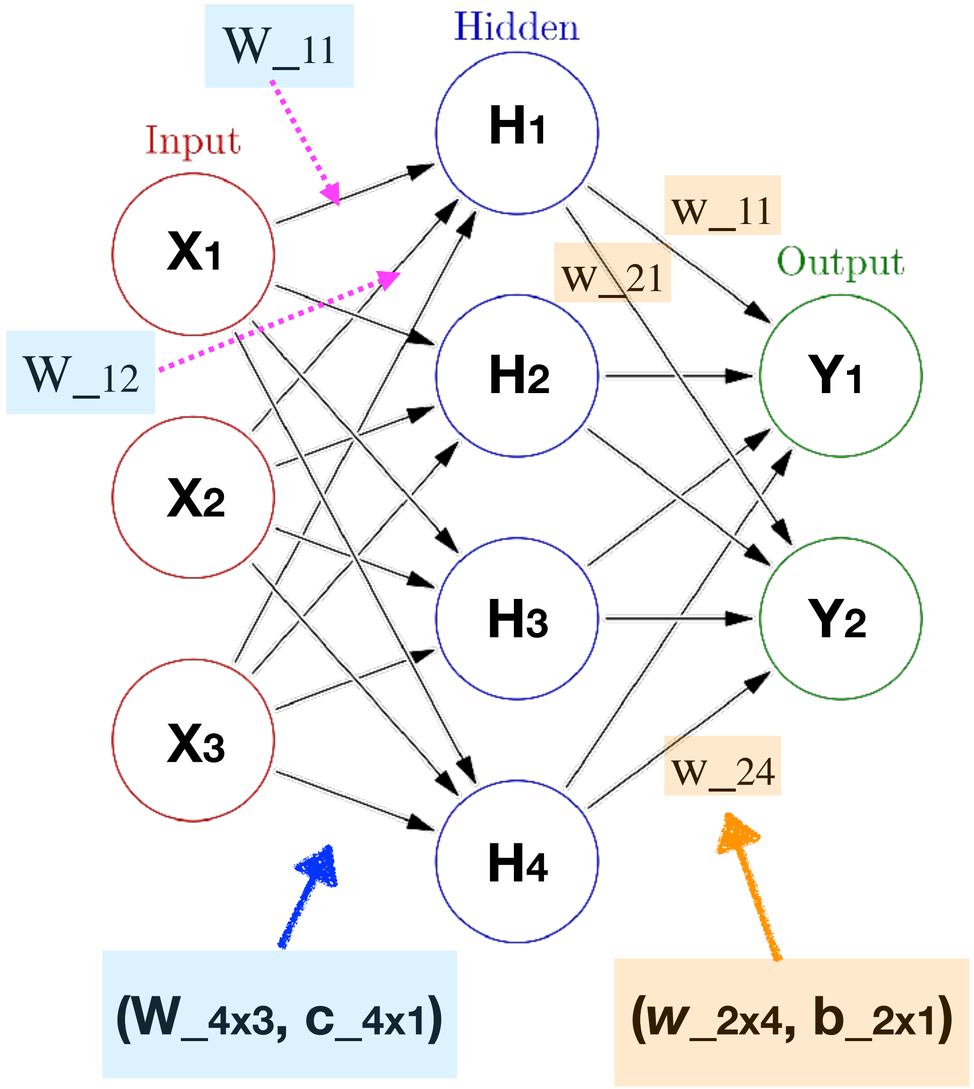
CNN



RNN



Chain Rule and Back-Propagation



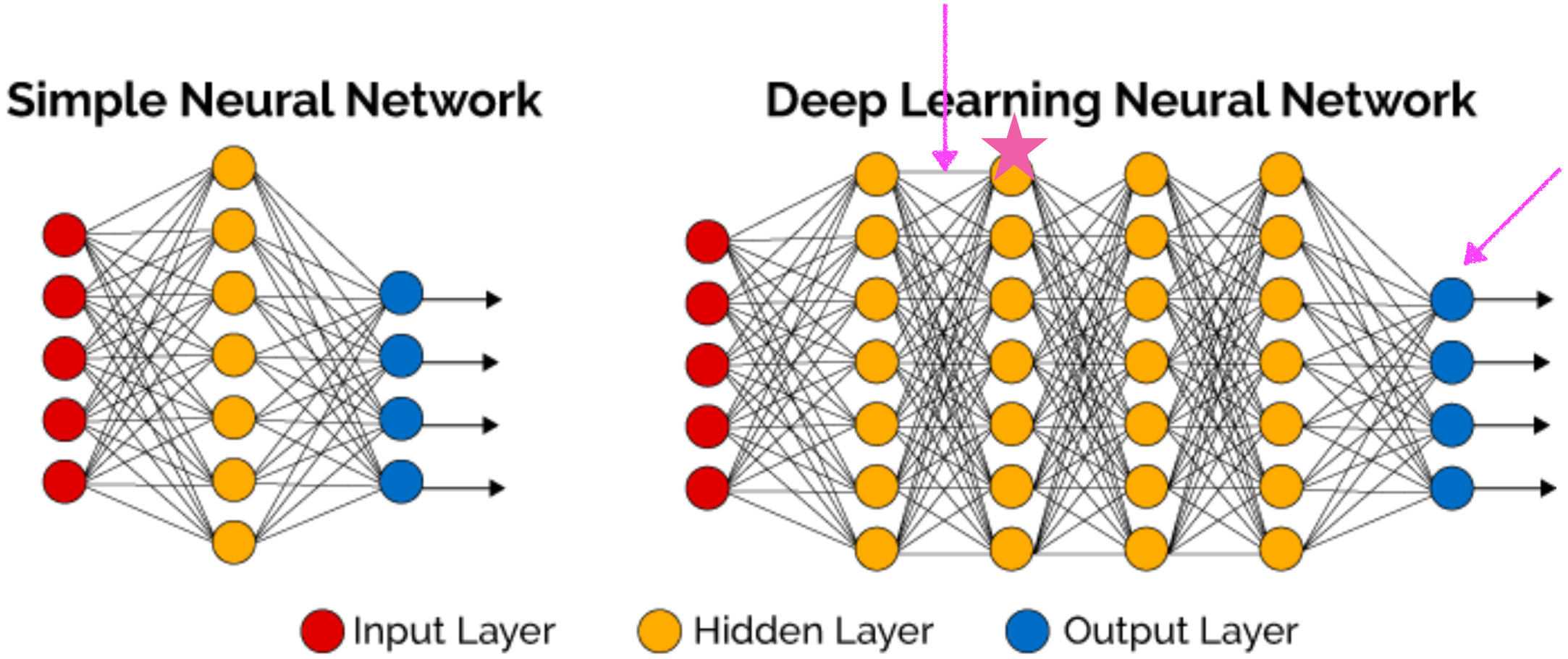
$$h_1 = \sigma(x_1 \cdot W_{1,1} + x_2 \cdot W_{1,2} + x_3 \cdot W_{1,3} + c_1)$$
$$h_2 = \sigma(x_1 \cdot W_{2,1} + x_2 \cdot W_{2,2} + x_3 \cdot W_{2,3} + c_2)$$
$$h_3 = \sigma(x_1 \cdot W_{3,1} + x_2 \cdot W_{3,2} + x_3 \cdot W_{3,3} + c_3)$$
$$h_4 = \sigma(x_1 \cdot W_{4,1} + x_2 \cdot W_{4,2} + x_3 \cdot W_{4,3} + c_4)$$

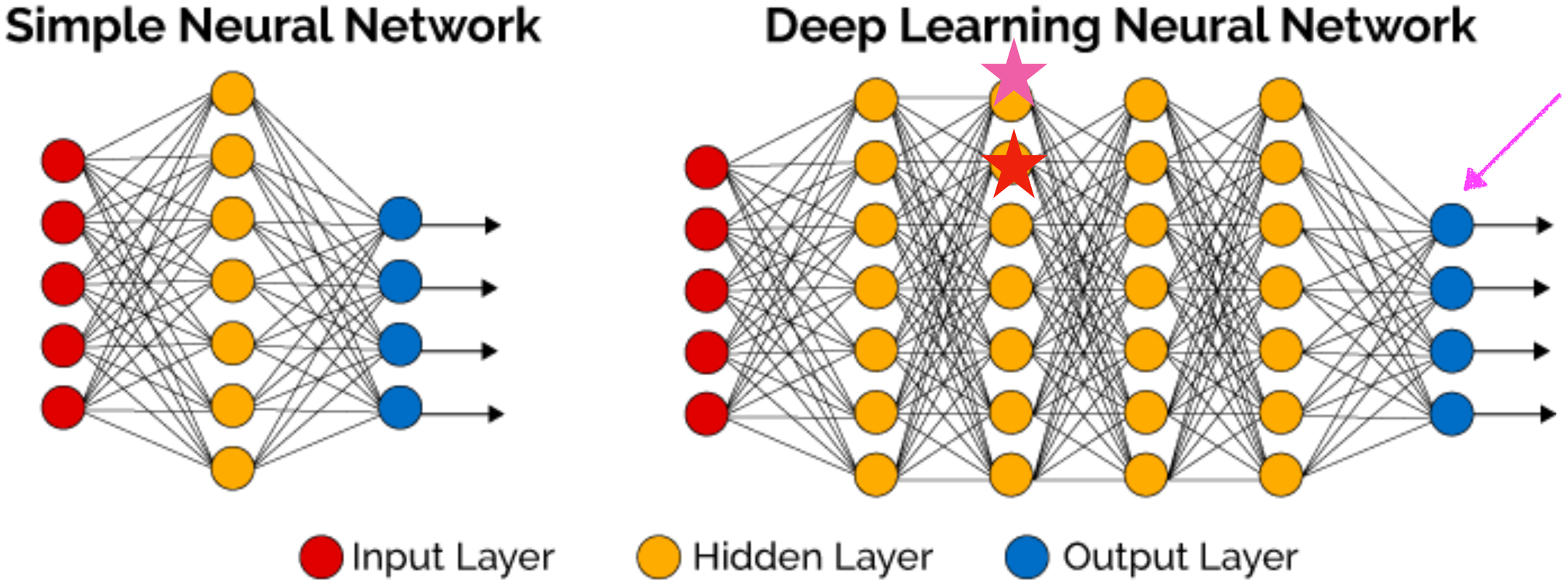
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{c}) \qquad \mathbf{y} = \mathbf{w}\mathbf{h} + \mathbf{b}$$

$$\frac{\partial y_1}{\partial w_{11}} = h_1$$
$$\frac{\partial y_2}{\partial w_{21}} = h_1$$
$$\frac{\partial y_i}{\partial w_{il}} = h_l$$

$$\frac{\partial y_1}{\partial W_{11}} = \frac{\partial y_1}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial W_{11}}$$
$$\frac{\partial y_1}{\partial W_{12}} = \frac{\partial y_1}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial W_{12}}$$
$$\frac{\partial y_i}{\partial W_{jl}} = \frac{\partial y_i}{\partial h_j}\frac{\partial h_j}{\partial z_j}\frac{\partial z_j}{\partial W_{jl}}$$

What about Multiple Hidden Layers?

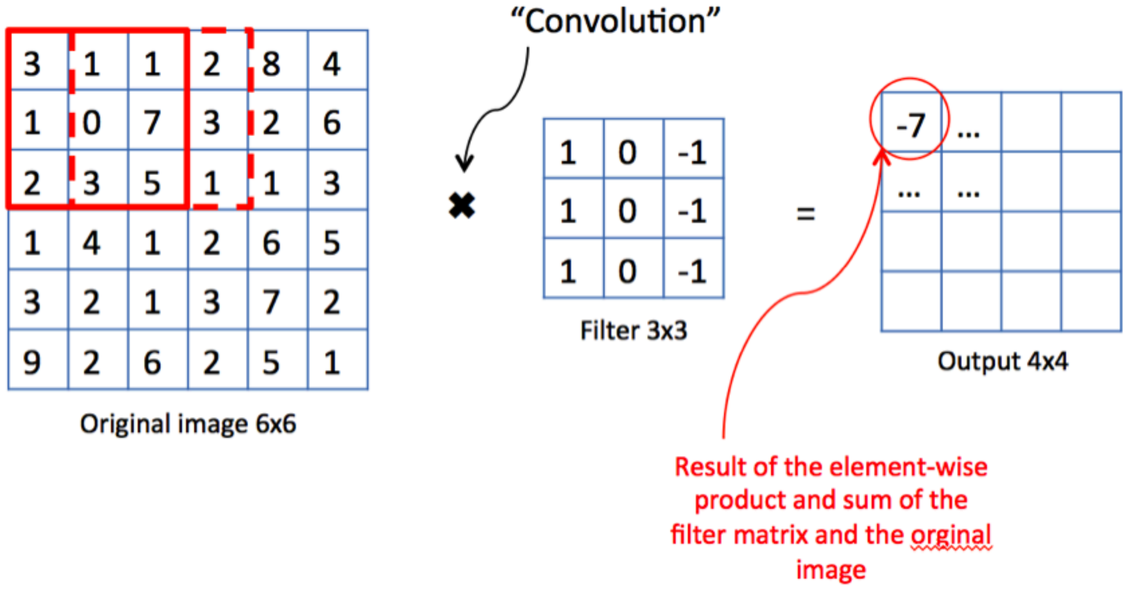

Simple Neural Network
Deep Learning Neural Network
Input Layer
Hidden Layer
Output Layer

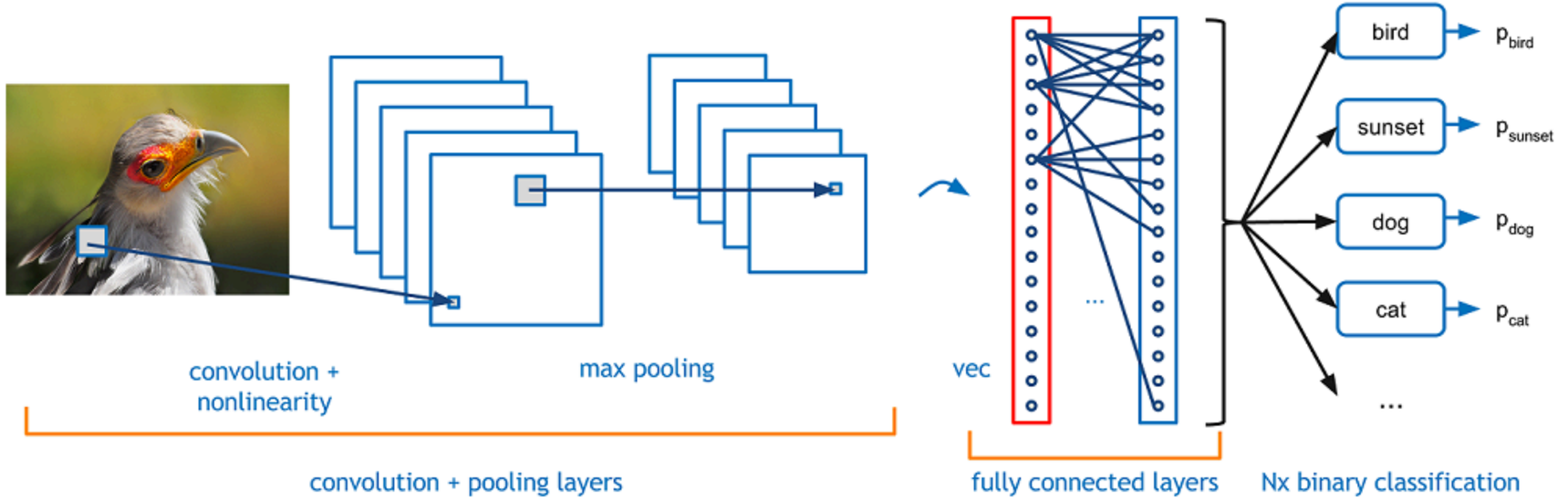


Recall how we recursively compute the forward/backward probabilities for HMM; the same trick can be used for efficient computation of the derivatives for DNN.

CNN

- Collect Data
- Pick a Network Architecture
- Pick a **loss function**
- Pick an **optimizer**





```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(
    filter=32, kernel_size=c(3,3),
               padding="same",
               input_shape=c(32,32,3)
    ) %>% layer_activation("relu") %>%

  layer_conv_2d(filter=32, kernel_size=c(3,3)) %>%
  layer_activation("relu") %>%

  layer_max_pooling_2d(pool_size=c(2,2)) %>%
  layer_dropout(0.25) %>%

  # Two additional hidden layers
  layer_conv_2d(filter=32, kernel_size=c(3,3),
                           padding="same") %>%
  layer_activation("relu") %>%

  layer_conv_2d(filter=32,kernel_size=c(3,3)) %>%
  layer_activation("relu") %>%
```

```
> (3*3*3+1)*32
[1] 896
```

```
> (3*3*32+1)*32
[1] 9248
```

```
> (3*3*32+1)*32
[1] 9248
```

```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(
    filter=32, kernel_size=c(3,3),
               padding="same",
               input_shape=c(32,32,3)
    ) %>% layer_activation("relu") %>%

  layer_conv_2d(filter=32,
                kernel_size=c(3,3)) %>%
  layer_activation("relu") %>%

  layer_max_pooling_2d(pool_size=c(2,2)) %>%
  layer_dropout(0.25) %>%

  # Two additional hidden layers
  layer_conv_2d(filter=32,
                kernel_size=c(3,3),
                padding="same") %>%
  layer_activation("relu") %>%

  layer_conv_2d(filter=32,
                kernel_size=c(3,3)) %>%
  layer_activation("relu") %>%
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
activation_1 (Activation)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
activation_2 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 15, 15, 32)	9248
activation_3 (Activation)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 32)	9248
activation_4 (Activation)	(None, 13, 13, 32)	0

```
# Max pooling and dropout
 layer_max_pooling_2d(pool_size=c(2,2)) %>%
 layer_dropout(0.25) %>%

 #Flatten the input and feed into dense layer
 layer_flatten() %>%
 layer_dense(512) %>%
 layer_activation("relu") %>%
 layer_dropout(0.5) %>%

 layer_dense(10) %>%
 layer_activation("softmax")
```

```
> 6*6*32
[1] 1152
> (1152+1)*512
[1] 590336
> (512+1)*10
[1] 5130
```

max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 32)	0
dropout_2 (Dropout)	(None, 6, 6, 32)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 512)	590336
activation_5 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130
activation_6 (Activation)	(None, 10)	0