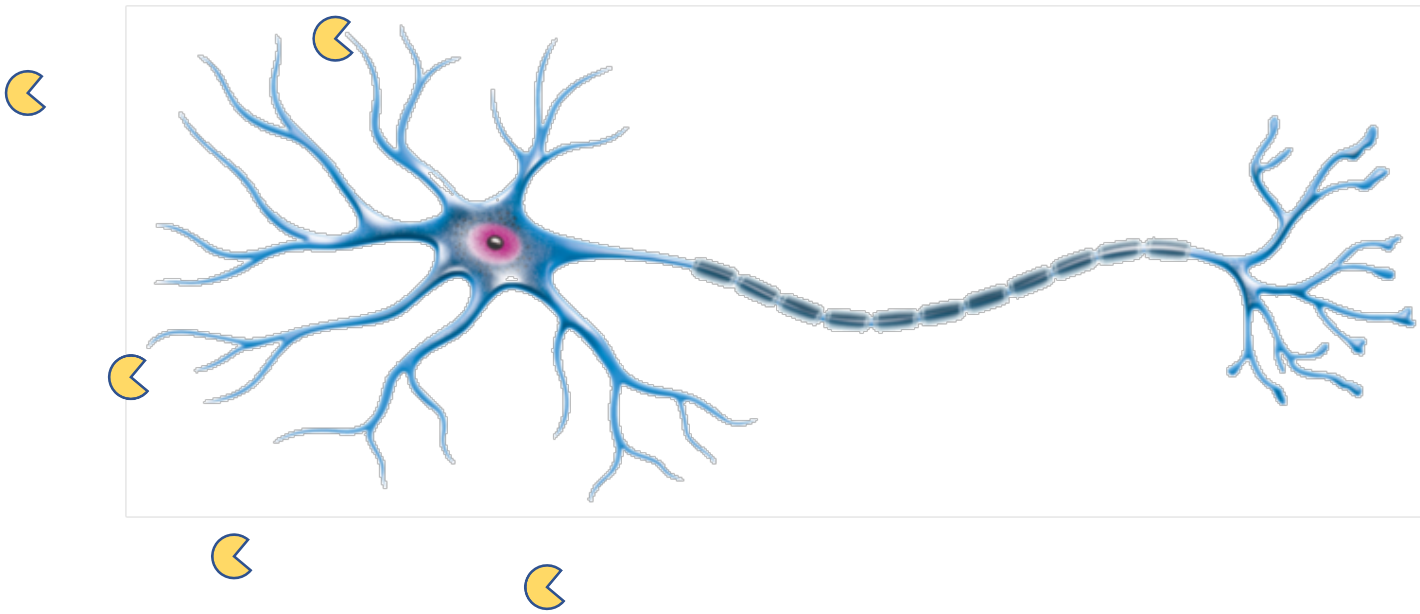
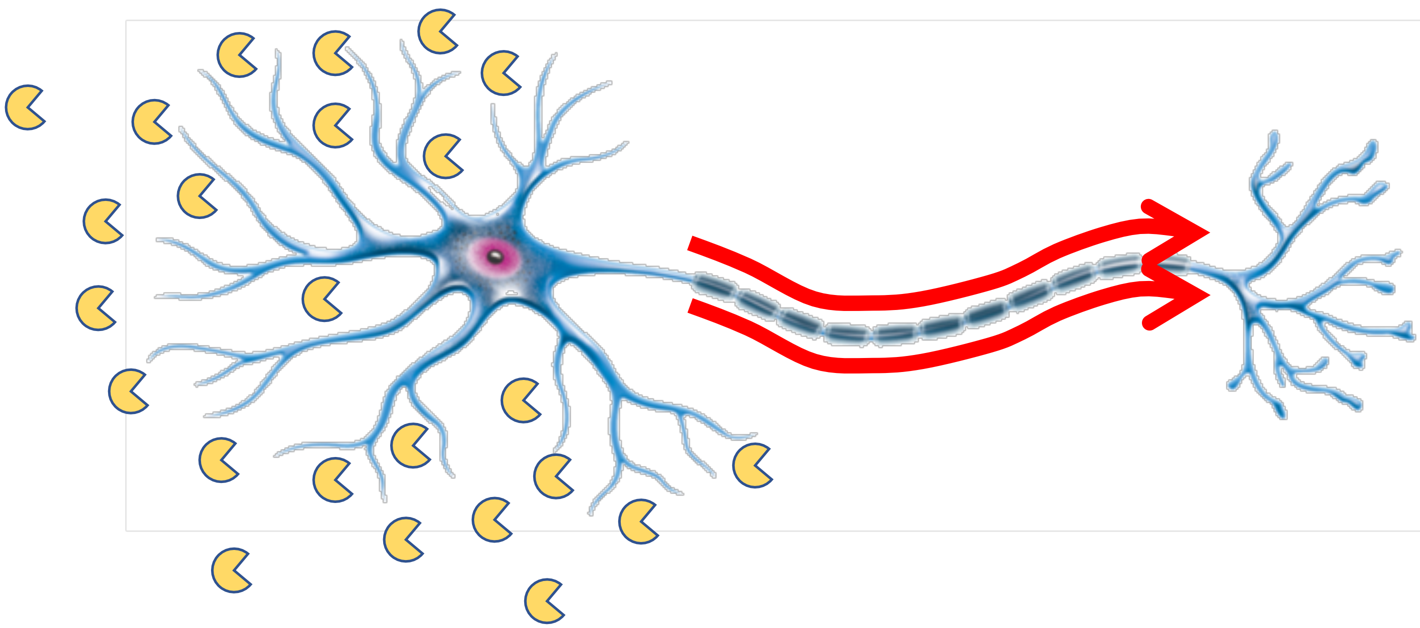
Inspiration Neuron

Neurotransmitter



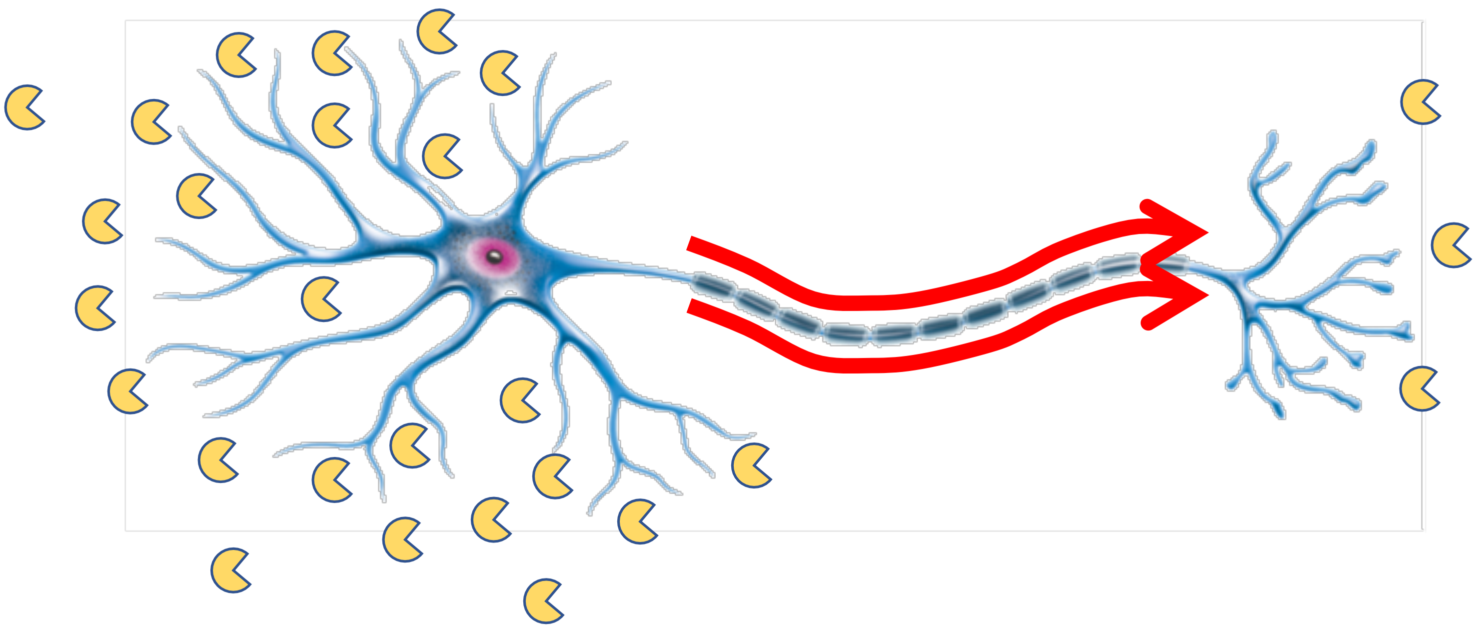
Inspiration Neuron

Neurotransmitter



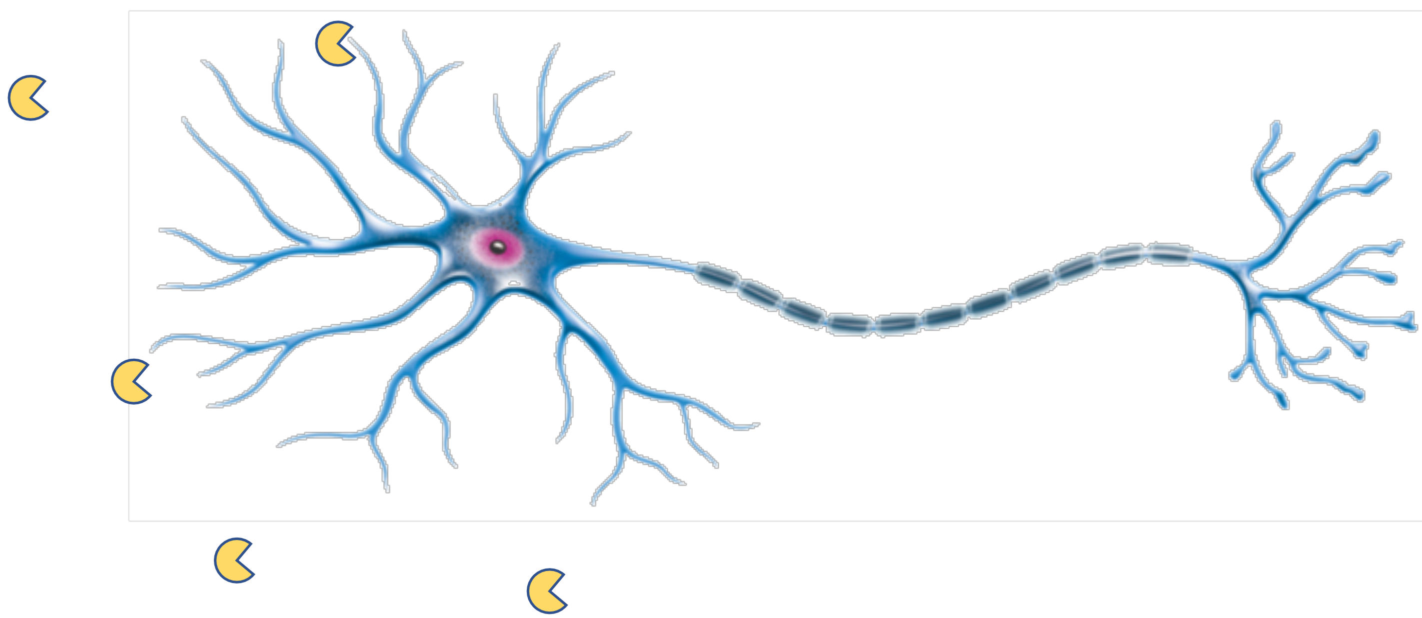
Inspiration Neuron

Neurotransmitter

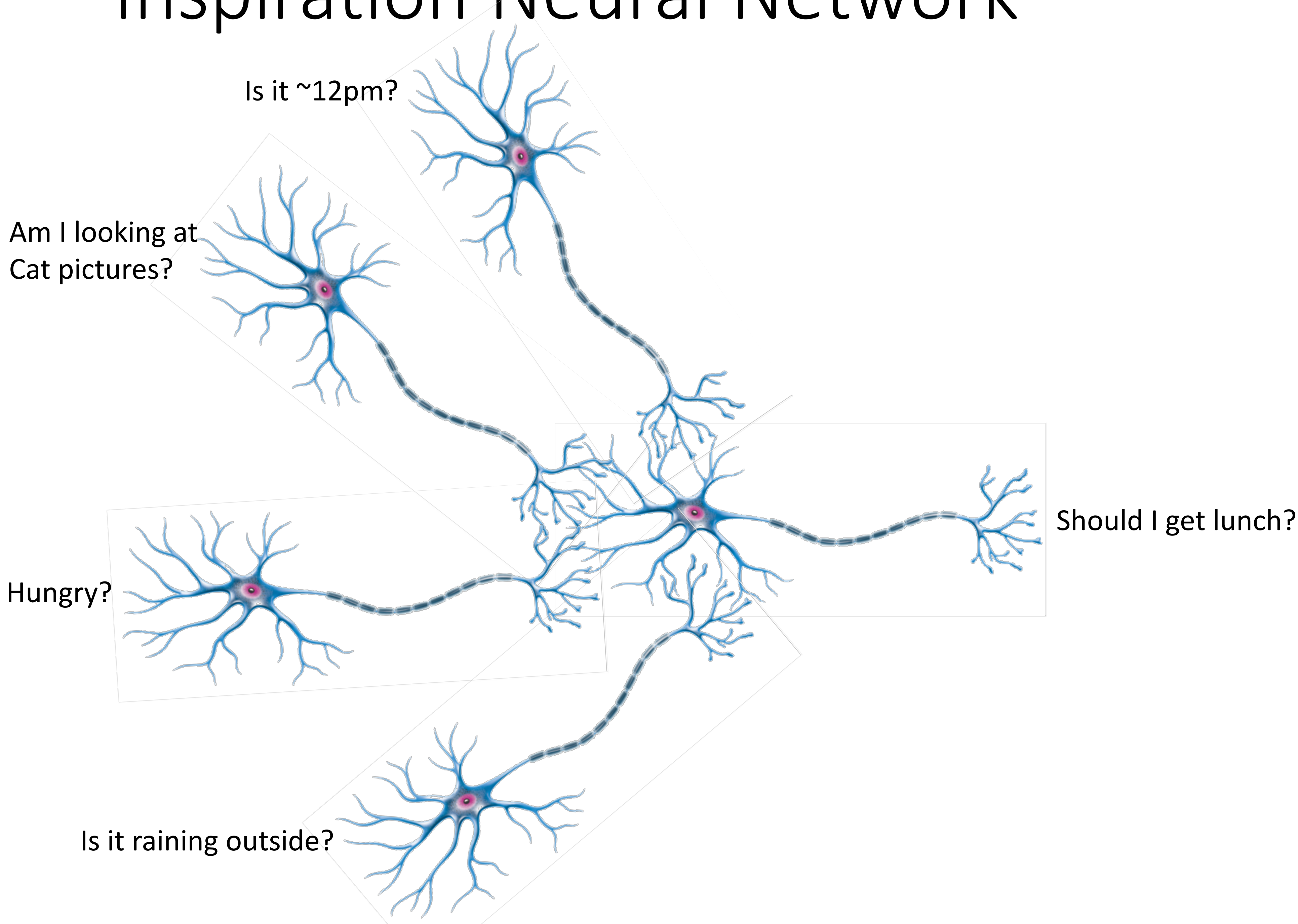


Inspiration Neuron

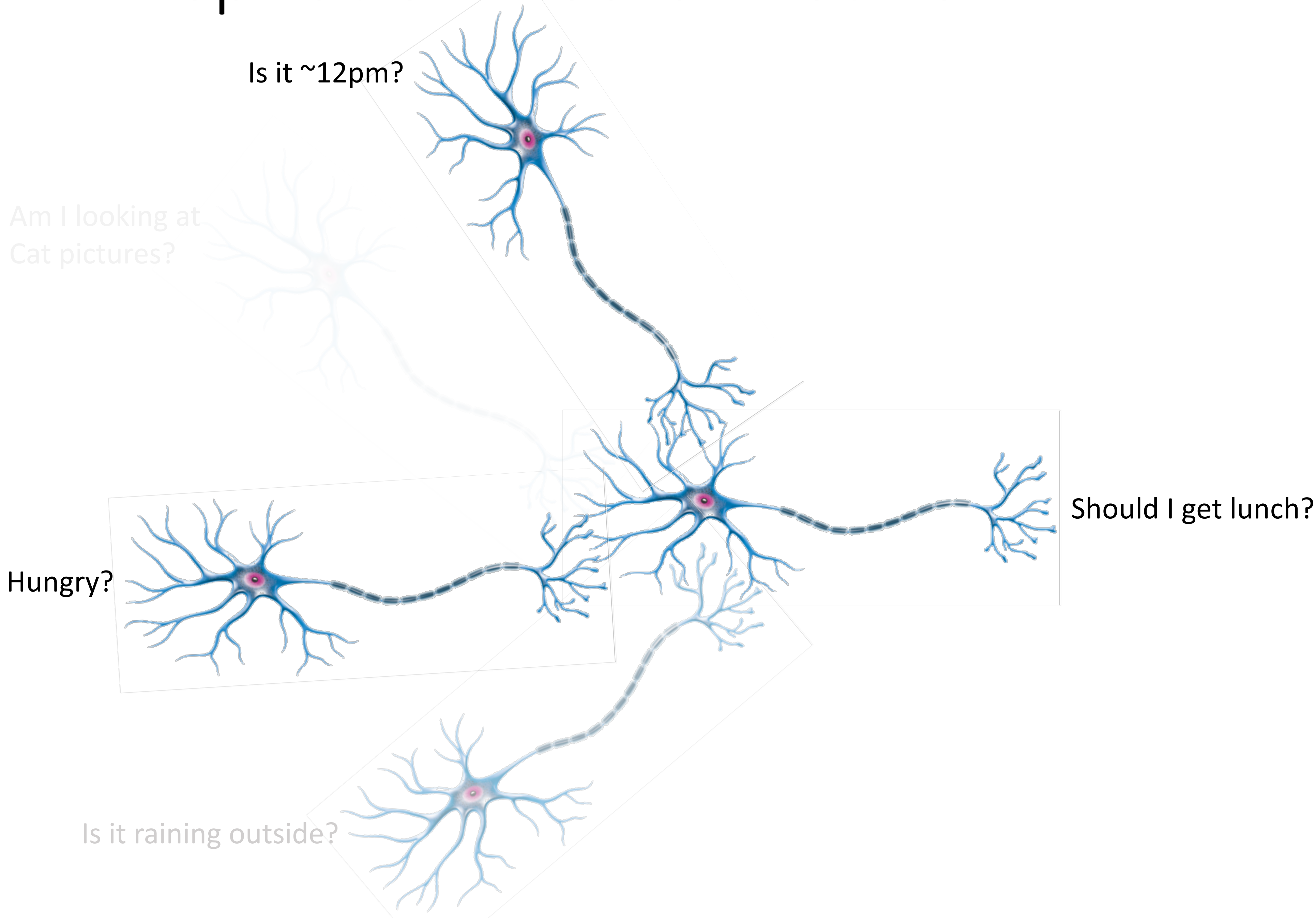
Neurotransmitter



Inspiration Neural Network

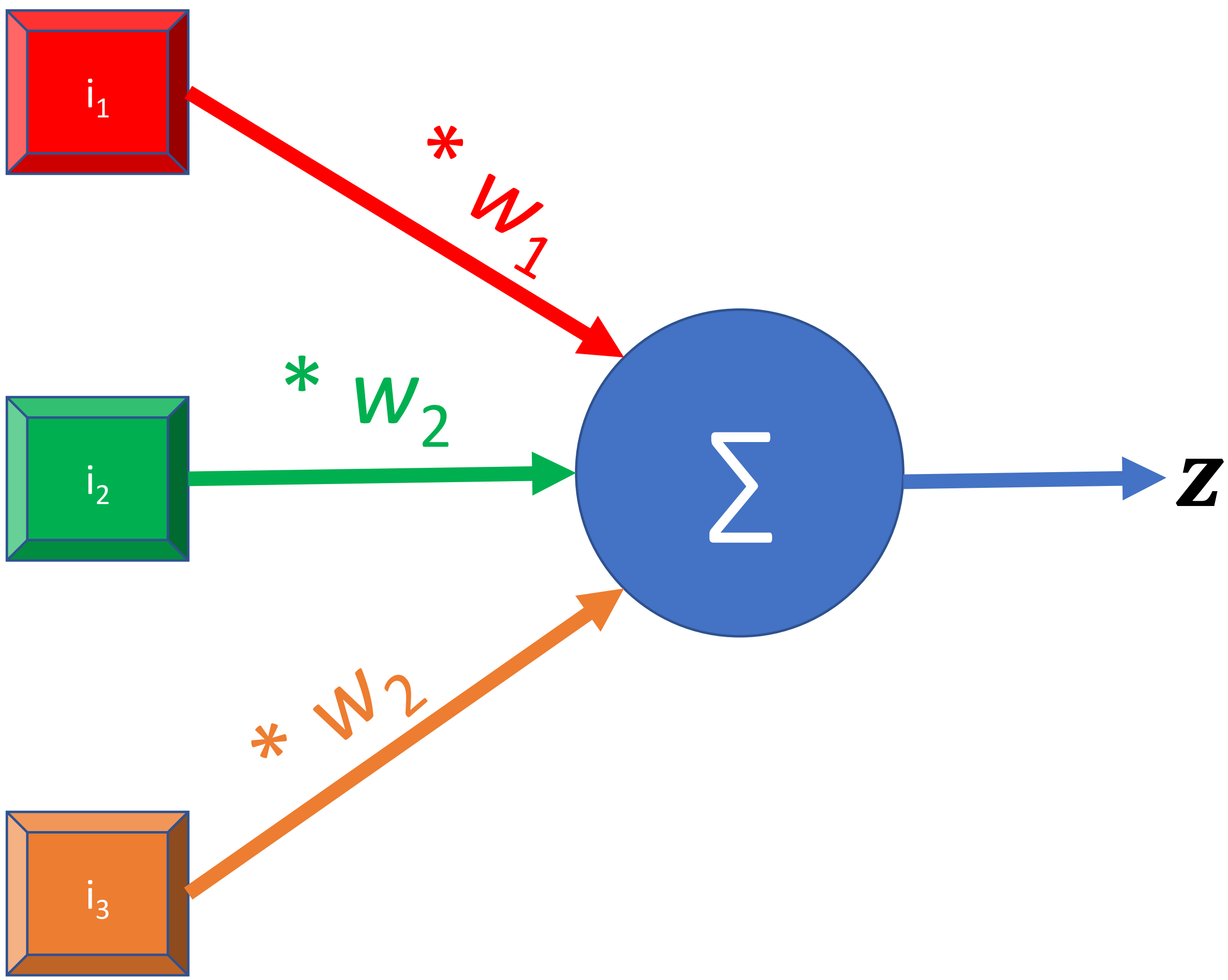


Inspiration Neural Network



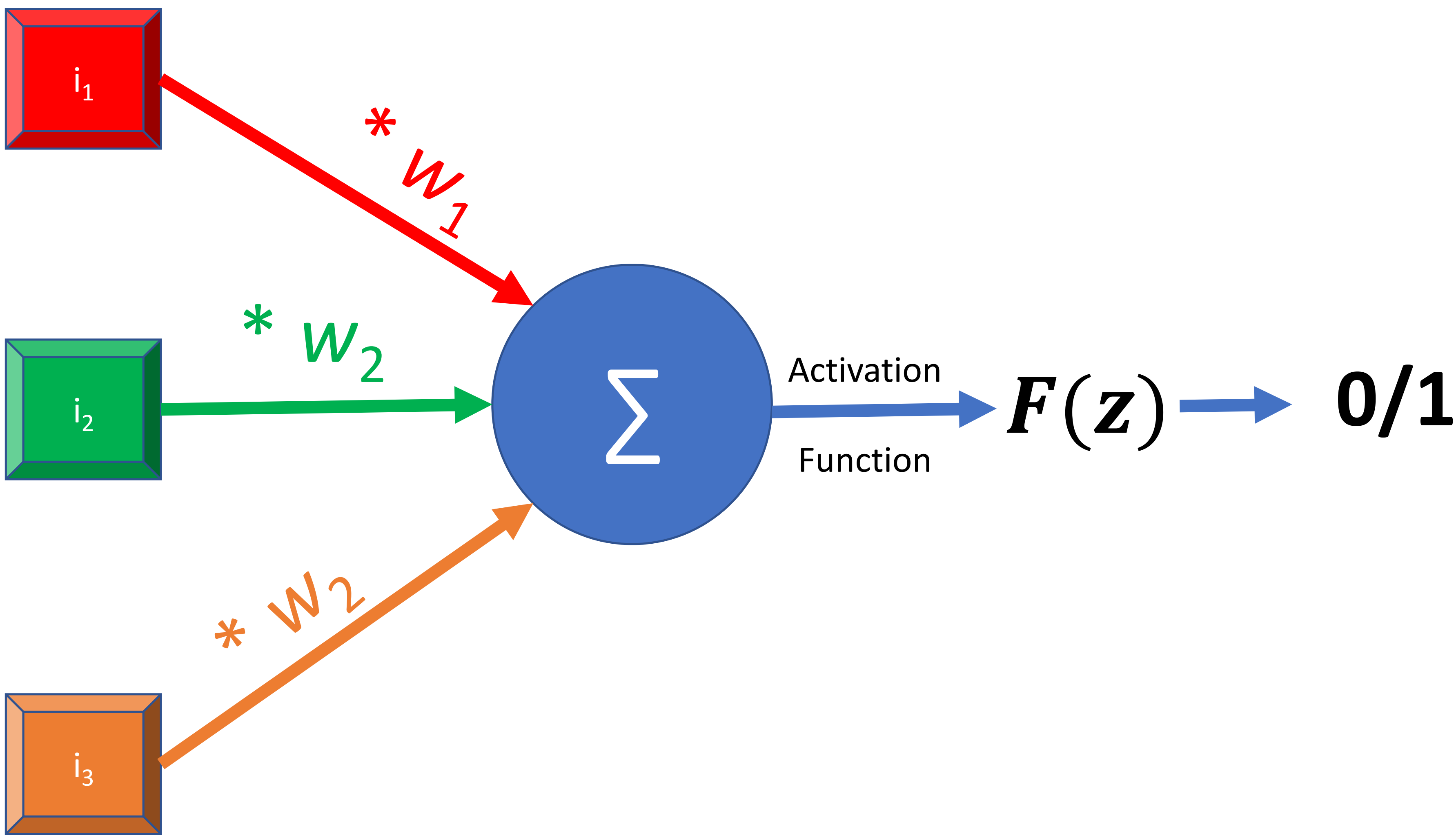
Synthetic Neuron: Perceptron

$$z = i_1 \cdot w_i + i_2 \cdot w_2 + i_3 \cdot w_3$$

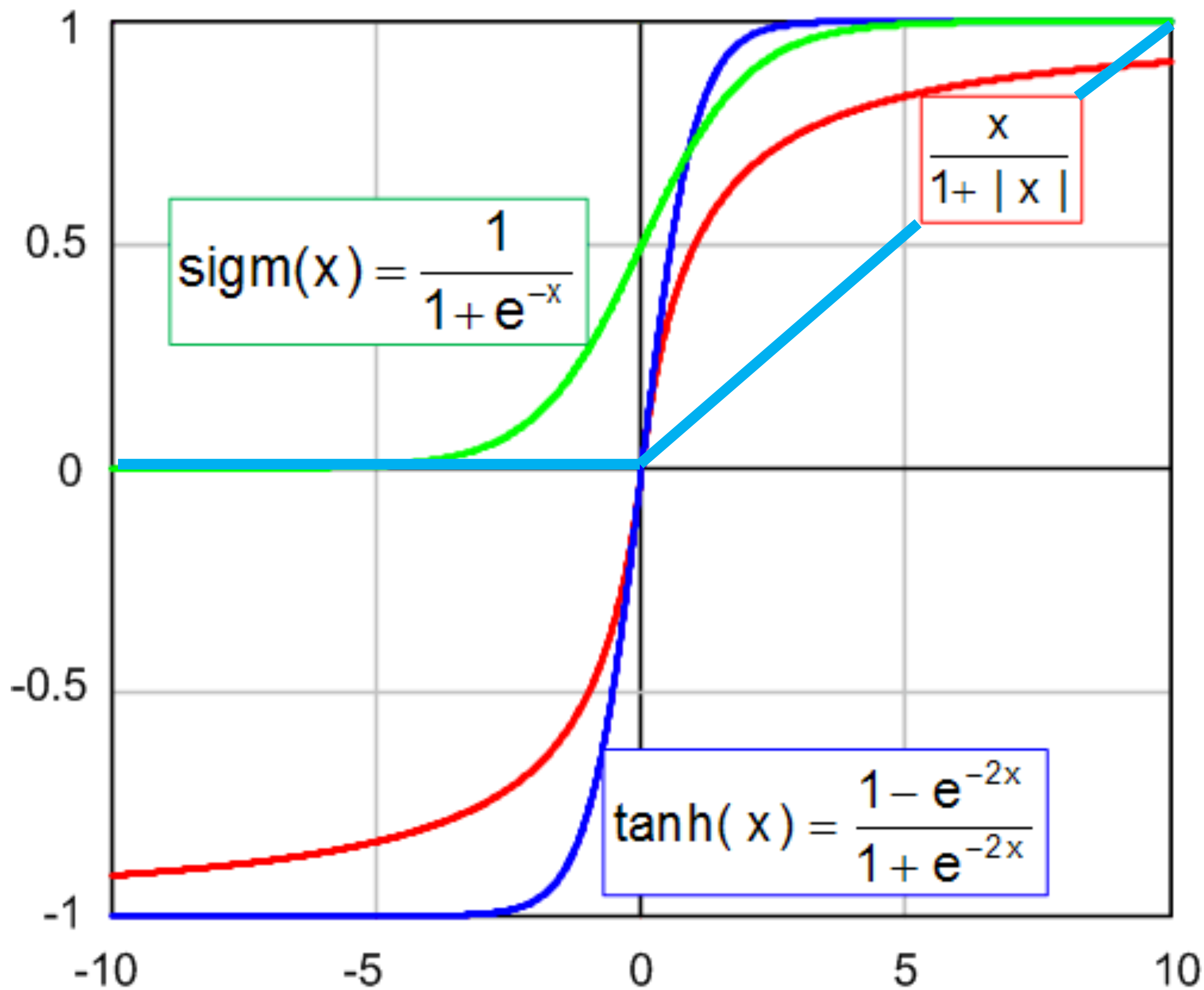


Synthetic Neuron: Perceptron

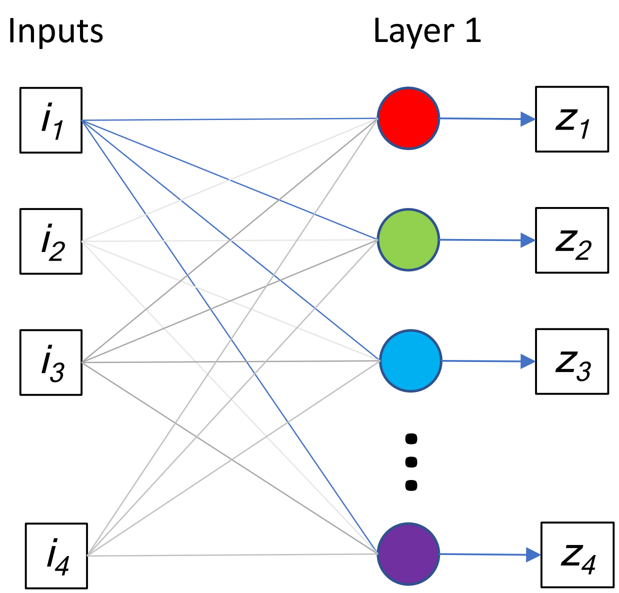
$$z = i_1 \cdot w_i + i_2 \cdot w_2 + i_3 \cdot w_3$$



Some Common Activation Functions

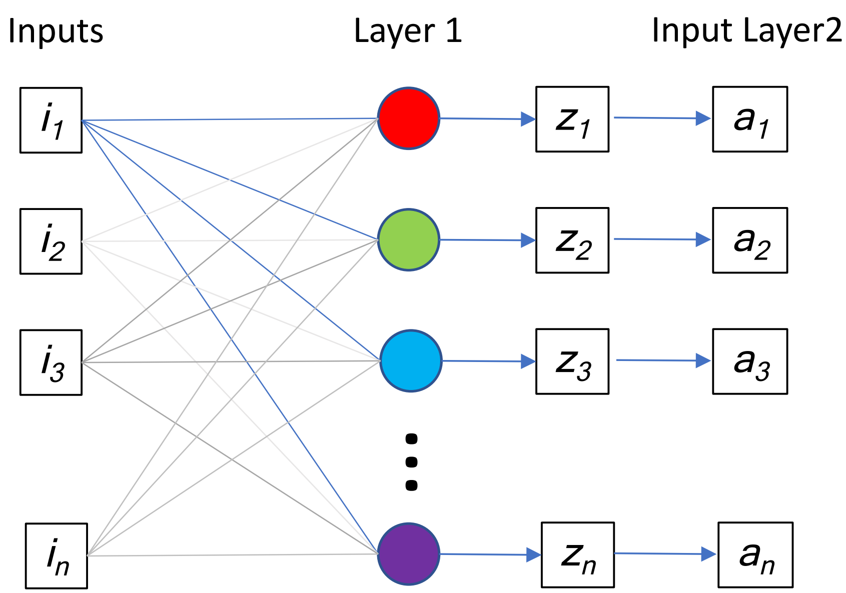


Forward Propagation: Layer 1



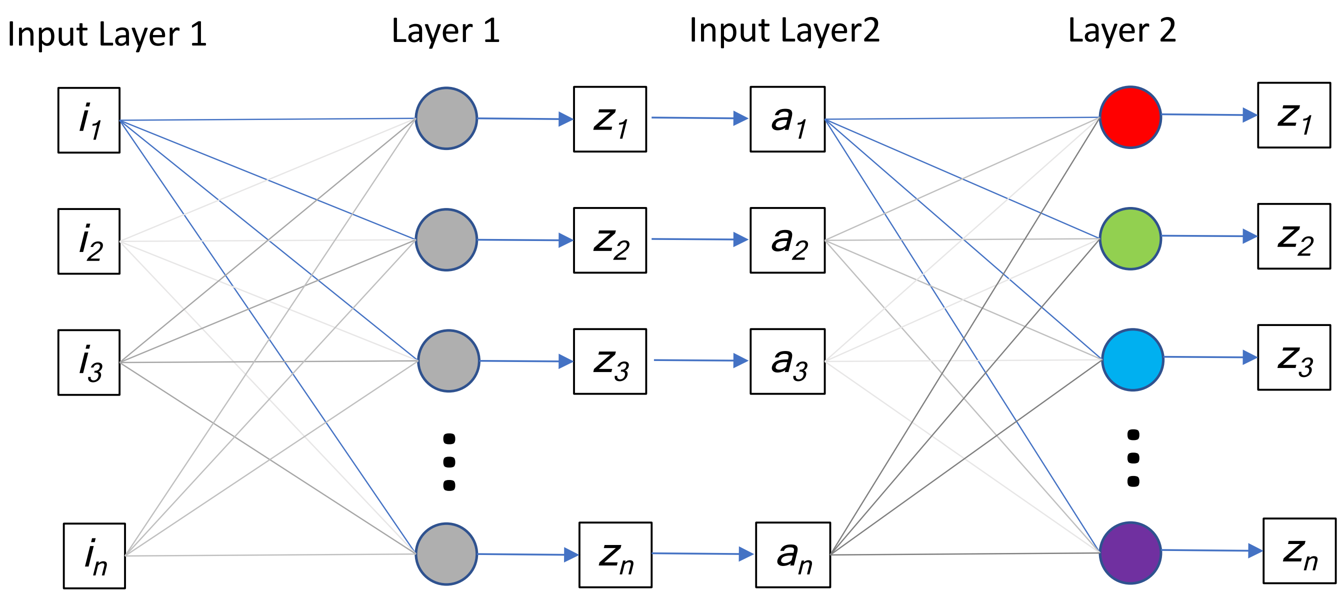
$$\begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix}$$

Forward Propagation: Layer 1



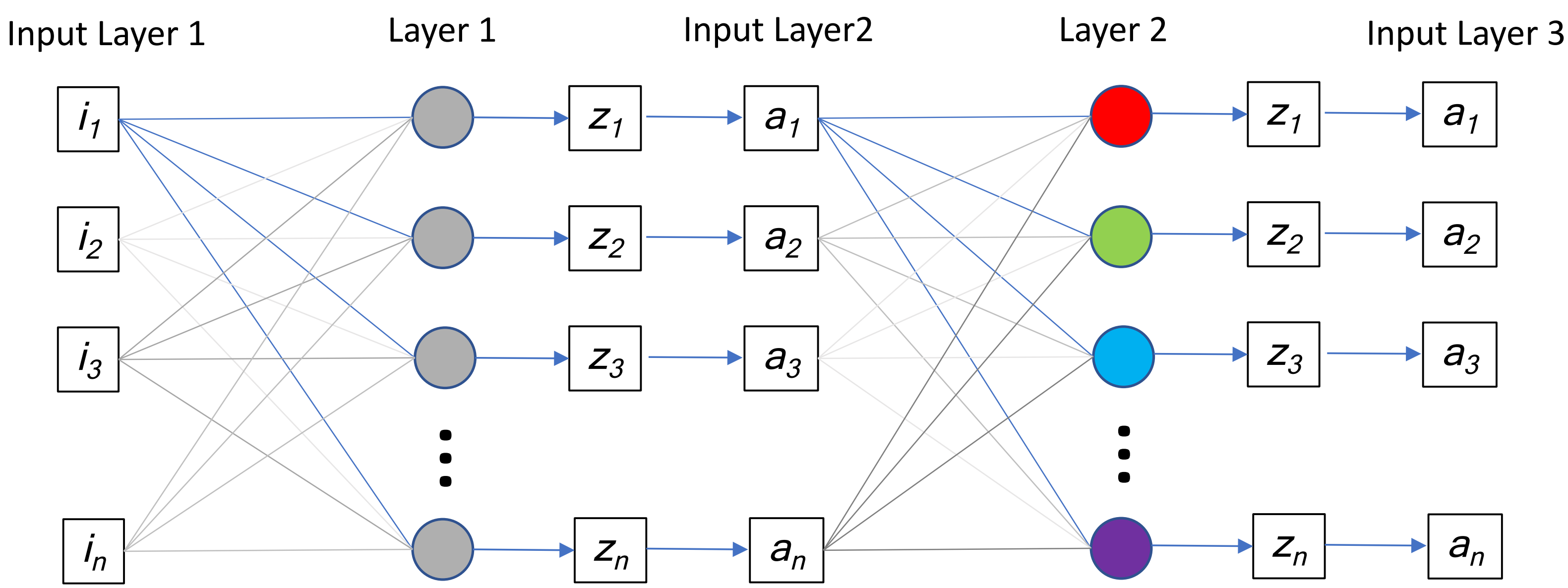
$$f\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{pmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

Forward Propagation: Layer 2



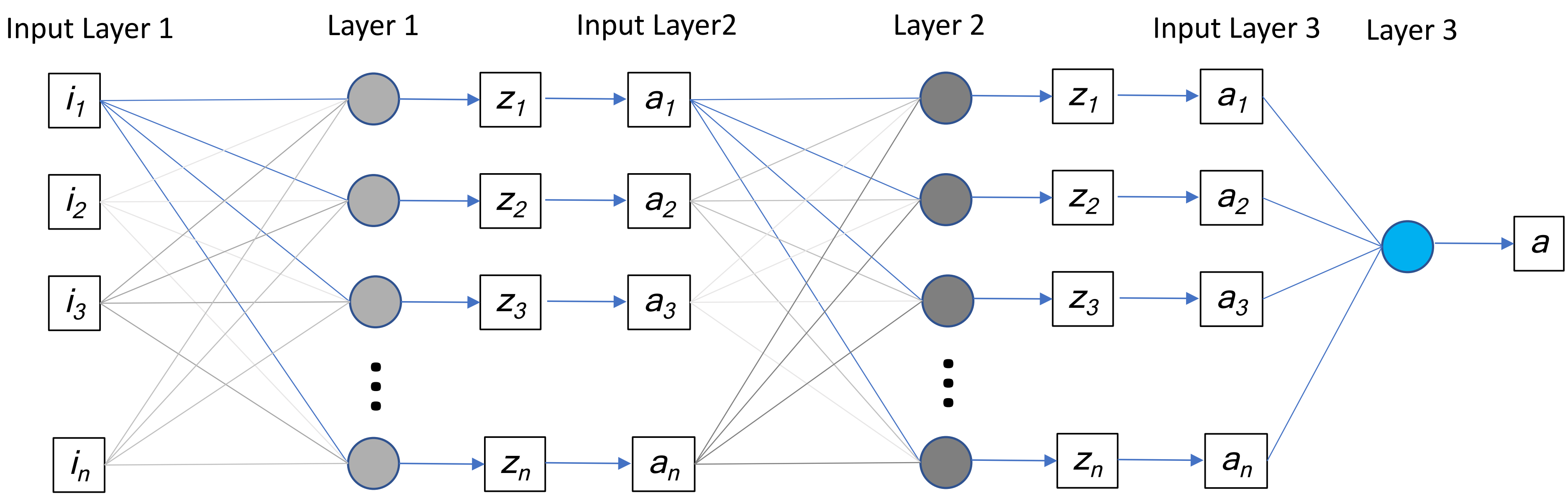
$$\begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix}$$

Forward Propagation: Layer 2



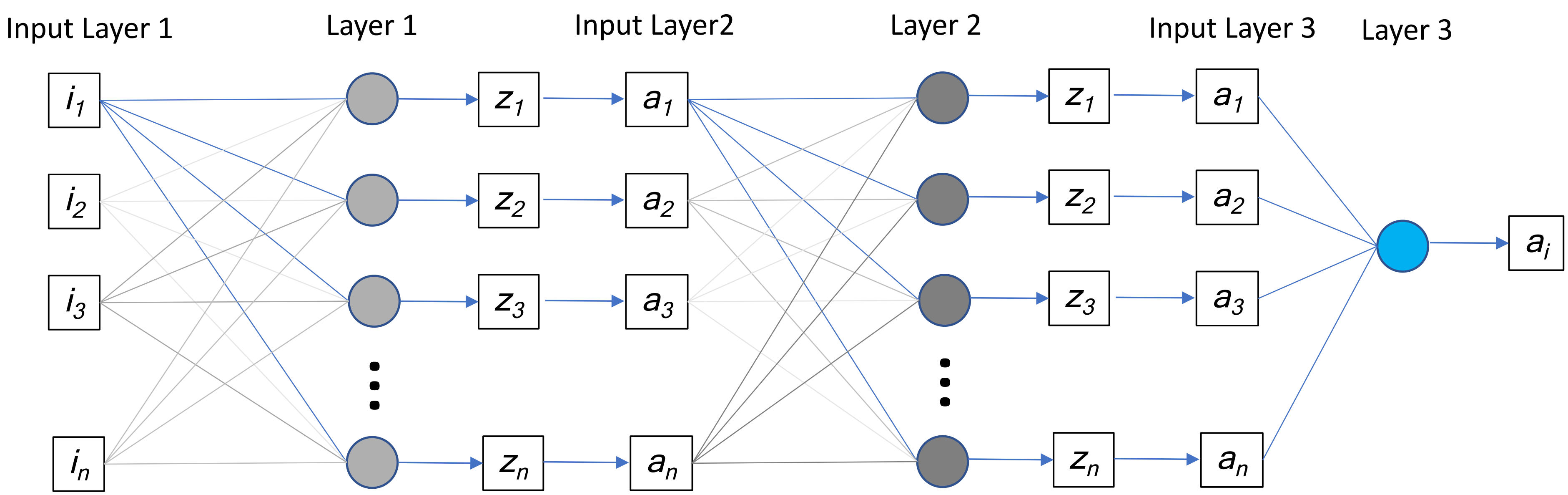
$$f\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{pmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}$$

Forward Propagation: Layer 3 (Output)



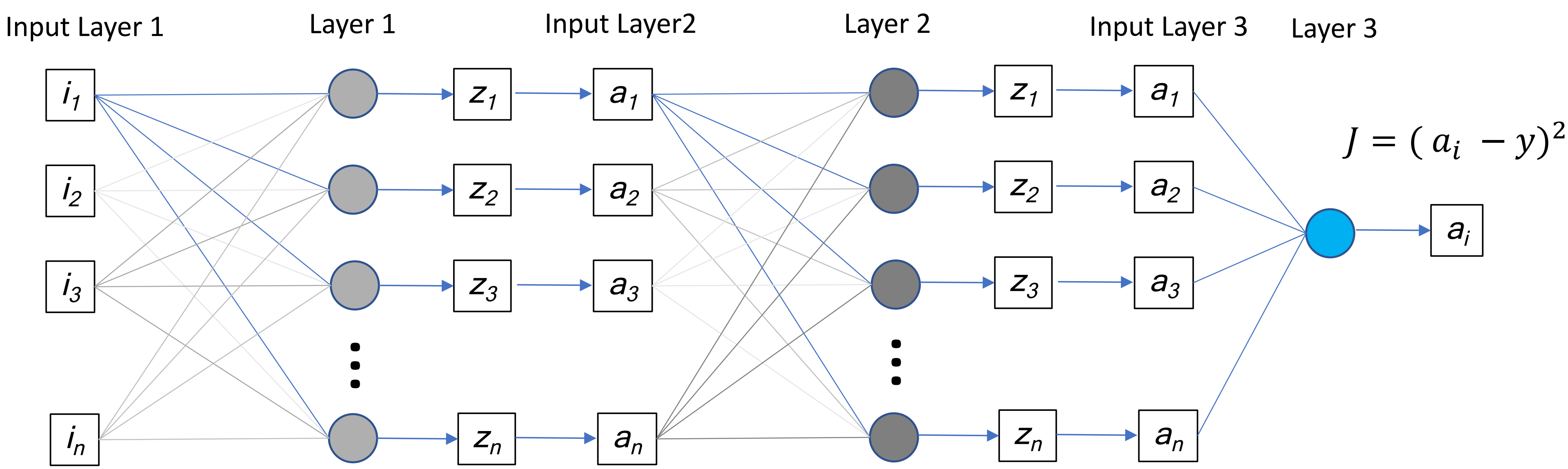
$$f\left(\begin{bmatrix} w_1^1 & w_2^1 & w_3^1 & \cdots & w_n^1 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix}\right) = a$$

Training Model Means Finding The Right Weights

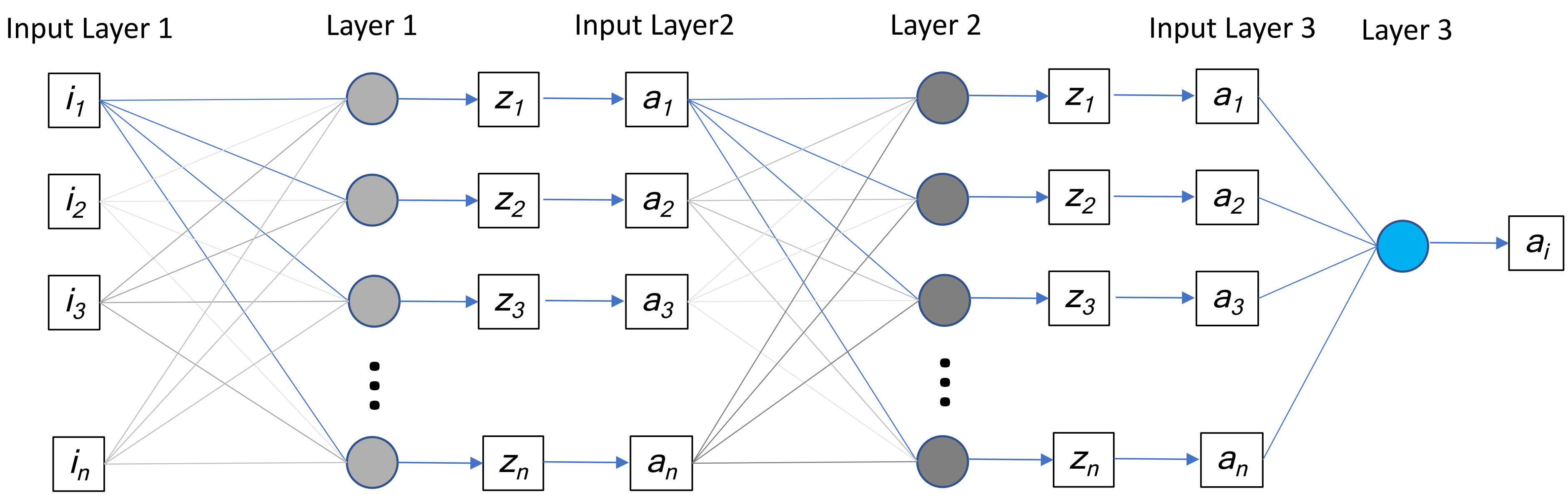


$$\begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \qquad \begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \qquad \begin{bmatrix} w_1^1 & w_2^1 & w_3^1 & \cdots & w_n^1 \end{bmatrix}$$

Backward Propagation: Define A Cost Function "J"

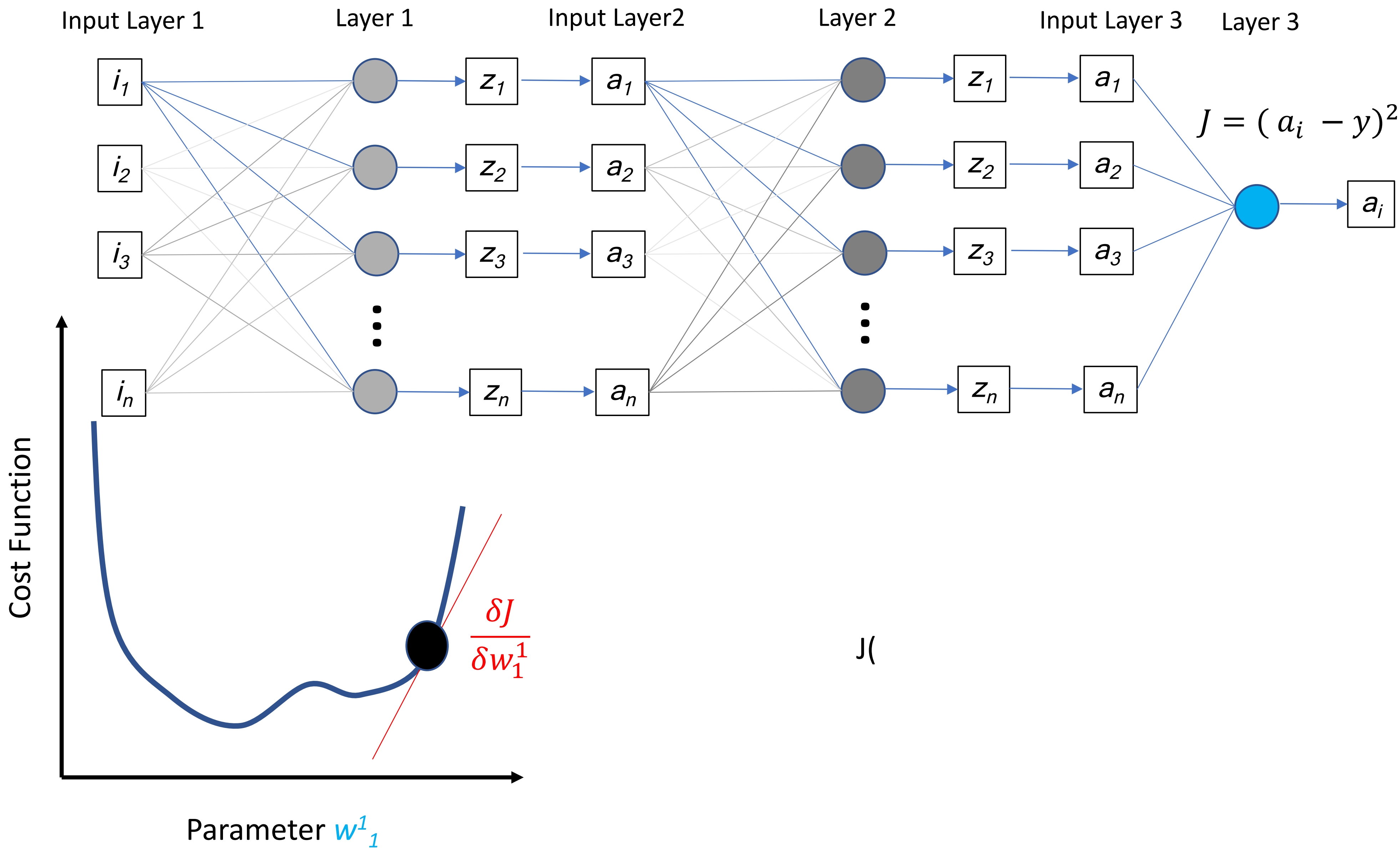


Backward Propagation: How do we find the best value for $w^1{}_1$ In the First Layer?

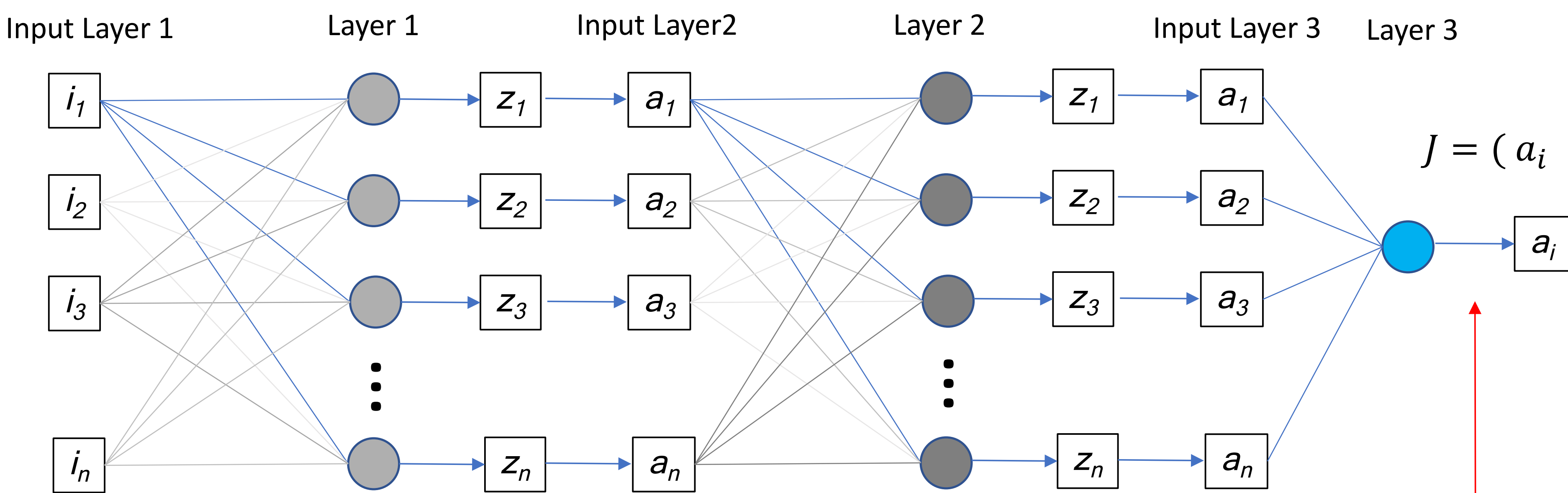


$$\begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \qquad \begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \qquad [w_1^1 \quad w_2^1 \quad w_3^1 \quad \cdots \quad w_n^1]$$

Same Way You Would By Any Gradient Descent Method



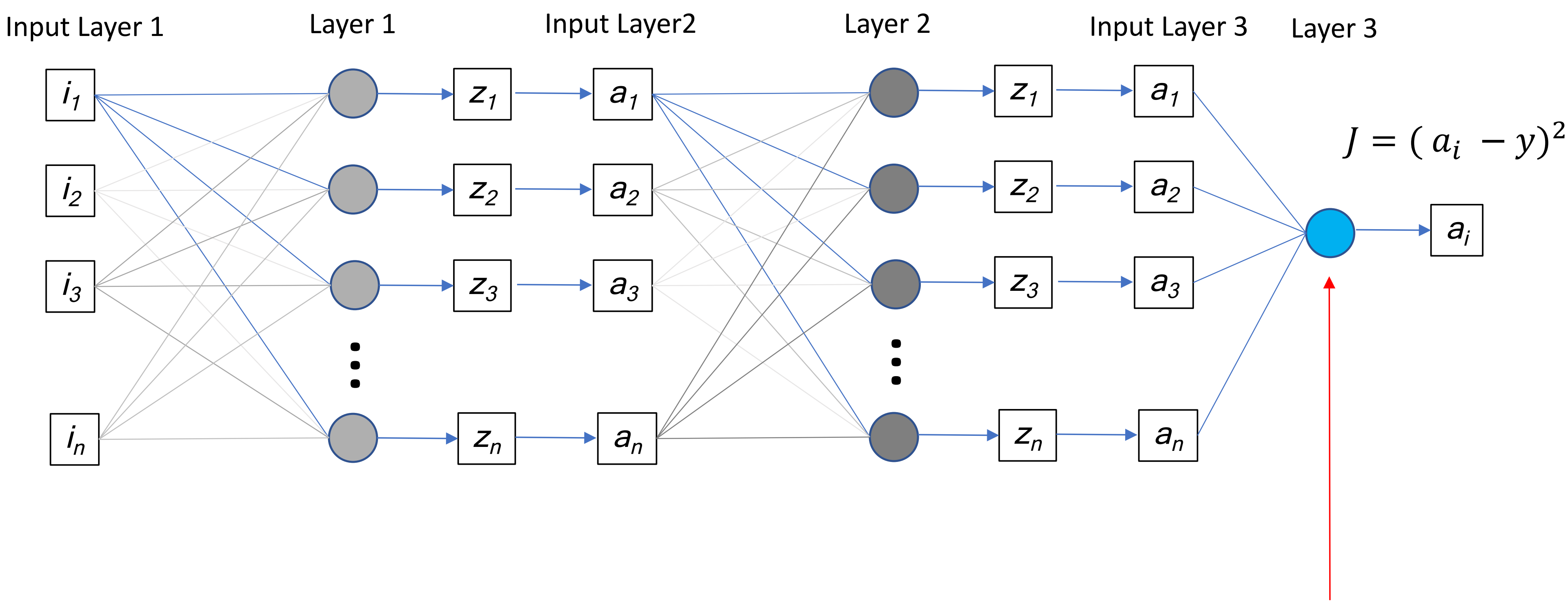
Backward Propagation: Calculate Derivative With Respect Layer 3 Activation



$$J = (a_i - y)^2$$

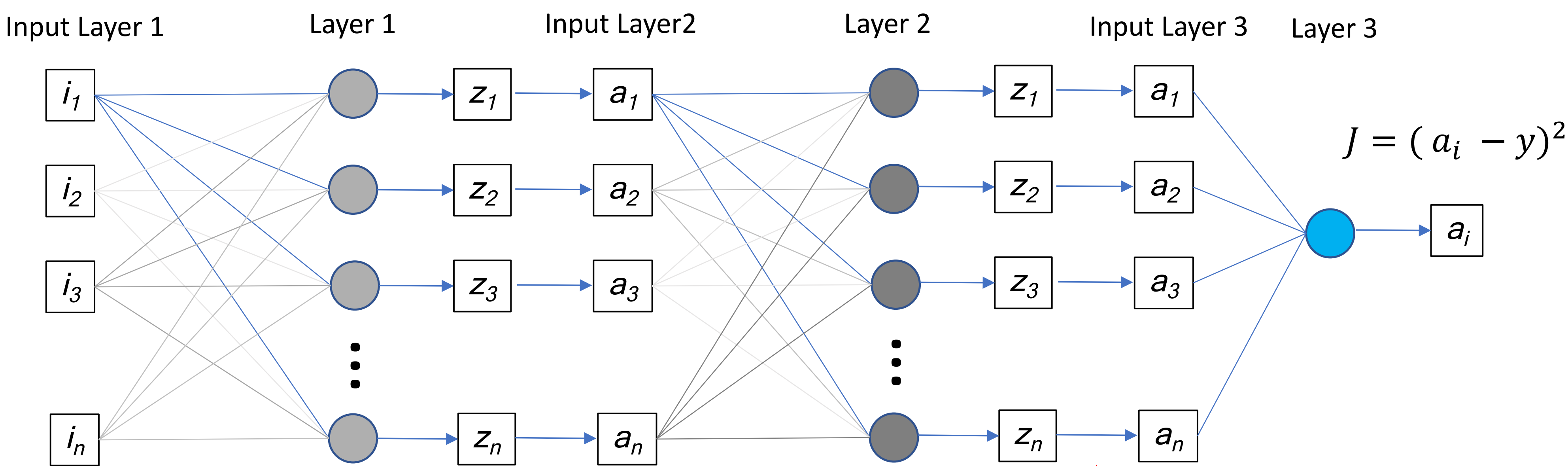
$$\left[\frac{\delta J}{\delta f(w_1^1)} \quad \frac{\delta J}{\delta f(w_2^1)} \quad \frac{\delta J}{\delta f(w_3^1)} \quad \cdots \quad \frac{\delta J}{\delta f(w_N^1)}\right]$$

Backward Propagation: Calculate Derivative With Respect to Layer 3 Weights



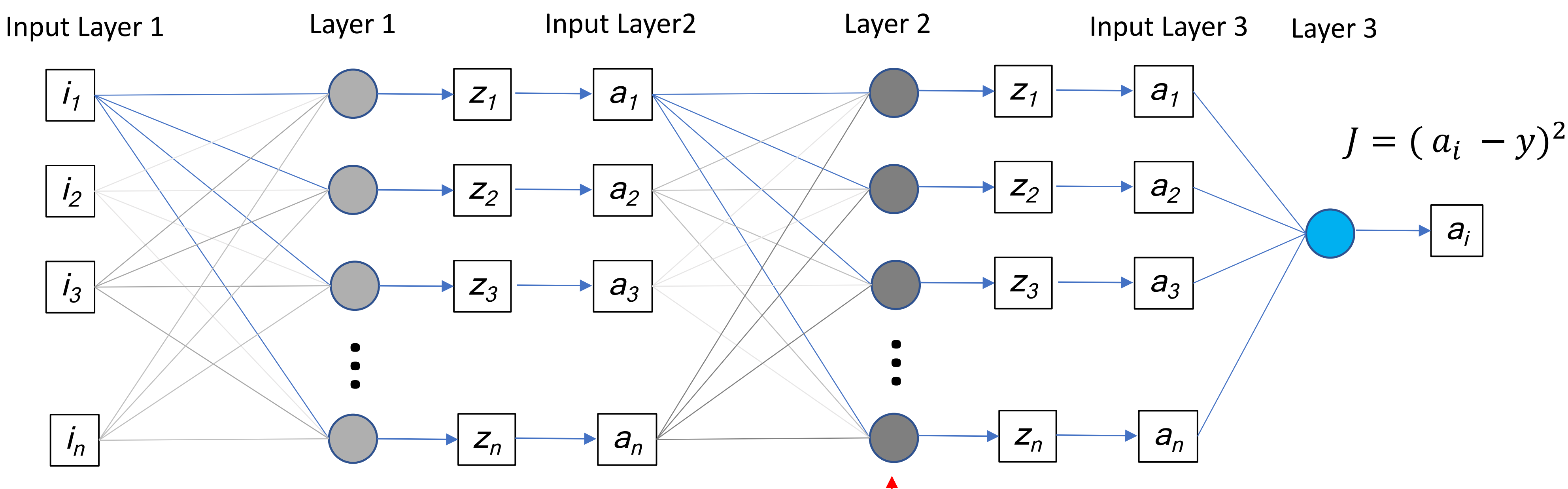
$$\left[\frac{\delta J}{\delta f(w_1^1)} \cdot \frac{\delta f(w_1^1)}{\delta w_1^1} \quad \frac{\delta J}{\delta f(w_2^1)} \cdot \frac{\delta f(w_2^1)}{\delta w_2^1} \quad \frac{\delta J}{\delta f(w_3^1)} \cdot \frac{\delta f(w_3^1)}{\delta w_3^1} \quad \cdots \quad \frac{\delta J}{\delta f(w_N^1)} \cdot \frac{\delta f(w_n^1)}{\delta w_n^1} \right]$$

Backward Propagation: Calculate Derivative With Respect to Layer 2 Activation Function!



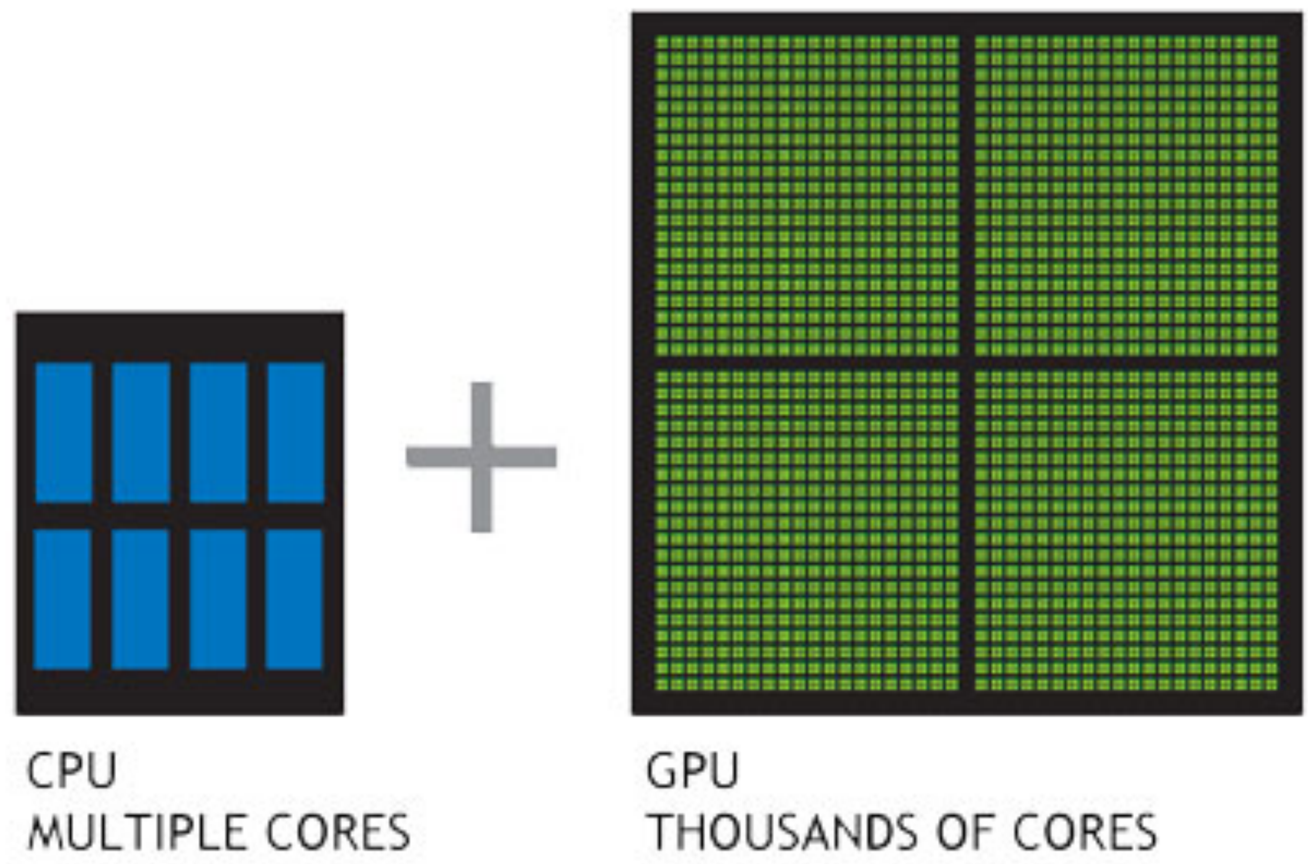
$$\left[\frac{\delta J}{\delta f(w_1^1)} \cdot \frac{\delta f(w_1^1)}{\delta w_1^1} \cdot \frac{\delta w_1^1}{\delta f(w_1^1 \ldots w_n^1)} \quad \cdots \quad \frac{\delta J}{\delta f(w_2^1)} \cdot \frac{\delta f(w_n^1)}{\delta w_n^1} \cdot \frac{\delta w_n^1}{\delta f(w_1^1 \ldots w_n^1)}\right]$$

Backward Propagation: Calculate Derivative With Respect to Layer 2 Activation Function!



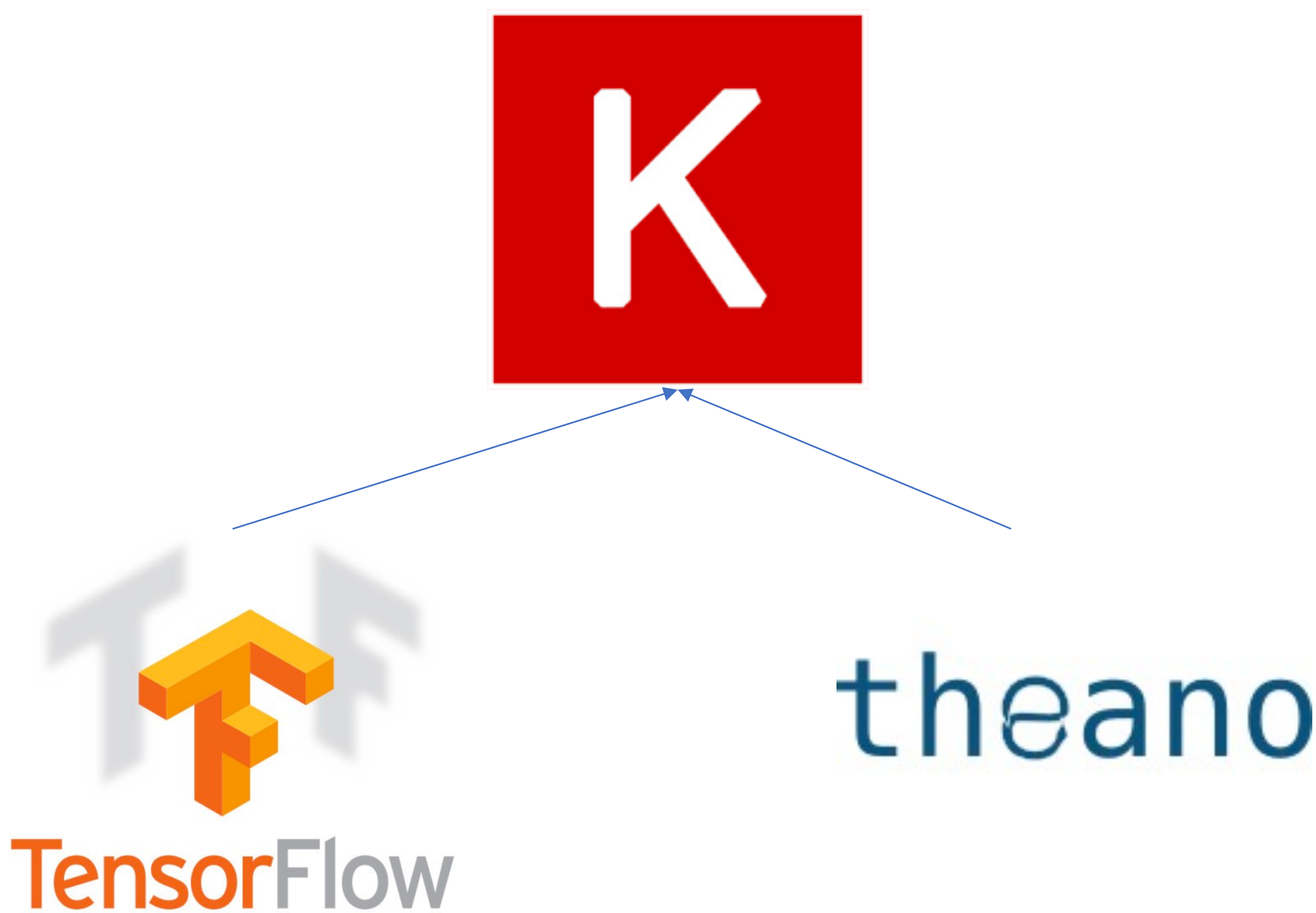
$$\left[\frac{\delta J}{\delta f(w_1^1)}\cdot\frac{\delta f(w_1^1)}{\delta w_1^1}\cdot\frac{\delta w_1^1}{\delta f(w_1^1 \dots w_n^1)} \quad \cdots \quad \frac{\delta J}{\delta f(w_2^1)}\cdot\frac{\delta f(w_n^1)}{\delta w_n^1}\cdot\frac{\delta w_n^1}{\delta f(w_1^1 \dots w_n^1)}\right] \mathrm{d}\begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix}$$

The Derivatives Are Mathematically Simple … Just *A Lot* Of Terms to Calculate, Many Many Times



GPU Programing Is A Full Time Job

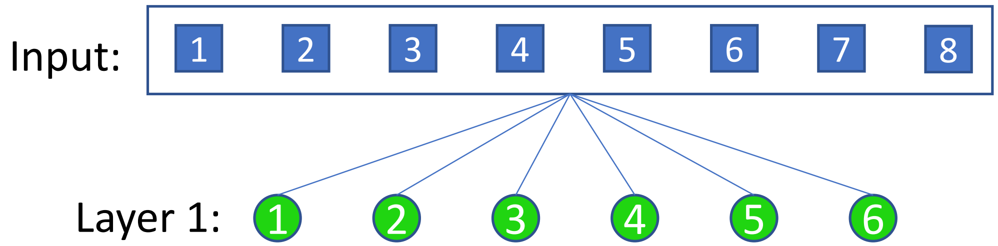
Nice Python API



The Keras API

```
model  = Sequential(   )
```

The Keras API



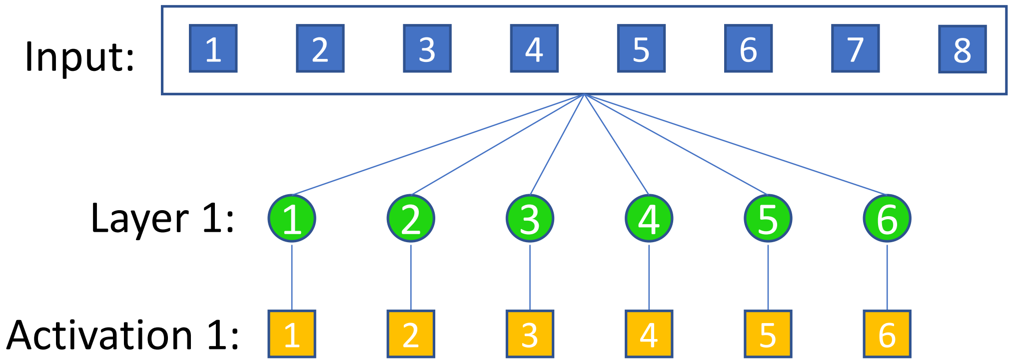
Number of Features /
Weights for Each Neuron

```
model  = Sequential(   )

model.add(   Dense(   6,    input_dim=8 )    )
```

Number of Neurons
In Layer

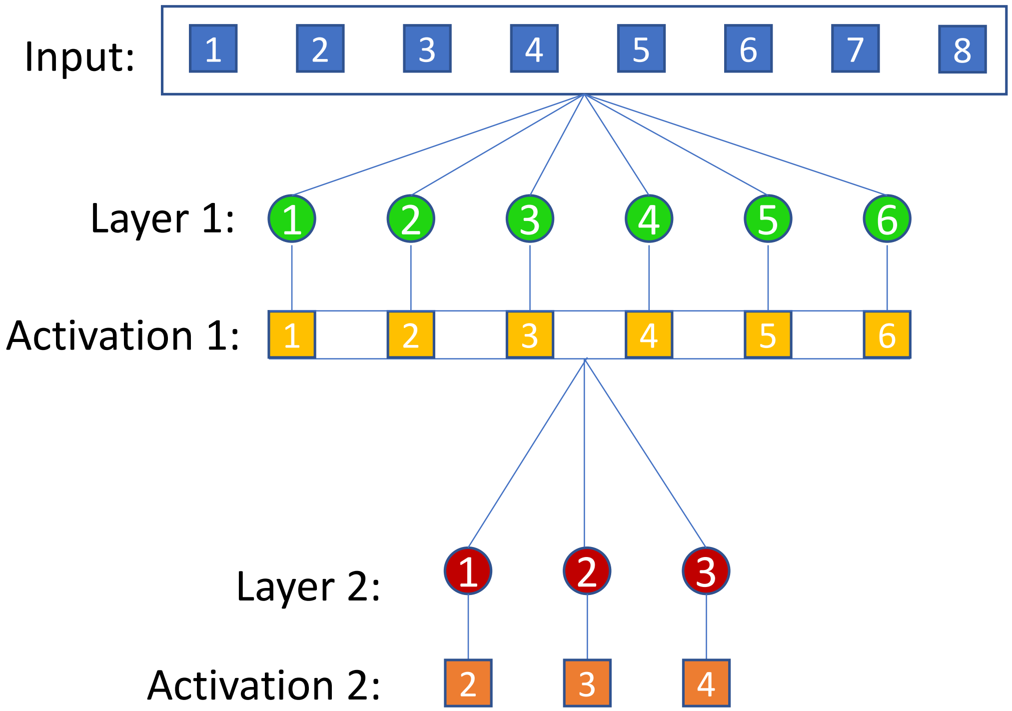
The Keras API

```
model  = Sequential(   )

model.add(   Dense(   6,    input_dim=8 )    )

model.add(   Activation( 'tanh'  )    )
```



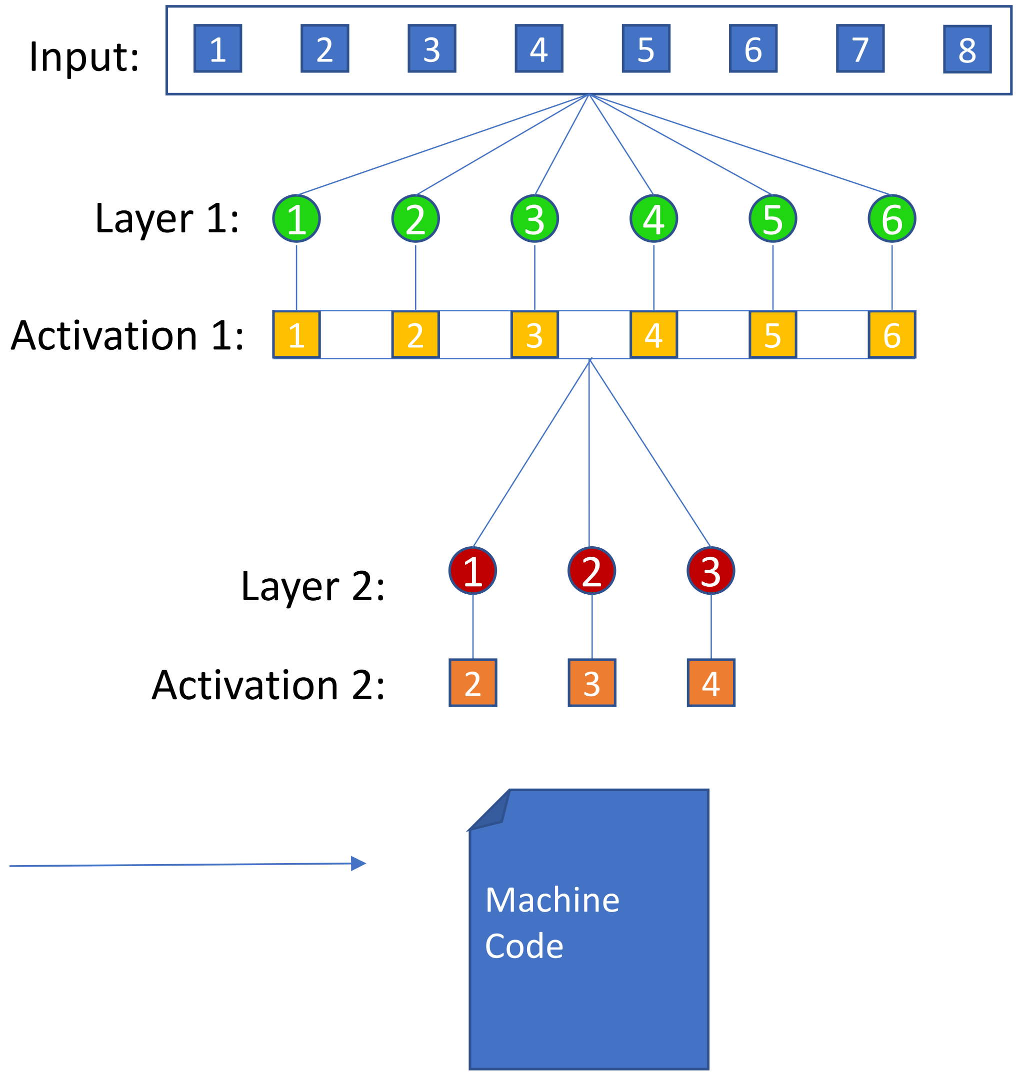
The Keras API

```
model  = Sequential(   )

model.add(   Dense(   6,    input_dim=8 )    )

model.add(   Activation( 'tanh'  )    )

model.add(    Dense(  3  )    )

model.add(   Activation( 'tanh'  )    )
```



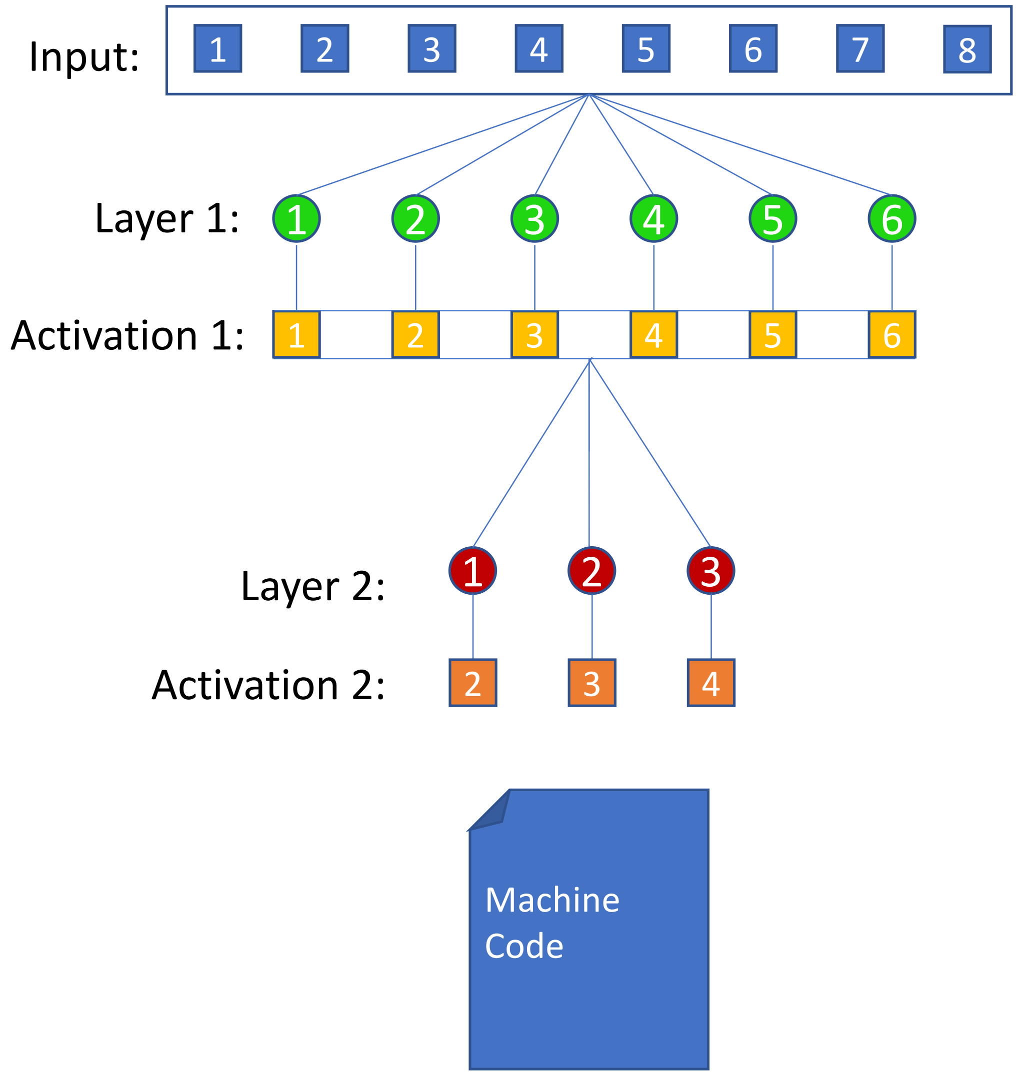
The Keras API

```
model  = Sequential(   )

model.add(   Dense(   6,    input_dim=8 )    )

model.add(   Activation( 'tanh'  )    )

model.add(    Dense(  3  )    )

model.add(   Activation( 'tanh'  )    )

model.compile( optimizer, loss='logloss', metrics  )
```



The Keras API

```
model  = Sequential(   )

model.add(   Dense(   6,    input_dim=8 )    )

model.add(   Activation( ‘tanh’  )    )

model.add(    Dense(  3  )    )

model.add(   Activation( ‘tanh’  )    )


model.compile( optimizer, loss=‘logloss’, metrics  )


model.fit( X_data, y_data )
```

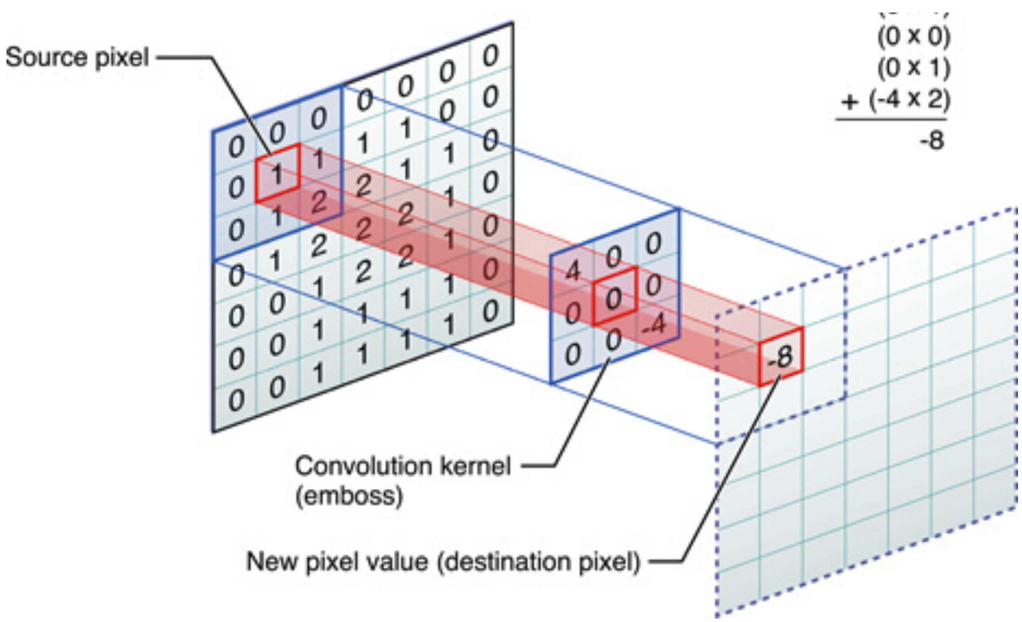


NN Invented In 1970's … Were Not Terribly Useful

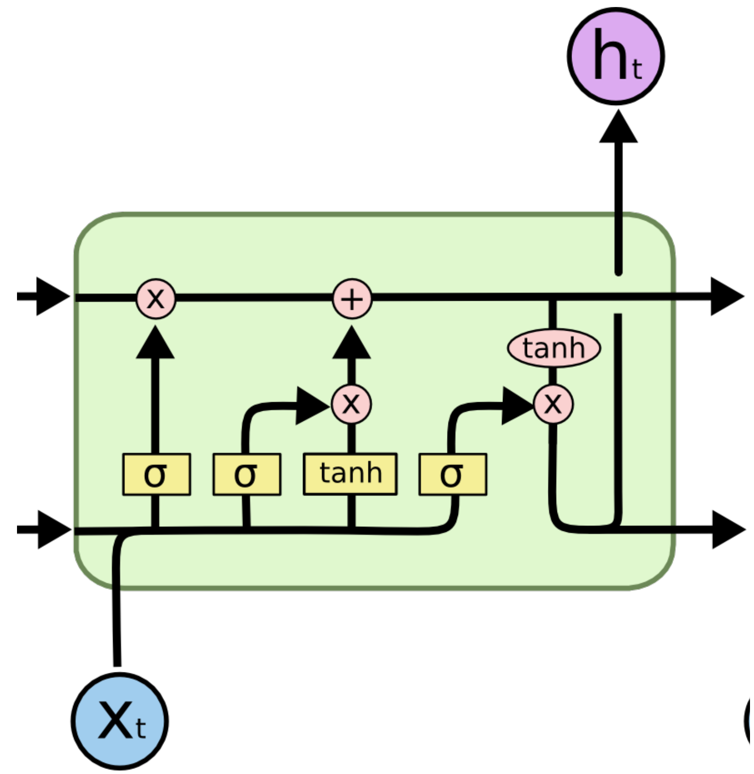
FEATURE ENGINEERING >> Algorithm

Then In The 2000's NN Learned To Engineer Features

Convolution



Recurrent Neural Network



Convolution

Kernel

-1	-1	-1
1	1	1
0	0	0

"Image"

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution

Convolution

Kernel

-1	-1	-1
1	1	1
0	0	0

"Image"

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution

12			

Convolution

Kernel

-1	-1	-1
1	1	1
0	0	0

"Image"

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution

12	4		

Convolution

Kernel

-1	-1	-1
1	1	1
0	0	0

“Image”

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution

12	4	-2	

Convolution

Kernel

-1	-1	-1
1	1	1
0	0	0

"Image"

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution

12	4	-2	-8

Convolution

Kernel

-1	-1	-1
1	1	1
0	0	0

“Image”

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

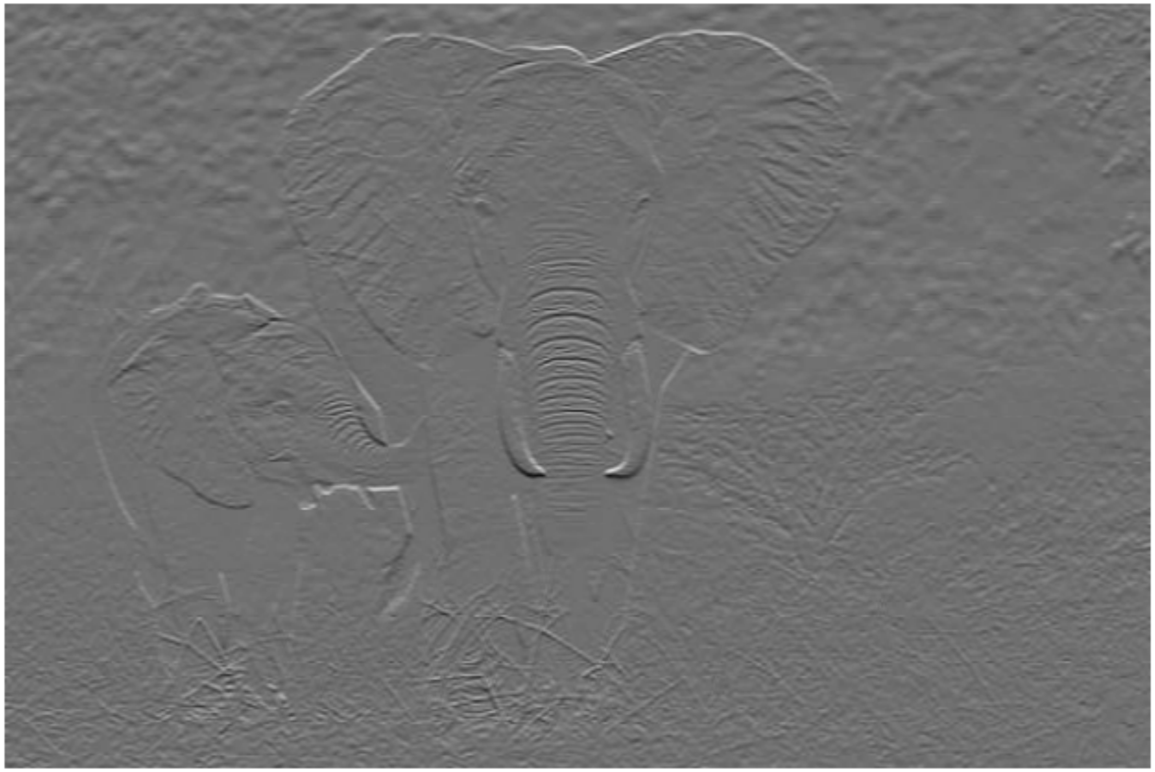
Convolution

12	4	-2	-8
0			

Convolution

Convolution

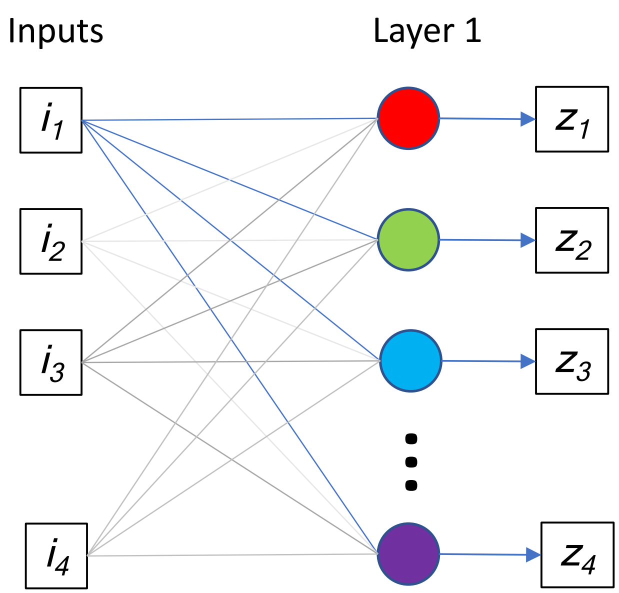
Top Edge Filter



-1	-1	-1
1	1	1
0	0	0

Make The NN Design The Convolution Kernels

One Layer



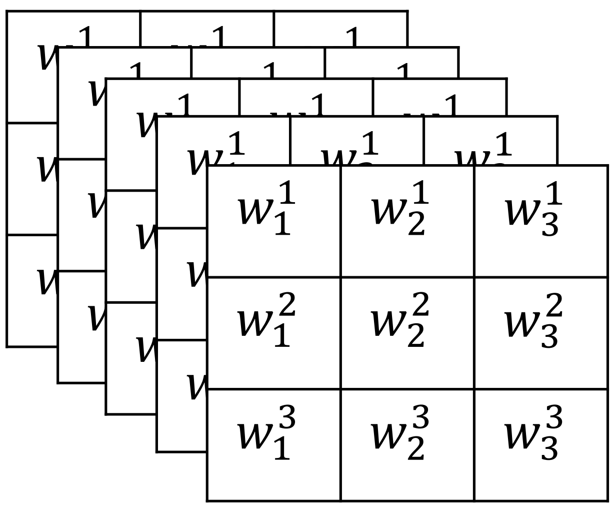
$$\begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_1^2 & \cdots & w_n^2 \\ w_1^3 & w_2^3 & \cdots & w_n^3 \\ \vdots & \vdots & & \vdots \\ w_1^P & w_2^P & \cdots & w_n^P \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \\ i_n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_n \end{bmatrix}$$

Design N Convolutional Kernels

w_1^1	w_2^1	w_3^1
w_1^2	w_2^2	w_3^2
w_1^3	w_2^3	w_3^3

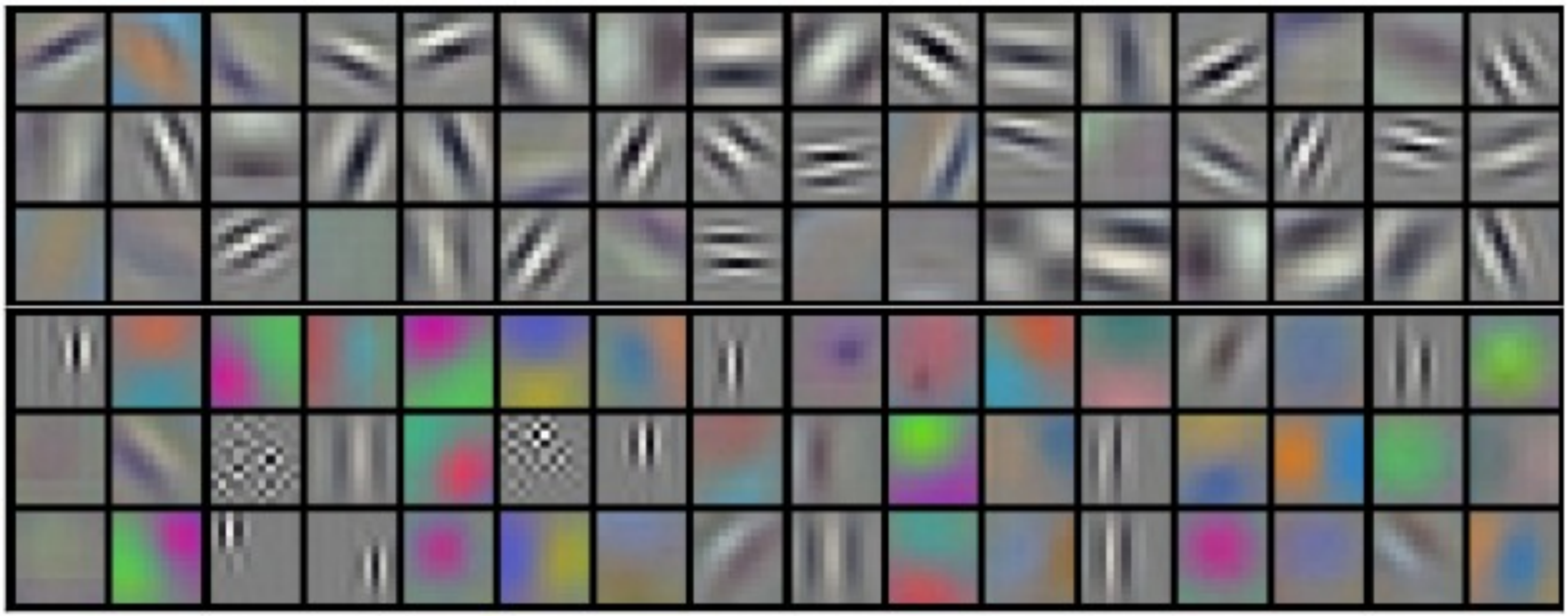
Apply Multiple Convolutional Kernels To Extract Different Kinds of Features

Design 5 (3x3) Convolutional Kernels




```
model  = Sequential(   )

model.add( Convolution2d( 5, 3, 3 ) )

model.add( Flatten() )

model.compile( optimizer, loss='logloss', metrics  )


model.fit( X_data, y_data )
```

Some of the 128 Kernels In The First Layer of VGG16



Apply Many Layers of Convolution

“Image”

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution 1

12	4	-2	-8
0	3	9	11
-13	-14	-14	-11
7	9	12	14

Convolution 2

Kernel

-1	-1	-1
1	1	1
0	0	0

Convolution

"Image"

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution 1

12	4	-2	-8
0	3	9	11
-13	-14	-14	-11
7	9	12	14

Convolution 2

-4	

Kernel

-1	-1	-1
1	1	1
0	0	0

Convolution

“Image”

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution 1

12	4	-2	-8
0	3	9	11
-13	-14	-14	-11
7	9	12	14

Convolution 2

-4	

Kernel

-1	-1	-1
1	1	1
0	0	0

Convolution

“Image”

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution 1

12	4	-2	-8
0	3	9	11
-13	-14	-14	-11
7	9	12	14

Convolution 2

-4	

Kernel

-1	-1	-1
1	1	1
0	0	0

Convolution

“Image”

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution 1

12	4	-2	-8
0	3	9	11
-13	-14	-14	-11
7	9	12	14

Convolution 2

-4	29

Kernel

-1	-1	-1
1	1	1
0	0	0

Convolution

"Image"

4	0	1	8	3	1
6	4	7	2	1	1
3	5	9	2	8	5
2	1	1	0	4	0
3	1	7	3	7	8
6	7	5	3	0	9

Convolution 1

12	4	-2	-8
0	3	9	11
-13	-14	-14	-11
7	9	12	14

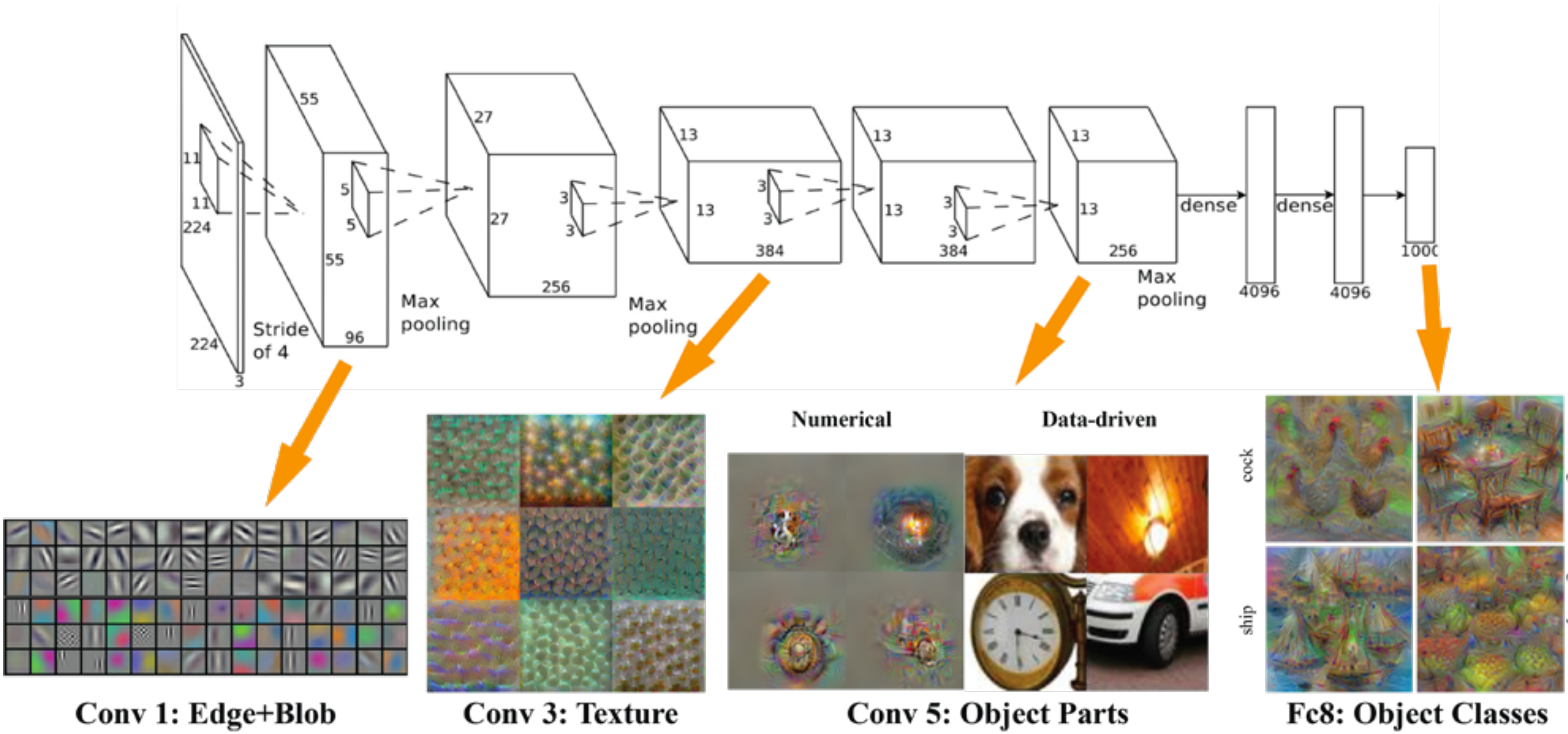
Convolution 2

-4	29
-53	

Kernel

-1	-1	-1
1	1	1
0	0	0

VGG16 – Each Convolutional Layer Learns To Recognize More Complex Features



Transfer Learning