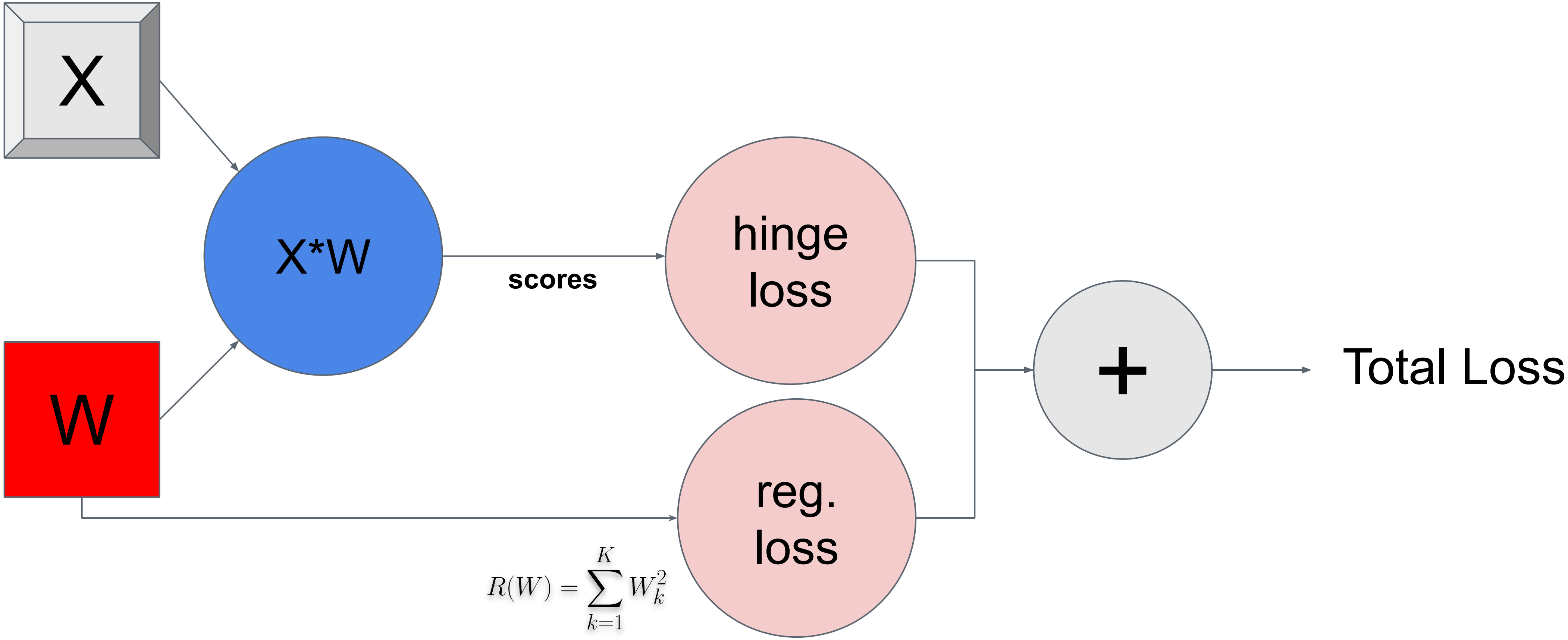
DATA 442: Neural Networks & Deep Learning

Dan Runfola – danr@wm.edu

icss.wm.edu/data442/

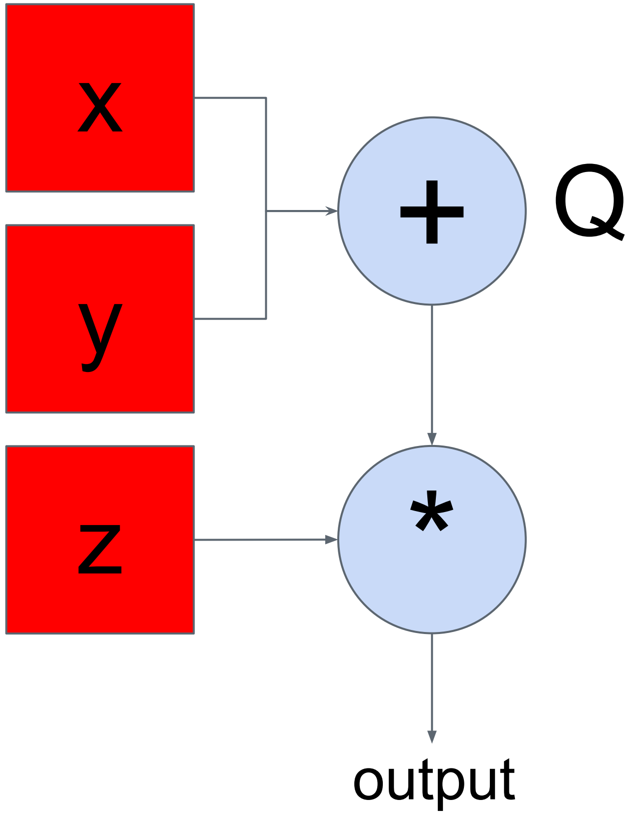
$$f(X, W)$$

$$\sum_{j \neq y_i}^{J} max(0, s_j - s_{y_i} + \varepsilon)$$



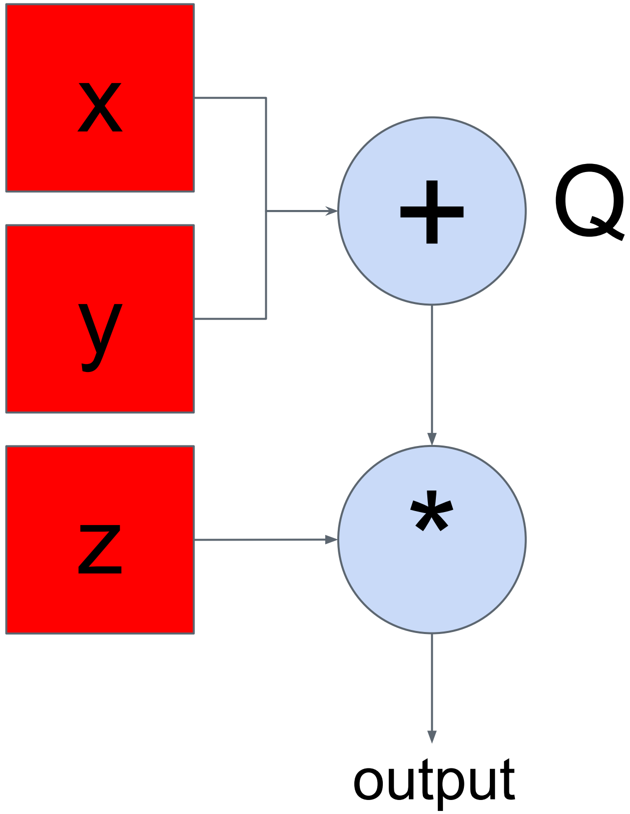
$$R(W) = \sum_{k=1}^{K} W_k^2$$

$$f(x, y, z) = (x + y) * z$$



$$Q = x + y$$

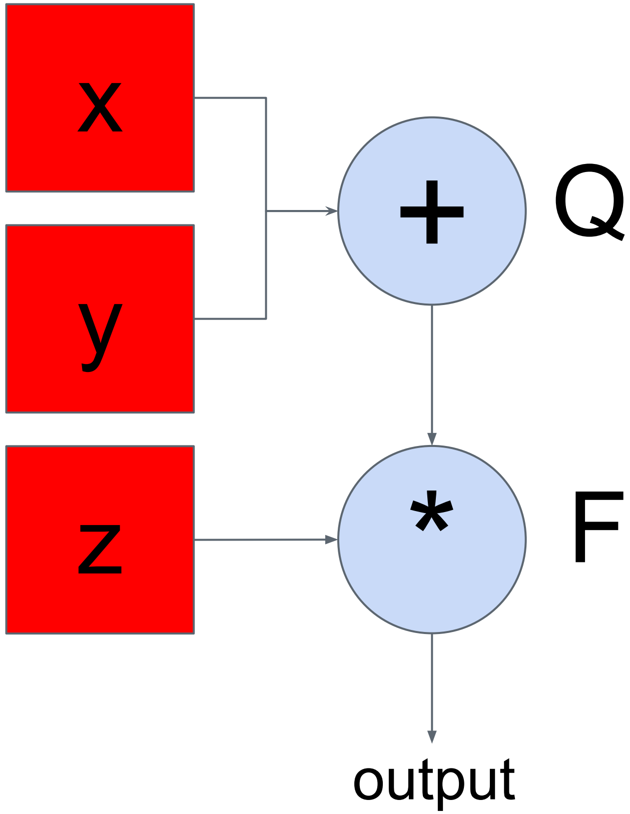
$$f(x, y, z) = (x + y) * z$$



$$Q = x + y$$

$$\frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$$

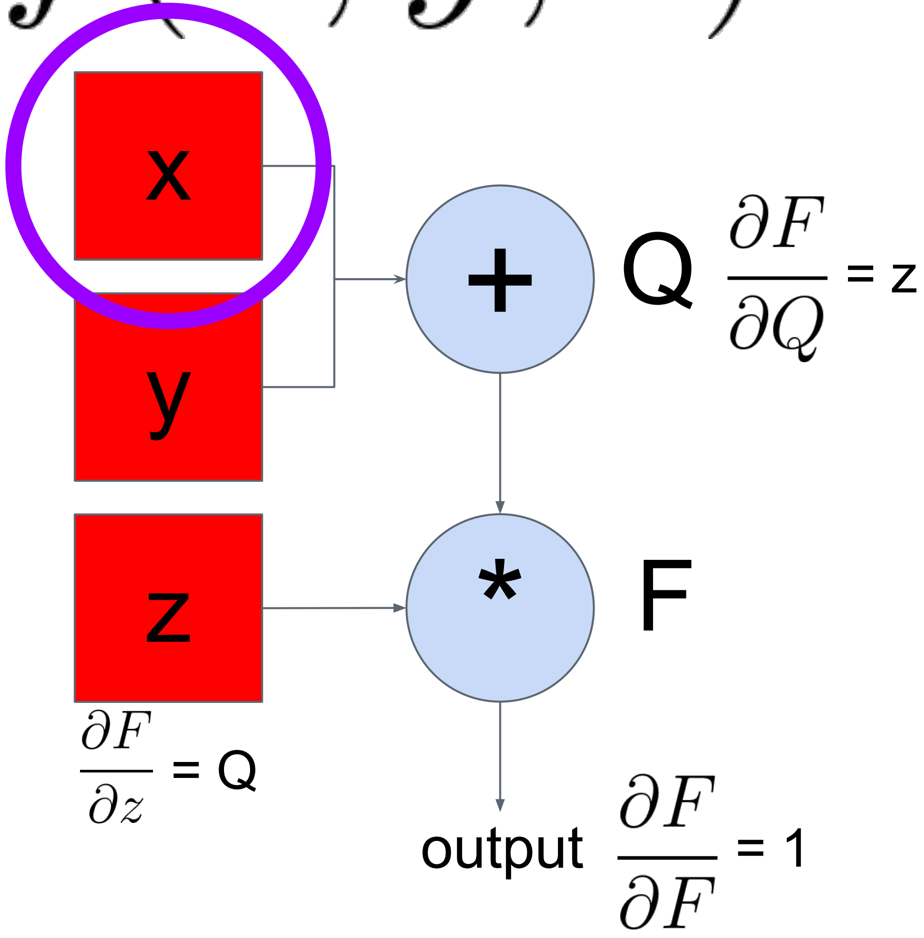
$$f(x,y,z) = (x+y)*z$$



$$F = qz$$

$$\frac{\partial f}{\partial Q} = z \quad \frac{\partial f}{\partial z} = Q$$

$$f(x, y, z) = (x + y) * z$$

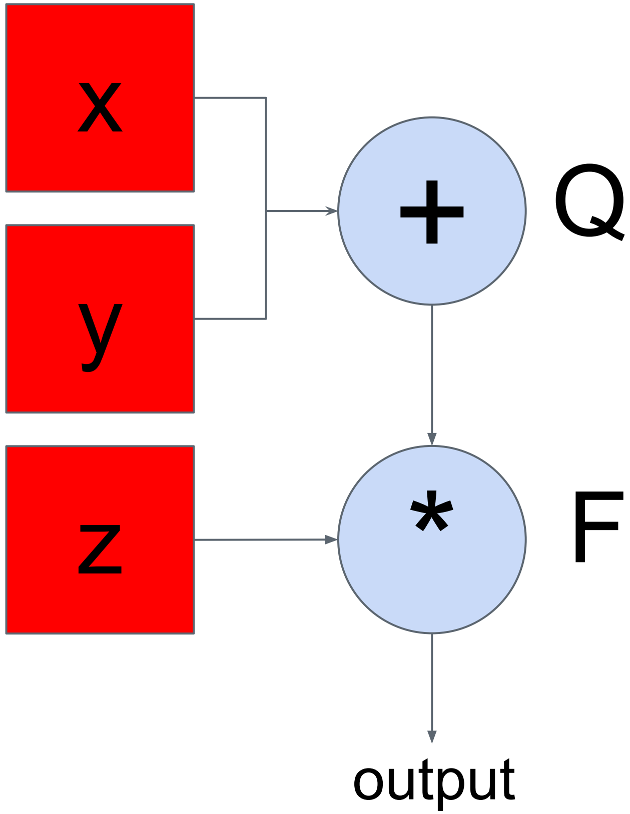

x
y
z
+
Q
$\frac{\partial F}{\partial Q}$ = z
*
F
$\frac{\partial F}{\partial z}$ = Q
output
$\frac{\partial F}{\partial F}$ = 1

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial q}\frac{\partial Q}{\partial x}$$

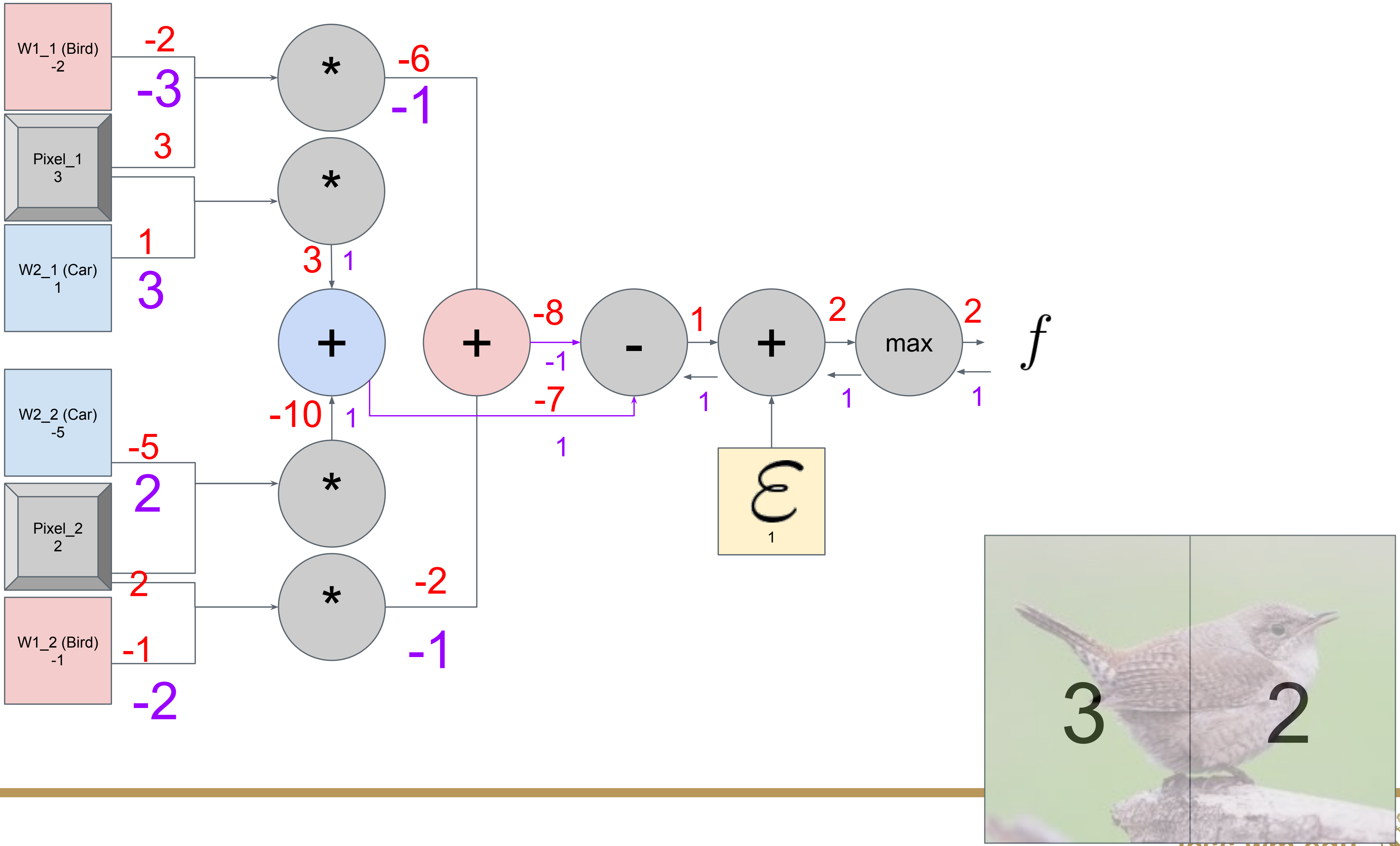
The Goal

$$\frac{\partial F}{\partial x} \quad \frac{\partial F}{\partial y} \quad \frac{\partial F}{\partial z}$$

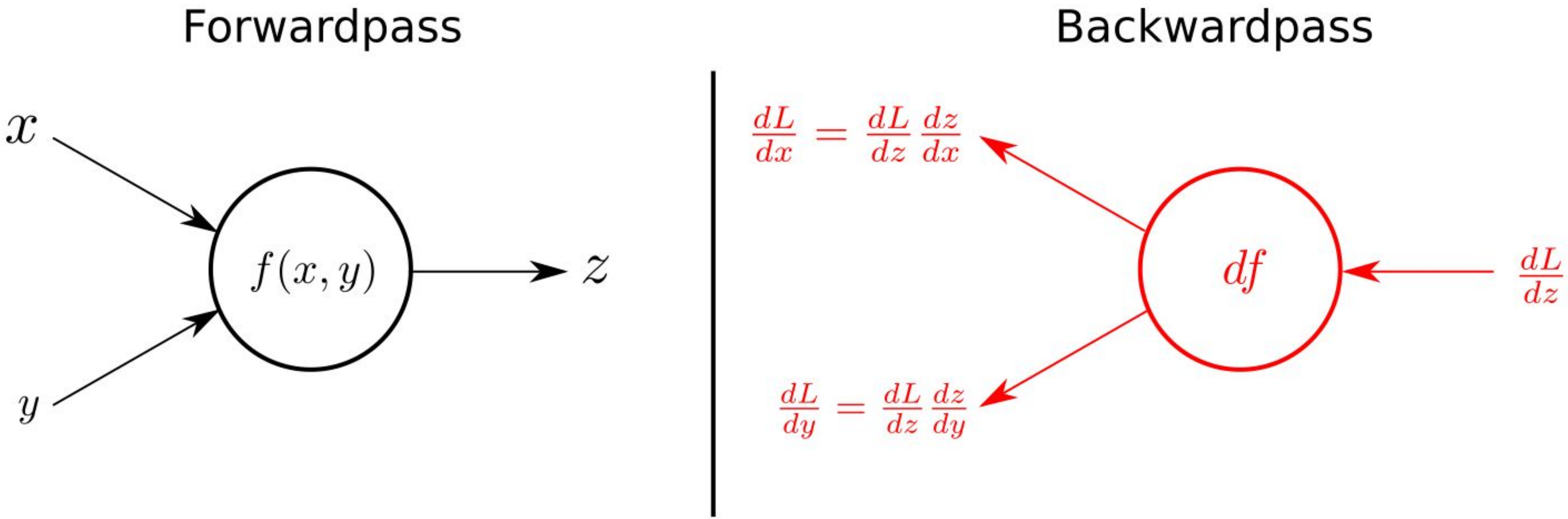
$$f(x, y, z) = (x + y) * z$$

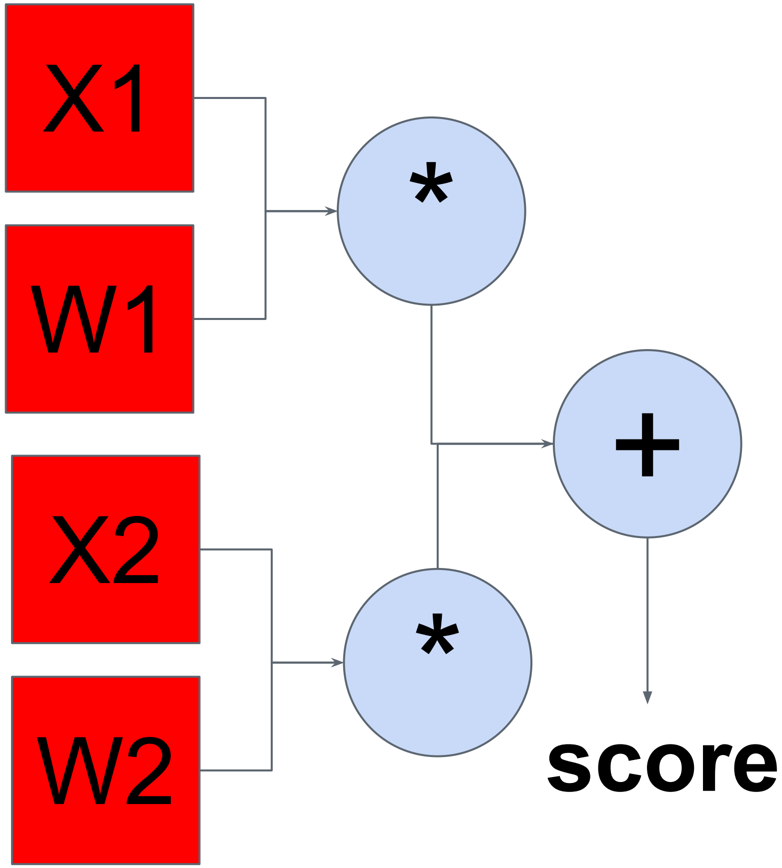


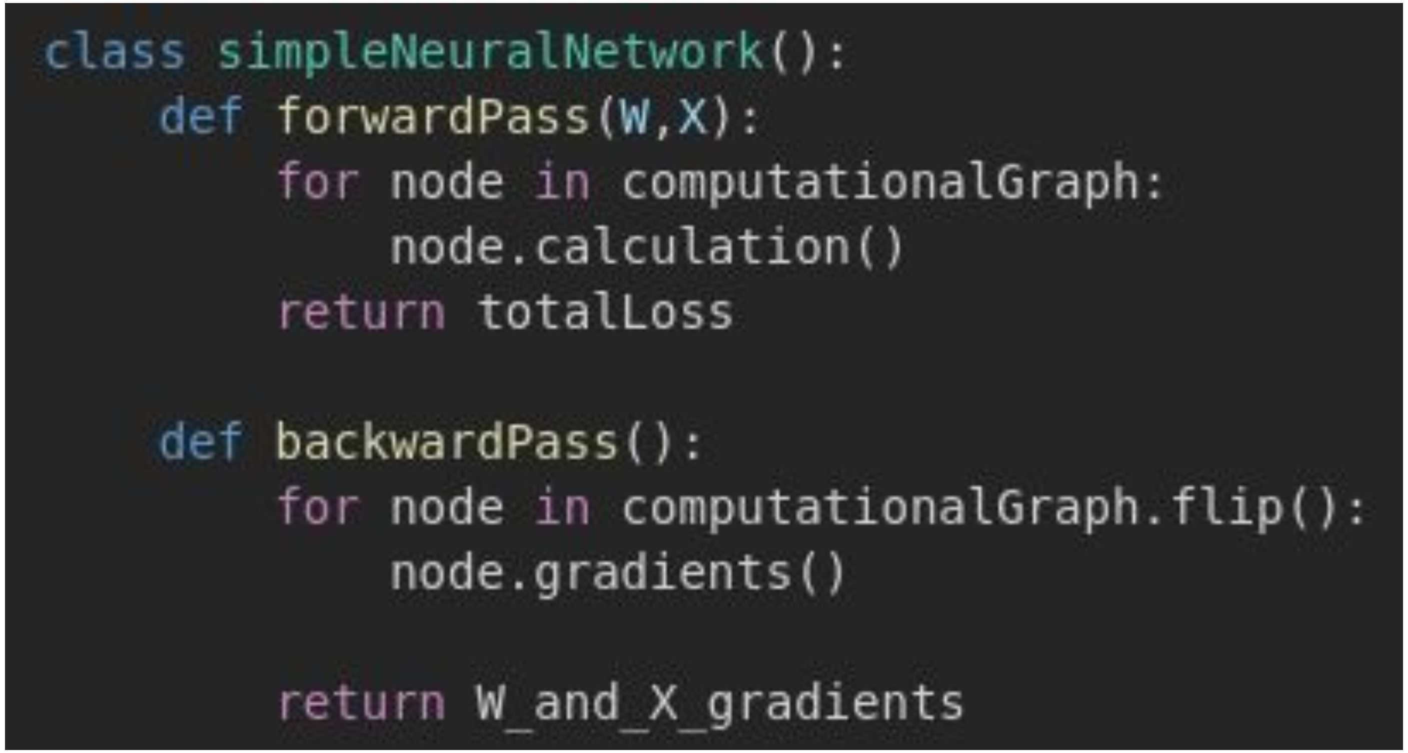
$$\frac{\partial F}{\partial x} = z * 1$$


W1_1 (Bird)
-2
-2
-3
Pixel_1
3
3
W2_1 (Car)
1
1
3
W2_2 (Car)
-5
-5
2
Pixel_2
2
2
W1_2 (Bird)
-1
-1
-2
*
-6
-1
*
3
1
+
-10
1
*
*
-2
-1
+
-8
-1
-7
1
-
1
1
+
2
1
max
2
1
f
$\mathcal{E}$
1
3
2

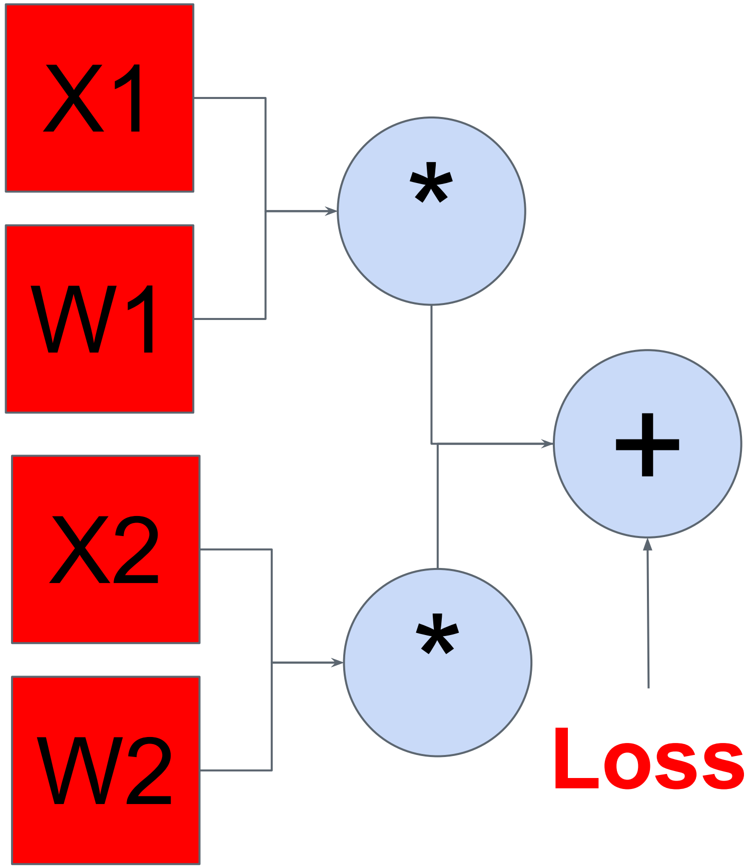
Forward vs. Backward Pass

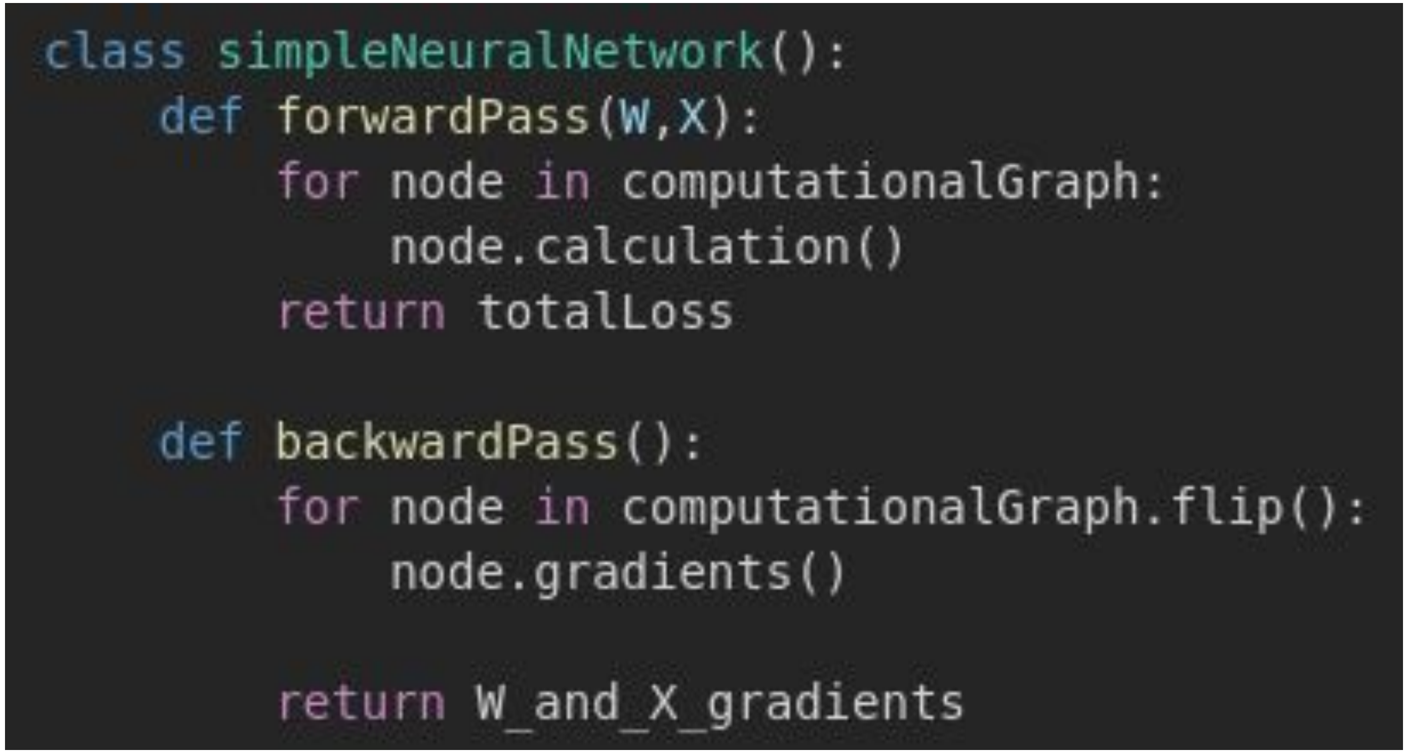

Forwardpass
x
y
$f(x,y)$
z
Backwardpass
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$
df
$\frac{dL}{dz}$



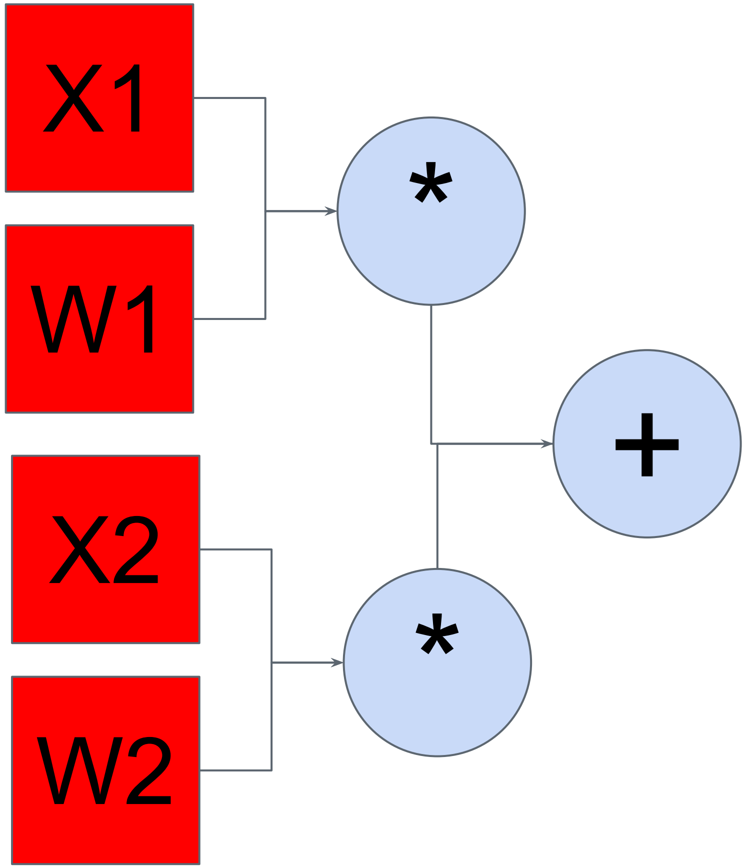


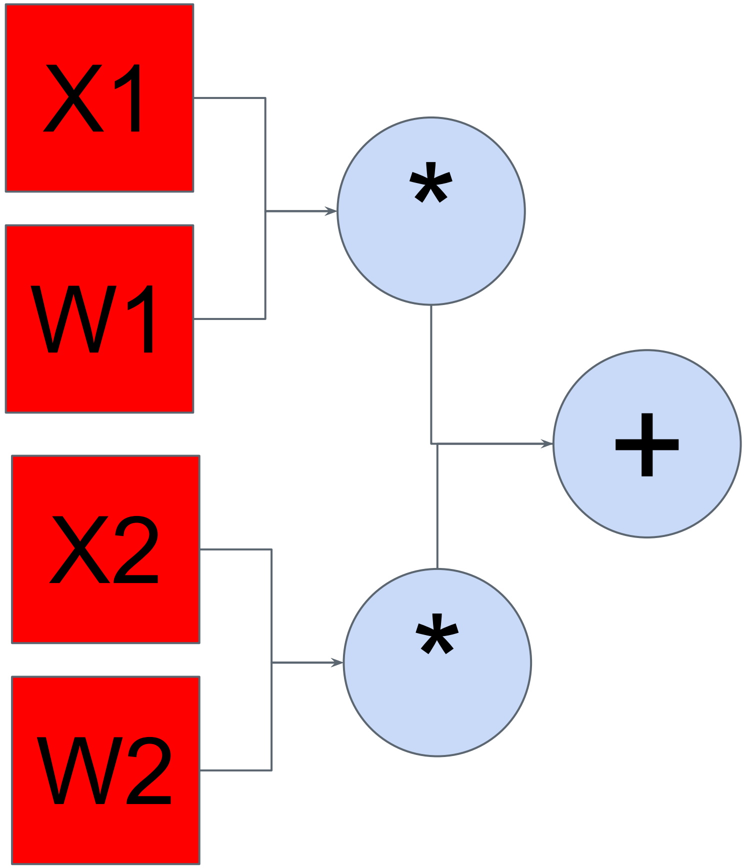
```
class simpleNeuralNetwork():
    def forwardPass(W,X):
        for node in computationalGraph:
            node.calculation()
        return totalLoss

    def backwardPass():
        for node in computationalGraph.flip():
            node.gradients()

        return W_and_X_gradients
```

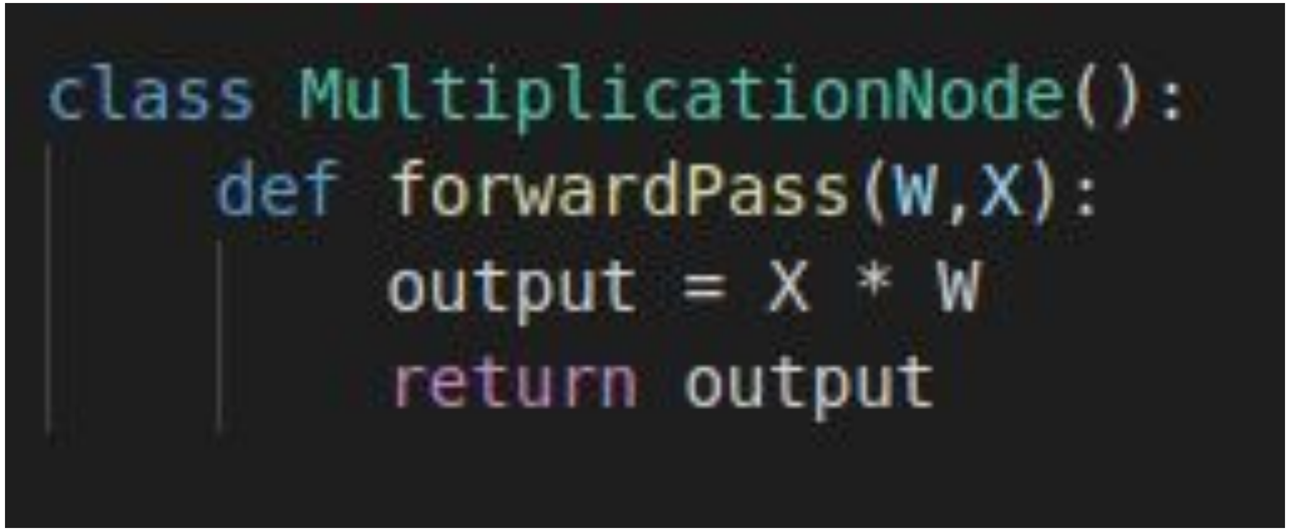




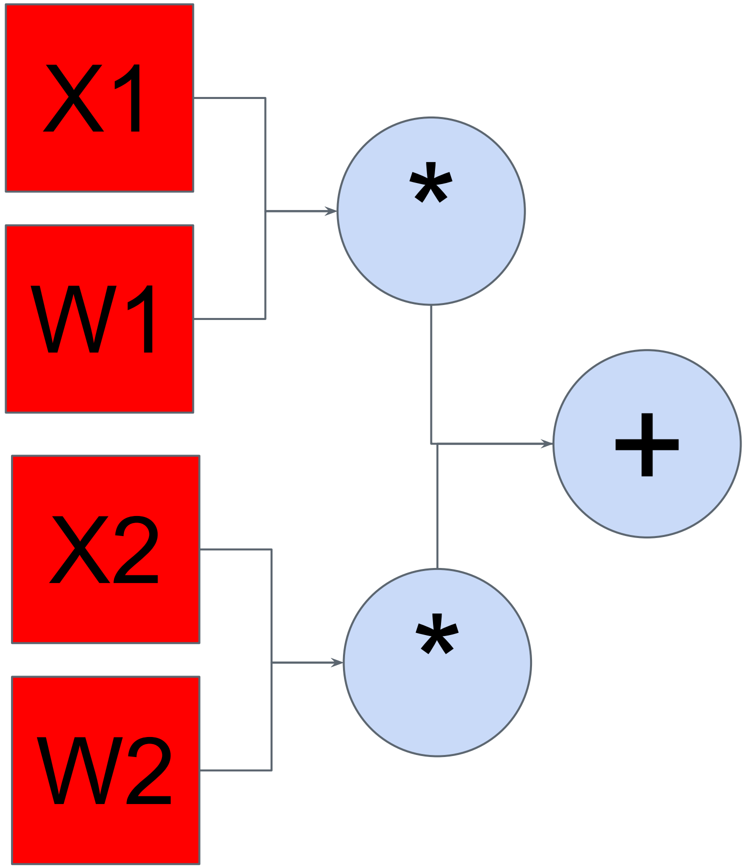
```
class simpleNeuralNetwork():
    def forwardPass(W,X):
        for node in computationalGraph:
            node.calculation()
        return totalLoss

    def backwardPass():
        for node in computationalGraph.flip():
            node.gradients()

        return W_and_X_gradients
```

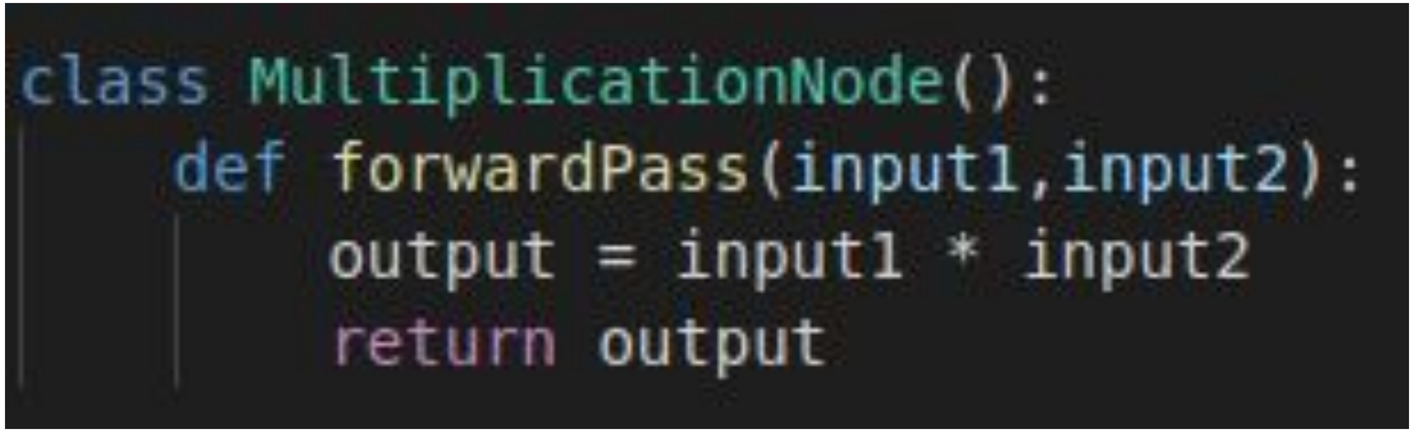

X1
W1
X2
W2
*
*
+



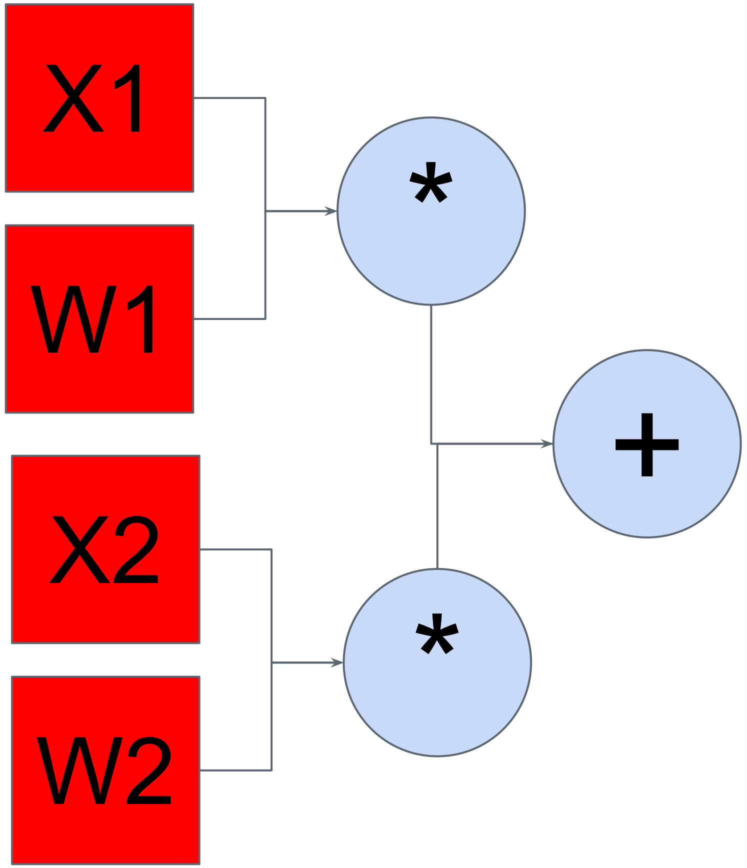


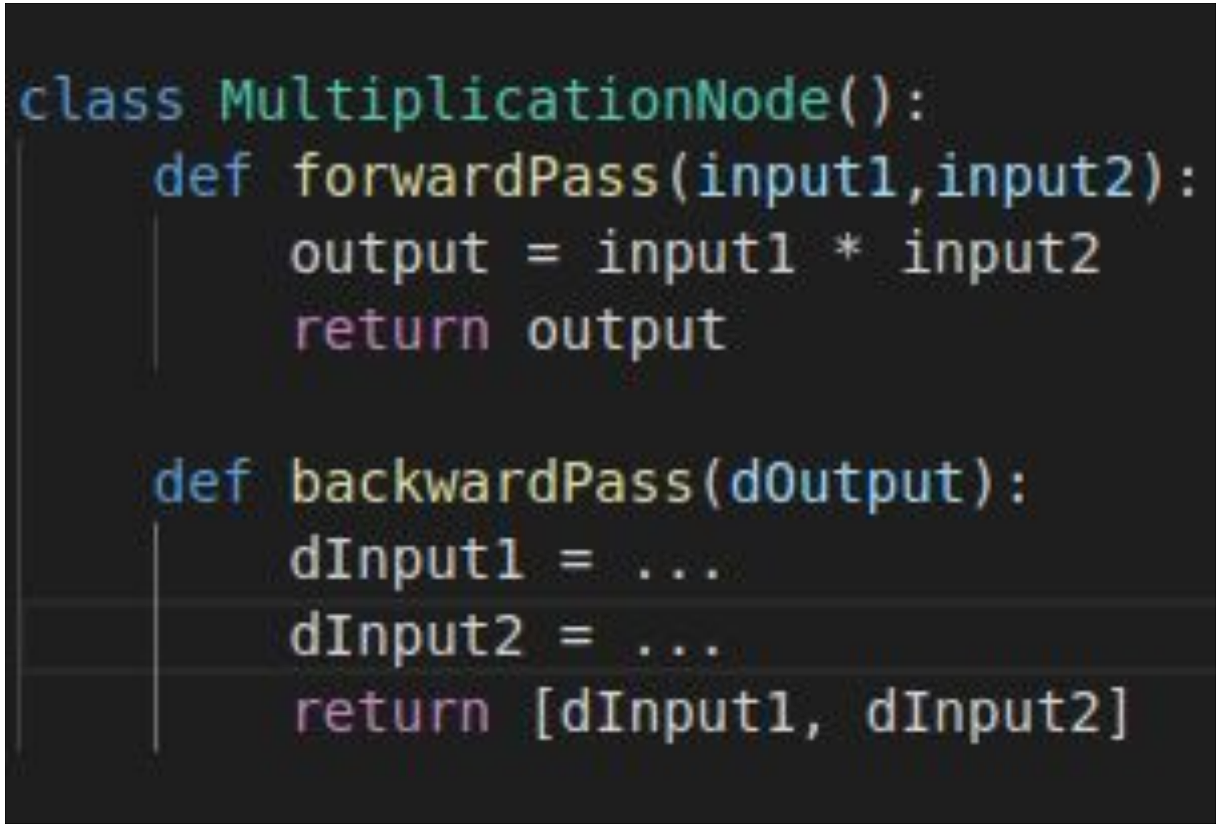
```
class MultiplicationNode():
    def forwardPass(W,X):
        output = X * W
        return output
```





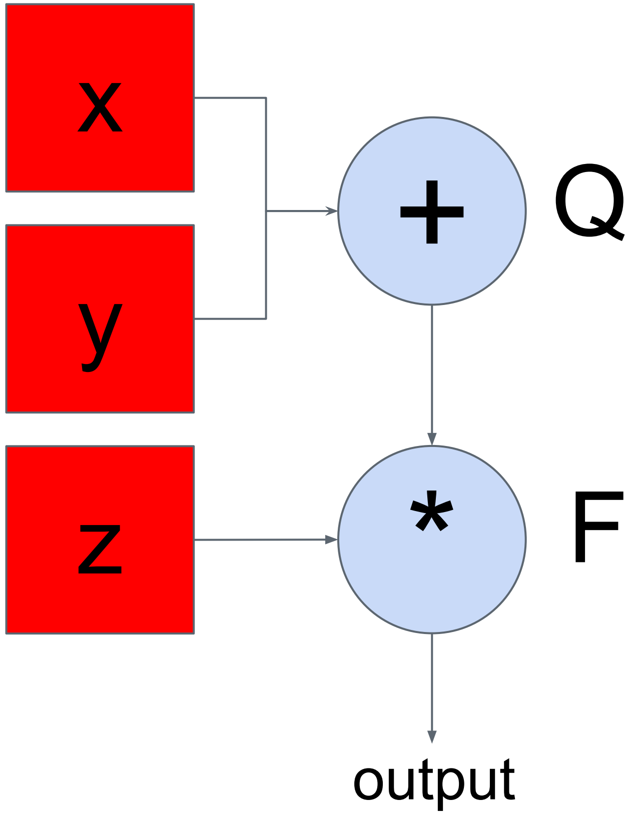
```
class MultiplicationNode():
    def forwardPass(input1,input2):
        output = input1 * input2
        return output
```





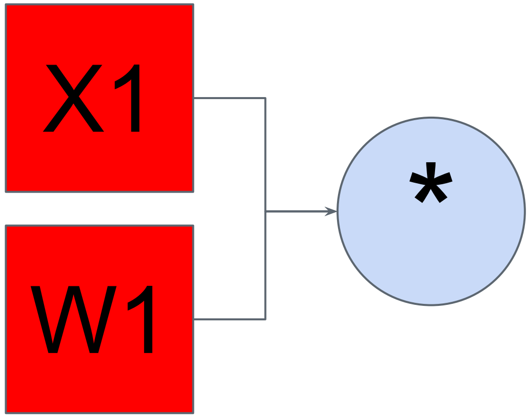
```
class MultiplicationNode():
    def forwardPass(input1,input2):
        output = input1 * input2
        return output

    def backwardPass(dOutput):
        dInput1 = ...
        dInput2 = ...
        return [dInput1, dInput2]
```

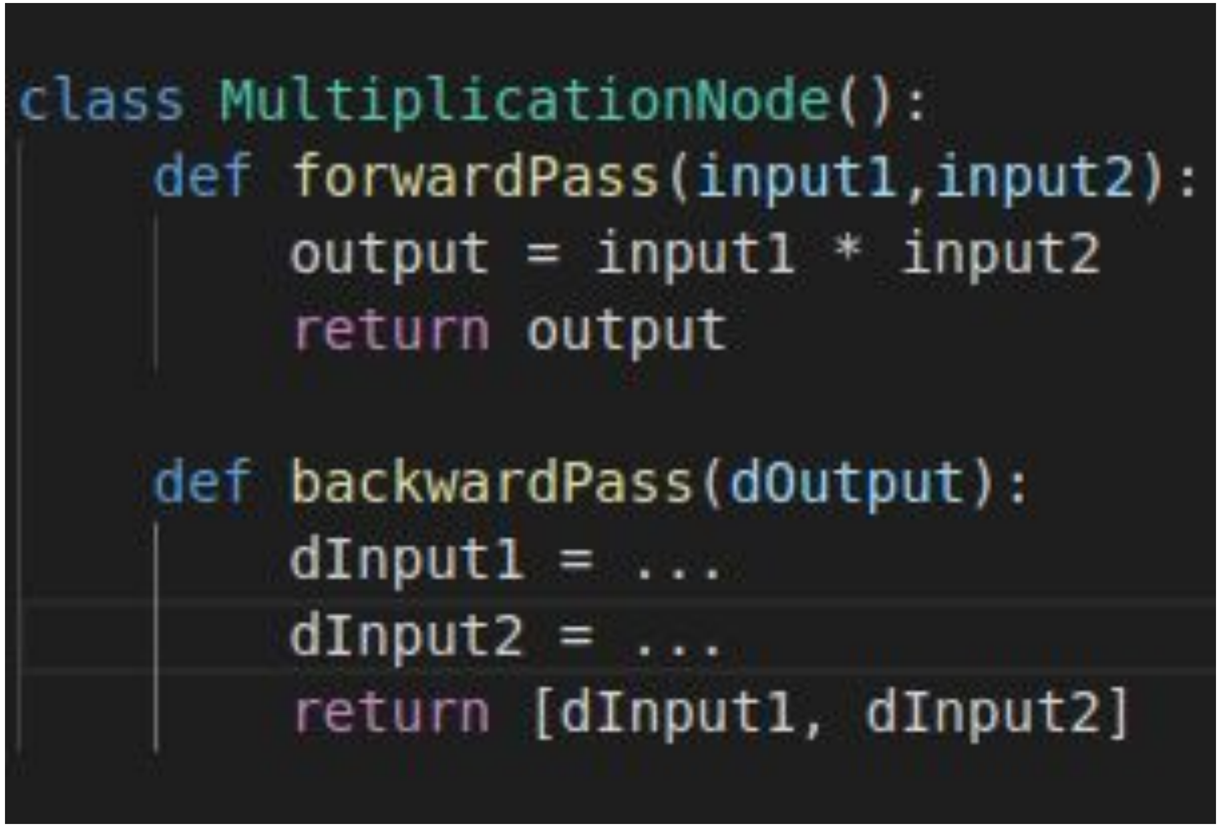
$$f(x, y, z) = (x + y) * z$$



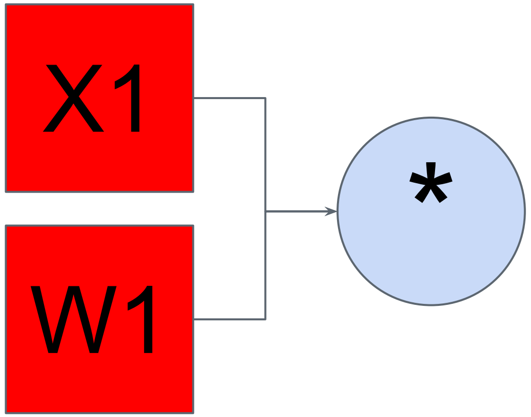
$$F = qz$$

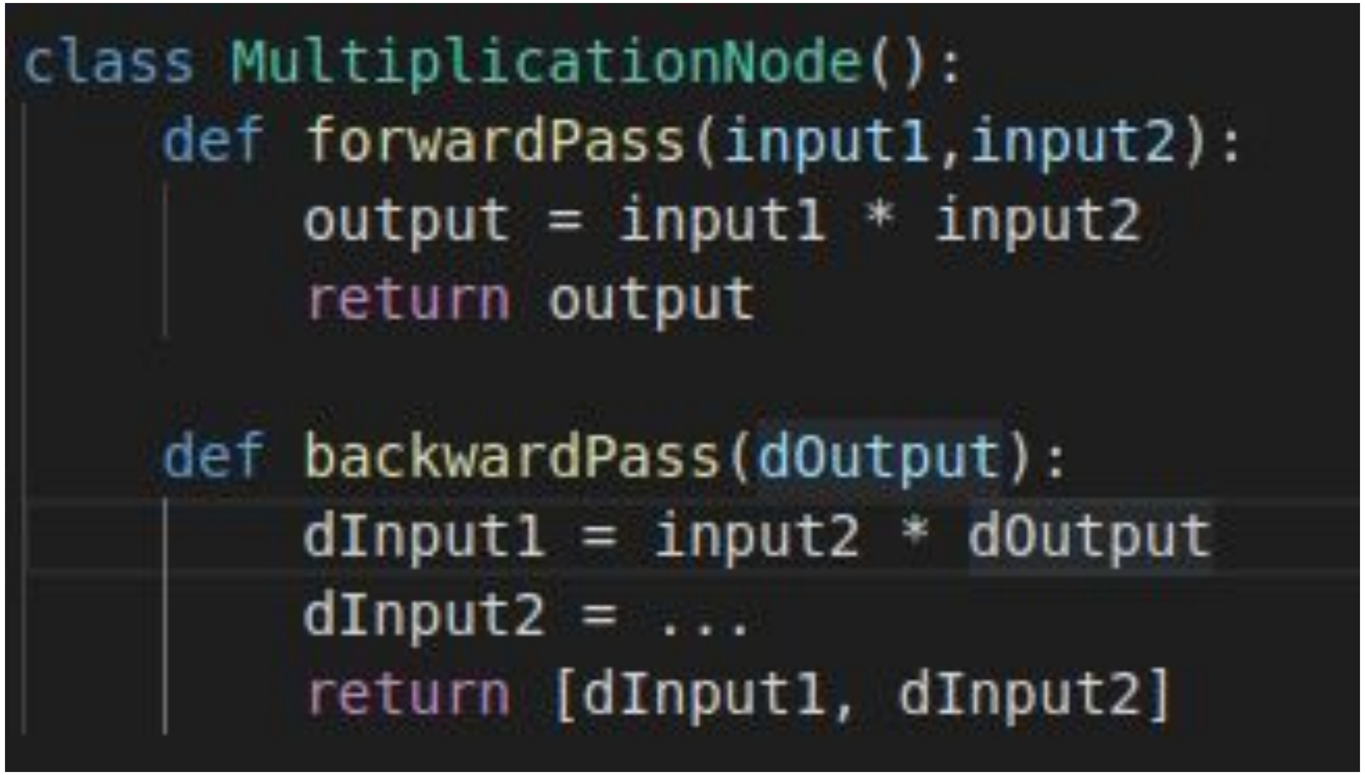
$$\frac{\partial f}{\partial Q} = z \quad \frac{\partial f}{\partial z} = Q$$



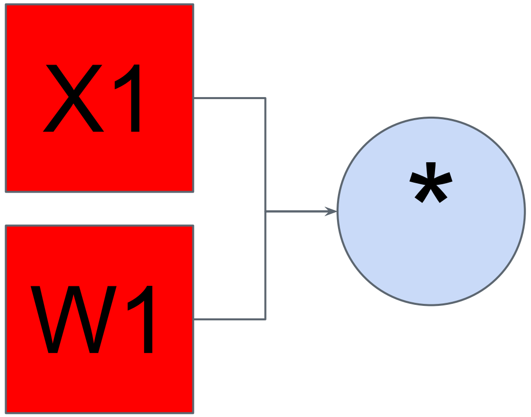


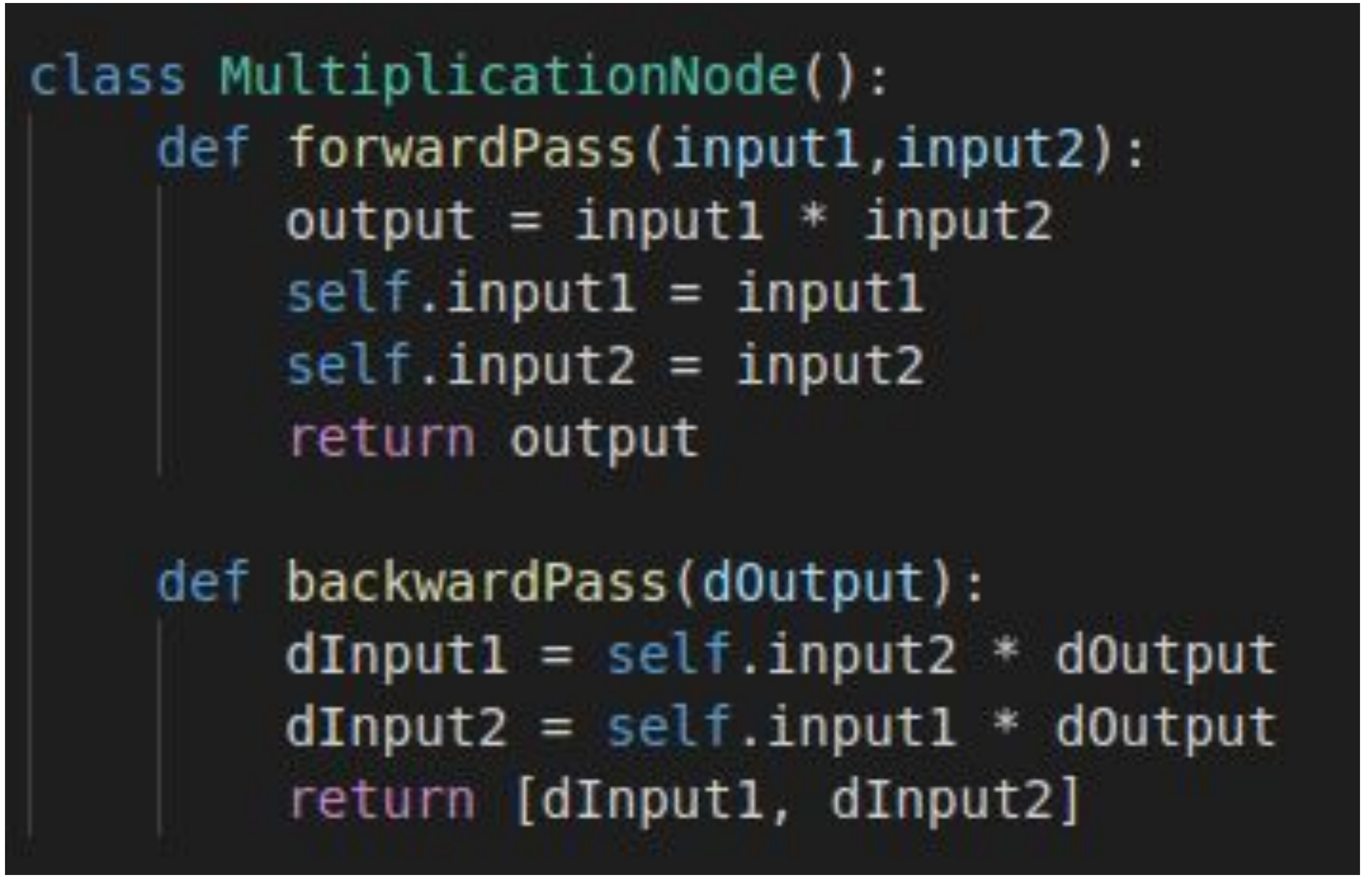
```
class MultiplicationNode():
    def forwardPass(input1,input2):
        output = input1 * input2
        return output

    def backwardPass(dOutput):
        dInput1 = ...
        dInput2 = ...
        return [dInput1, dInput2]
```





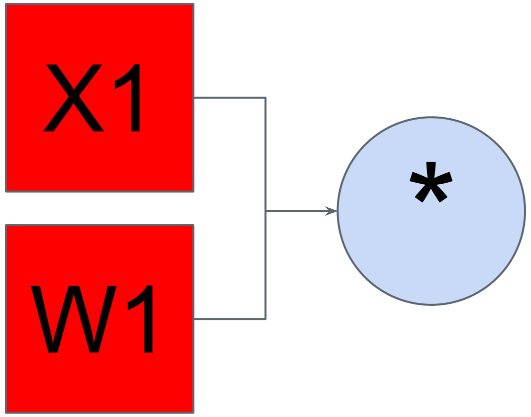
```
class MultiplicationNode():
    def forwardPass(input1,input2):
        output = input1 * input2
        return output

    def backwardPass(dOutput):
        dInput1 = input2 * dOutput
        dInput2 = ...
        return [dInput1, dInput2]
```





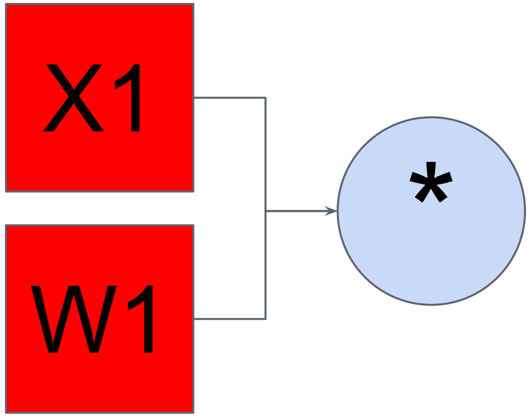
```
class MultiplicationNode():
    def forwardPass(input1,input2):
        output = input1 * input2
        self.input1 = input1
        self.input2 = input2
        return output

    def backwardPass(dOutput):
        dInput1 = self.input2 * dOutput
        dInput2 = self.input1 * dOutput
        return [dInput1, dInput2]
```

From Computational Graphs to Neural Nets



$$f(X, W) = X * W$$

From Computational Graphs to Neural Nets

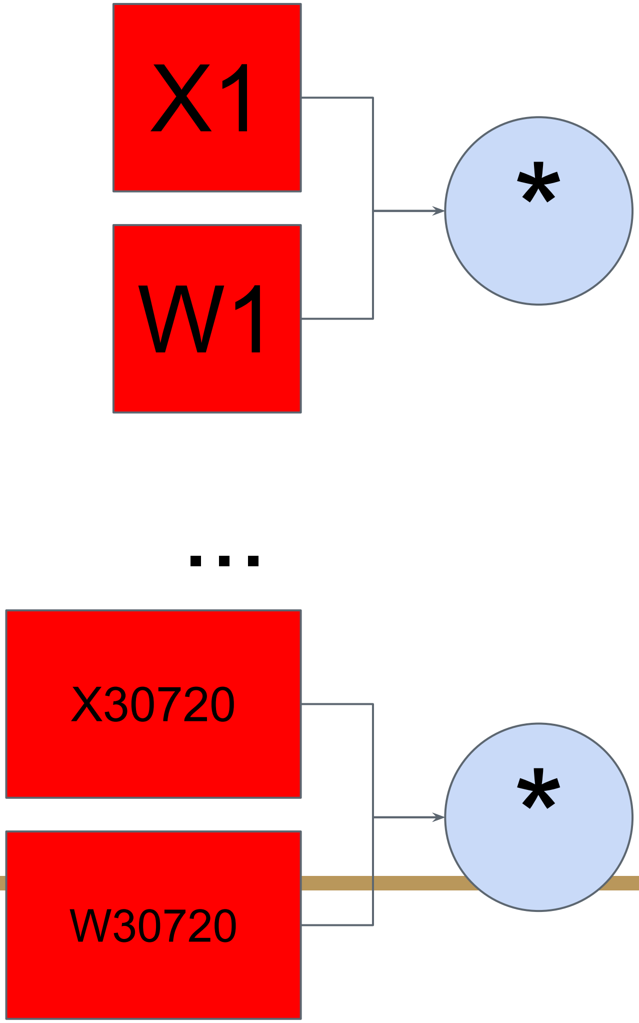


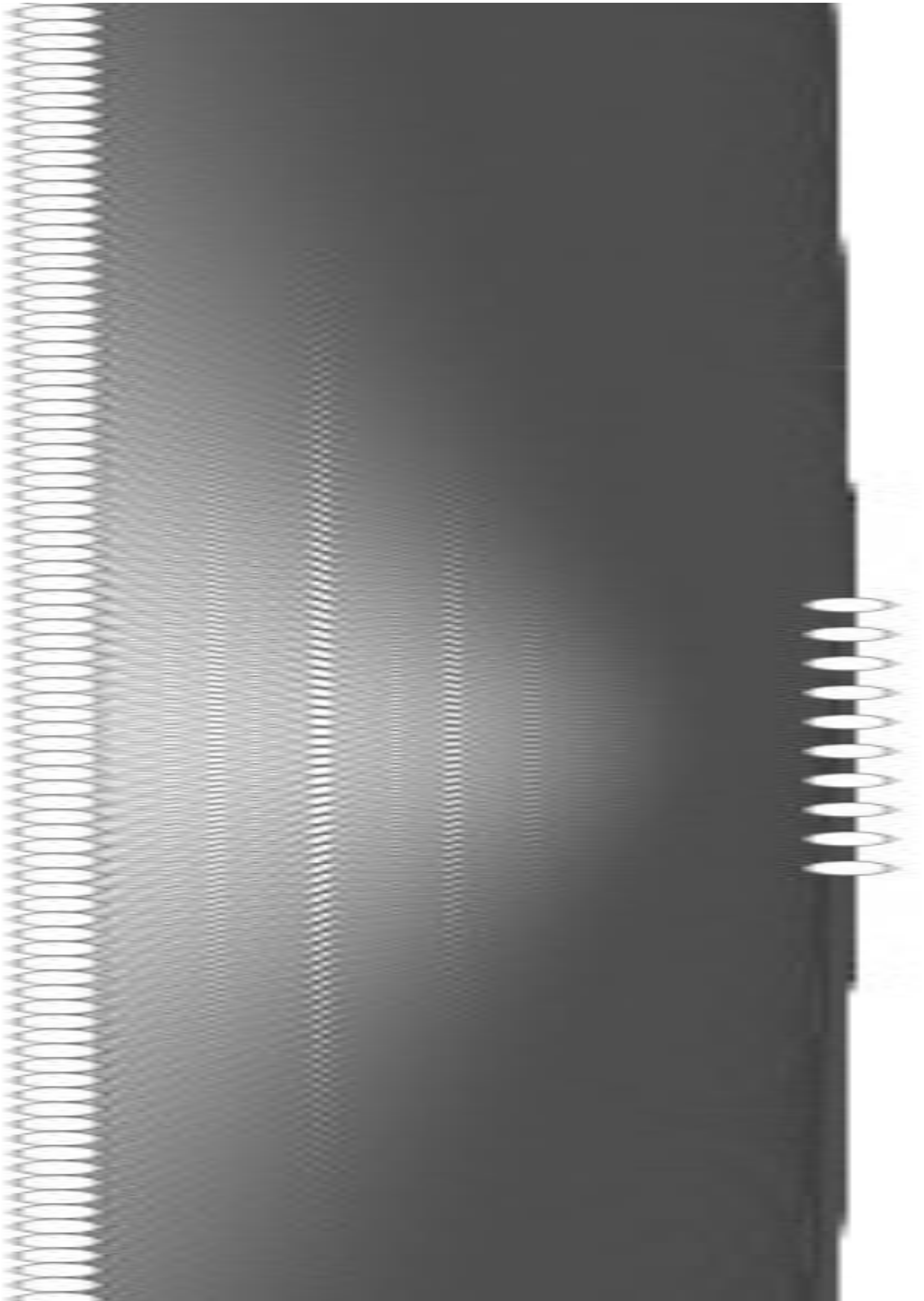
$$f(X, W) = X * W$$

3,072 Pixels in CIFAR-10 (32x32x3 colors)

3,072 Weights for each of 10 classes.

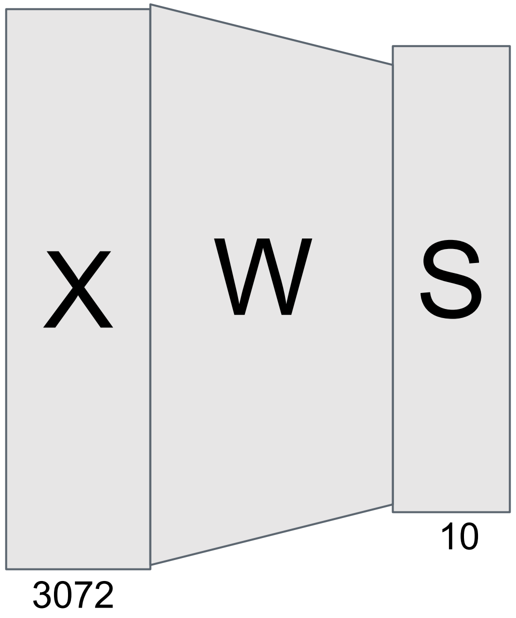
From Computational Graphs to Neural Nets





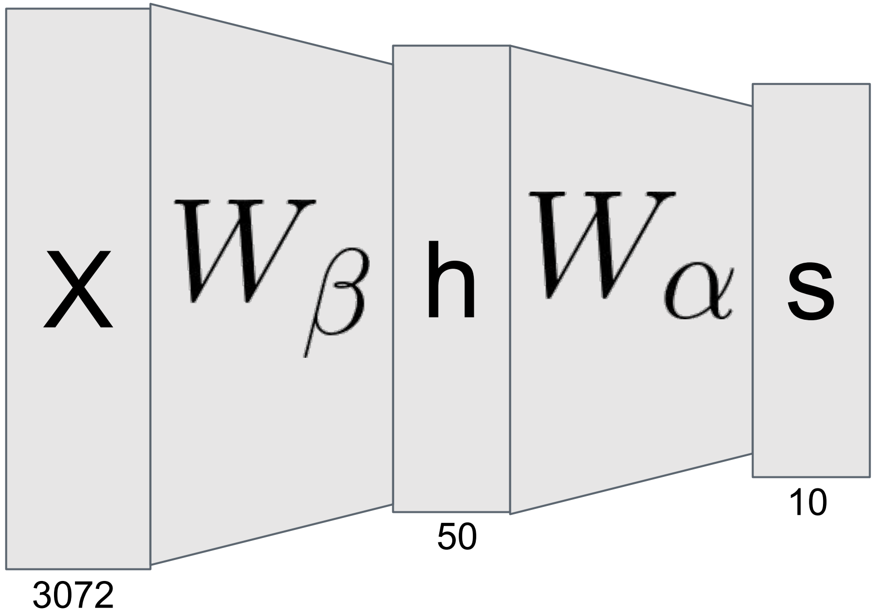
From Computational Graphs to Neural Nets

$$f(X, W) = X * W$$

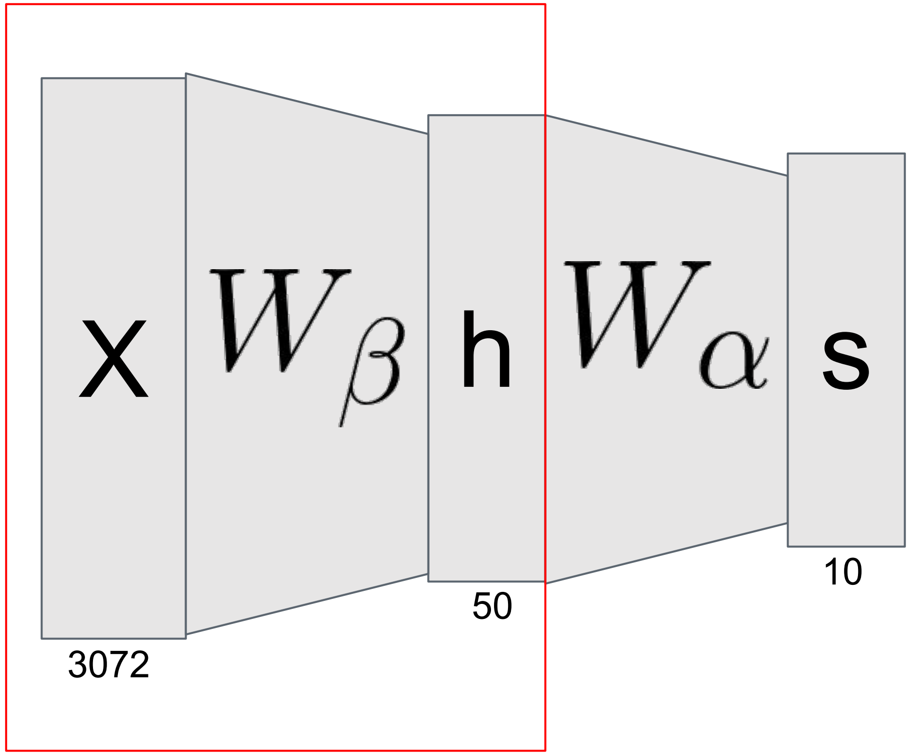


From Computational Graphs to Neural Nets

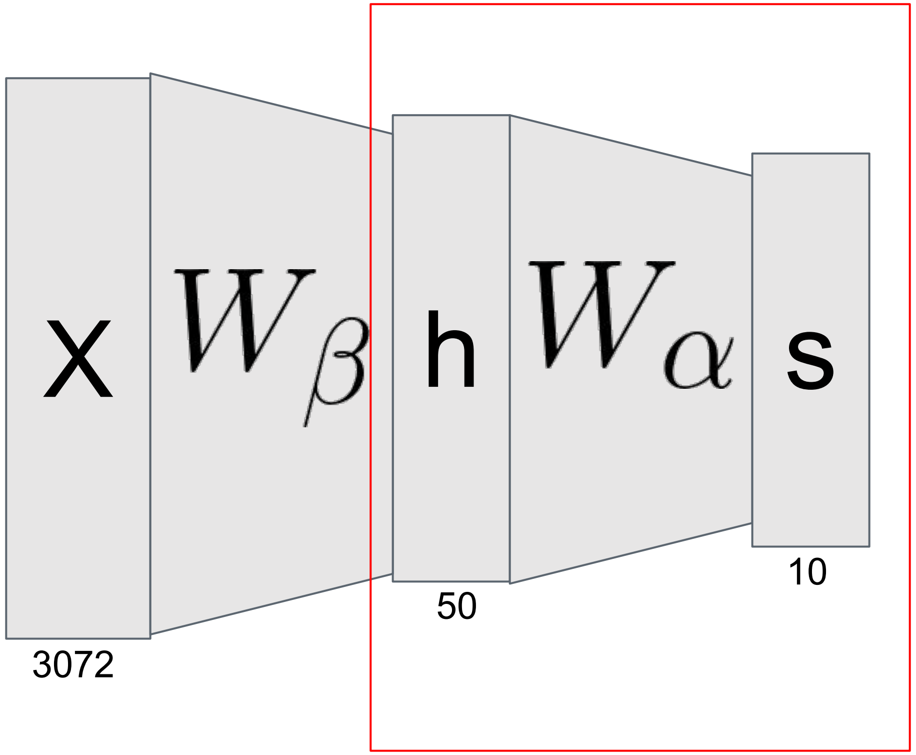
$$f = W_\alpha * max(0, W_\beta * X)$$



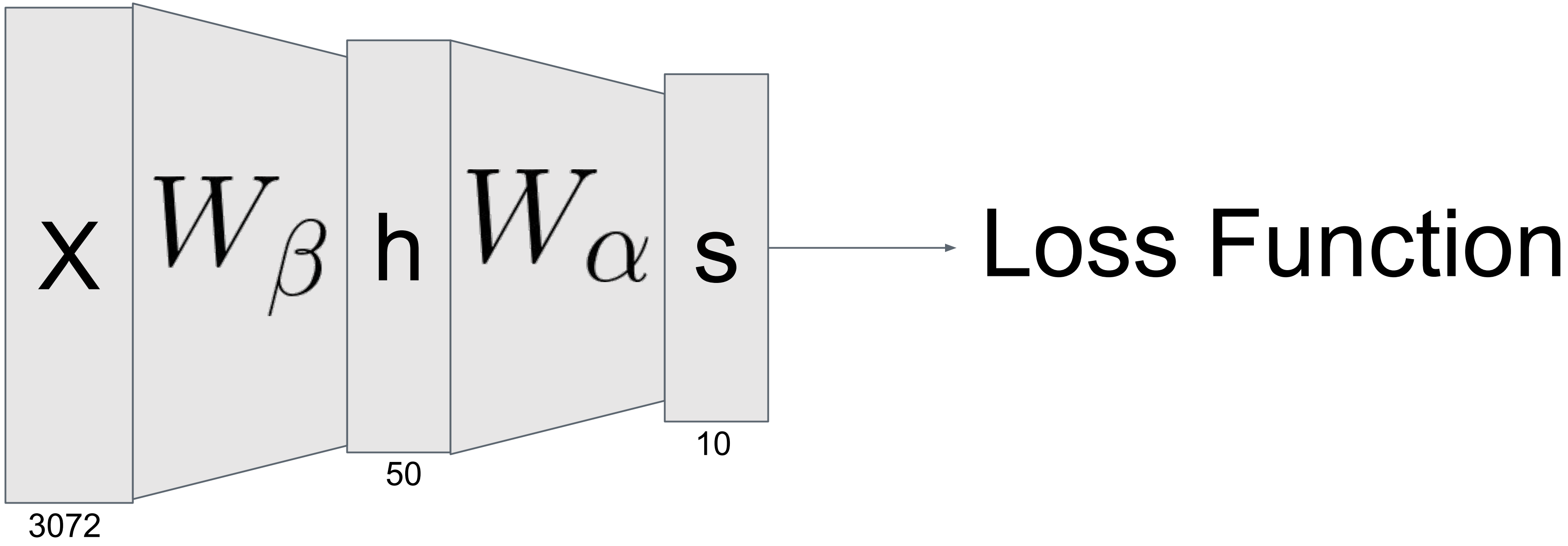
$$f = W_\alpha * max(0, W_\beta * X)$$



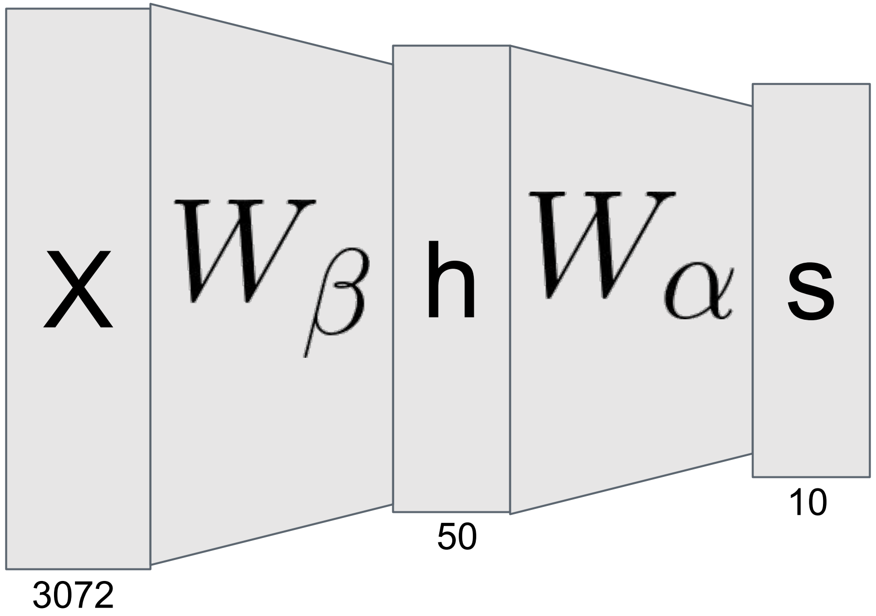
$$f = W_\alpha * max(0, W_\beta * X)$$



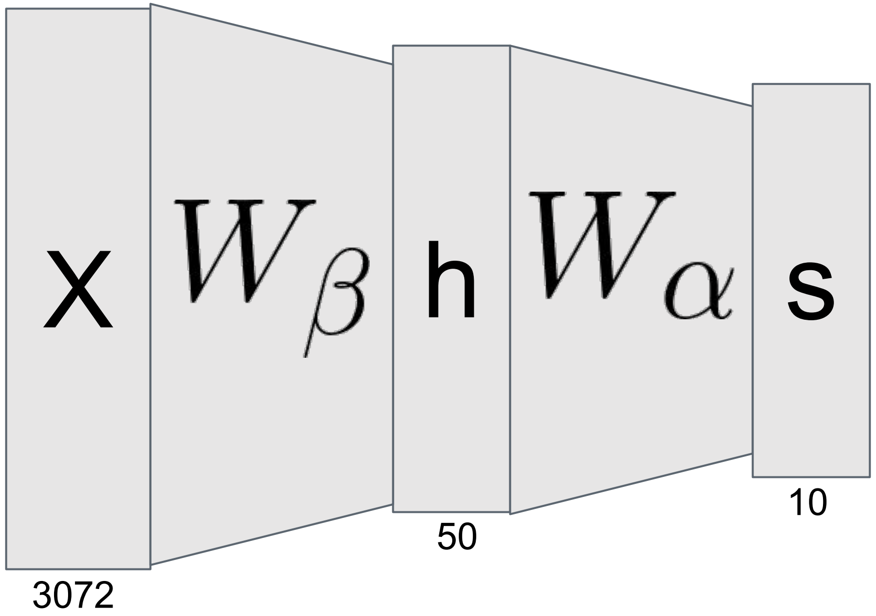
$$f = W_\alpha * max(0, W_\beta * X)$$



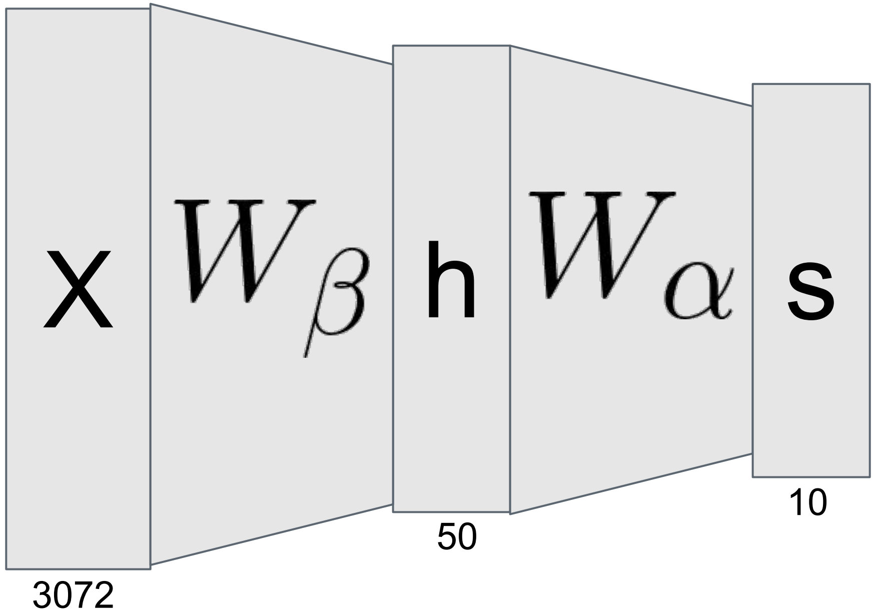
$$f = W_\alpha * max(0, W_\beta * X)$$



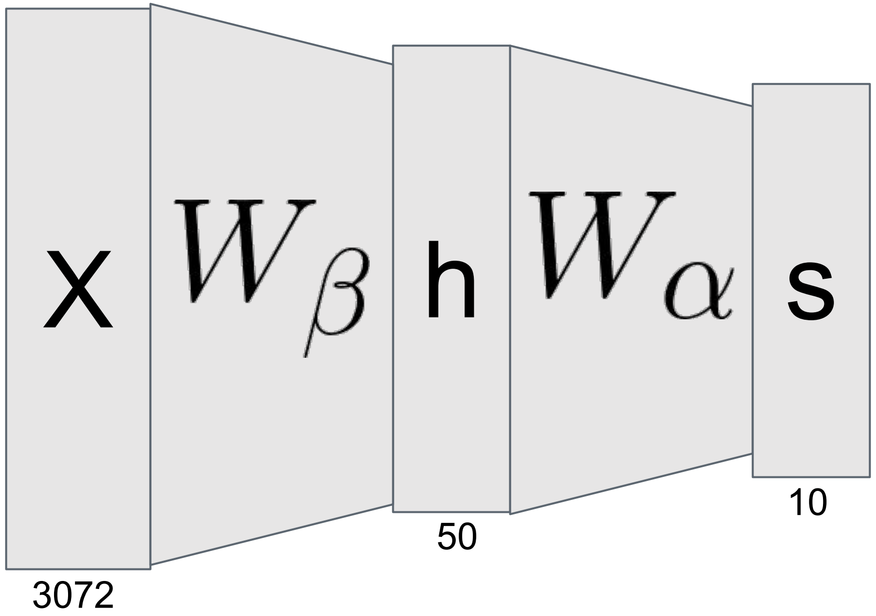
$$f = W_\alpha * max(0, W_\beta * X)$$



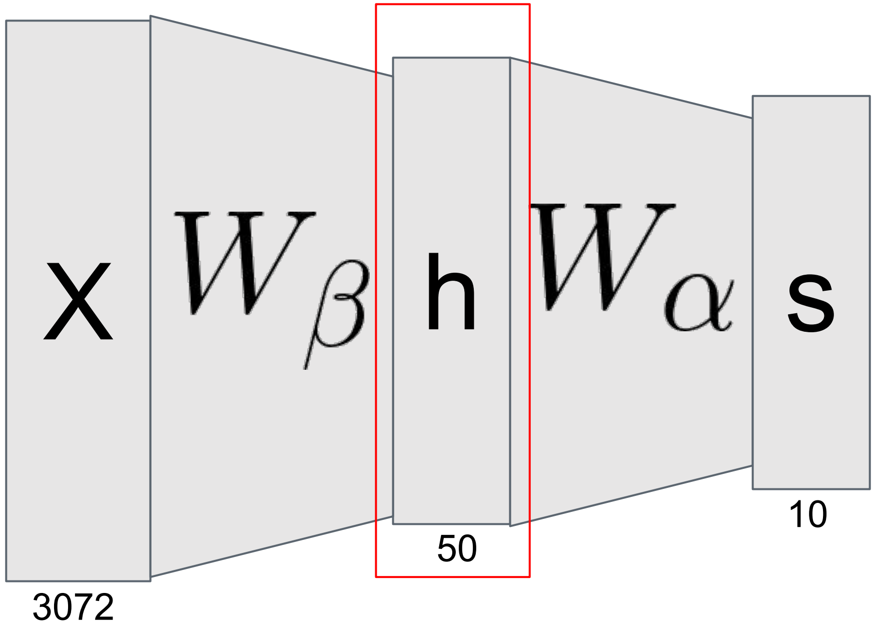
$$f = W_\alpha * max(0, W_\beta * X)$$



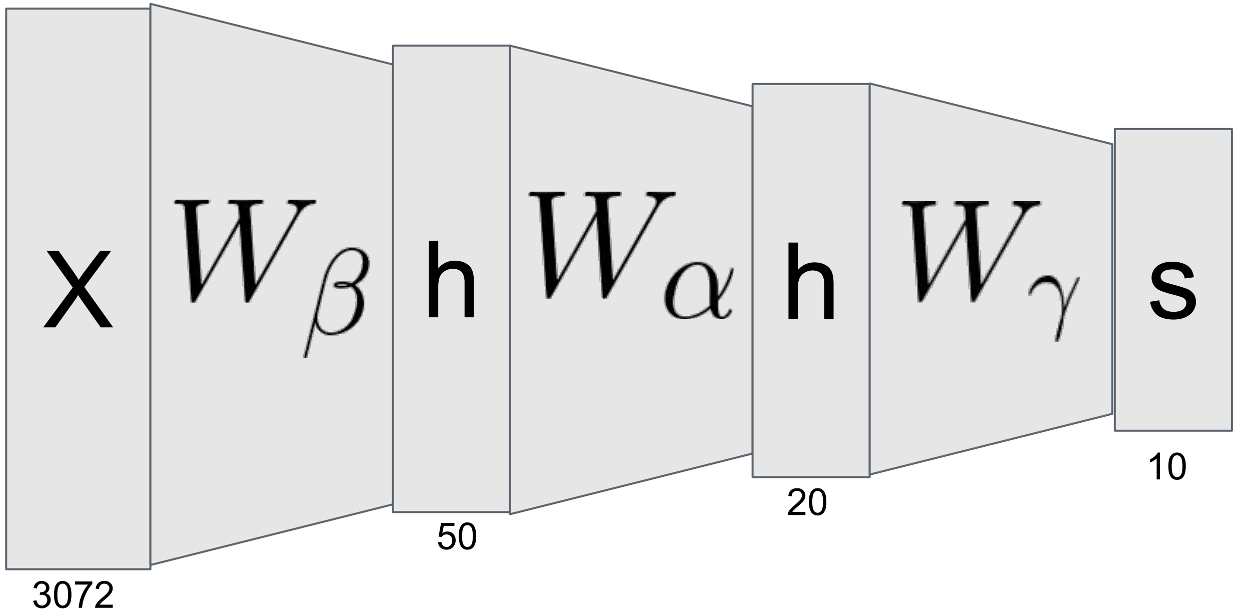
$$f = W_\alpha * max(0, W_\beta * X)$$

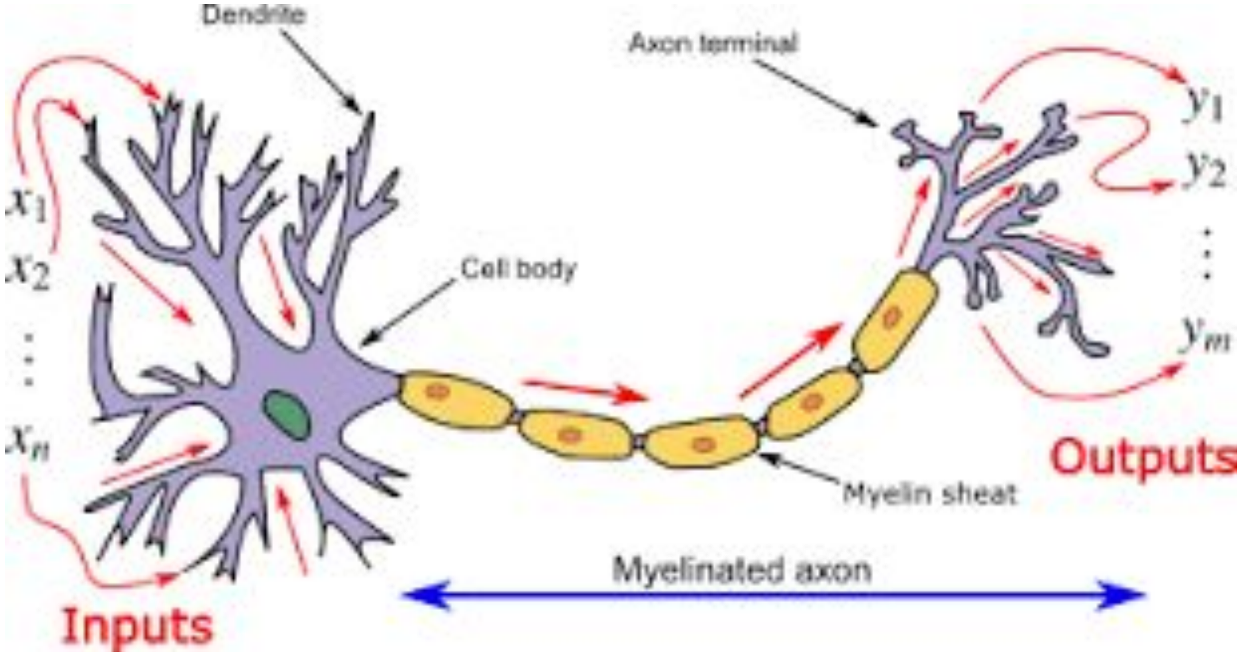


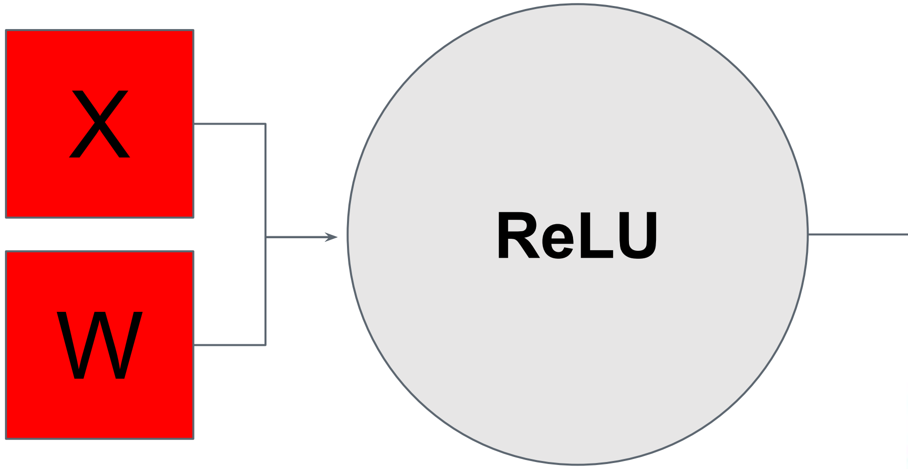
$$f = W_\alpha * max(0, W_\beta * X)$$

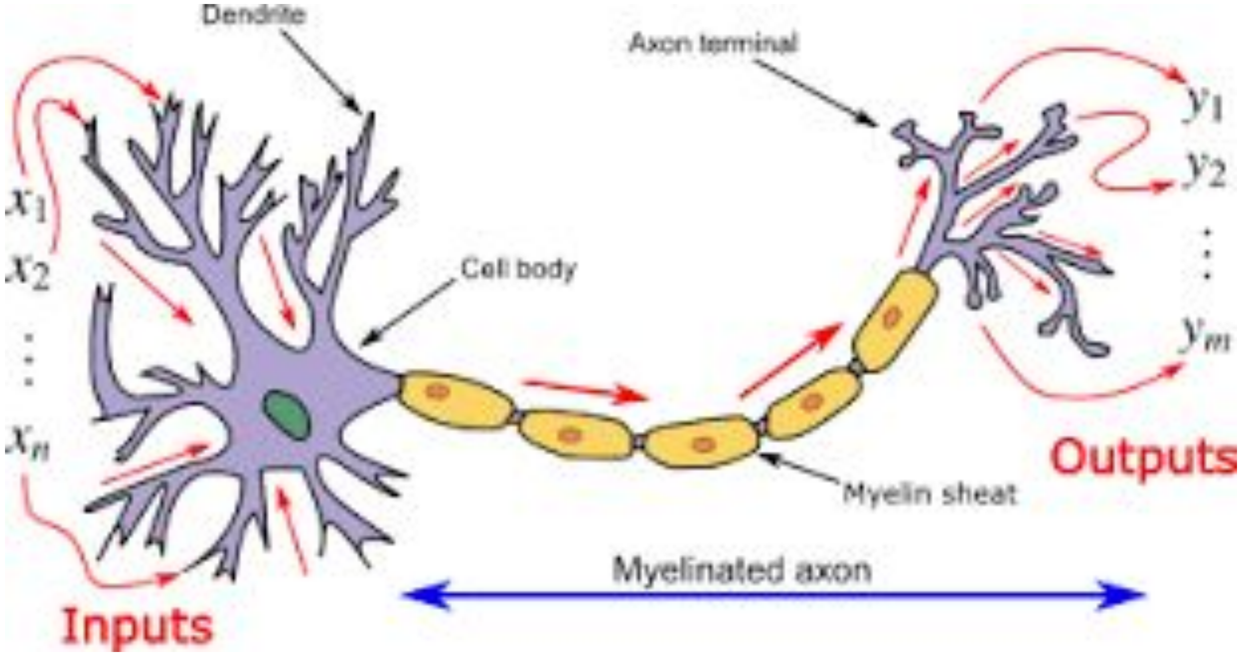


$$f = W_\gamma * max(0, W_\alpha * max(0, W_\beta * X))$$




Dendrite
Axon terminal
x_1
x_2
x_n
Cell body
y_1
y_2
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs


X
W
ReLU


Dendrite
Axon terminal
Cell body
x_1
x_2
⋮
x_n
y_1
y_2
⋮
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs

Summary

Forward and Backward propagation in code

Strategies for abstracting and communicating network architecture (layers)

How computational graphs relate to neural networks

What “deep learning” actually means

How “deeper learning” can help when you have horses that face in different directions.

A bit on the biological inspiration of neural networks.