Optional Lab: Back propagation using a computation graph

Working through this lab will give you insight into a key algorithm used by most machine learning frameworks. Gradient descent requires the derivative of the cost with respect to each parameter in the network. Neural networks can have millions or even billions of parameters. The *back propagation* algorithm is used to compute those derivatives. *Computation graphs* are used to simplify the operation. Let's dig into this below.

In [ ]:
```
from sympy import *
import numpy as np
import re
%matplotlib widget
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox
from matplotlib.widgets import Button
import ipywidgets as widgets
from lab_utils_backprop import *
```

Computation Graph

A computation graph simplifies the computation of complex derivatives by breaking them into smaller steps. Let's see how this works.

Let's calculate the derivative of this slightly complex expression, $J = (2+3w)^2$. We would like to find the derivative of J with respect to w or $\frac{\partial J}{\partial w}$.

In [ ]:
```
plt.close("all")
plt_network(config_nw0, "./images/C2_W2_BP_network0.PNG")
```

Above, you can see we broke the expression into two nodes which we can work on independently. If you already have a good understanding of the process from the lecture, you can go ahead and fill in the boxes in the diagram above. You will want to first fill in the blue boxes going left to right and then fill in the green boxes starting on the right and moving to the left. If you have the correct values, the values will show as green or blue. If the value is incorrect, it will be red. Note, the interactive graphic is not particularly robust. If you run into trouble with the interface, run the cell above again to restart.

If you are unsure of the process, we will work this example step by step below.

Forward Propagation

Let's calculate the values in the forward direction.

> Just a note about this section. It uses global variables and reuses them as the calculation progresses. If you run cells out of order, you may get funny results. If you do, go back to this point and run them in order.

In [ ]:
```
w = 3
a = 2+3*w
J = a**2
print(f"a = {a}, J = {J}")
```

You can fill these values in the blue boxes above.

Backprop

$J = a^2$

Backprop is the algorithm we use to calculate derivatives. As described in the lectures, backprop starts at the right and moves to the left. The first node to consider is $J = a^2$ and the first step is to find $\frac{\partial J}{\partial a}$

$$\frac{\partial J}{\partial a}$$

Arithmetically

Find $\frac{\partial J}{\partial a}$ by finding how J changes as a result of a little change in a. This is described in detail in the derivatives optional lab.

In [ ]:
```
a_epsilon = a + 0.001       # a epsilon
J_epsilon = a_epsilon**2    # J_epsilon
k = (J_epsilon - J)/0.001   # difference divided by epsilon
print(f"J = {J}, J_epsilon = {J_epsilon}, dJ_da ~= k = {k} ")
```

$\frac{\partial J}{\partial a}$ is 22 which is $2 \times a$. Our result is not exactly $2 \times a$ because our epsilon value is not infinitesimally small.