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 [1]: from numpy import *
        from numpy.linalg as np
        import math
        ImpSplotlib widget
        %matplotlib inline, inset as plt
        from matplotlib.widgets import TextBox
        from matplotlib.widgets import Button
        import ipywidgets as widgets
        from lab_utils_backprop import *
```

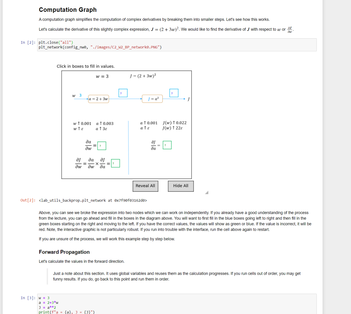
Computation Graph

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 [2]: plt.close('all')
        plt_network(config_nwk, "./images/C2_W2_BP_network0.PNG")
```

Click in boxes to fill in values.




```
Out[2]: <lab_utils_backprop.plt_network at 0x7f9df01a180>
```

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 fill in the first 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 in color. 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 [3]: w = 3
        b = 5.0e
        a = w*b
        print("a = {a}, J = {J}")
        a = 15, J = 121
```

You can fit these values in the blue boxes above.

Backprop

$\boxed{\text{J = a}^2}$ 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 [4]: e_epsilon = a + 0.001      # a epsilon
        J_epsilon = a_epsilon**2   # J epsilon
        k = 1_epsilon - J(a,b)     # difference divided by epsilon
        print("J = {J}, J_epsilon = {J_epsilon}, dJ_da ~= k = {k} ")
        J = 121, J_epsilon = 121.060000000000,  dJ_da ~= k = 60.9999999999815
```

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

Symbolically

Now, let's use SymPy to calculate derivatives symbolically as we did in the derivatives optional lab. We will prefix the name of the variable with an 's' to indicate this is a symbolic variable.

```
In [5]: sw,sa,sb = symbols('w,a,b')
        s1 = sa**2
        s1
Out[5]: 2
        a
In [6]: s1.subs({sa,s1})
Out[6]: 121
In [7]: dJ_da = diff(s1, sa)
        dJ_da
Out[7]: 2a
```

So, $\frac{\partial J}{\partial a} = 2a$. When $a = 11$, $\frac{\partial J}{\partial a} = 22$. This matches our arithmetic calculation above. If you have not already done so, you can go back to the diagram above and fill in the value for $\frac{\partial J}{\partial a}$.

$$\frac{\partial J}{\partial w}$$

$\boxed{\text{a = w*b}}$ Moving from right to left, the next value we would like to compute is $\frac{\partial J}{\partial w}$. To do this, we first need to calculate $\frac{\partial a}{\partial w}$ which describes how the output of this node, a, changes when the input w changes a little bit.

Arithmetically

Find $\frac{\partial a}{\partial w}$ by finding how a changes as a result of a little change in w.

```
In [8]: w_epsilon = w + 0.001      # w plus a small value, epsilon
        a_epsilon = J*w_epsilon
        k = a_epsilon - a(w,b)     # difference divided by epsilon
        print("J = {J}, a_epsilon = {a_epsilon}, da_dw ~= k = {k} ")
        w = 11, a_epsilon = 11.003, da_dw ~= k = 1.0000000000031537
```

Calculated arithmetically, $\frac{\partial a}{\partial w} = 1$. Let's try it with SymPy.

```
In [9]: sa = 2 * 3*sw
        sa
Out[9]: 3w*2
In [10]: da_dw = diff(sa,w)
         da_dw
Out[10]: 3
```

> The next step is the interesting part.
>
> - We know that a small change in w will cause a to change by 3 times that amount.
> - We know that a small change in a will cause J to change by 2 x a times that amount. (an11 in this example)
> - so, putting these together,
> - We know that a small change in w will cause J to change by 3 x 2 x a times that amount.
>
> These cascading changes go by the name of the chain rule. It can be written like this:
>
> $$\frac{\partial J}{\partial w} = \frac{\partial a}{\partial w}\frac{\partial J}{\partial a}$$
>
> Working right to left, the chain rule states that:
>
> $$\frac{\partial J}{\partial w} = 3 \quad \frac{\partial J}{\partial a} = 2a \quad \frac{\partial J}{\partial w} = 22 \quad J \uparrow 0.66\,w$$
>
> It's worth spending some time thinking this through if it is not clear. This is a key take-away.
>
> Let's try calculating it:

```
In [11]: dJ_dw = da_dw * dJ_da
         dJ_dw
Out[11]: 6w
```

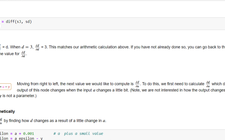
And a bit ?? in this example so $\frac{\partial J}{\partial w} = 66$. We can check this arithmetically:

```
In [12]: w_epsilon = w + 0.001
         a_epsilon = J *w_epsilon
         k = a_epsilon - a(w,b)     # difference divided by epsilon
         k = k - b
         print("J = {J}, J_epsilon = {J_epsilon}, dJ_dw ~= k = {k} ")
         J = 121, J_epsilon = 121.066006000000,  dJ_dw ~= k = 66.0000000000062
```

OK! You can now fill the values for $\frac{\partial a}{\partial w}$ and $\frac{\partial J}{\partial w}$ in the diagram if you have not already done so.

Another view

One could visualize these cascading changes this way:



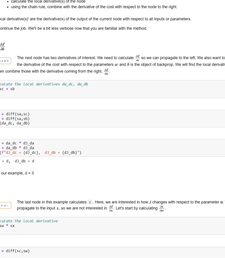
A small change in w is multiplied by $\frac{\partial a}{\partial w}$ resulting in a change that is 3 times as large. This larger change is then multiplied by $\frac{\partial J}{\partial a}$ resulting in a change that is now 3 x 22 = 66 times larger.

Computation Graph of a Simple Neural Network

Below is a graph of the neural network used in the lecture with different values. Try and fill in the values in the boxes. Note, the interactive graphic is not particularly robust. If you run into trouble with the interface, run the cell below again to restart.

```
In [13]: plt.close('all')
         plt_network(config_nwk, "./images/C2_W2_BP_network1.PNG")
```

Click in boxes to fill in values.




```
Out[13]: <lab_utils_backprop.plt_network at 0x7f9ec5c8e10>
```

Below, we will go through the computations required to fill in the above computation graph in detail. We start with the forward path.

Forward propagation

The calculations in the forward path are the ones you have recently learned for neural networks. You can compare the values below to those you calculated for the diagram above.

```
In [14]: # Inputs and parameters
         w = -2
         a = -2
         b = 4
         y = 1
         # Calculate per step values
         a = w*b
         a = a*y
         d = a - y
         J = d**2
         print("J={J}, a={d}, z={a}, c={b}")
         J=64.0, d=8, a=8, c=-4
```

Backward propagation (Backprop)

$\boxed{\text{J = d}^2}$ As described in the lectures, backprop starts at the right and moves to the left. The first node to consider is $J = d^2$ and the first step is to find $\frac{\partial J}{\partial d}$.

$$\frac{\partial J}{\partial d}$$

Arithmetically

Find $\frac{\partial J}{\partial d}$ by finding how J changes as a result of a little change in d.

```
In [15]: d_epsilon = d + 0.001
         J_epsilon = d_epsilon**2
         k = J_epsilon - J(d,a,b,y)     # difference divided by epsilon
         print("J = {J}, J_epsilon = {J_epsilon}, dJ_dd ~= k = {k} ")
         J = 64.0, J_epsilon = 64.016000000000,  dJ_dd ~= k = 16.0000000000727105
```

$\frac{\partial J}{\partial d}$ is 3, which is the value of d. Our result is not exactly d because our epsilon value is not infinitesimally small.

Symbolically

Now, let's use SymPy to calculate derivatives symbolically, as we did in the derivatives optional lab. We will prefix the name of the variable with an 's' to indicate this is a symbolic variable.

```
In [16]: sw,sa,sb,sy,s1 = symbols('w,a,b,y,1')
         s1 = sa**2
         s1.subs({sa,d})
         s1
Out[16]: d²
         d²
In [17]: s1.subs({s1,d})
Out[17]: 2
         d
In [18]: dJ_dd = diff(s1, sd)
         dJ_dd
Out[18]: d
```

So, $\frac{\partial J}{\partial d} = d$. When $d = 3$, $\frac{\partial J}{\partial d} = 3$. This matches our arithmetic calculation above. If you have not already done so, you can go back to the diagram above and fill in the value for $\frac{\partial J}{\partial d}$.

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

$\boxed{\text{d = a - y}}$ Moving from right to left, the next value we would like to compute is $\frac{\partial J}{\partial a}$. To do this, we first need to calculate $\frac{\partial d}{\partial a}$ which describes how the output of this node changes when the input a changes a little bit. (Note, we are not interested in how the output changes when y changes since y is not a parameter.)

Arithmetically

Find $\frac{\partial d}{\partial a}$ by finding how d changes as a result of a little change in a.

```
In [19]: a_epsilon = a + 0.001       # a plus a small value
         d_epsilon = a_epsilon - y
         k = d_epsilon - d(J,a,b,y)     # difference divided by epsilon
         print("J = {J}, d_epsilon = {d_epsilon}, dd_da ~= k = {k} ")
         d = 3, d_epsilon = 3.0010000000000,  dd_da ~= k = 1.0000000000001114
```

Calculated arithmetically, $\frac{\partial d}{\partial a} = 1$. Let's try it with SymPy.

Symbolically

```
In [20]: sd = sa - sy
         sd
Out[20]: a - y
In [21]: dd_da = diff(sd,sa)
         dd_da
Out[21]: 1
```

Calculated arithmetically, $\frac{\partial d}{\partial a}$ also equals 1.

> The next step is the interesting part, repeated again in this example:
>
> - We know that a small change in a will cause d to change by 1 times that amount.
> - We know that a small change in d will cause J to change by d times that amount. (d=3 in this example)
> - so, putting these together,
> - We know that a small change in a will cause J to change by 1 x d times that amount.
>
> This is again the chain rule. It can be written like this:
>
> $$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial d}\frac{\partial d}{\partial a}$$
>
> Let's try calculating it:

```
In [22]: dJ_da = dd_da * dJ_dd
         dJ_da
Out[22]: d
```

And d is 3 in this example so $\frac{\partial J}{\partial a} = 3$. We can check this arithmetically:

```
In [23]: a_epsilon = a + 0.001
         d_epsilon = a_epsilon - y
         J_epsilon = d_epsilon**2
         k = J_epsilon - J(d,a,b,y)
         print("J = {J}, J_epsilon = {J_epsilon}, dJ_da ~= k = {k} ")
         J = 8.5, J_epsilon = 8.5030000500000,  dJ_da ~= k = 3.0005000000002277
```

OK, they match! You can now fill the values for $\frac{\partial J}{\partial a}$ and $\frac{\partial d}{\partial a}$ in the diagram if you have not already done so.

> **The steps in backprop**
> Now that you have worked through several nodes, we can refine down the basic method:
> working right to left, for each node:
>
> - calculate the local derivative(s) of the node
> - using the chain-rule, combine with the derivative of the cost with respect to the node to the right.
>
> The 'local derivative(s)' are the derivative(s) of the output of the current node with respect to all inputs or parameters.
>
> Let's continue the job. We'll be a bit less verbose now that you are familiar with the method.
>
> $$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial a}$$
>
> $\boxed{\text{a = w*b}}$ The next node has two derivatives of interest. We need to calculate $\frac{\partial a}{\partial w}$ so we can propagate to the left. We also want to calculate $\frac{\partial a}{\partial b}$. Finding the derivative of the cost with respect to the parameters w and b is the object of backprop. We will find the local derivatives, $\frac{\partial a}{\partial w}$ and $\frac{\partial a}{\partial b}$ and then combine those with the derivative coming from the right, $\frac{\partial J}{\partial a}$.

```
In [24]: # calculate the local derivatives da_dw, da_db
         sa = sw * sb
         sa
Out[24]: b*w
         b*w
In [25]: da_dw = diff(sa,sw)
         da_dw = diff(sa,w)
         print(da_dw, da_db)
         b
In [26]: da_dw = diff(sa,w)
         da_db = diff(sa,b)
         print("{da_dw}, {da_db}")
         b
In [27]: # da_dw = da_db
         dJ_da = dJ_db
         dJ_da
Out[27]: d
```

And d is 3 in this example so $\frac{\partial J}{\partial a} = 3$. We can check this arithmetically:

```
In [28]: a_epsilon = a + 0.001
         d_epsilon = a_epsilon - y
         J_epsilon = d_epsilon**2
         k = J_epsilon - J(d,a,b,y)
         print("J = {J}, J_epsilon = {J_epsilon}, dJ_da ~= k = {k} ")
         J = 8.5, J_epsilon = 8.5030000500000,  dJ_da ~= k = 3.0005000000002277
```

OK, they match! You can now fill the values for $\frac{\partial J}{\partial a}$ and $\frac{\partial J}{\partial b}$ in the diagram if you have not already done so.

> **The steps in backprop**
> Now that you have worked through several nodes, we can refine down the basic method:
> working right to left, for each node:
>
> - calculate the local derivative(s) of the node
> - using the chain-rule, combine with the derivative of the cost with respect to the node to the right.
>
> The 'local derivative(s)' are the derivative(s) of the output of the current node with respect to all inputs or parameters.
>
> Let's continue the job. We'll be a bit less verbose now that you are familiar with the method.
>
> $$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a}$$
>
> $\boxed{\text{a = w*b}}$ The last node in this example calculates $\frac{\partial a}{\partial w}$. Here, we are interested in how J changes with respect to the parameter w. We will not back propagate to the input a, so we are not interested in $\frac{\partial a}{\partial a}$. Let's start by calculating $\frac{\partial a}{\partial w}$.

```
In [29]: # calculate the local derivative
         sa = sw * sb
         sa
Out[29]: a*b
In [30]: da_dw = diff(sa,sw)
         da_dw
Out[30]: b
```

This derivative is a bit more exciting than the last one. This will vary depending on the value of a. This is 2 in our example.

Combine this with $\frac{\partial J}{\partial a}$ to find $\frac{\partial J}{\partial w}$.

```
In [31]: dJ_dw = da_dw * dJ_da
         dJ_dw
Out[31]: d*b
In [32]: print("{J_dw = {dJ_dw.subs({sd,d},{sa,a})}}")
         dJ_dw = 2*8
```

$d = 3$, so $\frac{\partial J}{\partial w} = 6$ for our example.
Let's test this arithmetically:

```
In [33]: a_epsilon = ((w+0.001)*b - y)**2
         J_epsilon = (a_epsilon - y)**2
         print("J = {J}, J_epsilon = {J_epsilon}, dJ_dw ~= k = {k} ")
         J = 8.5, J_epsilon = 4.5030001,  dJ_dw ~= k = 6.0010000000610
```

They match! Great. You can add $\frac{\partial J}{\partial w}$ to the diagram above and our analysis is complete.

Congratulations!

You've worked through an example of back propagation using a computation graph. You can apply this to larger examples by following the same node by node approach.