Residual connections: a gradient highway

References

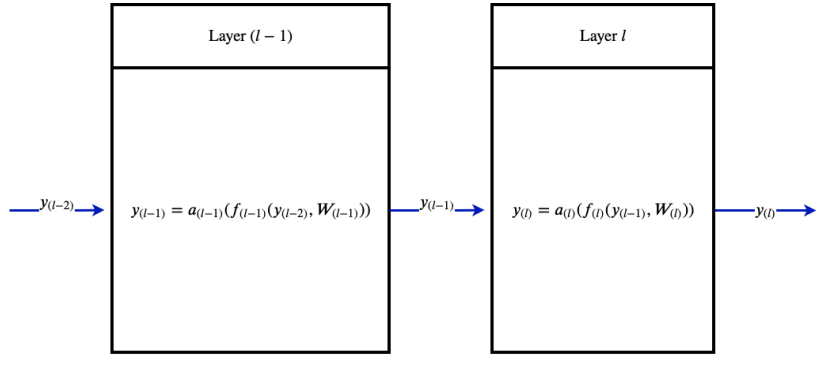
Deep Residual Learning (https://arxiv.org/abs/1512.03385)

There is a clever architectural "trick" to combat the problem of vanishing and exploding gradients.

Consider two layers of a Neural Network

$$\begin{aligned} \mathbf{y}_{(l-1)} &= a_{(l-1)} \left(f_{(l-1)}(\mathbf{y}_{(l-2)}, \mathbf{W}_{(l-1)})\right) \\ \mathbf{y}_{(l)} &= a_{(l)} \left(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)})\right) \end{aligned}$$

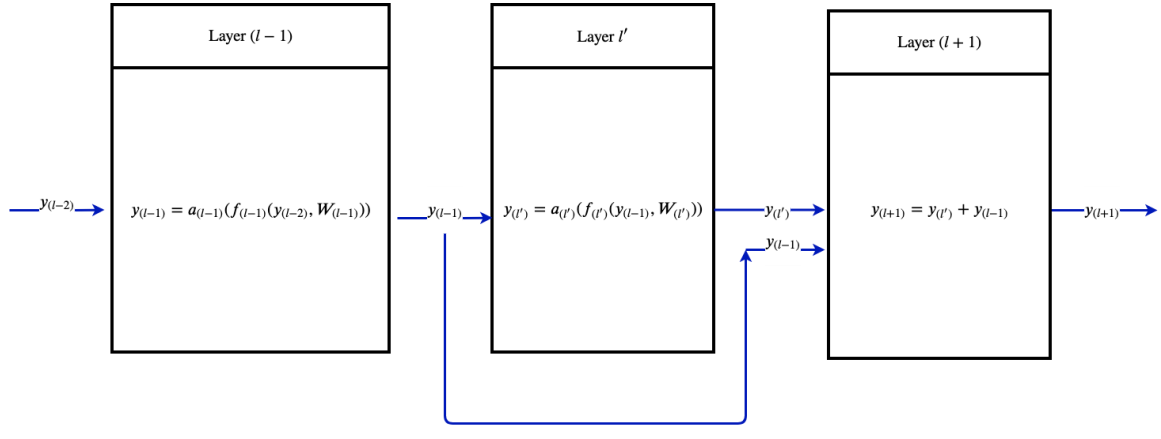
"Plain" Neural Network



Suppose we make a slight modification

- Allowing $\mathbf{y}_{(l-1)}$, the output of layer $(l-1)$
- To both flow into layer l
- And to "skip" over layer l
- Where it is added, in layer $(l+1)$ to the output of layer l

Residual Network with Skip Connection



Here are the equations defining the original network:

$$\mathbf{y}_{(l)} \quad = \quad a_{(l)} \quad \left(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)})\right)$$

and the modified network:

$$\begin{aligned} \mathbf{y}_{(l')} \quad &= \quad a_{(l')} \left(f_{(l')}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l')})\right) \\ \mathbf{y}_{(l+1)} \quad &= \quad \mathbf{y}_{(l')} + \mathbf{y}_{(l-1)} \end{aligned}$$

Now suppose we wanted the two networks to compute the same mapping from input $\mathbf{y}_{(l-1)}$ to output

$$\mathbf{y}_{(l+1)} = \mathbf{y}_{(l)}$$

Then

$$
\begin{aligned}
\mathbf{y}_{(l+1)} &= \mathbf{y}_{(l')} + \mathbf{y}_{(l-1)} && \text{definition of } \mathbf{y}_{(l+1)} \\
\mathbf{y}_{(l)} &= \mathbf{y}_{(l')} + \mathbf{y}_{(l-1)} && \text{requiring } \mathbf{y}_{(l+1)} = \mathbf{y}_{(l)} \\
\mathbf{y}_{(l')} &= \mathbf{y}_{(l)} - \mathbf{y}_{(l-1)} && \text{re-arranging terms}
\end{aligned}
$$

That is: layer l' computes the *residual* of $\mathbf{y}_{(l)}$ with respect to $\mathbf{y}_{(l-1)}$.

This type of architecture is called a *Residual Network* or *Residual Layer*.

The connection of $\mathbf{y}_{(l-1)}$ to the input of layer $(l+1)$ is called a *Skip connection*

This seems strange (and perhaps pointless) until you consider the Back Propagation process.

Recall how the Loss Gradient

$$\mathcal{L}'_{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}}$$

propagates backwards inductively

$$\mathcal{L}'_{(l-1)} \quad = \quad \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$$

It is modulated by Local Gradient $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$

In the original Neural Network, the Local Gradient relating the output and input $\mathbf{y}_{(l-1)}$ is

$$\begin{aligned}
\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} &= \frac{\partial a_{(l)}(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)})}{\partial \mathbf{y}_{(l-1)}} && \text{since } \mathbf{y}_{(l)} = a_{(l)}(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)}) \\
&= a'_{(l)} * f'_{(l)} && \text{where } a'_{(l)} = \frac{\partial a_{(l)}(\ldots)}{\partial f_{(l)}(\ldots)}; f'_{(l)} = \frac{\partial f_{(l)}(\ldots)}{\partial \mathbf{y}_{(l-1)}}
\end{aligned}$$

Whereas, in the modified neural network, the Local Gradient relating the output (now $\mathbf{y}_{(l+1)}$) to input $\mathbf{y}_{(l-1)}$ $$ \begin{array}[lll]\ \frac{\partial \y{(\ll+1)}}{\partial \y{(\ll-1)}} & = & \frac{\partial \y'\llp}{\partial \y{(\ll-1)}} + \frac{\partial \y{(\ll-1)}}{\partial \y{(\ll-1)}} } & \text{since } \y{(\ll+1)} = \y'\llp

- \y{(\ll-1)}\ & = & \frac{\partial a{(\ll')} (f{(\ll')}(\y{(\ll-1)}, \W{(\ll')}) }{\partial \y{(\ll-1)}} } + 1 & \text{since } y'\llp = a{(\ll')}(f{(\ll')}(\y{(\ll-1)}, \W{(\ll')}) \ & = & a'{(\ll')} * f'{(\ll')} + 1 & \text{where } a'{(\ll')} = \frac{\partial a{(\ll')}(\ldots)}{\partial f{(\ll')}(\ldots)}; f'{(\ll')} = \frac{\partial f{(\ll')}(\ldots)}{\partial \y_{(\ll-1)}} \end{array} $$

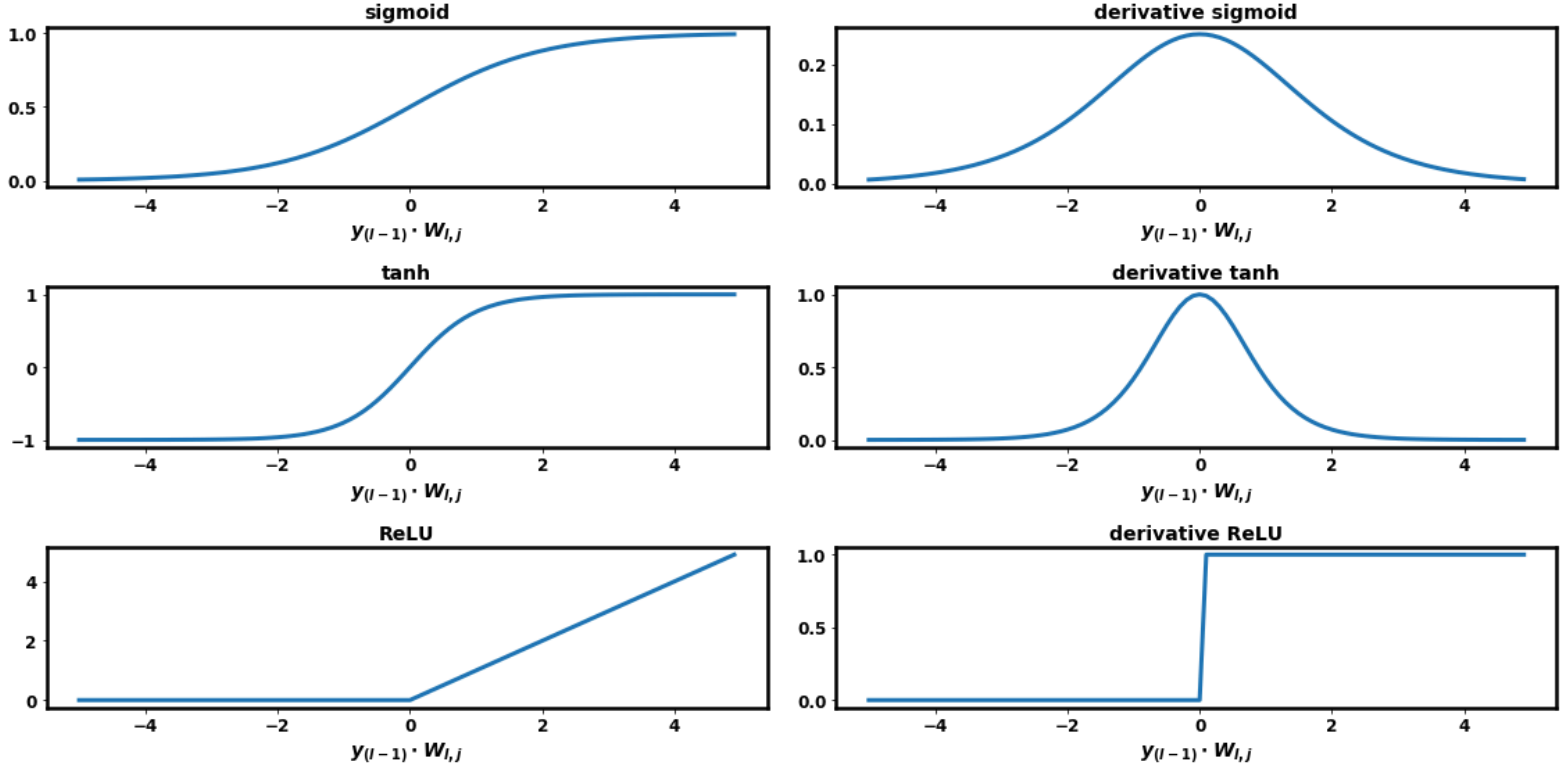
The main difference between the Local Gradients of the two networks

- is the "+1" term

Since the derivative a' of most activation functions is *less than one*

- The Local Gradient of the original network *diminishes* the Loss Gradient
- The Local Gradient of the modified network *does not* diminish the Loss Gradient

In [5]:
```
fig, _ = nnh.plot_activations( np.arange(-5,5, 0.1) )
```



The "+ 1" term in the Local Gradient of the Residual Network allows the Loss Gradient to flow backwards un-diminished

- It is like an "express lane" on the backward pass highway !

This simple trick vanquishes the vanishing gradient !

It is one of the major reasons that we are able to train extremely deep Neural Networks.

Another important fact

- Adding an additional *residual* layer to a Neural Network *cannot* increase the Loss !

Because there exists a set of weights $\mathbf{W}'_{(l')}$ for which

$$a_{(l')}\left(f_{(l')}(\mathbf{y}_{(l-1)}, \mathbf{W}'_{(l')})\right) = 0$$

The residual layer would choose these weights rather than suffering an increased loss !

By choosing these weight $\mathbf{W}'_{(l')}$ the residual layer computes the identity functon

$$\mathbf{y}_{(l+1)} = \mathbf{y}_{(l-1)}$$

Thus, adding more residual layers cannot hurt performance.

Without the skip connection, it has been observed empirically

- That it is difficult for a Neural Network to learn the identity function
- So the skip connection is quite useful

Technical aside

There is an unresolved debate where to place the "head" of the skip connection

- insider the activation function
- outside the activation function

We choose the latter to simplify the derivative expression for the loss gradient.

Conclusion and Preview: Skip connections in LSTM's, GRU's

The gradient highway also turns out to be useful in RNN's.

There are more powerful variants of the RNN called LSTM and GRU which avoid vanishing gradients, partially through the use of skip connections.

These variants enhance the power of skip connections by allowing selective skipping via the use of "gates".

We will see this shortly.

In [6]:
```
print("Done")
```

```
Done
```