Notation review

- Layer l: $0 \le l \le (L-1)$

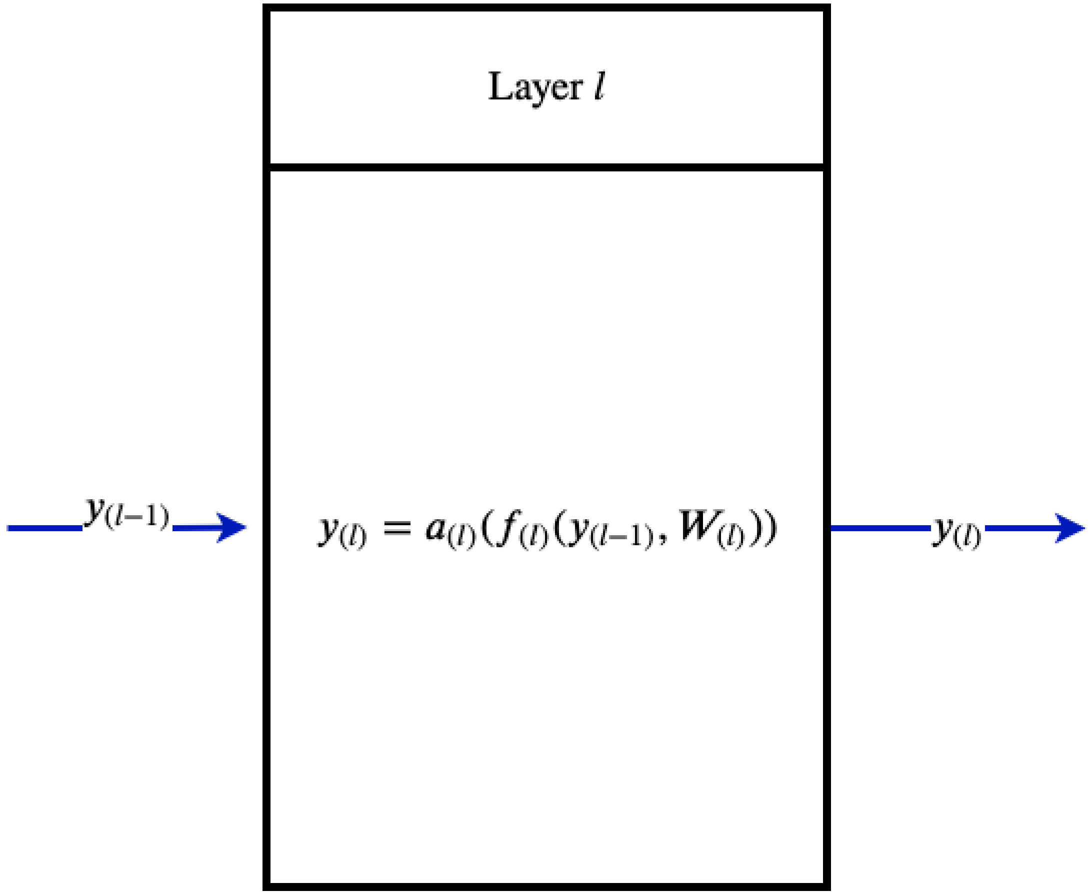
$$\mathbf{y}_{(l)} = a_{(l)} \left(f_{(l)}(\mathbf{y}_{(l-1)}, \mathbf{W}_{(l)}) \right)$$

- Layer 0 is input

$$\mathbf{y}_{(0)} = \mathbf{x}$$

- Layer L is *head* (Classifier/Regression)
- Layer $(L+1)$ is Loss layer
- We omit writing a separate bias term $\mathbf{b}_{(l)}$: we fold it into the weights $\mathbf{W}_{(l)}$

Layer notation



Back propagation

Gradient Descent updates weights $\mathbf{W}$ using the derivative of the loss $\mathcal{L}$ with respect to $\mathbf{W}_{(l)}$.

$$\mathbf{W} = \mathbf{W} - \alpha * \frac{\partial \mathcal{L}}{\partial \mathbf{W}}$$

where $\alpha \leq 1$ is the learning rate.

Since each layer l has its own weights $\mathbf{W}_{(l)}$ the derivatives needed are

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(l)}} \text{ for } l = 1, \ldots, L$$

We will show how to compute these derivatives via a procedure known as *Back propagation*.

It is really nothing more than an *iterated* application of the Chain Rule of Calculus.

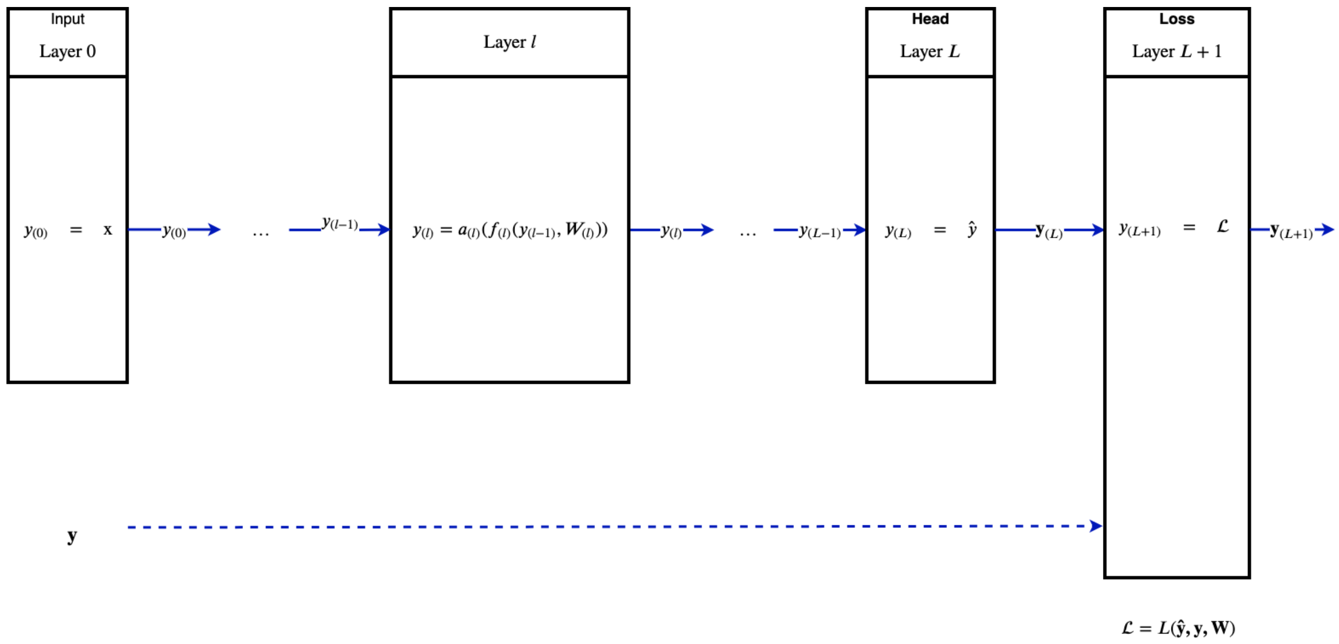
Recall that we created layer $(L+1)$ to compute the Loss function

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

where layer L is the "head" (Classifier/Regression).

Our computation thus looks like:

Additional Loss Layer (L+1)



We will compute the derivative of the Loss with respect to $\mathbf{y}_{(l)}$, for each $1 \leq l \leq (L+1)$

Let

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

denote the derivative of $\mathcal{L}$ with respect to the output of layer l, i.e., $\mathbf{y}_{(l)}$.

This is called the **loss gradient**.

The loss gradient can be computed for each layer sequentially in *reverse order*.

That is why the procedure is called *Backwards propagation:*

Starting at the end

$$
\begin{aligned}
\mathcal{L}'_{(L+1)} &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(L+1)}} \\
&= \frac{\partial \mathbf{y}_{(L+1)}}{\partial \mathbf{y}_{(L+1)}} \quad \text{by construction: } \mathbf{y}_{(L+1)} = \mathcal{L} \\
&= 1
\end{aligned}
$$

We inductively work our way backwards

- Given $\mathcal{L}'_{(l)}$
- Compute $\mathcal{L}'_{(l-1)}$
- Using the chain rule

$$
\begin{array}{lll}
\mathcal{L}'_{(l-1)} & = & \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l-1)}} \\
& = & \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} & \text{chain rule} \\
& = & \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} & \text{definition of } \mathcal{L}'_{(l)}
\end{array}
$$

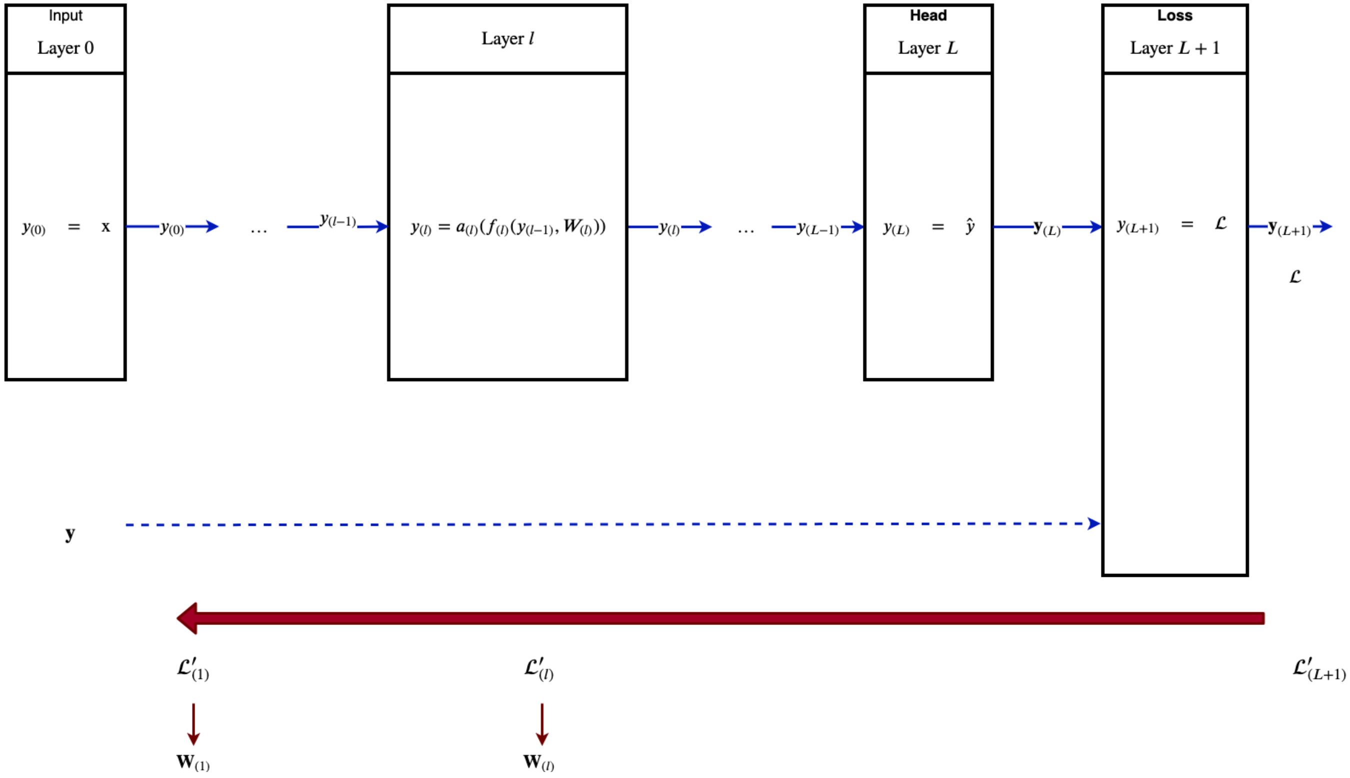
Observe: the loss gradient $\mathcal{L}'_{(l)}$ of layer l

- flows backward to layer $l-1$
- modulated by $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$

Expanding this logic: The loss gradient "flows backward", from $\mathbf{y}_{(L+1)}$ to $\mathbf{y}_{(1)}$.

This is referred to as the *backward pass*.

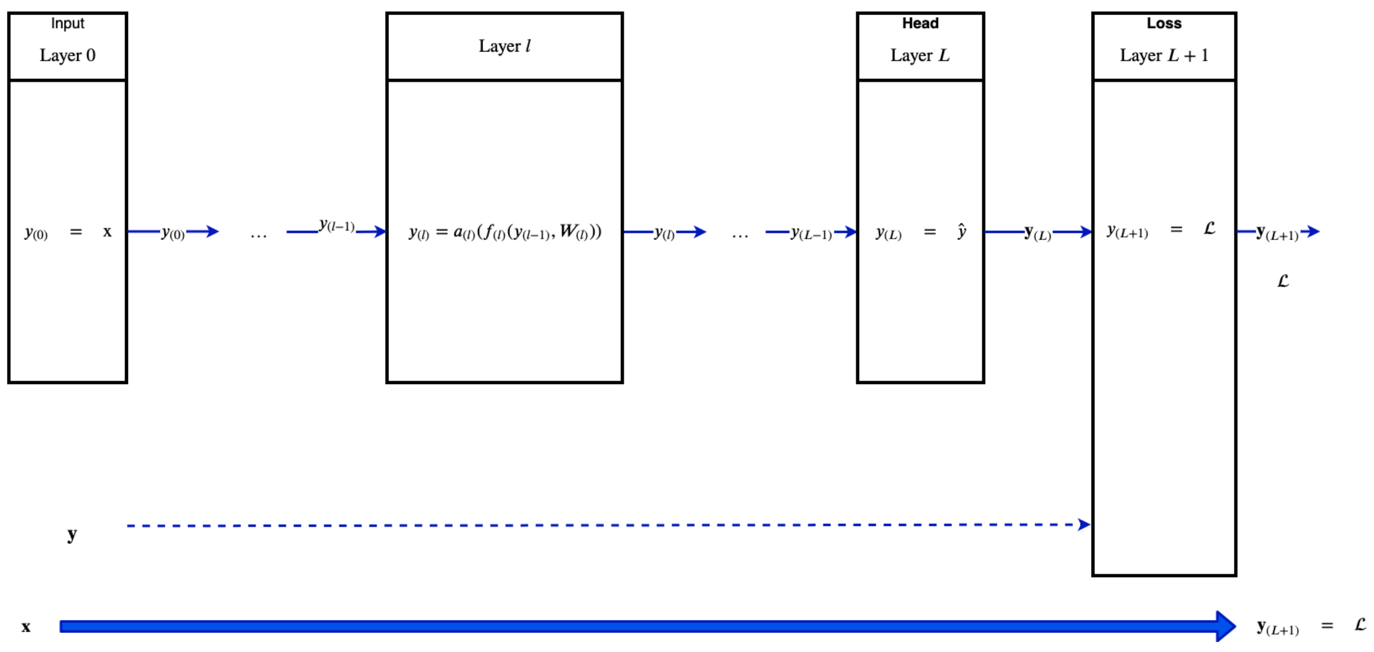
Backward pass: Loss to Weights



Contrast this to the information flow that leads to prediction $\hat{\mathbf{y}} = \mathbf{y}_{(L)}$

- Information flows forward, from input $\mathbf{x}$ to $\mathbf{y}_{(L)}$
- This is called the *forward pass*

Forward Pass: Input to Loss



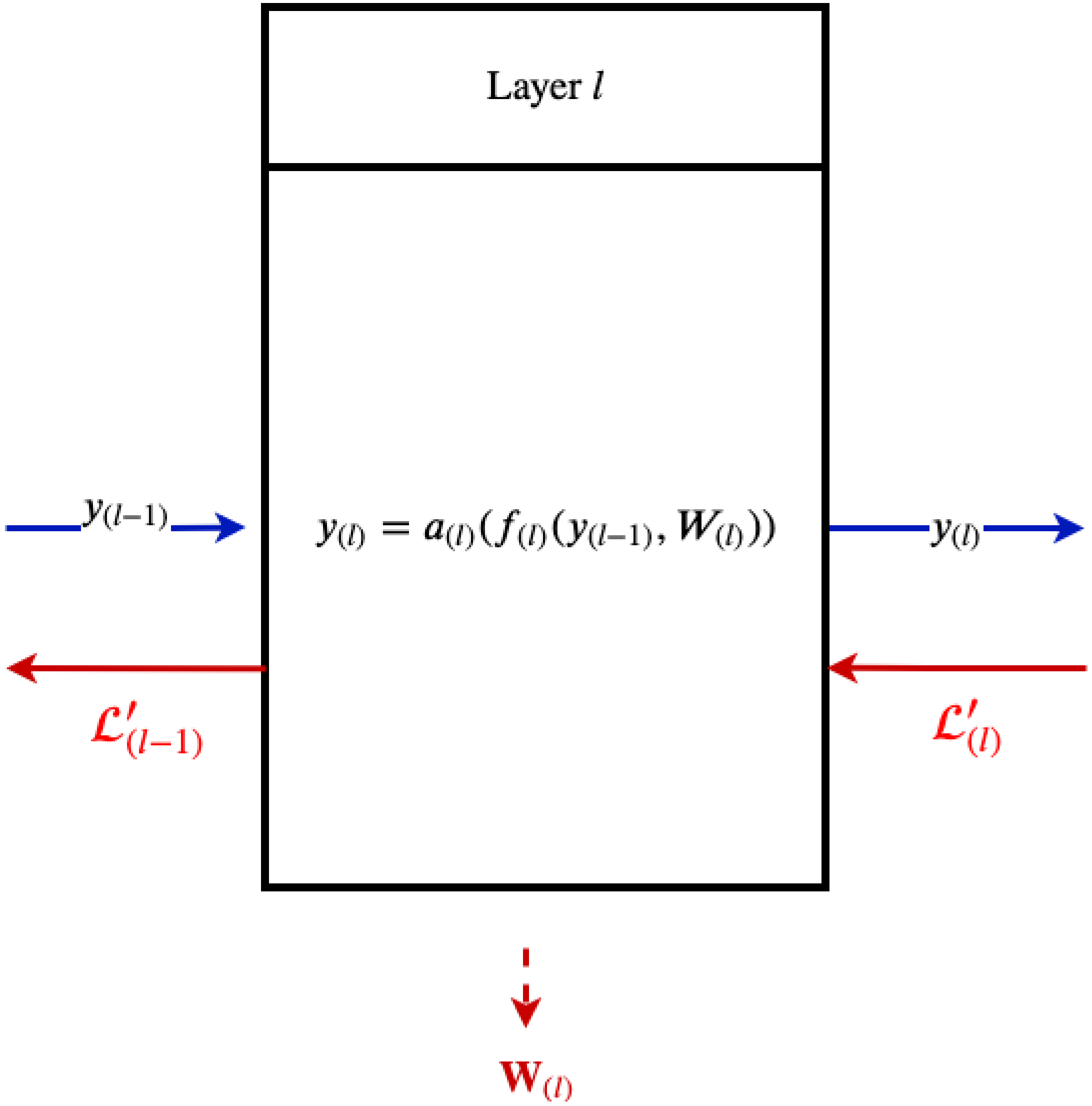
The purpose of flowing the loss gradient backwards is to find the optimal value for $\mathbf{W}_{(l)}$, the weights for each layer $l, 1 \le l \le L$

- Via Gradient Descent, which modifies the current estimate of $\mathbf{W}_{(l)}$
- Using the derivative of the loss with respect to $\mathbf{W}_{(l)}$
- Which can be obtained via another application of the Chain Rule

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

Here is a larger picture of the flow during the Forward and Backward pass at layer l.

Forward and Backward pass: Detail



Since $\mathbf{y}_{(l)}$ is a function of

- $\mathbf{y}_{(l-1)}$, the previous layer's output
- And $\mathbf{W}_{(l)}$, the weights of layer l.

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

the computation of $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$ depends on the functional form of $a_{(l)}$ and can be obtained via the rules of Calculus.

The derivatives of $\mathbf{y}_{(l)}$ with respect to each of its inputs

- $\dfrac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$
- $\dfrac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{W}_{(l)}}$

are called **local gradients**

Note that we can compute the local gradients

- During the **forward pass**
- Since the derivatives only depend on inputs and not on any value subsequent to layer l

We will take advantage of this fact when we demonstrate some pseudo-code for the Forward and Backward passes.

So we say that the loss gradient of the preceding layer is the product of

- The loss gradient of the current layer
- The local gradient with respect to the layer's inputs

Conclusion

Gradient Descent depends on the ability to compute

- The derivative of the Loss with respect to the weights

We demonstrated a procedure called Back Propagation to compute these derivatives.

The forward pass of a Neural Network is the process of computing outputs (predictions) from inputs.

Back propagation is what happens in the backward pass, which maps loss to weights.

In [4]:

```
print("Done")
```

```
Done
```