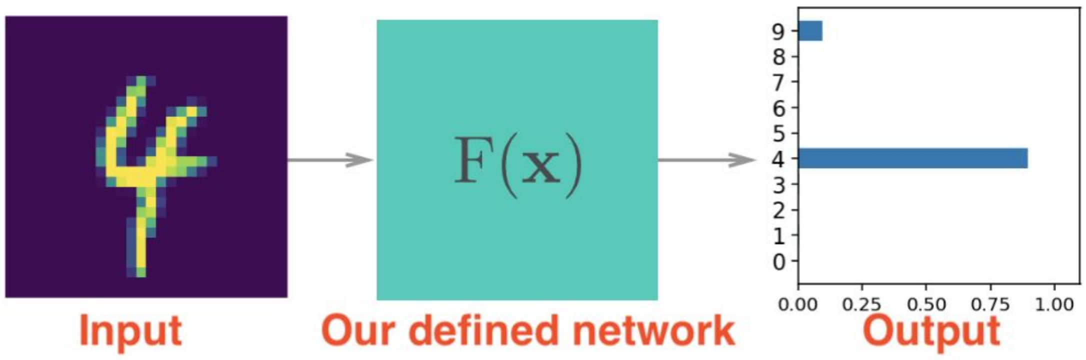
Reference: Lesson 5: Introduction of PyTorch: 9, 10, 11

- **Goal**: use the network we defined previous lesson and train them
- Neural networks with non-linear activations work like universal function approximators.

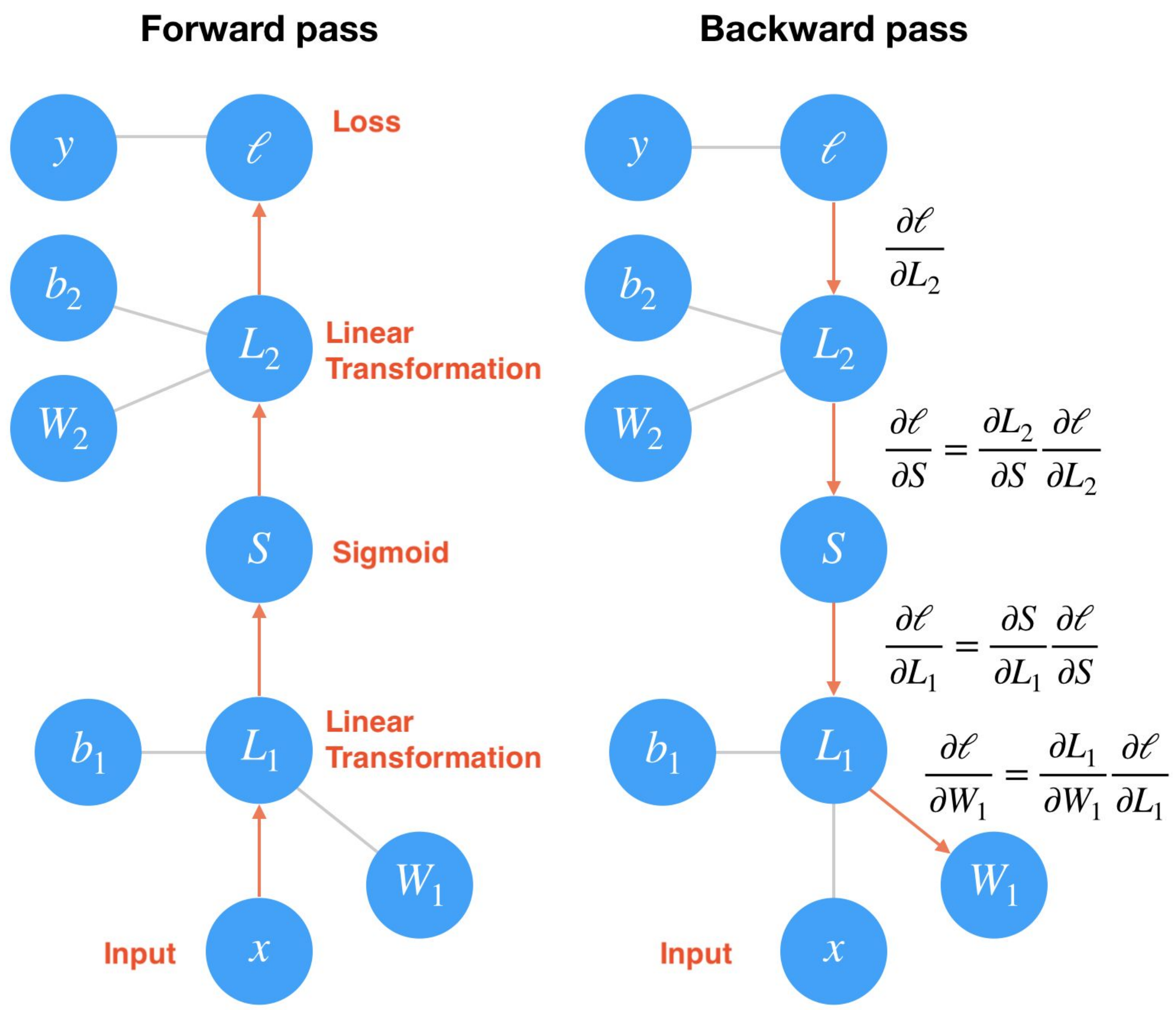


- **Problem**: How can we train the network ?
    - At first the network is naive, it doesn't know the function mapping the inputs to the outputs. We train the network by showing it examples of real data, then adjusting the network parameters such that it approximates this function.
    - we calculate a **loss function** (also called the cost), a measure of our prediction error. This is used to define how poor the network is when predict real outputs
    - Minimize the loss function with gradient descent process

This is a good time to link lesson 3 loss function to this :)

- **Backpropagation**

Goal: adjust the weights and biases to minimize the loss.



Forward pass
Backward pass
Loss
Linear Transformation
Sigmoid
Linear Transformation
Input
Input
y
ℓ
b_2
L_2
W_2
S
b_1
L_1
W_1
x
$\frac{\partial \ell}{\partial L_2}$
$\frac{\partial \ell}{\partial S} = \frac{\partial L_2}{\partial S}\frac{\partial \ell}{\partial L_2}$
$\frac{\partial \ell}{\partial L_1} = \frac{\partial S}{\partial L_1}\frac{\partial \ell}{\partial S}$
$\frac{\partial \ell}{\partial W_1} = \frac{\partial L_1}{\partial W_1}\frac{\partial \ell}{\partial L_1}$

- **How to calculate loss in PyTorch**
    - In nn module, we can use cross-entropy loss (nn.CrossEntropyLoss)
    - To actually calculate the loss, you first define the criterion then pass in the output of your network and the correct labels.
    - We need to pass in the raw output of our network into the loss, not the output of the softmax function. This raw output is usually called the *logits* or *scores*. We use the logits because softmax gives you probabilities which will often be very close to zero or one but floating-point numbers can't accurately represent values near zero or one (read more here). It's usually best to avoid doing calculations with probabilities, typically we use log-probabilities.

- Build nn.Sequential with LogSoftmax, dim = 1; criterion should be nn.NLLLoss

- **We calculated a loss, now what ?**
    - We are going to use it to calculate gradients of all our parameters with respect to the loss
    - Autograd works by keeping track of operations performed on tensors, then going backwards through those operations, calculating gradients along the way.
    - you need to set requires_grad = True on a tensor. You can do this at creation with the requires_grad keyword, or at any time with x.requires_grad_(True)
    - With PyTorch, we run data forward through the network to calculate the loss, then, go backwards to calculate the gradients with respect to the loss. Once we have the gradients we can make a gradient descent step.

- **Optimizer**
    - An optimizer that we'll use to update the weights with the gradients, PyTorch’s optim.package

===
We know about:

- Loss
- Gradient descent
- Backpropagation
- Optimizer

The general process for training:

- Make a forward pass through the network
- Use the network output to calculate the loss
- Perform a backward pass through the network with loss.backward() to calculate the gradients
- Take a step with the optimizer to update the weights