1. What is a neural network neuron?

   A Neural Network neuron, as opposed to brain neuron, is small storage unit or a signal. The computation unit, which is part of the brain neuron, is outside this storage unit, which involves two things: a weight and an activation function. The neuron is connected to other neurons through input and output weights.

   Here is a schematic diagram of a neural network neuron:

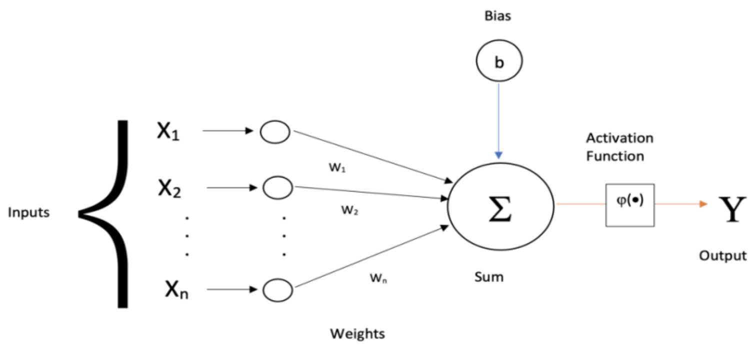

Image source: https://laptrinhx.com/artificial-neural-networks-and-its-intuition-2081057101/

2. What is the use of the learning rate?

   The learning rate is a (hyper)parameter which is tuned in such a way to ensure descent (or ascent) of weights so as to reach a point where the error reaches minimum in a gradual manner. A just right value of learning rate helps to reach to that global minimum of error in a way which is not too slow to reach to minimum or is not too fast that we overshoot that minimum point.

3. How are weights initialized?

   The weights are initialized randomly using Standard Normal probability distribution (which has mean 0 and variance 1).

   Normal probability distribution has a bell shaped curve with its maximum situated a 0 with a symmetric spread on both sides of zero. Most of the real world elements, e.g. height (or weights) of students in a class follow normal distributions (of course with different means and variance). The essential point here is that it is a symmetric distribution with (nearly) equal number values lying on both sides of the mean.

   This ensures that values of weights are not particularly biased towards one or the other value.

4. What is "loss" in a neural network?

   Loss in a neural network is deviation of the predicted output from the actual output in the data set. The value of loss can be calculated depending on how we have defined the loss function. If we simply define the loss function to be difference between actual value and the predicted (calculated) value, it may so happen (though rarely) the predicted values are in such way that differences cancel each other out leaving the loss to be at (almost) zero. Therefore (wrongly) assuming the model is a good fit.

Hence it is absolutely essential to define the loss function very carefully to be able to get a fair estimate of the loss. Some of the most commonly used loss functions (among others) are sum of absolute differences,
$loss = \sum_{i=1}^{n} |y_i - \widehat{y_i}|$ or
sum of square of differences
$loss = \sum_{i=1}^{n} (y_i - \widehat{y_i})^2$, where
$n$ is number of samples in the data set, $y_i, \widehat{y_i}$ are actual and predicted values of the output respectively.

5. What is the "chain rule" in gradient flow?

From what I understand from the question, there are two things here: what a chain rule is, and (sort of) its role in gradient flow.

A chain rule is used when we are calculating derivative (gradient) of a function of variables which are dependent on other variables. For example, let

$z = f(x, y)$ and let $y = g(x)$. Then

$\frac{\partial z}{\partial x} = \frac{\partial f}{\partial x} . \frac{\partial x}{\partial x} + \frac{\partial f}{\partial y} . \frac{\partial y}{\partial x}$ which (of course ) will be $\frac{\partial z}{\partial x} = \frac{\partial f}{\partial x} . + \frac{\partial f}{\partial y} . \frac{\partial y}{\partial x}$ as $\frac{\partial x}{\partial x} = 1$ .

Now if $x = k(t)$ and we want to differentiate $z$ with respect to $t$, then we will have

$$\frac{\partial z}{\partial t} = \frac{\partial f}{\partial x} . \frac{\partial x}{\partial t} + \frac{\partial f}{\partial y} . \frac{\partial y}{\partial x} . \frac{\partial x}{\partial t}$$

That is the chain rule you need to apply when you differentiating a function which is function of another function (which is function of…..) of another variable, say ‘a’, with respect to ‘a’.

Now, Gradient tells the rate of change of error in the output with respect to (set of) weights, which is why we need to differentiate. The output finally delivered is a culmination of activated values in the previous neurons (in the previous layers) based on the weights there, thus is a function of function (of function of….) of weights (variables as in the chain rule discussion).

So if we have to calculate the gradient of the final output (to update the parameters), we realize that this output is (flowing) from input layer towards the output layer (via intermediate layer(s)) and hence is (in a way) dependent on the all (or some of) the weights going back to the input layer. Since the output is not just dependent on the flow from immediate previous layers but all the previous layers which makes it a function of function of (function of ….) of weight with respect to which we want to calculate the gradient.

So if we take the scenario as discussed in the class where output, lets say $y_{hat}$, was a function of previous two layers and was written as

$y_{hat} = \tanh(w_{311}.(\tanh(w_{111}x_1 + w_{121}x_2 + b_1)) + w_{321}.(\tanh(w_{211}x_1 + w_{221}x_2 + b_2)) + b_3)$

This can be written as, (in terms of previous discussion)

$y_{hat} = f_1(w_{311,}w_{321,}b_3, f_2(w_{211,}w_{221,}b_2), f_3(w_{111,}w_{121,}1))$

Therefore

$\frac{\partial y_{hat}}{\partial w_{111}} = \frac{\partial f_1}{\partial f_3} \times \frac{\partial f_3}{\partial w_{111}}$

If $f_3$ is further a function of $f_4\ (w_{111},\ldots.)$ (i.e. if there more layers in between), then the above equation will change to

$$\frac{\partial y_{hat}}{\partial w_{111}} = \frac{\partial f_1}{\partial f_3} \times \frac{\partial f_3}{\partial f_4} \times \frac{\partial f_4}{\partial w_{111}}$$

and so on.

This is what is “chain rule” in the gradient flow.