Non-linearity with activation functions (Tanh, Sigmoid, Relu, PRelu)

A neural network is modelled after the human brain that consists of neurons. To obtain the output, a neural network accepts an input and weights summed with bias before arriving at the output. An activation function is one of the most important factors in a neural network, which is applied to the input before deriving an output.

This activation function decides whether or not a neuron will be activated and transferred to the next layer. It is also referred to as threshold or transformation for the neurons as it decides if the neuron's input is relevant in the prediction process or not.

The main job of an activation function is to introduce non-linearity in a neural network. One way to look at this is that without a non-linear activation function, a neural network will behave just like a single-layer perceptron; it does not matter how many layers it has.

Activation Function & Non-Linearity

At the most basic level, a neural network consists of three main layers:

Input layer: This layer accepts input from the outside world to the network. No computation is performed here, and the only job is to pass the received information to the hidden layer.

Hidden layer: This layer accepts information from the input layer and performs all computations. This layer is hidden from the outside world and transfers the result to the next layer.

Output layer: It accepts the result from the hidden layer and relays it to the outside world.

The activation function is present in the hidden layer. The activation layer cannot be linear because irrespective of how complex the architecture is, a linear activation function is effective only one layer deep. Also, the real world and associated problems are highly non-linear. The only situation where using linearity may prove beneficial is in case of regression problems (think predicting housing prices).

A linear activation function lacks in performing backpropagation. Thus, it is not recommended to be used in a neural network. While a model may perform a task even without the presence of an activation function in a linear manner, it would lack efficiency and accuracy. The significance of the activation function lies in making a given model learn and execute difficult tasks. Further, a non-linear activation function allows the stacking of multiple layers of neurons to create a deep neural network, which is required to learn complex data sets with high accuracy.

Non-Linear Activation Functions

Examples of non-linear activation functions include:

Sigmoid function: The Sigmoid function exists between 0 and 1 or -1 and 1. The use of a sigmoid function is to convert a real value to a probability. In machine learning, the sigmoid function is generally used to refer to the logistic function, also called the logistic sigmoid function; it is also the most widely used sigmoid function (others are the hyperbolic tangent and the arctangent).

A sigmoid function is placed as the last layer of the model to convert the model's output into a probability score, which is easier to work with and interpret.

Another reason to use it mostly in the output layer is that it can otherwise cause a neural network to get stuck in training time.

TanH function: It is the hyperbolic tangent function whose range lies between -1 and 1, hence also called the zero-centred function. Because it is zero centred, it is much easier to model inputs with strongly negative, positive or neutral values. TanH function is used instead of sigmoid function if the output is other than 0 and 1. TanH functions usually find applications in RNN for natural language processing and

speech recognition tasks.

On the downside, in the case of both Sigmoid and TanH, if the weighted sum input is very large or very small, the function's gradient becomes very small and closer to zero.

ReLU function: Rectified Linear Unit, also called ReLU, is a widely favoured activation function for deep learning applications. Compared to Sigmoid and TanH activation functions, ReLU offers an upper hand in terms of performance and generalisation. In terms of computation too, ReLU is faster as it does not compute exponentials and divisions. The disadvantage is that ReLU overfits more, as compared with Sigmoid.

Softmax function: It is used to build a multi-class classifier to solve the problem of assigning an instance to one class when the number of possible classes is larger than two. Softmax ensures that the sum of outputs is 1. The softmax function squeezes the outputs for each class between 0 and 1 and divides it by the sum of outputs.

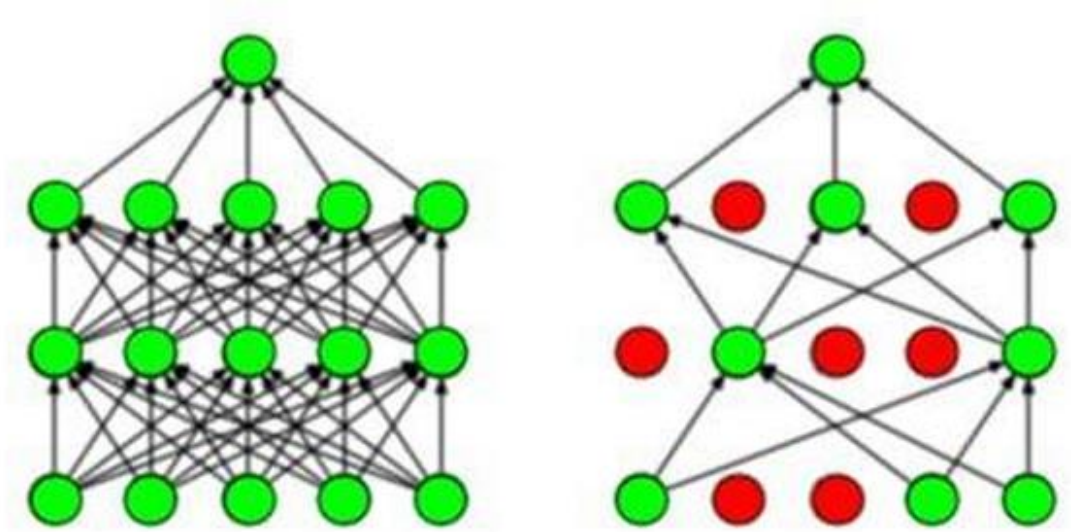
Parametric ReLU (PReLU)

ReLU has been one of the keys to the recent successes in deep learning. Its use has lead to better solutions than that of sigmoid. This is partially due to the vanishing gradient problem in case of sigmoid activations. But, we can still improve upon ReLU. LeakyReLU was introduced, which doesn't zero out the negative inputs as ReLU does. Instead, it multiplies the negative input by a small value (like 0.02) and keeps the positive input as is. But this has shown a negligible increase in the accuracy of our models. What if we can learn that small value during training so that our activation function can better adapt to the other parameters (like weights and biases). This is where PReLU comes in. We can learn the slope parameter using backpropagation at a negligible increase in the cost of training. In feed-forward networks, each layer learns a single slope parameter. In CNNs we can learn them for each layer or we can learn them for each channel for each layer. Number of slope parameters to be learned=Number of layers (or sum of all channels in every layer). This is negligible as compared to the number of weights and biases to be learned.

Dropout as regularization

What is Dropout?

"Dropout" in machine learning refers to the process of randomly ignoring certain nodes in a layer during training. In the figure below, the neural network on the left represents a typical neural network where all units are activated. On the right, the red units have been dropped out of the model — the values of their weights and biases are not considered during training.



(a) Standard Neural Net (b) After applying dropout.

Dropout is used as a regularization technique — it prevents overfitting by ensuring that no units are codependent.

Other Common Regularization Methods When it comes to combating overfitting, dropout is definitely not the only option. Common regularization techniques include:

1. Early stopping: stop training automatically when a specific performance measure (eg. Validation loss, accuracy) stops improving

2. Weight decay: incentivize the network to use smaller weights by adding a penalty to the loss function (this ensures that the norms of the weights are relatively evenly distributed amongst all the weights in the networks, which prevents just a few weights from heavily influencing network output)

3. Noise: allow some random fluctuations in the data through augmentation (which makes the network robust to a larger distribution of inputs and hence improves generalization)

4. Model combination: average the outputs of separately trained neural networks (requires a lot of computational power, data, and time) Dropout remains an extremely popular protective measure against overfitting because of its efficiency and effectiveness.

How Does Dropout Work?

When we apply dropout to a neural network, we're creating a "thinned" network with unique combinations of the units in the hidden layers being dropped randomly at different points in time during training. Each time the gradient of our model is updated, we generate a new thinned neural network with different units dropped based on a probability hyperparameter p. Training a network using dropout can thus be viewed as training loads of different thinned neural networks and merging them into one network that picks up the key properties of each thinned network.

It is observed that the models with dropout had a lower classification error than the same models without dropout at any given point in time. A similar trend was observed when the models were used to train other datasets in vision, as well as speech recognition and text analysis. The lower error is because

dropout helps prevent overfitting on the training data by reducing the reliance of each unit in the hidden layer on other units in the hidden layers.