**Andrew Ng’s Model**

|  |  |
| --- | --- |
|  | Terms:  CONV-1D   * Filter size * Stride * Number of filters   GRU   * Dropout * Batch Norm   Output Layer   * Dense |

**Dropout**:

*According to*[*Wikipedia*](https://en.wikipedia.org/wiki/Dropout_(neural_networks))*—  
The term “dropout” refers to dropping out units (both hidden and visible) in a neural network.*

At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

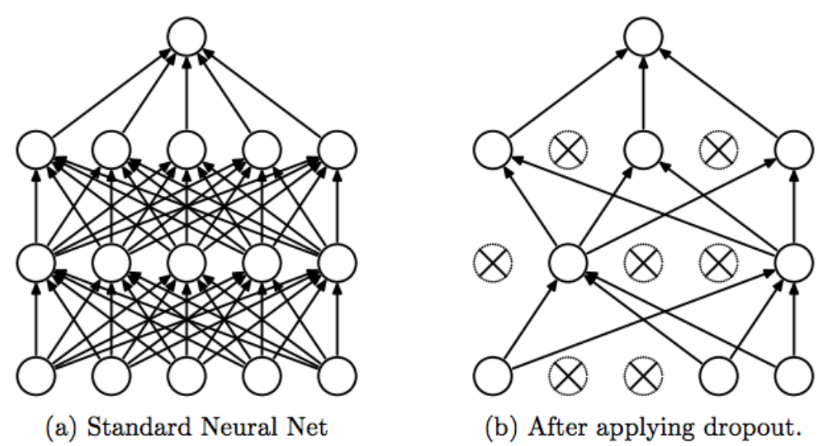
Purpose: to prevent overfitting. Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

**Training Phase:**

Training Phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).

**Testing Phase:**

Use all activations but reduce them by a factor *p* (to account for the missing activations during training).



1. Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
2. Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
3. With H hidden units, each of which can be dropped, we have  
   2^H possible models. In testing phase, the entire network is considered and each activation is reduced by a factor *p.*

**What is ADAM**

<https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

**The Vanishing Gradient Problem**

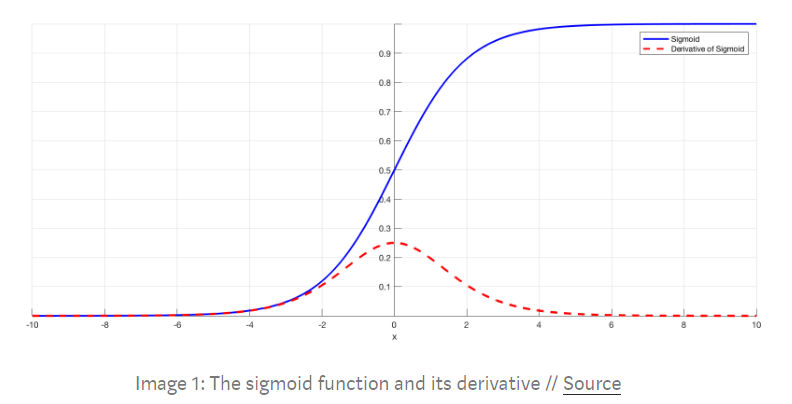
<https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>

**The problem:**

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

**Why:**

Certain activation functions, like the sigmoid function, squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.



As an example, Image 1 is the sigmoid function and its derivative. Note how when the inputs of the sigmoid function becomes larger or smaller (when |x| becomes bigger), the derivative becomes close to zero.

**Why it’s significant:**

For shallow network with only a few layers that use these activations, this isn’t a big problem. However, when more layers are used, it can cause the gradient to be too small for training to work effectively.

Gradients of neural networks are found using backpropagation. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers.

However, when *n* hidden layers use an activation like the sigmoid function, *n* small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers.

A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

**Solutions:**

The simplest solution is to use other activation functions, such as ReLU, which doesn’t cause a small derivative.

**Difference between sigmoid and reLu**:

<https://www.quora.com/What-is-the-difference-between-sigmoid-and-ReLU>

Some context: sigmoid and reLu are commonly used as activation functions

A sigmoid function will transform an input value into an output between 0.0 and 1.0. Any input larger than 1.0 will be transformed to 1.0, and inputs smaller than 0.0 will be transformed to 0.0. When used in a neural network, this leads to saturation around 1.0 and 0.0 and makes the midpoint quite sensitive to change. This means that when using the function to train a neural network, especially one with many layers, it becomes increasingly more difficult for the neural network to adapt and it’s weights and thus imporve performance. The sigmoid function can also cause neural networks to suffer from the vanishing gradient problem since error is backpropagated through the layers and decreases dramatically with each hidden layer.

ReLU takes an input and directly outputs the input if positive and outputs 0 if negative. ReLU combines the benfits of a linear activation function (no vanishing gradient) while allowing for complex relationships to be modeled in the function. Unlike sigmoid, reLU is called a piecewise function, because half of the output is linear (the positive output) while the other half is nonlinear. The ReLU function is also much less computationally taxing than sigmoid.

**LSTM vs GRU**

<https://datascience.stackexchange.com/questions/14581/when-to-use-gru-over-lstm>

The key difference between a GRU and an LSTM is that a GRU has two gates (*reset* and *update* gates) whereas an LSTM has three gates (namely *input*, *output* and *forget* gates).

GRU is related to LSTM as both are utilizing different way if gating information to prevent vanishing gradient problem. Here are some pin-points about GRU vs LSTM-

* The GRU controls the flow of information like the LSTM unit, but without having to use a ***memory unit***. It just exposes the full hidden content without any control.
* GRU is relatively new, and from my perspective, the performance is on par with LSTM, but computationally ***more efficient*** (*less complex structure as pointed out*). So we are seeing it being used more and more.
* **GRUs train faster** and perform better than LSTMs on **less training data** if you are doing language modeling (not sure about other tasks).
* **GRUs are simpler** and thus easier to modify, for example adding new gates in case of additional input to the network. It's just less code in general.
* **LSTMs** should in theory **remember longer sequences** than GRUs and outperform them in tasks requiring modelling long-distance relations.