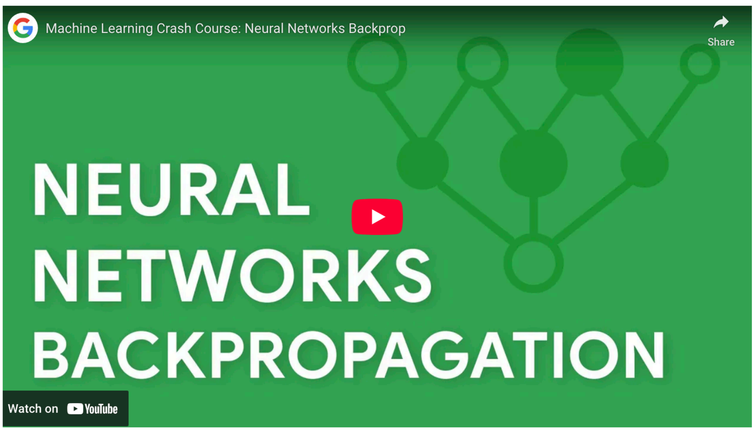
Neural Networks: Training using backpropagation

Backpropagation is the most common training algorithm for neural networks. It makes gradient descent feasible for multi-layer neural networks. Many machine learning code libraries (such as Keras) handle backpropagation automatically, so you don't need to perform any of the underlying calculations yourself. Check out the following video for a conceptual overview of how backpropagation works:



To learn more about building image models, check out the Image Classification course.

Best practices for neural network training

This section explains backpropagation's failure cases and the most common way to regularize a neural network.

NOTE: The backpropagation training algorithm makes use of the calculus concept of a gradient to adjust model weights to minimize loss. Understanding and debugging the issues below usually requires some background in calculus.

Vanishing Gradients

The **gradients** for the lower neural network layers (those closer to the input layer) can become very small. In **deep networks** (networks with more than one hidden layer), computing these gradients can involve taking the product of many small terms.

When the gradient values approach 0 for the lower layers, the gradients are said to "vanish". Layers with vanishing gradients train very slowly, or not at all.

The ReLU activation function can help prevent vanishing gradients.

Exploding Gradients

If the weights in a network are very large, then the gradients for the lower layers involve products of many large terms. In this case you can have exploding gradients: gradients that get too large to converge.

Batch normalization can help prevent exploding gradients, as can lowering the learning rate.

Dead ReLU Units

Once the weighted sum for a ReLU unit falls below 0, the ReLU unit can get stuck. It outputs 0, contributing nothing to the network's output, and gradients can no longer flow through it during backpropagation. With a source of gradients cut off, the input to the ReLU may not ever change enough to bring the weighted sum back above 0.

Lowering the learning rate can help keep ReLU units from dying.

There are also many variants of ReLU that were designed to address this specific problem, such as LeakyReLU, which you may want to consider using as an activation function to prevent dead ReLU units.

Dropout Regularization

Yet another form of regularization, called **dropout regularization**, is useful for neural networks. It works by randomly "dropping out" unit activations in a network for a single gradient step. The more you drop out, the stronger the regularization:

- 0.0 = No dropout regularization.
- 1.0 = Drop out all nodes. The model learns nothing.
- Values between 0.0 and 1.0 = More useful.

Key terms:

- Backpropagation
- Dropout regularization
- Exploding gradient problem
- Vanishing gradient problem