Neural Nets: An Educational Guide

By Matthew Escobar

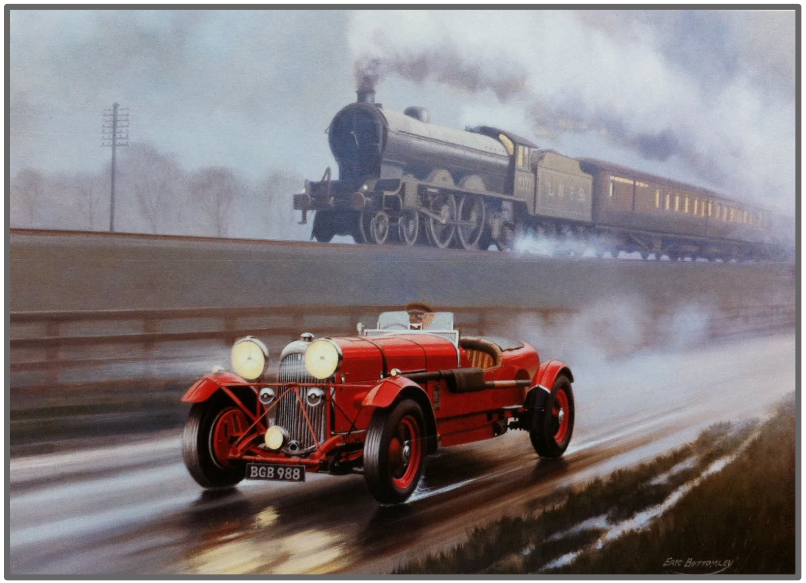
Table of Contents

What is Machine Learning?

Machine Learning (ML) is a branch of **Artificial Intelligence** (AI) where computers are applied to execute a task. The key difference between the two is that AIs is explicitly programmed to execute a task. ML algorithms are programmed to “learn” how to execute a task.

But why waste time learning?

ML algorithms aren’t limited by hard-coded processes. It’s like comparing a car and a train; the car can take optimal roads to get from point A to point B while the train is limited to its tracks.



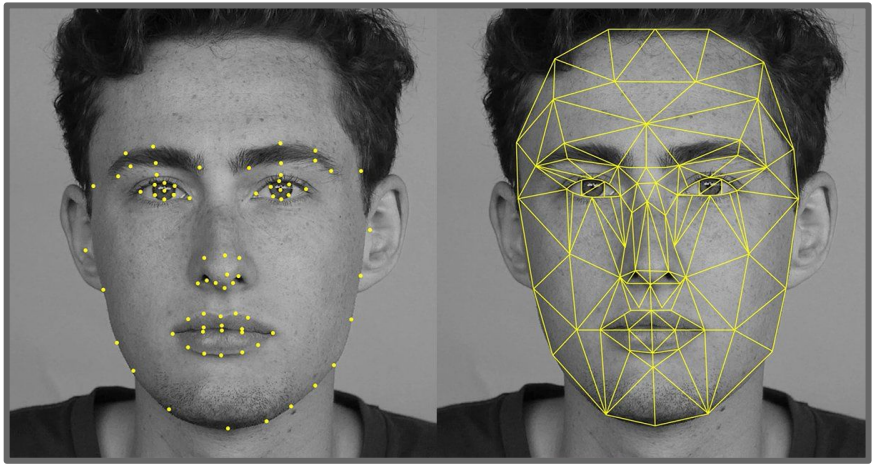
Flavors of Machine Learning: Supervised

Supervised learning algorithms are given input data with corresponding outputs/labels. They attempt to predict outputs/labels given the inputs.

Supervised learning is applied to both regression and classification problems.

Ex. Facial Recognition

You give the algorithm pictures of people and identify where their faces are. The goal of the algorithm is to find their faces.

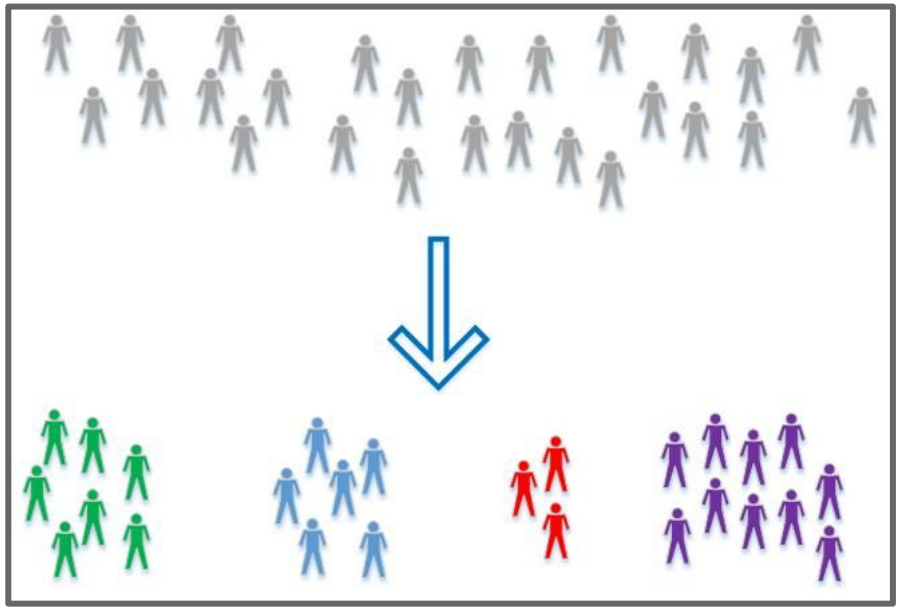


Flavors of Machine Learning: Unsupervised

Unsupervised learning algorithms are given input data without labels. They classify the data into different categories that usually adhere to some hidden structure.

Ex. Photo Sorting

You give an algorithm pictures of four people but you don't know which is which. The goal of the algorithm is to categorize each picture into four categories without labels.



Flavors of Machine Learning: Reinforcement

Reinforcement learning algorithms make actions that are assigned a reward or punishment. The algorithm aims to take actions that maximize its reward.

Ex. ML Chess Agents

You want an algorithm to win a chess game so you give it the chess move set. Moves that lead to a victory are rewarded while moves that lead to a defeat are punished.

Deep Learning

Deep learning algorithms are a subclass of machine learning algorithms that use multiple layers that process, extract, and transform data. Each layer uses the previous layer's outputs as inputs.

Typically, deep learning is used for classifying and predicting data by analyzing patterns.

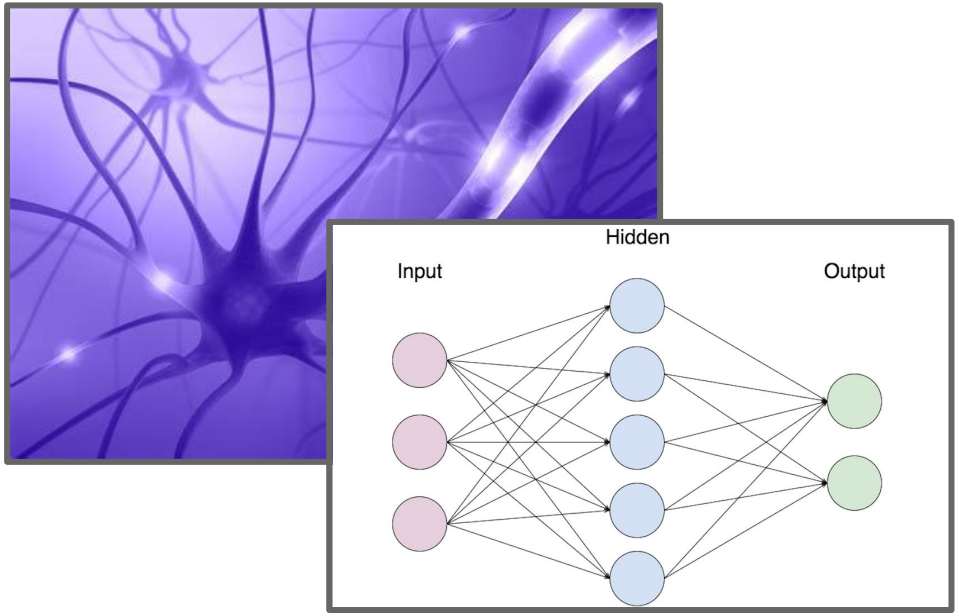
The most well-known form of deep learning algorithms are deep neural networks or Multi Layer Perceptrons (MLPs). However, there are variants to this algorithm that will be covered later like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). For now, we'll focus on MLPs.

Multi Layer Perceptrons: The Classic

Neural Networks/MLPs

A classic neural network model is called a **Multi Layer Perceptron** (MLP). Their name is quite self-descriptive—MLPs consist of multiple layers of perceptrons or neurons that are networked together. Of course, the “neurons” are not biological, but the concept of neural networks are indeed inspired from biology. Much like the neurons in the brain, neurons in a neural network are interconnected and manipulating those neural connections are crucial to a MLP’s ability to learn.

Additionally, MLPs bear structures and functions that carry over to other neural network variants. Thus, mentioning MLPs first is fitting to understand the other variants.



Structure of MLPs: Data Representation

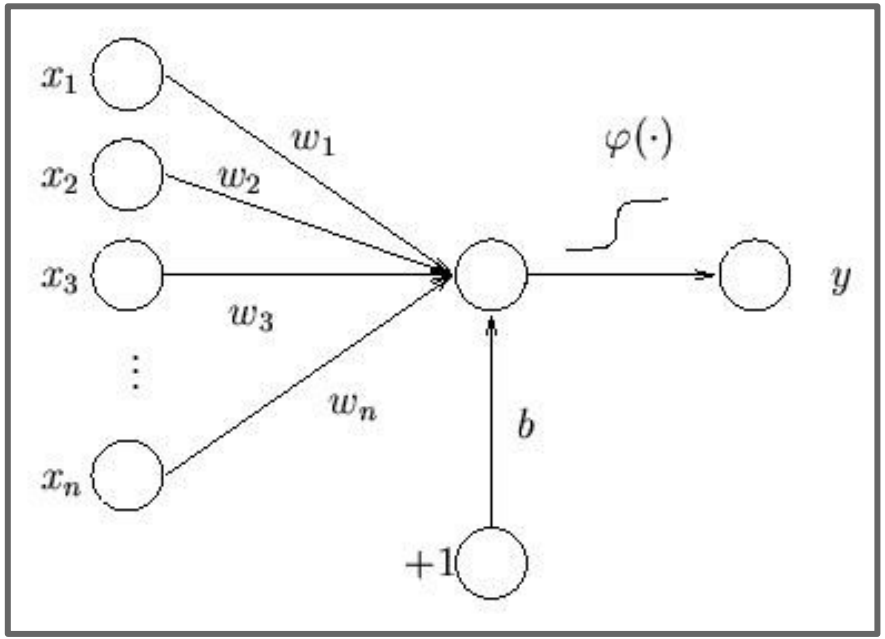
Before we talk about the intricacies of MLP structure, it's important to understand how data is represented.

All neural networks operate on numerical data. Even when pictures and texts are involved, they're broken down into numbers. However, MLPs are optimal for operating on numerical data where no overarching pattern that contributes to the data's meaning.

For instance, sentences with words out of sequence have no meaning. Thus, sequential patterns lend to interpreting information in a sentence. Data that's patternized like this are NOT optimal for MLPs. They prefer lists of unrelated elements.

Structure of MLPs: The Neuron

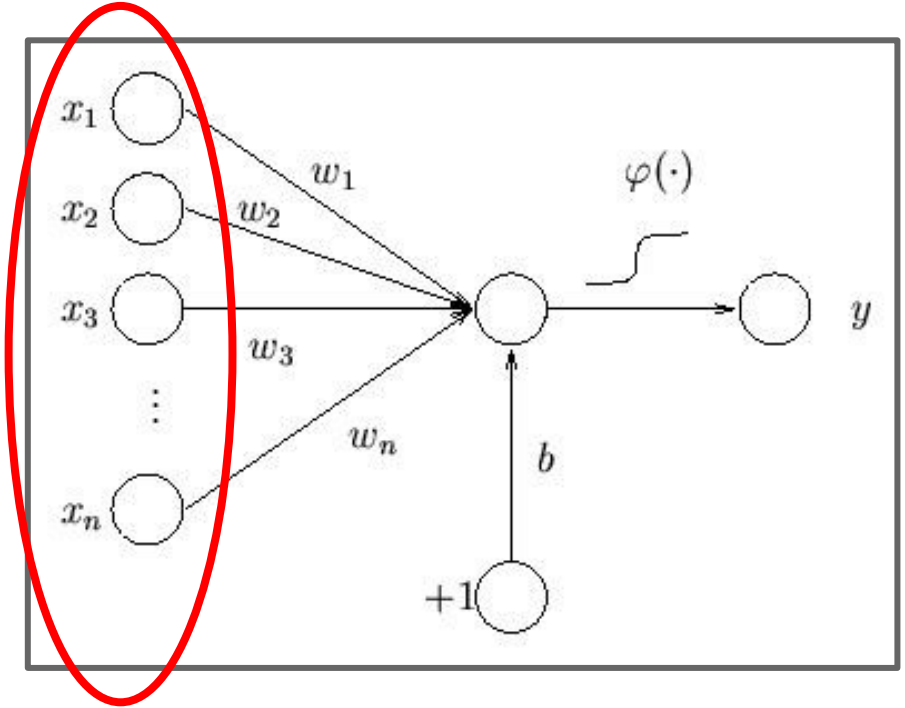
The **neuron**, also known as the **perceptron**, is the functional unit of an MLP and any neural network. On its own, the neuron is rather unremarkable. But in layers, the calculations executed by the neurons can perform operations that are vital to the MLPs success. We will talk about those operations later. For now, we will talk about the structure of the neuron.



The Neuron: Inputs

To accommodate for a vast amount of data that can influence the MLPs performance of a task (like classification), the neuron can take as many inputs as required. And, so the neurons can manipulate the data, the inputs must be numerical.

From a mathematical standpoint, “x” in “y = mx + b” represents the inputs.



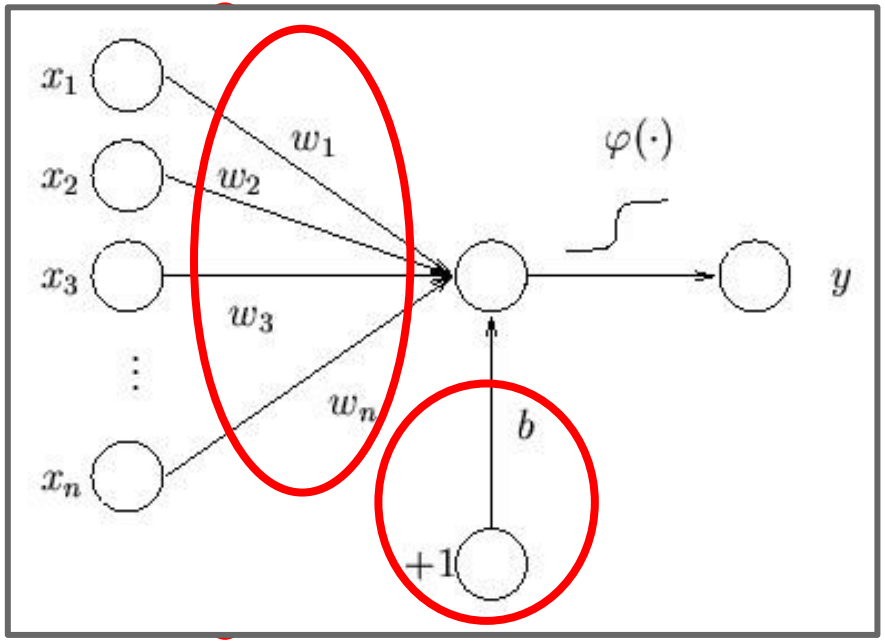
The Neuron: Weights & Biases

Neurons are equipped with weights and biases to manipulate the inputs.

Weights multiply the inputs; they determine how much an individual input is valued in accomplishing the MLP's task.

Biases add to the inputs; they are the MLP's predetermined value of a certain element in accomplishing the MLP's task.

Mathematically, weights are "m" and biases are "b" in "y = mx + b".

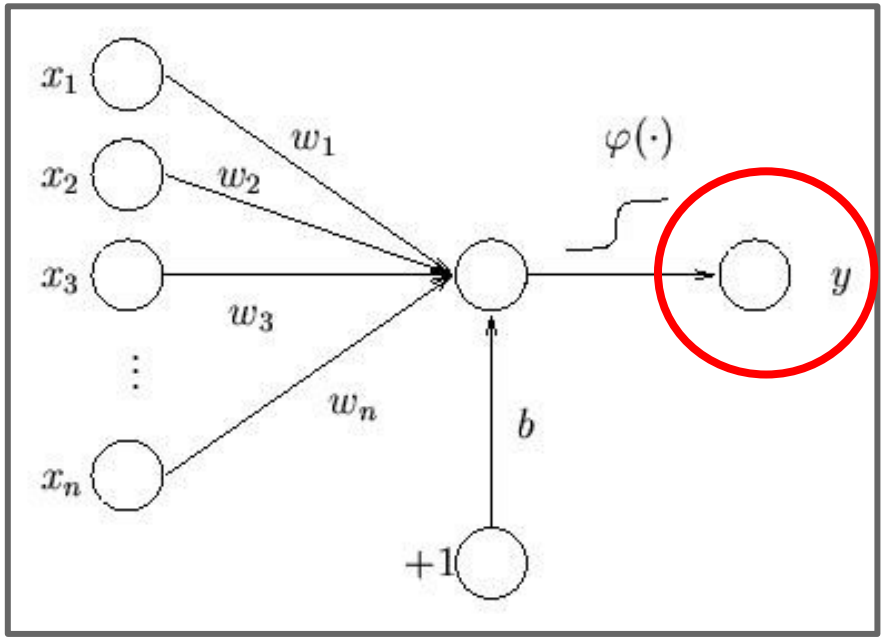


The Neuron: Outputs

Unlike the inputs, neurons only produce one output which lets the neural network funnel data to make a simple decision.

For example, let's say an image classifier must categorize an image into ten categories of different animals. That classified will have to take in all of the image's pixels as inputs and narrow it down to ten outputs at the end of the neural network which correspond to the ten categories.

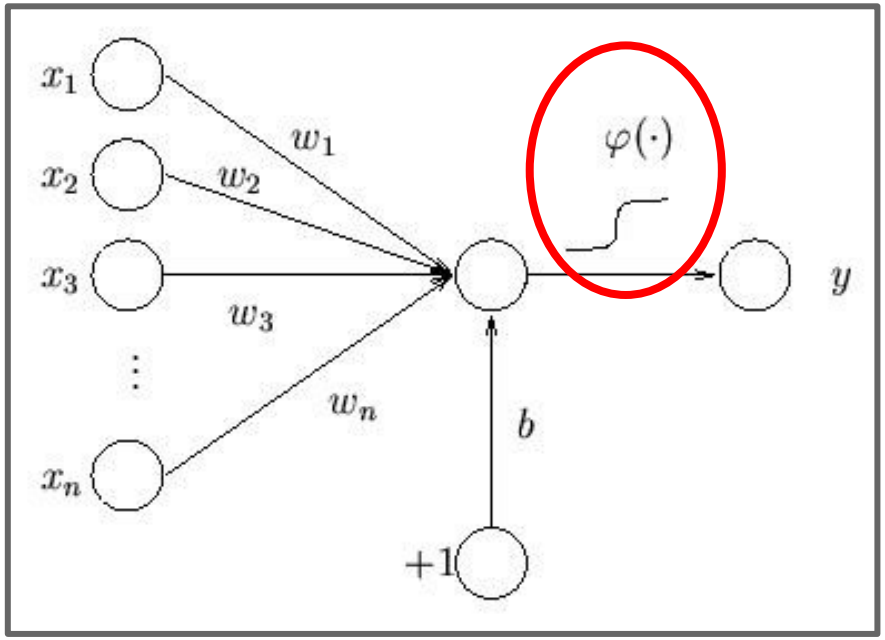
Mathematically, outputs are the "y" in "y = mx + b".



The Neuron: Activation Functions

The neuron's **activation function** is responsible for using all the inputs, weights, and biases to calculate one output. For the past three slides, I've used a linear activation function to represent the other elements in a neuron, but there are a lot of more complex activation functions used for equally complex tasks.

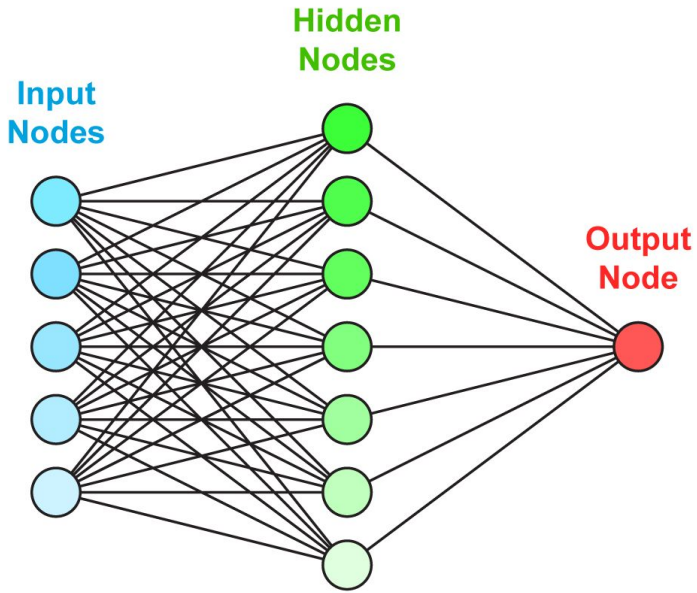
For example, the sigmoid function is often used for binary cross entropy while the softmax function is used for categorical cross entropy. We'll get into those later.



Structure of MLPs: Layers

Layers are sets of neurons in parallel. By grouping neurons together, they can perform operations individual neurons would be incapable of—like the image classifying example.

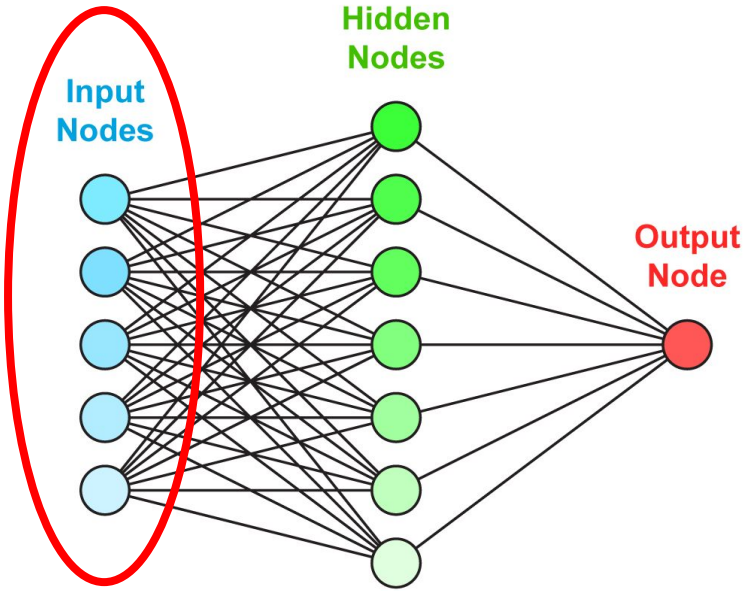
Inside of a layer, neurons typically share some similarities: they take the same inputs, possess the same activation functions, and work toward a similar goal. However, neurons are tuned individually in a process called "backpropagation"—another topic we'll discuss later. Just keep in mind that layers are permanent while neurons are dynamic as the program runs.



Layers: Input Layer

The **input layer** is the only required layer that isn't really a "layer" since it's not made of neurons. The input layer represents the inputs that are going into the network and how they are distributed.

When the data is passed to the network, every neuron in the next layer gets all of the input data from the input layer. This way of "feeding forward" data continues between layers: the outputs of each neuron in the previous layer are the inputs of each neuron in the next layer. But since the input layer isn't made of neurons, no data manipulation happens when passing the input data from the input layer into the hidden layers.

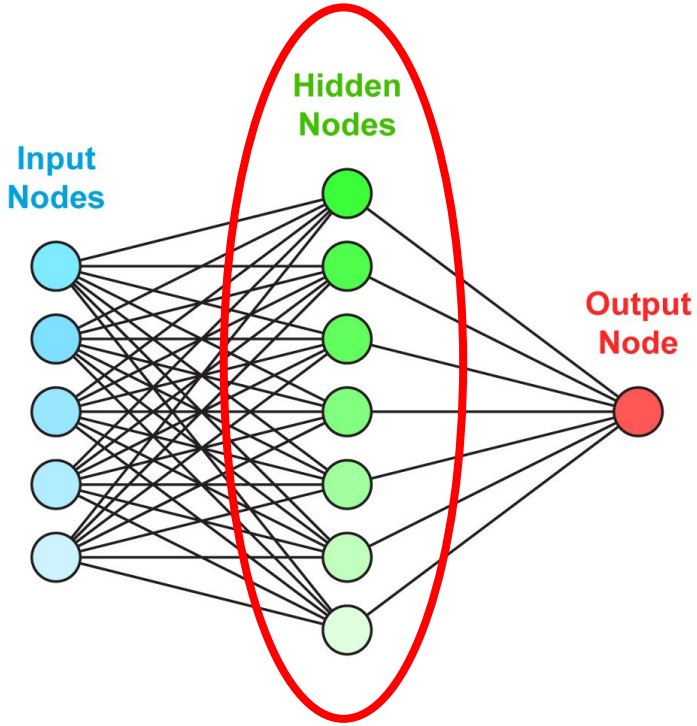


Layers: Hidden Layers

Hidden layers are responsible for conducting the majority of operations on data—passing all of the inputs through neural activation functions to produce outputs for the next layer. Common layers in basics MLPs are the “dense” and “dropout” layers.

Dense layers are normal layer of normal neurons. Their main function is to manipulate the data with their activation functions.

Dropout layers take a randomly selected percentage of neurons in a network and deactivates them. This helps the network learn more by forcing it to use other neurons to learn. Dropout layers aren’t made of neurons.

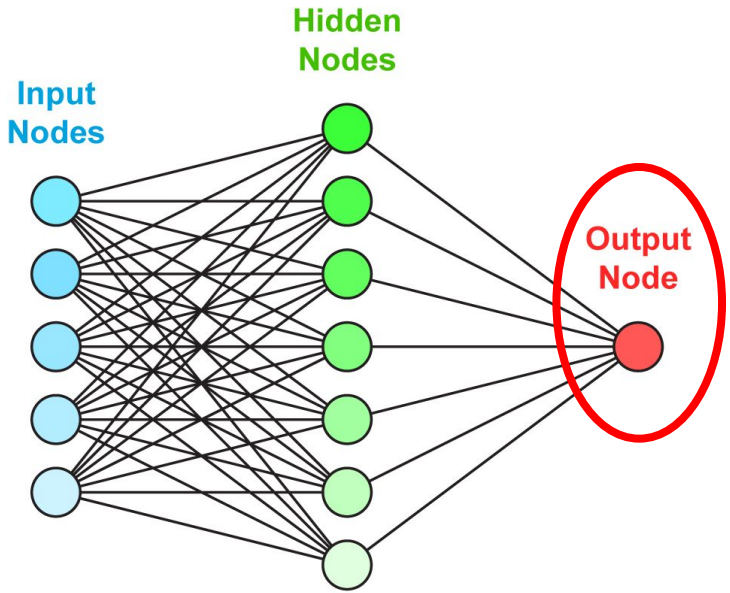


Layers: Output Layer

The **output layer** is the final hidden layer that gives the neural network's final output(s). They are typically the smallest layer in the neural network since all of the data has been condensed to determine these last output(s). Output layers are typically "dense" layers with the sigmoid or softmax activation functions.

Sigmoid dense layers have one neuron that classify data into two categories. This is called binary cross entropy.

Softmax dense layers have numerous neurons equal to the number of categories to classify data. This is called categorical cross entropy.



Operation of MLPs

Now that we know how MLPs and basic neural networks are structure, but how do they learn? The answer lies in backpropagation; a complex set of equations that fine-tune a network to its task. However, before we discuss backpropagation, we'll need to understand the processes of training, testing, and all the elements that go with it. Additionally, these operations apply to all neural networks.

Operation of MLPs: Training

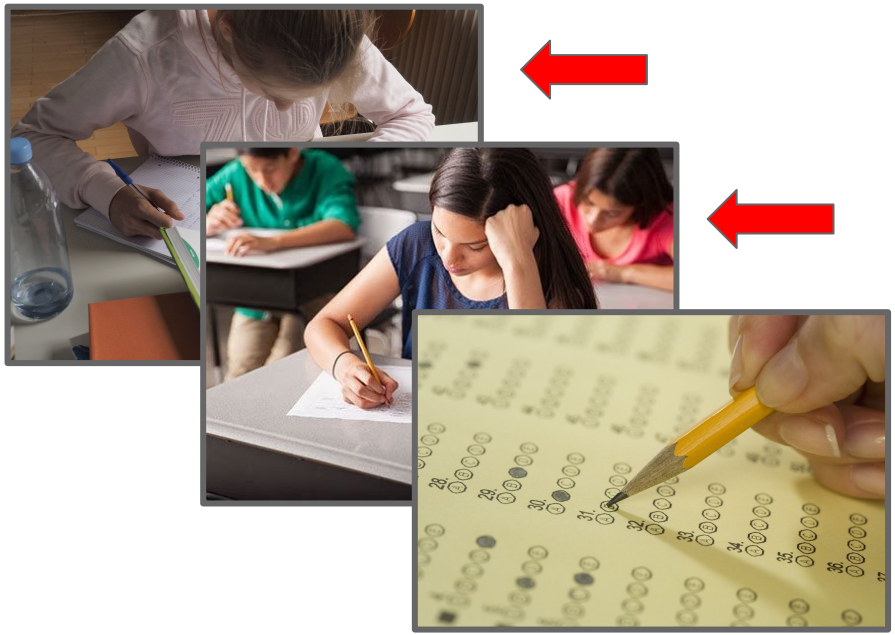
Training is the only period when backpropagation happens. The goal of training is to create a model fit to operate with real-world data by first letting it run on practice data. It's exactly like training a person: athletes need to practice before a game or match where the stakes are raised.

Training: Training Data

Training data is a proportion of data separated from the full data set or corpus. Training data, as the name implies, is data fed into the network during training. Out of all the types of data, training data is the largest proportion of data out of the corpus.

Validation data is a smaller proportion of data that is set aside to conduct miniature tests during the network's training process. Sometimes, however, the term "validation data" is swapped for "testing data" which will be explained later.

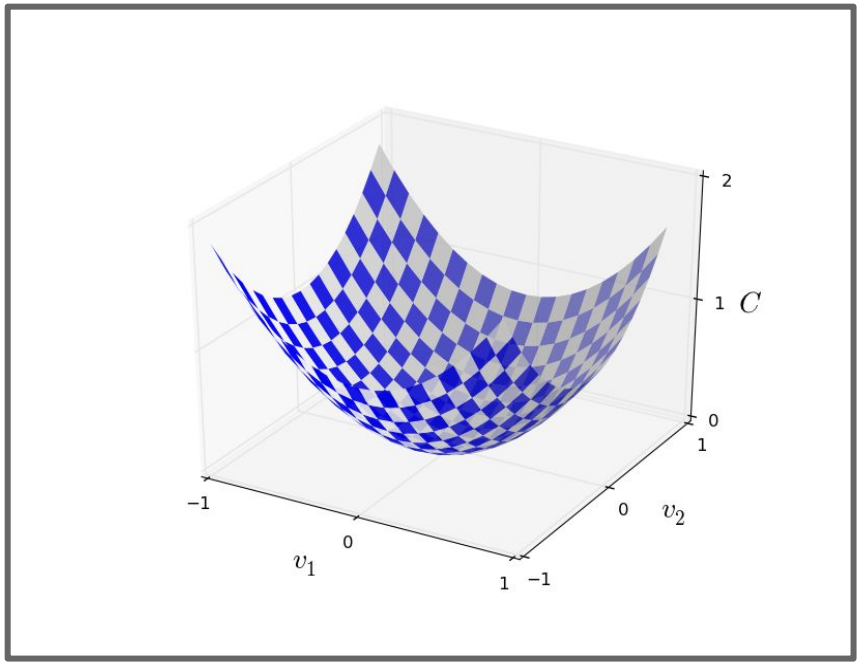
Metaphorically, training data is like homework, validation data are like quizzes, and testing data are like tests.



Training: Loss

Once the training data is fed through the entire network, the outputs of the neural network are compared to the desired outputs of the neural network. This is where loss functions come into play. **Loss functions** mathematically measures **loss** by comparing the network's experimental values with the actual values. Loss is then fed into backpropagation.

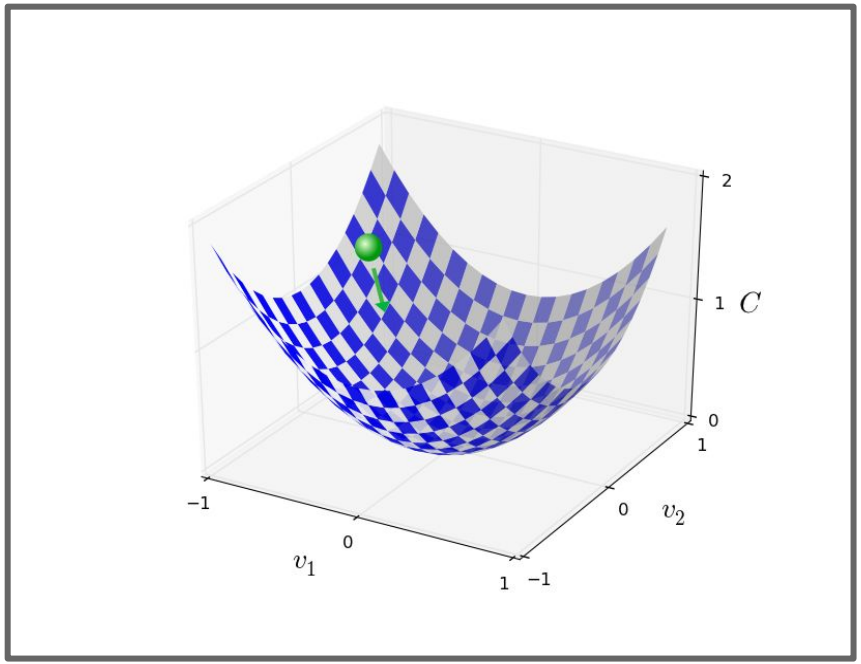
Like activation functions, there are multiple loss functions. Binary cross entropy has a different loss function from categorical cross entropy because their outputs are different and must be interpreted differently.



Here is a graph of loss given two variables.

Training: Backpropagation

Backpropagation takes the loss value and edits the neural network's by changing its weights. The goal of backpropagation is to reduce the loss value as much as possible. So when backpropagation occurs after the next set of data is fed through the network, the loss value will be less than the previous loss value. Thus, as the loss decreases, the network's experimental output values will become more and more similar to the actual values; the network becomes more accurate/proficient at its task. This involves a fair bit of multivariable calculus that won't be discussed here.



The ball rolls down the graph to find a lower loss value. Backpropagation works like this.

Training: Batches & Epochs

Rather than backpropagate after every training example, a network can backpropagate after **batches**, or groups of training examples, to save time. When the network runs through a batch, it's errors are saved up and loss is calculated all at once. Once it's run through the entire batch, backpropagation happens only once.

Epochs are when the network runs through the entire training set once. A network will often run through the training set multiple times, but if it runs to long, it risks overfitting—a topic that we'll get to in testing.

The dough is like an epoch, it's the blend that has all the cookies. The batch is like the tray, it's how many cookies you can make from the blend at a time.

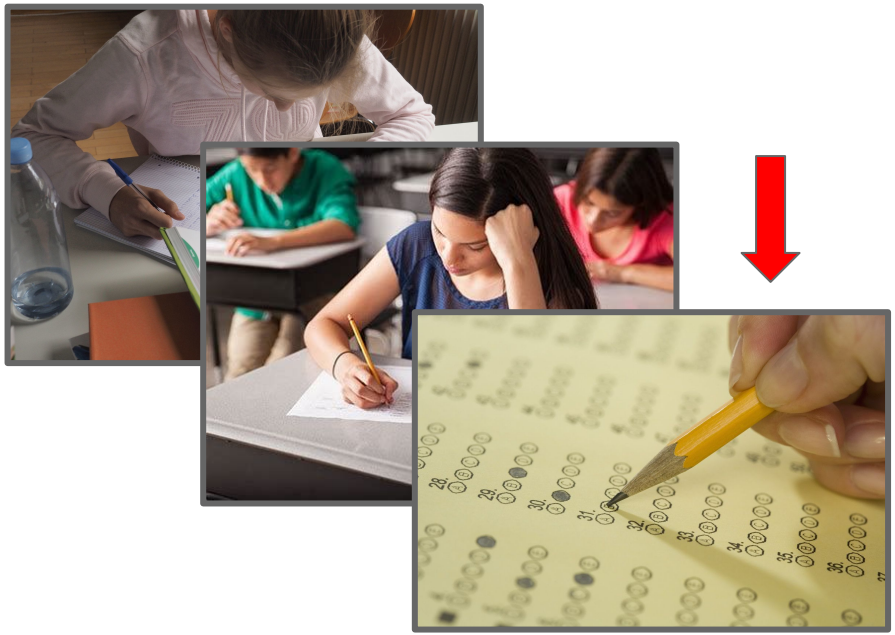
Operation of MLPs: Testing

Testing exists to verify that the neural network is ready to be applied to real-world data; it's like a student's final exam before graduating to the next grade. In this period, backpropagation does not occur and the network relies solely on what it has learned thus far.

Testing: Testing Data

Testing data, as opposed to training data, is used during testing, makes up a smaller proportion of the corpus, and doesn't happen alongside backpropagation. Thus, while a network trains, testing data is reserved until the network is fully fit. However, it is fed through the network exactly like training data.

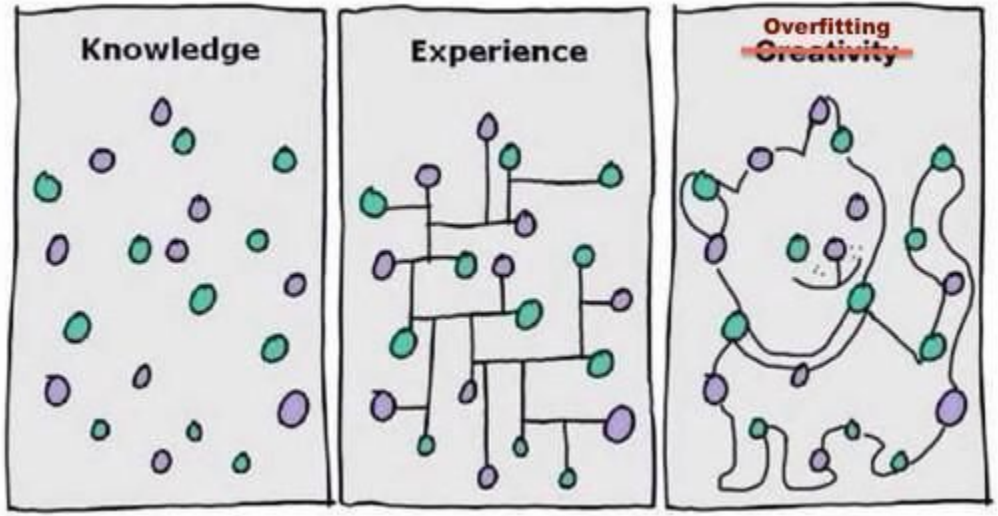
Again, "validation data" and "testing data" may be mistaken for one another.



Testing: Overfitting

Overfitting is a risk of training a network too long where the network becomes so familiar with the training data that it begins to overanalyze the training data outliers that don't apply to every data set. Thus, the network loses generality and is no longer applicable to other data.

Overfitting is more difficult to verify during training, especially without validation data. With testing, identifying an overfitted network is simple because network's classifications will be much less accurate in the test data than in the training data.



Operation of MLPs: Optimization

We've determined how a neural network learns, but there are limits to a neural network's potential that deal with variables only the programmer can alter: **hyperparameters**. The process of fine-tuning a network's hyperparameters to unlock more of its potential is called **hyperparameter optimization**. This process, however, is tedious. There are a vast number of hyperparameters to edit even in the simplest networks: layer size, activation functions, loss functions, learning rate decay, gradient decay, *etc*. There are automated ways to do this, but that's beyond the scope of this kiosk.

Operation of MLPs: Application

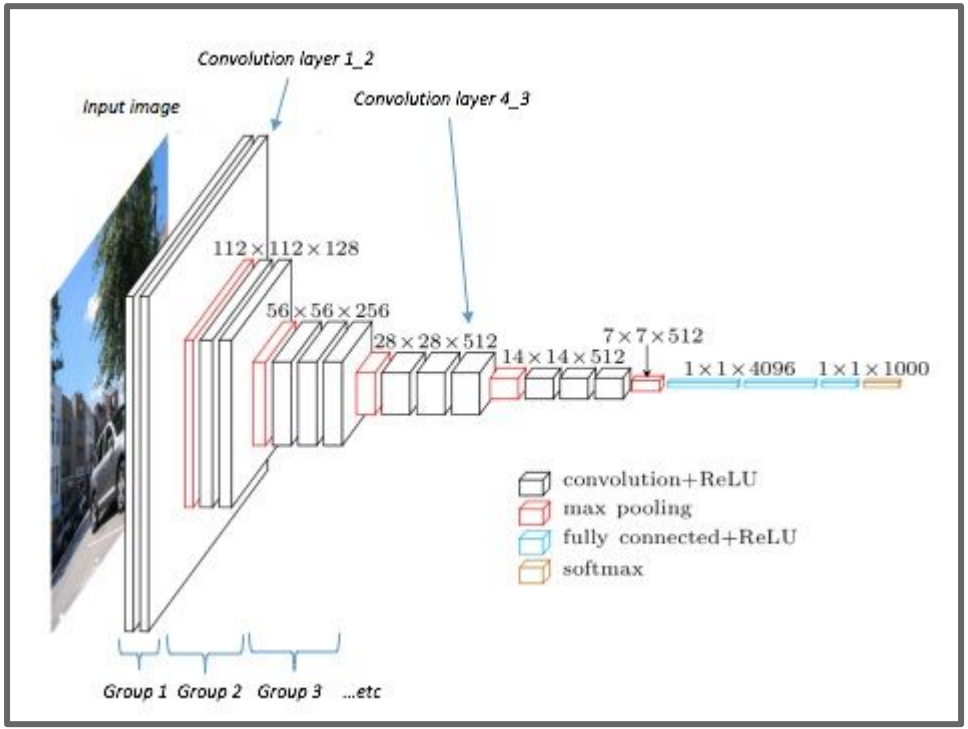
MLPs are good generalists. Like a lot of neural networks, they're great at classification problems, but again, the complexity of the problem depends on the data that's being classified (images, texts, audio). They excel in crunching numerical data but lack in their ability to solve those more complex problems with patternized data. At some point, a specialist network is required which are less versatile but adept at working specific problems.

This is a section of data from the MNIST handwritten number data set. These images have a very low resolution so MLPs can classify these numbers adequately.

Convolutional Neural Networks: Pixel Perfect

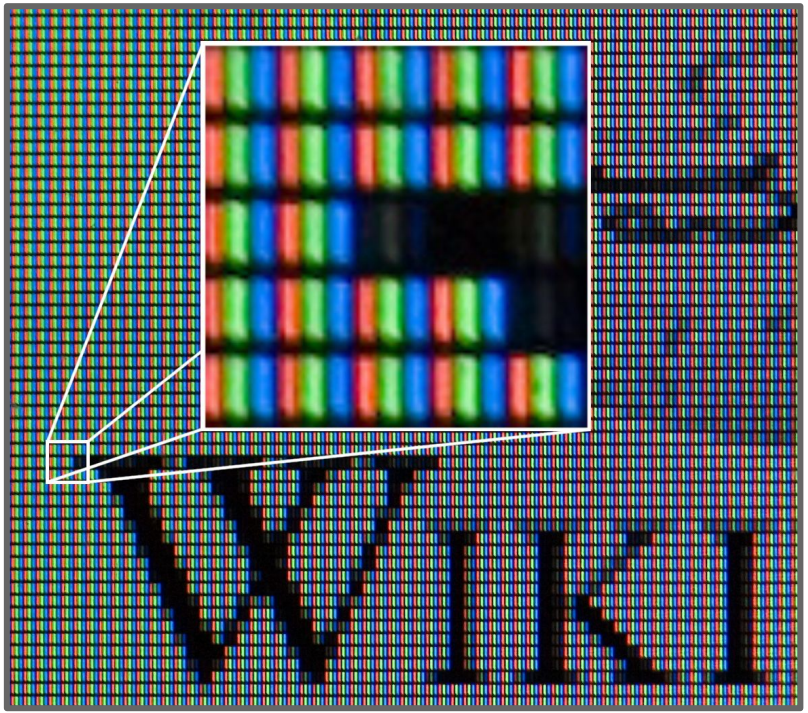
Convolutional Neural Networks (CNNs)

CNNs are networks that specialize in processing images. Before continuing, it's important to note that CNNs, MLPs, and many other types of neural networks bear many resemblances. They all train; they all backpropagate; they're all tested; at least, from an operational standpoint, all neural networks are similar because they all rely on the magic of backpropagation. Specialist networks branch off from MLPs in their structure and data; in order to process images, CNNs are equipped with special tools that MLPs and other networks don't have.



CNNs: Data Representation

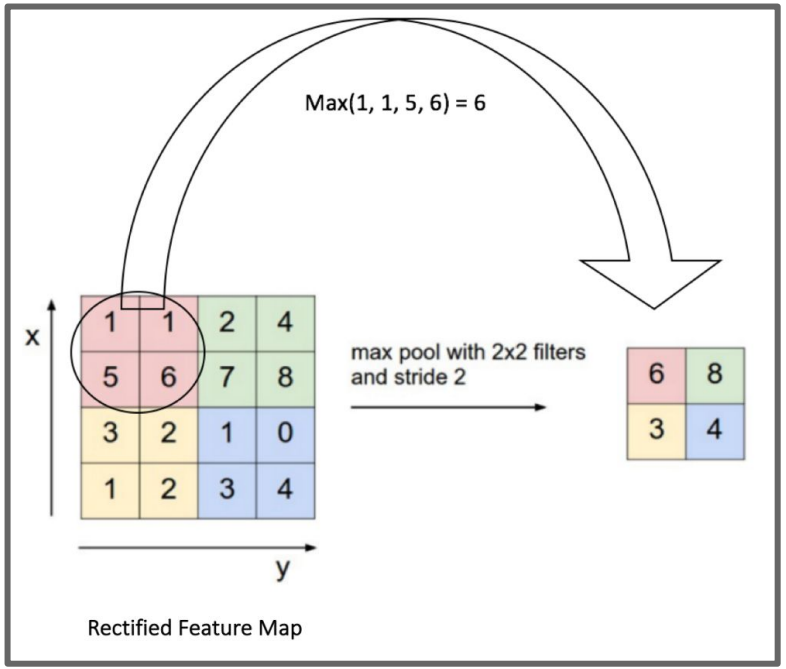
CNN's operate on images or data with spatial patterns. But, as stated before, all data must be numerical. So how are images broken down into numbers? Well, every digital image is represented on a monitor and occupies a specific number of pixels in a two dimensional space. Thus, the pixels are elements in a two dimensional array or grid. Each pixel in a colored image also contains a color that's represented by RGB (red green blue) values which are numbers from 0 to 255. Now we have a third dimension: channels of color. In conclusion, a colored image is a three dimensional array where each element is a pixel that contains a number from 0 to 255. In a black and white image, though, there is only one channel.



CNNs: Pooling Layers

Pooling layers are found in CNNs that are responsible for condensing data. Like dropout layers, pooling layers aren't "layers" because they're not made of neurons, but they manipulate the data passing into the next layer. For images, pooling uses a **kernel**, a small two dimensional grid, to scan the image left right up and down. Everywhere the kernel stops, a value is generated that condenses the information of the pixels within the grid to one value which effectively creates a feature map of an image that represents an image's most prominent features.

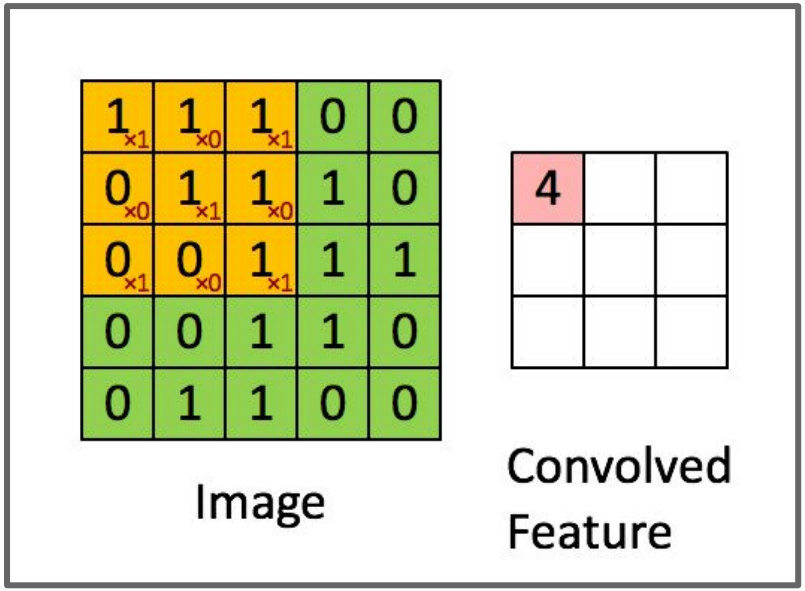
Varied pooling layers exist: max pooling passes the largest value in a kernel; average pooling passes the average of values in a kernel. Max is more common.



CNNs: Convolution Layers

Convolution layers are akin to pooling layers, but they have neurons. A kernel slides across the image to scan it, but instead of pooling the image, the values in the kernel are passed through a neuron that manipulates the data to produce a single output. The data can still be arranged in two dimensions for the kernel, but convolution will make the image unrecognizable. Outputs still refer to a two-dimensional grid, but not the original image.

Convolutional layers are much different from dense layers where every output of the previous layer is passed into every neuron of the next. In this case, only a select group of outputs, the outputs in the kernel, are passed into the neuron of the next layer.

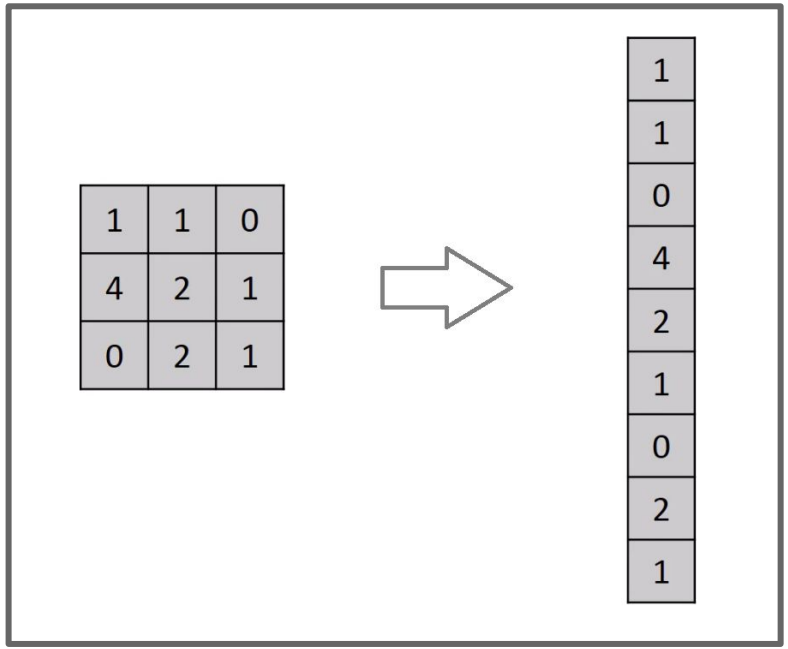


The yellow square is the kernel and it slides along the image to find features. In this case, it's looking for an "x" shape as seen by the multipliers. The sum of its findings are then placed in the feature map.

CNNs: Flatten Layer

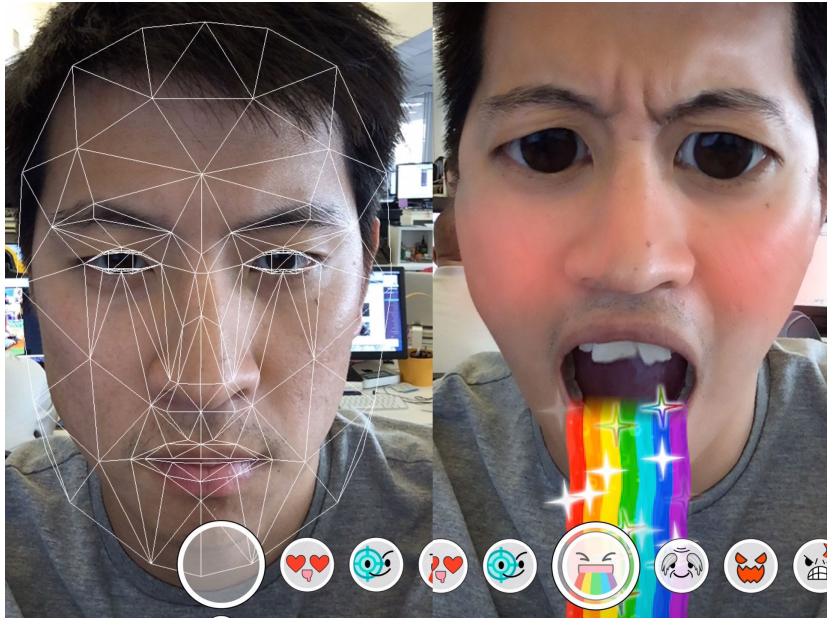
Pooling and convolution happen at the start of a CNN and almost always operate one after another. Eventually, the network makes a smaller image than the original that is simple enough to be interpreted by dense layers. After all, a CNN must end in a dense layer if it is classifying the image. Before then, though, the image must be flattened. A **flatten layer** is another purely operational layer that doesn't use neurons. It's responsible for flattening the image's channels into one one-dimensional array.

This especially crucial in colored images because convolution and pooling can occur multiple times per channel which effectively multiply the number of outputs the network passes layer to layer.



CNNs: Application

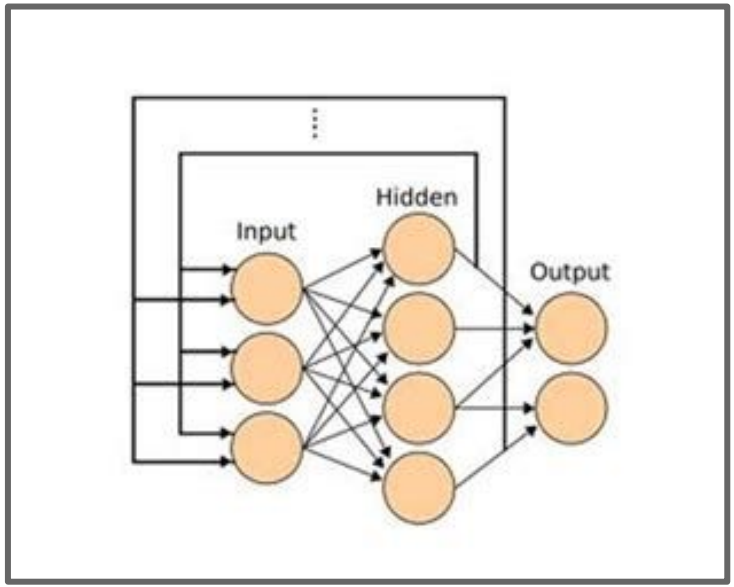
Because CNNs specialize in image classification, they can be used to create facial recognition algorithms used by SnapChat or Apple, algorithms that use images of a patient to diagnose them, or algorithms that identify objects in videos. Indeed, a lot of our decisions are made with the help of visual stimuli, so even though CNNs are specialized for tackling visual data, they are still a neural network with plenty of applications.



Recurrent Neural Networks: Past, Present, & Prediction

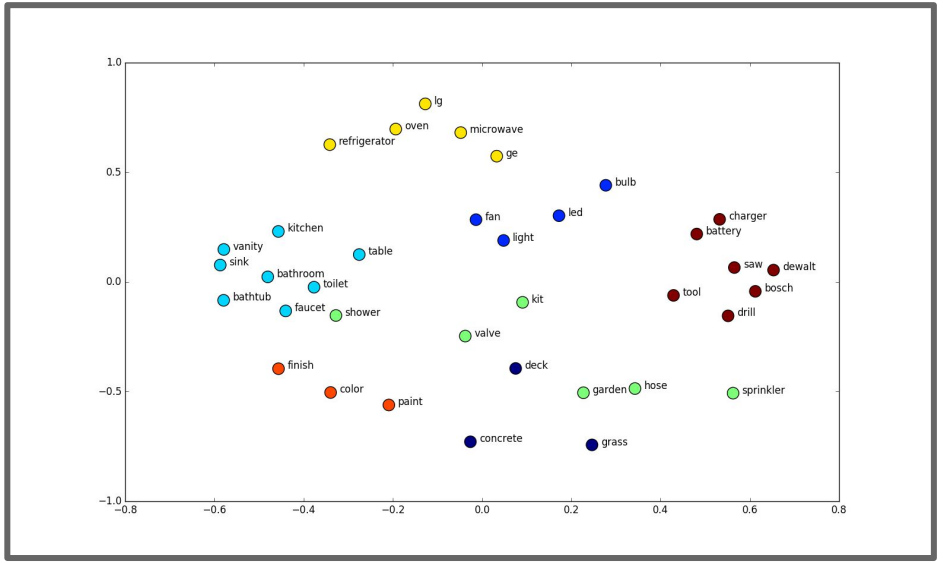
Recurrent Neural Networks (RNNs)

RNNs operate on data that have sequential patterns. Like CNNs and MLPs, they share the same operations regarding backpropagation, training, *etc.*, but there's one key difference. MLPs and CNNs are feedforward networks because data only moves in one direction: input to output. RNNs move data forward, but once inputs become outputs, they're fed back into the RNN. This makes the network more susceptible to directly making decisions based on its past experiences. Normal MLPs can't do this, so like CNNs, RNNs require some special tools.



RNNs: Data Representation

RNNs operate on data with sequential patterns: sentences, videos, trends, *etc*. Trends are numbers in series and videos are a series of images, but how are sentences broken down into numbers? An RNN can't understand words, but it can plot a word's contextual meaning on a multidimensional vector. For instance, the network can slowly learn that some words are adjectives, nouns, and verbs because of where they occur in the sentence. The network, though, will never know a word's deeper meaning like connotation, sarcasm, irony, *etc*.

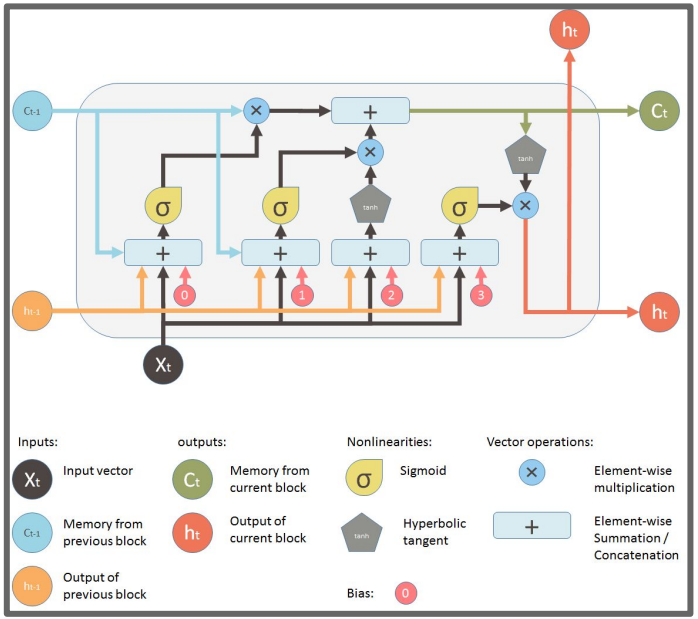


Above is an example chart of how hardware can be embedded based off of contextual meaning.

RNNs: LSTM Units

Long-Short Term Memory Units (LSTMs) are modified neurons that RNNs use as operational units. Unlike a classic neuron, they have what's called a forget gate which determines what past data the RNN needs to remember and what it needs to forget.

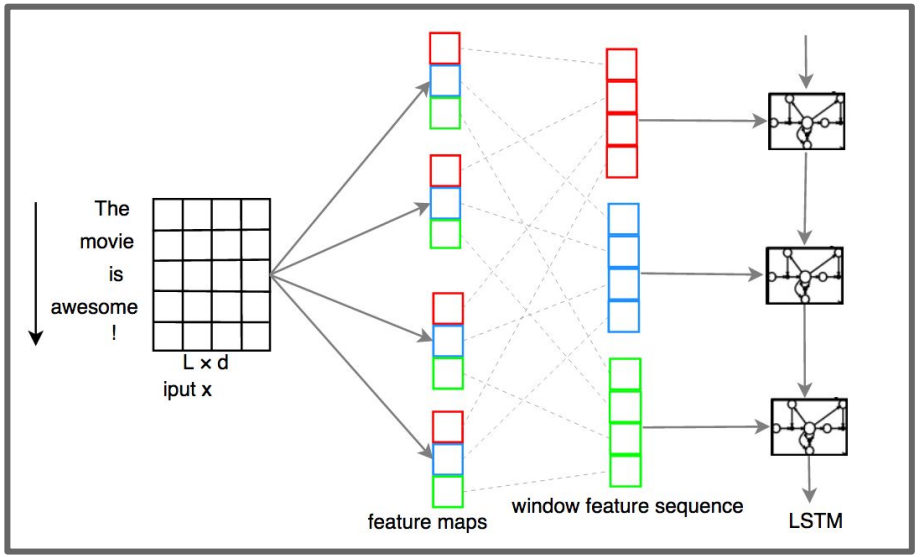
This solves a dilemma known as the **vanishing gradient** which happens when a network without LSTMs accumulates too much past data to feed back into itself. Since all the past data are multiplied by weights that are between 0 and 1 every time the data is fed back, the outputs of the RNN eventually approach a limit of zero. By forgetting select examples, LSTMs can avoid this dilemma.



RNNs: Embedding Layer

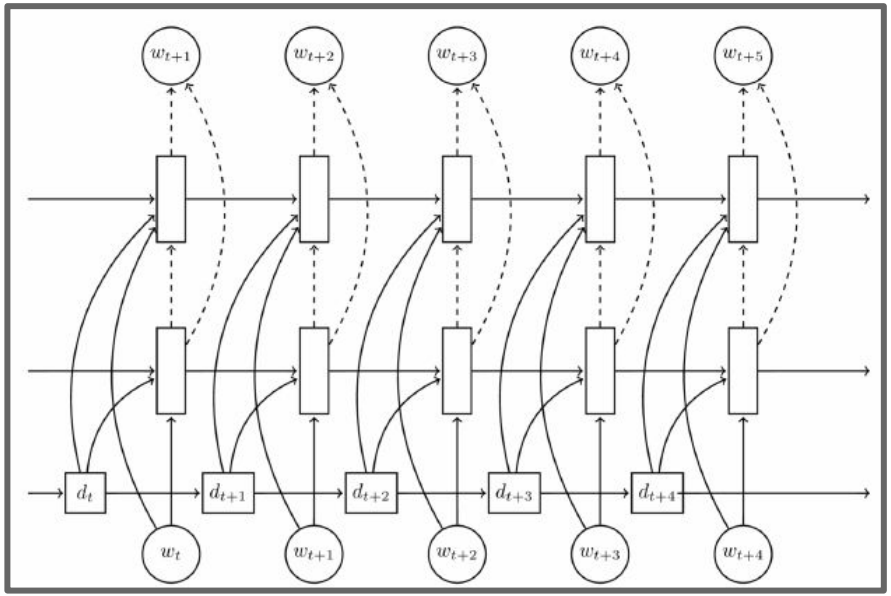
The **embedding layer** is a layer in networks that analyzes words, so not all RNNs require an embedding layer. Nonetheless, this layer is responsible for taking raw words and placing them in a multidimensional vector based on their contextual meaning.

Again, not all RNNs require an embedding layer. In fact, most RNNs process time-based data, but it's important to note that RNNs operate on data in sequence and are not exclusively limited to time-based data.



RNNs: LSTM Layers

LSTM layers are layers made of LSTMs rather than normal neurons. They usually make up the majority of hidden layers in an RNN until the very end when a classification or prediction must be made. Unlike the convolutional and pooling layers in a CNN, the operation of an LSTM layer doesn't require two distinct layers. Thus, they can be stacked directly on top of each other and can placed in more locations—similar to dense layers in an MLP.



An "unrolled" LSTM layer that shows inputs and connections into an LSTM layer as time proceeds.

RNNs: Application

RNNs are often applied to predicting trends in the stock market, identifying actions in videos or images, detecting insurance fraud, and voice recognition or natural language processing.

It’s important to note that MLPs, RNNs, and CNNs, can be mixed together in order to apply neural networks to more complex problems. For instance, identifying actions in videos and images require processes of both a CNN and RNN. Likewise, the flavors of machine learning can be mixed together. For instance, Elon Musk’s OpenAI plays Dota II, a video game, to near perfection. That would require elements of a CNN, RNN, and reinforcement learning.

Conclusion

And those are the main three neural networks! If you want to read more about these networks, tamper with one, or perhaps create on yourself, I recommend finding resources online. The deep learning library Keras for the coding language Python is a fantastic place to start, and you can find my reference to interpreting Keras for MLPs, CNNs, and RNNs.