Deep Learning and CNNs

Andrés Ponce

June 10, 2020

Deep Learning methods allow us to perform other functions such as object recognition obtained from image datasets.

Conventional Approach to Object Recognition

The key steps in object recognition involve **feature extraction** from several images. However, features may look different when recognizing different types of objects.

We could have a mdoel with **hand-crafted** feature detection. Here, we could define by hand the kind of features we would like to see in our model. After we go through the feature extractor we could have a simple classifier.

Another approach is instead to have a **trainable feature extractor**, along with a trainable classifier.

Neural Networks

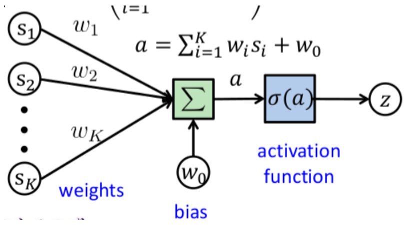


Figure 1: In neural networks, we have a set **s** of input values, which are then summed up through the different layers. Afterwards, we pass it through an **activation function** $\sigma(a)$, which then leads to the class we predict it into.

If we recall, **neural networks** are a series of layers, composed of individual **neurons** and which connect and communicate to neurons the layer before or after it.

Training the neural network consists in trying to minimzize the loss function. For classification, the loss function might be the distance between the correct classification and our predicted values. The formula might look something like:

$$L(\theta) = \sum_{i=1}^{N} \|\mathbf{y}_i - g_w(\mathbf{x_i})\|^2$$

To minimize it, we use the technique called **gradient descent**. This technique involves finding the values for which the **derivative** of the loss function decreases. Even within gradient descent, there are some different types we might wish to use. For example, **steepest descent** uses batch processing to change the parameter vector **w**. [1]

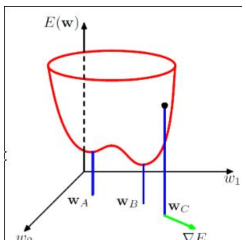
[1] Remember batch methods operate on the whole training set to modify weights.

Another method for gradient descent involves selecting a subset of the dataset to use when measuring error. This is called **stochastic** gradient descent. Here we use a formula such as

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2$$

also known as **sum of squares**. [2]

[2] Sum of squares errors are more useful for problems resembling regression.



Calculating the error at every single node each iteration may be too intensive to repeat at every step, so insted we can calculate the error via an algorithm called **backpropagation**. The backpropagation algorithm serves us to calculate the error in the nodes. Remember the value of the activaiton function is a function of the weights of the layers incoming, so something like:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

There are a couple of steps for calculating the error:

1. Calculate the error for each of the k classes. This is done for each output layer.
$$\delta_k = y_k - t_k$$

2. Calculate the error for the **hidden layers**. We use the formula
$$\delta_j = h'(a_j) \sum_k w_{kj}^{(2)} \delta_k$$
Remember $h(a)$ is the function which further modifies the activation function a of each neuron in the hidden layer. [3]

[3] Still have some steps missing.

What is a Deep Neural Network?

Essentially it is a neural network with many hidden layers. [4]

[4] Deep, get it?

We have multiple different possible architectures for neural networks.

For object recognition such as pictured above, we can model an image just as a grid of pixel values. Then we can use the values as input to the neural network. However, for a *nxm* image, we would need to have $(nxm)^l$ total connetions, where l is the number of layers in our network Thus we can use **Convolutional Neural Networks** for help with this.

Convolutional Neural Networks

A convolutional neural network is a multi-layer neural network with **local connectivity** and **weight-sharing**.

Local Connectivity

The main idea of a CNN is that a normal neural network does not scale that well when applied to larger and larger inputs such as images. So we instead map a *region* of the input image to a given neuron in a convolution layer. This is due to the fact that a given pixel in an image will likely be of use in its local region rather than in the entire image. Thus having a fully connected network would most likely have a lot of overfitting or simply unimportant weights. This would reduce the number of parameters we need in our model.

Once we take a section of the image as a neuron, we have what's called a **convolutional layer**.

Weight Sharing

We can usually calculate the weight over several layers of information.

We have a couple aspects we want to focus on:

- **Input layer**: The input layer contains n rows and m columns. The input also contains **depth** information, maybe like pixel color values.
- **Convolutional Layer**: We compute the dot product between the image values and the weights in the connected region.
- **Pool Layer**: The pool layer performs downsampling along some of the dimensions of the input.
- **Fully Connected Layer**: The final layer which is used to determine the class output. Similar to a regular NN, every node in layer j is connected to every node in the previous layer.

Once we take the image and pass it through the convolutional layer, we can pass it through several **filters**. These filters can apply a different weight to particular portions of the image. [5] A convolutional layer will then perform convolution and then pass the output through an activation function. For a map of size nxn, and a filter of size fxf, we will have a resulting matrix of size $n - f + 1$.

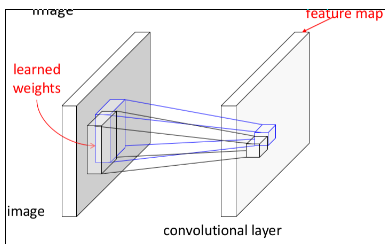


Figure 3: Weight sharing among various small regions of the input image.

[5] A **convolution** is a type of function that describes how two different functions are to be applied.