CS231n Convolutional Neural Networks for Visual Recognition

Table of Contents:

Quick intro

It is possible to introduce neural networks without appealing to brain analogies. In the section on linear classification we computed scores for different visual categories given the image using the formula $s = Wx$, where W was a matrix and x was an input column vector containing all pixel data of the image. In the case of CIFAR-10, x is a [3072x1] column vector, and W is a [10x3072] matrix, so that the output scores is a vector of 10 class scores.

An example neural network would instead compute $s = W_2 \max(0, W_1 x)$. Here, W_1 could be, for example, a [100x3072] matrix transforming the image into a 100-dimensional intermediate vector. The function $max(0, -)$ is a non-linearity that is applied elementwise. There are several choices we could make for the non-linearity (which we'll study below), but this one is a common choice and simply thresholds all activations that are below zero to zero. Finally, the matrix W_2 would then be of size [10x100], so that we again get 10 numbers out that we interpret as the class scores. Notice that the non-linearity is critical computationally - if we left it out, the two matrices could be collapsed to a single matrix, and therefore the predicted class scores would again be a linear function of the input. The non-linearity is where we get the *wiggle*. The parameters W_2, W_1 are learned with stochastic gradient descent, and their gradients are derived with chain rule (and computed with backpropagation).

A three-layer neural network could analogously look like $s = W_3 \max(0, W_2 \max(0, W_1 x))$, where all of W_3, W_2, W_1 are parameters to be learned. The sizes of the intermediate hidden

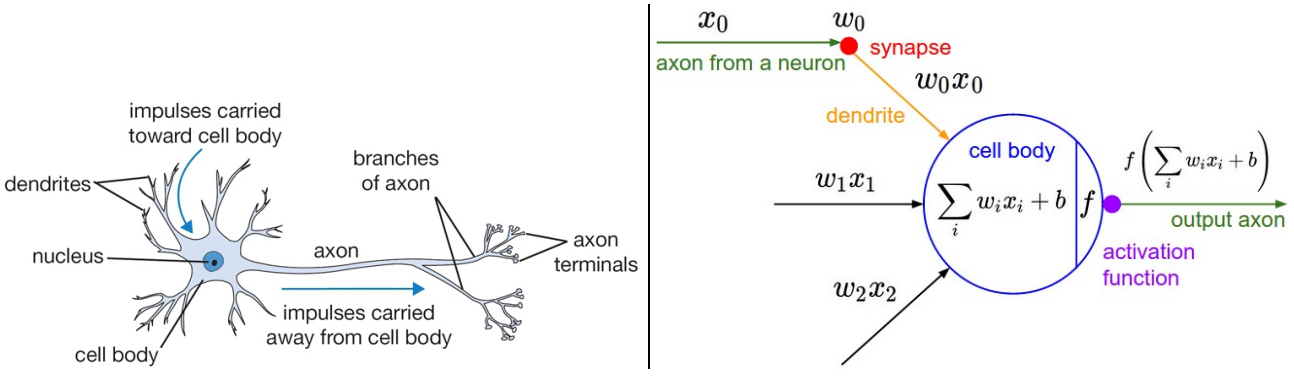
vectors are hyperparameters of the network and we'll see how we can set them later. Lets now look into how we can interpret these computations from the neuron/network perspective.

Modeling one neuron

The area of Neural Networks has originally been primarily inspired by the goal of modeling biological neural systems, but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks. Nonetheless, we begin our discussion with a very brief and high-level description of the biological system that a large portion of this area has been inspired by.

Biological motivation and connections

The basic computational unit of the brain is a **neuron**. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^14 - 10^15 **synapses**. The diagram below shows a cartoon drawing of a biological neuron (left) and a common mathematical model (right). Each neuron receives input signals from its **dendrites** and produces output signals along its (single) **axon**. The axon eventually branches out and connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g. x_0) interact multiplicatively (e.g. $w_0 x_0$) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence (and its direction: excitory (positive weight) or inhibitory (negative weight)) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can *fire*, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this *rate code* interpretation, we model the *firing rate* of the neuron with an **activation function** f, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the **sigmoid function** σ, since it takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1. We will see details of these activation functions later in this section.



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

An example code for forward-propagating a single neuron might look as follows:

```
class Neuron(object):
  # ...
  def forward(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a num
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activ
    return firing_rate
```

In other words, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function), in this case the sigmoid $\sigma(x) = 1/(1 + e^{-x})$. We will go into more details about different activation functions at the end of this section.

Coarse model. It's important to stress that this model of a biological neuron is very coarse: For example, there are many different types of neurons, each with different properties. The dendrites in biological neurons perform complex nonlinear computations. The synapses are not just a single weight, they're a complex non-linear dynamical system. The exact timing of the output spikes in many systems in known to be important, suggesting that the rate code approximation may not hold. Due to all these and many other simplifications, be prepared to hear groaning sounds from anyone with some neuroscience background if you draw analogies between Neural Networks and real brains. See this review (pdf), or more recently this review if you are interested.

Single neuron as a linear classifier

The mathematical form of the model Neuron's forward computation might look familiar to you. As we saw with linear classifiers, a neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space. Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

Binary Softmax classifier. For example, we can interpret $\sigma(\sum_i w_i x_i + b)$ to be the probability of one of the classes $P(y_i = 1 \mid x_i; w)$. The probability of the other class would be $P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$, since they must sum to one. With this interpretation, we can formulate the cross-entropy loss as we have seen in the Linear Classification section, and optimizing it would lead to a binary Softmax classifier (also known as *logistic regression*). Since the sigmoid function is restricted to be between 0-1, the predictions of this classifier are based on whether the output of the neuron is greater than 0.5.

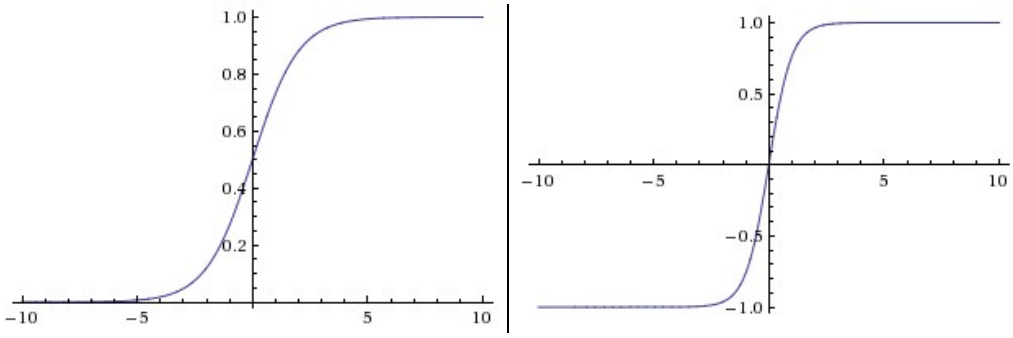
Binary SVM classifier. Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

Regularization interpretation. The regularization loss in both SVM/Softmax cases could in this biological view be interpreted as *gradual forgetting*, since it would have the effect of driving all synaptic weights w towards zero after every parameter update.

> *A single neuron can be used to implement a binary classifier (e.g. binary Softmax or binary SVM classifiers)*

Commonly used activation functions

Every activation function (or *non-linearity*) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions you may encounter in practice:



Left: Sigmoid non-linearity squashes real numbers to range between [0,1] **Right:** The tanh non-linearity squashes real numbers to range between [-1,1].

Sigmoid. The sigmoid non-linearity has the mathematical form $\sigma(x) = 1/(1 + e^{-x})$ and is shown in the image above on the left. As alluded to in the previous section, it takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

- *Sigmoids saturate and kill gradients*. A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate's output for the whole objective. Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must