Backpropagation for sequential neural networks with densely connected layers

Pawel Wocjan

February 20, 2019

Abstract

We introduce neural networks with densely connected layers and describe the backpropagation algorithm for efficiently training these networks.

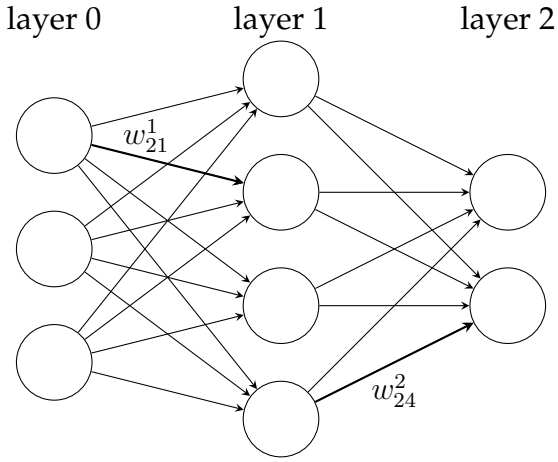
1 Forward propagation

We consider a sequential neural netowork consiting of the L densely connected layers of sizes $n^{[\ell]}$, where $\ell \in \{0, \ldots, L-1\}$ is used to enumerate the layers.

For $\ell \geq 1$, we denote the weight of the connection from the kth neuron in the $(\ell-1)$th layer to the jth neuron in the ℓth layer by

$$w_{jk}^{[\ell]} \tag{1}$$

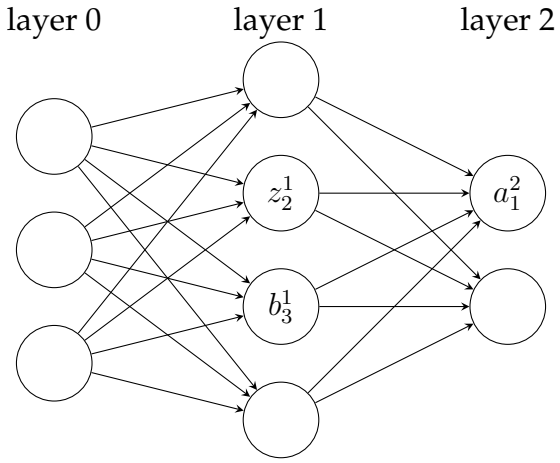
The diagram below shows two examples. The considered connections are drawn thicker.



We use a similar notation for the network's biases and activations. Let

$$b_j^{[\ell]}, \quad z_j^{[\ell]}, \quad a_j^{[\ell]} \tag{2}$$

denote the bias, weighted input, and activation of the jth neuron in the ℓth layer, respectively.



To keep the discuss genearl, we use $f^{[(\ell)]}$ to denote the activation function of the neurons in the ℓth layer. This is because the activation functions for different layers do not always have to be equal.[1]

The forward propagation is described by the following formulas in index notation. We use $k \in \{0, \ldots, n^{[(0)]} - 1\}$ to enumerate all neurons the 0th layer, which is the input layer. For each neuron k of the input layer, we set its activation to

$$a_k^{[0]} = x_k \tag{3}$$

where the value x_k is the kth entry of feature vector $\boldsymbol{x} \in \mathbb{R}^{n^{[0]}}$ that is input the network. (Obviously, the input neurons do not have any weights, biases, or weighted inputs.)

We use $j \in \{0, \ldots, n^{[\ell]} - 1\}$ to enumerate all neurons of the ℓ layer, where $\ell \geq 1$. For each neuron of the layer ℓ, we have

$$z_j^{[\ell]} = \sum_{k=0}^{n^{[\ell]}-1} w_{jk}^{[\ell]} \cdot a_k^{[\ell-1]} + b_j^{[\ell]} \tag{4}$$

$$a_j^{[\ell]} = f^{[\ell]}(z_j^{[\ell]}) \tag{5}$$

where the sum is taken over all neurons in the $(\ell - 1)$th layer.

[1] We exlude the softmax activation function for now. We only consider activation functions that are applied by each neuron independently. These include sigmoid, tangent hyperbolicus, relu activation functions. We treat the softmax activation function later.

To rewrite the above formulas in a matrix form we define a weight matrix

$$W^{[\ell]} = \left(w_{jk}^{[\ell]}\right) \in \mathbb{R}^{n^{[\ell-1]} \times n^{[\ell]}} \tag{6}$$

for each layer ℓ. Similarily, we define the bias vector $\boldsymbol{b}^{[\ell]} \in \mathbb{R}^{[\ell]}$ and the activation vector $\boldsymbol{a}^{[\ell]} \in \mathbb{R}^{[\ell]}$ for each layer ℓ.

The forward propagation is matrix notation is as follows. For the input layer, we set

$$\boldsymbol{a}^{[0]} = \boldsymbol{x} \tag{7}$$

For the ℓth layer, where $\ell \geq 1$, we have

$$\boldsymbol{z}^{[\ell]} = W^{[\ell]}\boldsymbol{a}^{[\ell-1]} + \boldsymbol{b}^{[\ell]} \tag{8}$$

$$\boldsymbol{a}^{[\ell]} = f^{[\ell]}(\boldsymbol{z}^{[\ell]}) \tag{9}$$

2 Loss function

The goal of backpropagation is to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{[\ell]}} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b_{j}^{[\ell]}} \tag{10}$$

with respect to any weight and bias of the network. We need these partial derivatives to be able to apply stochastic gradient descent.

Let $\boldsymbol{x} \in \mathbb{R}^{n^{[0]}}$ be an feature vector and $\boldsymbol{y} \in \mathbb{R}^{n^{[L-1]}}$ its corresponding target vector.[2] For obvious reasons, we need to assume that the loss function $\mathcal{L}$ is a function of the target vector $\boldsymbol{y}$ and the output of the neural network $\boldsymbol{a}^{[L-1]}$ that is produced when $\boldsymbol{x}$ isthe input. Also, we assume that the loss for a mini-batch is given by the mean of the losses for the individual examples of the mini-batch.

[2]For instance, this could be the the one-hot-encoding vector.