Backpropagation for sequential neural networks with densely connected layers

Pawel Wocjan

February 23, 2019

Abstract

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

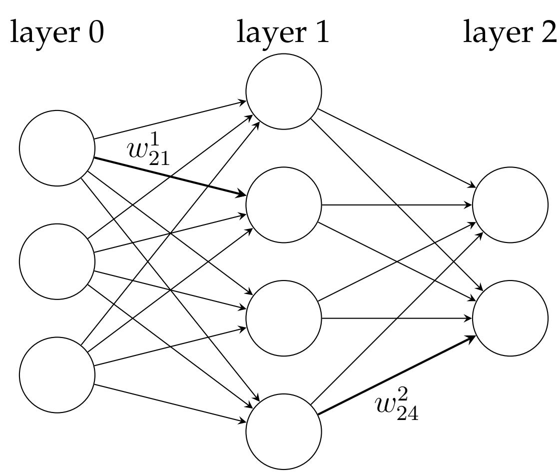
1 Forward propagation

We consider a sequential neural netowork consiting of the L densely connected layers of sizes n^ℓ, 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

$$w^\ell_{jk} \tag{1}$$

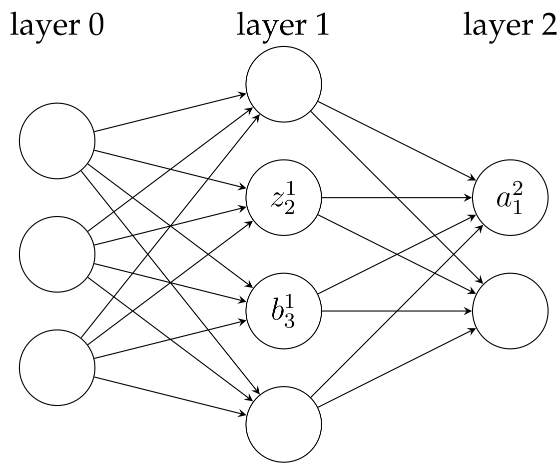
The diagram below shows two examples. The considered connections are drawn thicker.



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^ℓ 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] We use g^ℓ to denote the derivative of f^ℓ.

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

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

for each layer ℓ. Similarly, 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 in 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

$$\frac{\partial \mathcal{L}}{\partial w^\ell_{jk}} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b^\ell_j} \tag{10}$$

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 a 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.

3 Backpropagation

Assume that a friendly demon sits in the jth neuron in layer ℓ. As the weighted input $\boldsymbol{z}^\ell_j$ is formed, the demon messes with the neuron's operation. It adds a little change $\Delta\boldsymbol{z}^\ell_j$ to $\boldsymbol{z}^\ell_j$, so the neuron outputs $f^\ell(\boldsymbol{z}^\ell_j + \Delta\boldsymbol{z}^\ell_j)$ instead of $f^\ell(\boldsymbol{z}^\ell_j)$. This changes is propagated through the subsequent layers, finally causing the loss to change by an amount

$$\frac{\partial \mathcal{L}}{\partial z^\ell_j} \cdot \Delta z^\ell_j. \tag{11}$$

This demon is good, and it is trying to decrease the loss, that is, to find a Δz^ℓ_j that makes the loss smaller.

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

- Suppose the partial derivative $\partial \mathcal{L}/\partial z_j^\ell$ is a large value (either positive or negative). Then the demon can decrease the lost substantially by choosing Δz_j^ℓ to have the opposite sign to $\partial \mathcal{L}/\partial z_j^\ell$.
- In contrast, suppose that the partial derivative $\partial \mathcal{L}/\partial z_j^\ell$ is close to zero. Then the demon cannot decrease the loss much by perturbing the weighted input z_j^ℓ. So far the helpful demon can tell, the neuron is already pretty near optimal. So there is a heuristic sense in which the partial derivative $\partial \mathcal{L}/\partial z_j^\ell$ is a measure for the error in the neuron.

We define the delta (error) δ_j^ℓ of the jth neuron in the ℓth layer by

$$\delta_j^\ell = \frac{\partial \mathcal{L}}{\partial z_j^\ell}. \tag{12}$$

We also define $\boldsymbol{\delta}^\ell$ the vector of all errors associated with layer ℓ by

$$\boldsymbol{\delta}^\ell = \nabla_{\boldsymbol{z}^\ell} \mathcal{L}, \tag{13}$$

where $\nabla_{\boldsymbol{z}^\ell} \mathcal{L}$ the vector whose entries are the partial derivatives $\partial \mathcal{L}/\partial z_j^\ell$.

We will see backpropagation gives us an efficient way of calculating $\boldsymbol{\delta}^\ell$ for every layer. We use the following notation. The symbol $\odot$ denotes the Hadamard product[3] of two vectors. We use $\nabla_{\boldsymbol{a}^\ell} \mathcal{L}$ is defined to the vector whose entries are the partial derivatives $\partial \mathcal{L}/\partial a_j^\ell$. Similarly, we define $\nabla_{\boldsymbol{b}^\ell} \mathcal{L}$ and $\nabla_{W^\ell} \mathcal{L}$. Note that the former is a vector of the same shape as $\boldsymbol{b}$ and the latter a matrix of the same shape as W^ℓ. We use $(W^\ell)^T$ to denote the transpose of the weight matrix W^ℓ.

(BP1) equation for the error in the output layer

We show that the error $\boldsymbol{\delta}^{L-1}$ in the output layer is given by

$$\boldsymbol{\delta}^{L-1} = g(\boldsymbol{z}^{L-1}) \odot \nabla_{\boldsymbol{a}^{L-1}} \mathcal{L}, \tag{14}$$

vectors and To see this, we compute the components of $\boldsymbol{\delta}^{L-1}$ as follows

$$\begin{aligned} \delta_j^{L-1} &= \frac{\partial \mathcal{L}}{\partial a_j^{L-1}} \cdot \frac{\partial a_j^{L-1}}{\partial z_j^{L-1}} && (15) \\ &= \frac{\partial \mathcal{L}}{\partial a_j^{L-1}} \cdot g^{L-1}(z_j^{L-1}). && (16) \end{aligned}$$

Note that everything is easily computed. In particular, we compute the weighted input z_j^{L-1} during forward propagation, and it's only a small additional cost to compute $g^{L-1}(z_j^\ell)$. The exact form of $\partial \mathcal{L}/\partial a_j^{L-1}$ depends on the particular loss function used. However, since the loss function is known, it should be easy to compute $\partial \mathcal{L}/\partial a_j^{L-1}$.

[3] The Hadamard product (also called Schur product) is the entrywise product.

(BP2) equation for the error in terms of the error of the successor layer

We show that

$$\boldsymbol{\delta}^{\ell-1} = g^{\ell-1}(\boldsymbol{z}^{\ell-1}) \odot (W^{\ell})^T \boldsymbol{\delta}^{\ell}. \tag{17}$$

First, we have

$$\frac{\partial \mathcal{L}}{\partial a_k^{\ell-1}} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j^{\ell}} \cdot \frac{\partial z_j^{\ell}}{\partial a_k^{\ell-1}} \tag{18}$$

$$= \sum_j \delta_j^{\ell} \cdot w_{jk}^{\ell} \tag{19}$$

and in matrix notation

$$\nabla_{\boldsymbol{a}^{\ell-1}} \mathcal{L} = (W^{\ell})^T \nabla_{\boldsymbol{z}^{\ell}} \mathcal{L} \tag{20}$$

$$= (W^{\ell})^T \boldsymbol{\delta}^{\ell}. \tag{21}$$

Second, we have

$$\frac{\partial L}{\partial z_k^{\ell-1}} = \frac{\partial \mathcal{L}}{\partial a_k^{\ell-1}} \cdot \frac{\partial a_k^{\ell-1}}{\partial z_k^{\ell-1}} \tag{22}$$

and in matrix notation

$$\boldsymbol{\delta}^{\ell-1} = g^{\ell-1}(\boldsymbol{z}^{\ell-1}) \odot (W^{\ell})^T \boldsymbol{\delta}^{\ell}. \tag{23}$$

(BP3) equation for the rate of change of loss wrt to any bias

We show that

$$\nabla_{\boldsymbol{b}^{\ell}} \mathcal{L} = \boldsymbol{\delta}^{\ell}. \tag{24}$$

To see this, compute the entries of $\nabla_{\boldsymbol{b}^{\ell}}$ as follows

$$\frac{\partial \mathcal{L}}{\partial b_j^{\ell}} = \frac{\partial \mathcal{L}}{\partial z_j^{\ell}} \cdot \frac{\partial z_j^{\ell}}{b_j^{\ell}} \tag{25}$$

$$= \delta_j^{\ell} \cdot 1. \tag{26}$$

(BP4) equation for the rate of change of loss wrt to any weight

We show that

$$\nabla_{\boldsymbol{b}^{\ell}} \mathcal{L} = \boldsymbol{\delta}^{\ell} (\boldsymbol{a}^{\ell-1})^T. \tag{27}$$

Note that $\boldsymbol{\delta}^{\ell}$ is a column vector and $(\boldsymbol{a}^{\ell-1})^T$ is a row vector so that $\boldsymbol{\delta}^{\ell}(\boldsymbol{a}^{\ell-1})^T$ is a matrix. To prove the above equation, compute the entry of $\nabla_{\boldsymbol{w}^{\ell}}$ as follows

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{\ell}} = \frac{\partial \mathcal{L}}{\partial z_j^{\ell}} \cdot \frac{\partial z_j^{\ell}}{w_{jk}^{\ell}} \tag{28}$$

$$= \delta_j^{\ell} \cdot a_k^{\ell-1}. \tag{29}$$