S&DS 265 / 365

Introductory Machine Learning

Some Notes on Backpropagation

These notes describe how to carry out backpropagation for neural networks, using only a simple form of the chain rule from high school calculus. Don't worry about following all of the calculations, but it's spelled out here if you are interested.

The linear case

If $\mathcal{L}$ is a scalar function and $A = BC$, then

$$\frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial A} C^T \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial C} = B^T \frac{\partial \mathcal{L}}{\partial A}. \tag{2}$$

This can be shown directly using the "usual" chain rule. You should check that the dimensions match up!

The function $\mathcal{L}$ is our loss function. The use of this is called "backpropagation" because we start with the derivatives in the last layer of the network, and recursively send these "back" to compute the derivatives in the earlier part of the network.

So, if $f = Wx + b$ then

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial f} x^T. \tag{3}$$

If the loss $\mathcal{L}$ is squared error $\mathcal{L} = \frac{1}{2}(y - f)^2$ then

$$\frac{\partial \mathcal{L}}{\partial f} = (f - y). \tag{4}$$

Now, if

$$f = W_2 h + b_2 \tag{5}$$

$$h = W_1 x + b_1 \tag{6}$$

then

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} h^T = (f - y) h^T \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \frac{\partial \mathcal{L}}{\partial f} = W_2^T (f - y) \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial f} = (f - y). \tag{9}$$

Then, we have that

$$\frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial h} x^T = W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T = W_2^T (f - y) x^T \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial h} = W_2^T (f - y). \tag{11}$$

This is just another linear model, so it will train to be equivalent to least squares regression.

Using an activation function

To get something nonlinear, we need to include an activation function. We need an extension of our previous use of the chain rule.

If $\mathcal{L}$ is a scalar function and $A = \varphi(BC)$, applied component-wise. Then

$$\frac{\partial \mathcal{L}}{\partial B} = \left(\varphi'(BC) \star \frac{\partial \mathcal{L}}{\partial A} \right) C^T \tag{12}$$

$$\frac{\partial \mathcal{L}}{\partial C} = B^T \left(\varphi'(BC) \star \frac{\partial \mathcal{L}}{\partial A} \right) \tag{13}$$

where $P \star Q$ denotes the component-wise product.

If we add a ReLU nonlinearity, we get the two-layer neural network

$$h = \text{ReLU}(W_1 x + b_1) \tag{14}$$

$$f = W_2 h + b_2. \tag{15}$$

Let the loss for an example (x, y) be $\mathcal{L} = \frac{1}{2}(y - f(x))^2$. From the above calculations we then have

for the second layer

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} h^T \tag{16}$$

$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \frac{\partial \mathcal{L}}{\partial f} \tag{17}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial f}. \tag{18}$$

For the hidden layer, we just need to note that the derivative of the ReLU is $\text{ReLU}'(u) = \mathbb{1}(u > 0)$. We then have

$$\frac{\partial \mathcal{L}}{\partial W_1} = \left(\mathbb{1}(W_1 x + b_1 > 0) \star \frac{\partial \mathcal{L}}{\partial h} \right) x^T \tag{19}$$

$$= \left(\mathbb{1}(h > 0) \star \frac{\partial \mathcal{L}}{\partial h} \right) x^T \tag{20}$$

$$= \left(\mathbb{1}(h > 0) \star W_2^T \frac{\partial \mathcal{L}}{\partial f} \right) x^T \tag{21}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \left(\mathbb{1}(W_1 x + b_1 > 0) \star \frac{\partial \mathcal{L}}{\partial h} \right) \tag{22}$$

$$= \left(\mathbb{1}(h > 0) \star \frac{\partial \mathcal{L}}{\partial h} \right) \tag{23}$$

$$= \left(\mathbb{1}(h > 0) W_2^T \star \frac{\partial \mathcal{L}}{\partial f} \right). \tag{24}$$

If we change the activation function to $\varphi(u) = \tanh(u)$ then we just need to calculate (or Wiki-up) the derivative as $\varphi'(u) = 1 - \tanh^2(u)$. We would then change the above to

$$\frac{\partial \mathcal{L}}{\partial W_1} = \left((1 - \tanh^2(W_1 x + b_1)) \star \frac{\partial \mathcal{L}}{\partial h} \right) x^T \tag{25}$$

$$= \left((1 - \tanh^2(W_1 x + b_1)) \star W_2^T \frac{\partial \mathcal{L}}{\partial f} \right) x^T \tag{26}$$

and similarly for $\frac{\partial \mathcal{L}}{\partial b_2}$.

Classification

For a two-layer network *for classification* we have

$$h = \text{ReLU}(W_1 x + b_1) \tag{27}$$

$$f = W_2 h_1 + b_2 \tag{28}$$

$$p = \text{Softmax}(f) \tag{29}$$

where again W_j and b_j are matrices or vectors of the appropriate dimensions. Let the loss for an example (x, y) be the log-loss $\mathcal{L} = -\log p(y \mid x)$. Then

$$\frac{\partial \mathcal{L}}{\partial f} = \begin{pmatrix} p_1 - \mathbb{1}(y=1) \\ p_2 - \mathbb{1}(y=2) \\ p_3 - \mathbb{1}(y=3) \end{pmatrix} \in \mathbb{R}^3. \tag{30}$$

To see this, note that we can write the loss function as

$$\mathcal{L}(x, y) = \log\left(e^{f_1} + e^{f_2} + e^{f_3}\right) - \mathbb{1}(y=1) f_1 - \mathbb{1}(y=2) f_2 - \mathbb{1}(y=3) f_3. \tag{31}$$

Then, note that

$$\frac{\partial \mathcal{L}}{\partial f_1} = \frac{e^{f_1}}{e^{f_1} + e^{f_2} + e^{f_3}} - \mathbb{1}(y=1) \tag{32}$$

using the chain rule with $d\log(u)/du = 1/u$ and $de^u/du = e^u$.

Backpropagating to the second layer we have

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial f} h_1^T \tag{33}$$

$$\frac{\partial \mathcal{L}}{\partial h} = W_2^T \frac{\partial \mathcal{L}}{\partial f} \tag{34}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial f} \tag{35}$$

Then backpropagating to the first layer

$$\frac{\partial \mathcal{L}}{\partial W_1} = \left(\mathbb{1}(h > 0) \star \frac{\partial \mathcal{L}}{\partial h}\right) x^T \tag{36}$$

$$= \left(\mathbb{1}(h > 0) \star W_2^T \frac{\partial \mathcal{L}}{\partial f}\right) x^T \tag{37}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \left(\mathbb{1}(h > 0) \star \frac{\partial \mathcal{L}}{\partial h}\right) \tag{38}$$

$$= \left(\mathbb{1}(h > 0) \star W_2^T \frac{\partial \mathcal{L}}{\partial f}\right). \tag{39}$$

We'll leave it to you to extend the calculations (and implementation!) to a three-layer network, if you are so-inclined.