# A Better Error Function

Looking at our backpropagation algorithm from versions 1 and 2, we find something undesirable. Namely, both the change in weights and change in biases depend on the term:

$$\vec{\delta}_{\vec{z}} C = \vec{\delta}_{\vec{a}} C \odot \vec{\sigma}'(\vec{z}) = (\vec{a} - \vec{y}) \odot \vec{\sigma}'(\vec{z})$$

where all quantities refer to the last layer. If we analyze this a bit, we'll find a problem.

Intuitively, we would like backpropagation to have two properties:

1) When the network's output $\vec{a}$ is close to the correct answer $\vec{y}$, the weights/biases change very little.

2) When $\vec{a}$ is very off from $\vec{y}$, the weights/biases change a lot to improve.

Our algorithm satisfies the first property, since the term $\vec{a} - \vec{y} \to \vec{0}$ when $\vec{a} \to \vec{y}$, so $\vec{\delta}_{\vec{z}} C \to \vec{0}$ if $\vec{a}$ and $\vec{y}$ are close, and thus the weights/biases don't change much.

But what about when $\vec{a}$ and $\vec{y}$ are far apart? Component-wise, each $y$ in $\vec{y}$ is a 0 or 1, so for $\vec{a}$ to be far off, its components would also need to be near 0 or 1 (opposite the corresponding $y$):

$$\vec{a} = \begin{bmatrix} \text{near } 1 \\ \text{near } 0 \\ \vdots \\ \text{near } 0 \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

In this case, $\vec{a} - \vec{y}$ may be relatively large component-wise (at least, in the second component, where $\vec{a}$ is very starkly wrong), but what about $\vec{\sigma}'(\vec{z})$?

Recall:

$$\vec{\sigma}'(\vec{z}) = \vec{\sigma}(\vec{z}) \odot \left(\vec{1} - \vec{\sigma}(\vec{z})\right) = \vec{a} \odot (\vec{1} - \vec{a})$$

Or component-wise:

$$\sigma'(z) = a(1-a)$$

If $a$ is close to 0 or 1, this number approaches 0. Thus, it drives the term $\vec{\delta}_{\vec{z}} C$, and the deltas in weights/biases, closer to 0 (at least, in the components that are starkly wrong). So our network does not improve a lot when $\vec{a}$ is very far off.

We can remedy this with a simple "hack": just drop the $\vec{\sigma}'(\vec{z})$ term. That is, declare:

$$\vec{\delta}_{\vec{z}} C := \vec{a} - \vec{y}$$

during the algorithm. This will satisfy both properties. See the revised algorithm for the implementation of this change, with the error function parameter set to "cross entropy".

---

But this inherently changes the theory underlying the network. If we impose this property, we are no longer using Euclidean distance as our error heuristic. So what function is it that gives us this property?

As we will see, we arrive at the <u>cross entropy function</u>. We'll derive this fact here, and conclude that cross-entropy is generally a "better" choice of error function because it satisfies the two properties described above.

For each component of $\vec{\delta}_{\vec{z}} C$, we want:

$$\frac{dC}{dz} = a - y$$

Substitute $\frac{dC}{dz} = \frac{dC}{da} \cdot \frac{da}{dz}$ by the chain rule:

$$\frac{dC}{da} \cdot \frac{da}{dz} = a - y$$

$$\frac{dC}{da} \sigma'(z) = a - y$$

$$\frac{dC}{da} a(1-a) = a - y$$

$$\frac{dC}{da} = \frac{a-y}{a(1-a)}$$

We can integrate this to find $C$:

$$C = \int \frac{a-y}{a(1-a)} \, da$$

We can use a clever trick to simplify this integral:

$$= \int \frac{a - ay + ay - y}{a(1-a)} \, da$$

$$= \int \frac{a(1-y) - y(1-a)}{a(1-a)} \, da$$

$$= \int \left[ \frac{1-y}{1-a} - \frac{y}{a} \right] da$$

$$= (1-y)\left(-\ln(1-a)\right) - y \ln a + c$$

where $c$ is the integration constant.

$$= -\left[ y \ln a + (1-y) \ln(1-a) \right] + c$$

To solve for $c$, we make use of a key property of error functions in general:

$$\text{As } a \to y, \quad C \to 0$$

Since $y \in \{0, 1\}$, we only need to inspect the cases $a \to 0$ and $a \to 1$.

Case 1 $a \to 0$, $y = 0$

$$-\left[ 0 \cdot \ln a + (1-0) \ln(1-a) \right] + c \to 0$$

$$-\ln(1-a) + c \to 0$$

Let $a \to 0$:

$$-\ln(1) + c \to 0$$

$$c \to 0$$

(Similar results follow in case 2, when $a \to 1$, $y = 1$.)

Thus, setting $c = 0$ satisfies the properties of the error function. So, for one component, we have:

$$C = -\left[ y \ln a + (1-y) \ln(1-a) \right]$$

which is known as the <u>cross-entropy</u> between $y$ and $a$.*

The most natural way to apply this over all components is simply to sum over the components:

$$C = -\sum_{a,y} y \ln a + (1-y) \ln(1-a)$$

And this still satisfies the property that $\frac{dC}{dz} = a - y$ for each component $z, a, y$ of $\vec{z}, \vec{a}, \vec{y}$.

* Technically, cross-entropy is:

$-\sum_i p_i \ln q_i$

for two probability distributions $\{p_1, \ldots p_n\}$ $\{q_1, \ldots, q_n\}$. If we treat $\{a, 1-a\}$ and $\{y, 1-y\}$ as these prob. distributions, then this is cross-entropy.

This turns out to be a very natural measure of error — if $\{y_i, 1-y_i\}$ are the correct "probabilities" that the input's label is/isn't $i$, then this error measure tells us how close the network's computed probabilities $\{a_i, 1-a_i\}$ are to the correct ones. This is simply a different perspective on how good the network is at predicting the correct label.

You can verify that all the rules/quantities from our original backpropagation algorithm still apply:

| Last layer | Previous layer |
| --- | --- |
| ($\nabla_{\vec{a}} C$ value is irrelevant) | $\nabla_{\vec{a}'} C = W^t \nabla_{\vec{z}} C$ |
| $\nabla_{\vec{z}} C = \vec{a} - \vec{y}$ by construction | $\nabla_{\vec{z}'} C = \nabla_{\vec{a}'} C \odot \vec{\sigma}'(\vec{z}')$ |
| $\nabla_{\vec{b}} C = \nabla_{\vec{z}} C$ | $\nabla_{\vec{b}'} C = \nabla_{\vec{z}'} C$ |
| $D_W C = \nabla_{\vec{z}} C \otimes \vec{a}'$ | $D_{W'} C = \nabla_{\vec{z}'} C \otimes \vec{a}''$ |

See the revised backpropagation algorithm for the implementation.