Supplementary examples of neural networks

These notes are meant to give a few concrete examples of tiny *tiny* neural networks where it's possible to more easily understand both the *forward propagation* (compute output prediction y from input features x) and the *backward propagation* (compute gradient of a loss function with respect to network parameters).

The notes contain a few comments on how these examples connect to convolution, max pooling and average pooling. However, these notes are by no means sufficient to understand those topics; refer to Bishop and the slides.

Neural networks

The purpose of going through the following examples is to explain the nature of how backpropagation computes a gradient, and to try to connect simple examples to the notation for δ_k and δ_j used in Bishop Chapter 5.

Example 1

Suppose we have a little 1-1-1 neural network with activation function $h(\cdot)$ and no bias parameters, just weights:

$$\begin{aligned}
a &= w_1 x \\
z &= h(a) \\
y &= w_2 z \\
\ell &= \tfrac{1}{2}(y-t)^2
\end{aligned}$$

Then the partial derivative of the loss ℓ with respect to the second-layer weight is:

$$\begin{aligned}
\frac{\partial \ell}{\partial w_2} &= \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w_2} \\
&= (y-t)z \\
&= \delta_k z
\end{aligned}$$

where in Bishop we used shorthand $\delta_k = \frac{\partial \ell}{\partial y} = (y-t)$ with symbol k implying an index over the **output** layer, which in this case just one value y.

$$\begin{aligned}
\frac{\partial \ell}{\partial w_1} &= \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial a}\frac{\partial a}{\partial w_1} \\
&= (y-t)w_2 h'(a)x \\
&= \delta_k w_2 h'(a)x \\
&= \delta_j x
\end{aligned}$$

where as in Bishop we used shorthand $\delta_j = \frac{\partial \ell}{\partial a} = \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial a} = \delta_k w_2 h'(a)$ where again j indexes over the sole activation a.

Specific activation functions

If $h(a) = \sigma(a)$ then we would have $h'(a) = \sigma(a)(1 - \sigma(a)) = z(1 - z)$.

If $h(a) = \tanh(a)$ then we would have $h'(a) = 1 - \tanh^2(a) = 1 - z^2$.

If $h(a) = \mathrm{ReLU(a)} = \max(0, a)$ then we would have $h'(a) = \begin{cases} 0 & a < 0 \\ 1 & a > 0 \\ \text{undefined} & a = 0 \end{cases}$, but in practice we can just define $h'(0) \equiv 0$ and things work fine.

Example 2

Suppose we had a network two outputs in the final layer, rather than just one:

$$\begin{aligned} a &= w_1 x \\ z &= h(a) \\ y_1 &= w_2 z \\ y_2 &= w_3 z \\ \ell &= \tfrac{1}{2}(y_1 - t_1)^2 + \tfrac{1}{2}(y_2 - t_2)^2 \end{aligned}$$

Then since ℓ is a function of only y_1 and y_2 by chain rule of partial differentiation the partial derivatives with respect to second layer weights w_2 and w_3 are

$$\begin{aligned} \frac{\partial \ell}{\partial w_2} &= \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial w_2} + \frac{\partial \ell}{\partial y_2}\frac{\partial y_2}{\partial w_2} = (y_1 - t_1)(z) + (y_2 - t_2)(0) = \delta_{k_1} z \\ \frac{\partial \ell}{\partial w_3} &= \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial w_3} + \frac{\partial \ell}{\partial y_2}\frac{\partial y_2}{\partial w_3} = (y_1 - t_1)(0) + (y_2 - t_2)(z) = \delta_{k_2} z \end{aligned}$$

where, trying to connect to Bishop's notation, we have used shorthand $\delta_{k_1} = (y_1 - t_1)$ to denote the δ_k of the first output y_1 and $\delta_{k_2} = (y_2 - t_2)$ to denote the δ_k of the second output y_2.

Similarly the partial derivative for weight w_1

$$
\begin{aligned}
\frac{\partial \ell}{\partial w_1} &= \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial w_1} + \frac{\partial \ell}{\partial y_2}\frac{\partial y_2}{\partial w_1} \\
&= \frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial z}\frac{\partial z}{\partial a}\frac{\partial a}{\partial w_1} + \frac{\partial \ell}{\partial y_2}\frac{\partial y_2}{\partial z}\frac{\partial z}{\partial a}\frac{\partial a}{\partial w_1} \\
&= \left(\frac{\partial \ell}{\partial y_1}\frac{\partial y_1}{\partial z} + \frac{\partial \ell}{\partial y_2}\frac{\partial y_2}{\partial z}\right)\frac{\partial z}{\partial a}\frac{\partial a}{\partial w_1} \\
&= \big((y_1 - t_1)w_2 + (y_2 - t_2)w_3\big)h'(a)x \\
&= (\delta_{k_1} w_2 + \delta_{k_2} w_3)h'(a)x \\
&= \delta_j x
\end{aligned}
$$

where again we used shorthand $\delta_j = \frac{\partial \ell}{\partial a} = \sum_k \frac{\partial \ell}{\partial y_k}\frac{\partial y_k}{\partial a} = (\delta_{k_1} w_2 + \delta_{k_2} w_3)h'(a)$.

Example 3

Suppose we had a network one output but two hidden units, each connected to only one feature (so, the first layer is not "fully-connected").

$$
\begin{aligned}
a_1 &= w_1 x_1 \\
a_2 &= w_2 x_2 \\
z_1 &= h(a_1) \\
z_2 &= h(a_2) \\
y &= w_3 z_1 + w_4 z_2 \\
\ell &= \tfrac{1}{2}(y - t)^2
\end{aligned}
$$

Then since ℓ is a function of only y_1 and y_2 by chain rule of partial differentiation the partial derivatives with respect to second layer weights w_2 and w_3 are

$$
\begin{aligned}
\frac{\partial \ell}{\partial w_3} &= \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w_3} = (y - t)z_1 = \delta_k z_1 \\
\frac{\partial \ell}{\partial w_4} &= \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w_4} = (y - t)z_2 = \delta_k z_2
\end{aligned}
$$

where we use shorthand $\delta_k = \frac{\partial \ell}{\partial y} = (y - t)$ as usual. Now we have two weights in the final layer, each having a slightly different partial derivative (multiply δ_k by z_1 versus z_2).

Now taking the partial derivative of ℓ with respect to w_1 gives

$$
\begin{aligned}
\frac{\partial \ell}{\partial w_1} &= \frac{\partial \ell}{\partial y}\frac{\partial y}{\partial w_1} \\
&= \frac{\partial \ell}{\partial y}\left(\frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial w_1} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial w_1}\right) \quad \text{because } y \text{ is a function of } z_1 \text{ and } z_2 \\
&= \frac{\partial \ell}{\partial y}\left(\frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial a_1}\frac{\partial a_1}{\partial w_1} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial a_2}\frac{\partial a_2}{\partial w_1}\right) \quad \text{by chain rule} \\
&= (y-t)\big(w_3 h'(a_1)x_1 + w_4 h'(a_2)(0)\big) \\
&= \delta_k w_3 h'(a_1) x_1 \\
&= \delta_{j_1} x_1
\end{aligned}
$$

where we have used shorthand $\delta_{j_1} = \frac{\partial \ell}{\partial a_1}$ to denote the "error" backpropagated to a_1 and likewise $\frac{\partial \ell}{\partial w_2} = \delta_k w_4 h'(a_2) x_2 = \delta_{j_2} x_2$

Convolutional neural networks

Example 1

Let's revisit Example 3 from neural networks, except this time we'll add an initial step in the first layer where we compute weights w_1 and w_2 from a single *shared* w in the first layer. This turns our first layer into a "convolutional" layer with "filter size 1" (i.e. activation a_j is connected directly to just one feature x_j)

$$
\begin{aligned}
w_1 &= w \\
w_2 &= w \\
a_1 &= w_1 x_1 \\
a_2 &= w_2 x_2 \\
z_1 &= h(a_1) \\
z_2 &= h(a_2) \\
y &= w_3 z_1 + w_4 z_2 \\
\ell &= \tfrac{1}{2}(y-t)^2
\end{aligned}
$$

The values of $\frac{\partial \ell}{\partial w_3}$ and $\frac{\partial \ell}{\partial w_4}$ are the same as for Exercise 3 earlier.

Since ℓ is dependent on (functions of) weights w_1 and w_2, both of which are dependent on (functions of) shared weight w, then by the chain rule of partial differentiation we have

$$
\begin{aligned}
\frac{\partial \ell}{\partial w} &= \frac{\partial \ell}{\partial w_1}\frac{\partial w_1}{\partial w} + \frac{\partial \ell}{\partial w_2}\frac{\partial w_2}{\partial w} \\
&= (\delta_{j_1} x_1)(1) + (\delta_{j_2} x_2)(1) \quad \text{from Exercise 3 of neural nets} \\
&= \delta_{j_1} x_1 + \delta_{j_2} x_2
\end{aligned}
$$

Since we could obviously have just eliminated w_1 and w_2 when writing the original problem, we're really computing the gradient for an equivalent "network":

$$
\begin{aligned}
a_1 &= wx_1 \\
a_2 &= wx_2 \quad \leftarrow \text{ shares same weight as } a_1! \\
z_1 &= h(a_1) \\
z_2 &= h(a_2) \\
y &= w_3 z_1 + w_4 z_2 \\
\ell &= \tfrac{1}{2}(y-t)^2
\end{aligned}
$$

where the gradient component for w is $\frac{\partial \ell}{\partial w} = \delta_{j_1} x_1 + \delta_{j_2} x_2$. So, this is the gradient component for our little "filter" of length 1 that "scans" across the input features (all two of them 😃).

Using terminology from convolutional neural networks, this is a convolutional neural network with:

- a single convolutional layer (1 dimensional, 1 filter, filter size 1, no padding, no bias, activation h), and
- a single fully-connected layer (no bias).

Example 2

We revisit Example 1 except this time turn the final layer into a *max pooling* layer over all (both) of the inputs z_1 and z_2.

$$
\begin{aligned}
a_1 &= wx_1 \\
a_2 &= wx_2 \\
z_1 &= h(a_1) \\
z_2 &= h(a_2) \\
y &= \max(z_1, z_2) \quad \leftarrow \text{ only one contributes to } y! \\
\ell &= \tfrac{1}{2}(y-t)^2
\end{aligned}
$$

Using terminology from convolutional neural networks, this is a convolutional neural network with:

- a single convolutional layer (1 dimensional, 1 filter, filter size 1, no padding, no bias, activation h), and
- a single max pooling layer (1 dimensional, pooling region size 2, no padding)

Since for any fixed input $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ we will have either $\max(z_1, z_2) = z_1$ or $\max(z_1, z_2) = z_2$, let's assume that $z_1 > z_2$. Then for that particular $\mathbf{x}$ the above network is equivalent to

$$
\begin{aligned}
a_1 &= wx_1 \\
a_2 &= wx_2 \\
z_1 &= h(a_1) \\
z_2 &= h(a_2) \\
y &= z_1 \\
\ell &= \tfrac{1}{2}(y-t)^2
\end{aligned}
$$

Then the gradient for this *particular* $\mathbf{x}$ is a special case of Example 1 above where you fix second layer weights $w_3 = 1$ and $w_4 = 0$. In other words, max pooling is like using a different "one-hot" weight encoding that depends on the particular $\mathbf{z}$ used as input, which itself depends on the particular input features $\mathbf{x}$. For another $\mathbf{x}$ we might have had $z_2 > z_1$ and therefore it would be equivalent to setting $w_3 = 0$ and $w_4 = 1$.

Max operator

The $\max(\mathbf{z})$ operator applied over a vector of elements $\mathbf{z} = \left[z_1, \ldots, z_M\right]^T$ has the following gradient:

$$\nabla \max(\mathbf{z}) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \text{ where the sole 1 entry is at index } j' = \arg\max_j z_j$$

Max pooling

Max pooling applies the max operator to individual subsets of the z_j, similar to convolution. This means that implementing backpropagation correctly requires us to remember which particular z_j was selected by each application of the $\max$ operator; either that or we must re-compute the max to figure out which z_j is the max (and therefore relevant for computing the gradient).

Of course one *could* take the max over an entire layer (all $\mathbf{z}$) but this is seldom done in convolutional neural networks.

Average pooling

In "average pooling", rather than taking the local max of the z_j we take their average. This is equivalent to a convolutional layer where all the weights are $\frac{1}{n}$ where n is the number of weights in each filter. In our example above with just z_1 and z_2 a single output y, an average pooling layer with would compute $y = \frac{1}{2}z_1 + \frac{1}{2}z_2$