CS 189/289A Introduction to Machine Learning
Fall 2024 Jennifer Listgarten, Saeed Saremi DIS4

1 Backrop in Practice: Staged Computation

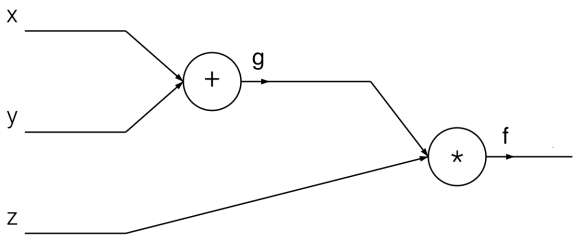
For the function $f(x, y, z) = (x + y)z$:

(a) Decompose f into two simpler functions.

Solution: We can conveniently express f as: $g = x + y$, $f = gz$. The original $f(x, y, z)$ expression is simple enough to differentiate directly, but this particular approach of thinking about it in terms of its decomposition can help with understanding the intuition behind backprop.

(b) Draw the network that represents the computation of f.

Solution:



(c) Write the forward pass and backward pass (backpropagation) in the network.

Solution:

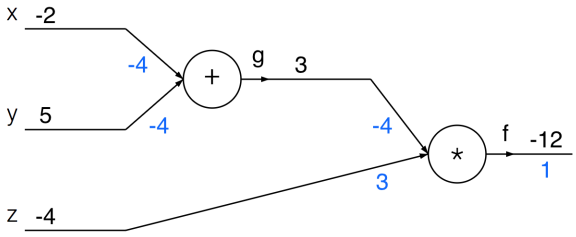
```
## forward pass
g = x + y
f = g * z

## backward pass
# backprop through f = gz first
dfdz = g
dfdg = z
# backprop through g = x + y, using chain rule
dfdx = 1.0 * dfdg # df/dx = dg/dx * df/dg
dfdy = 1.0 * dfdg # dg/dy = dg/dx = 1
```

(d) Update your network drawing with the intermediate values in the forward and backward pass. Use the inputs $x = -2$, $y = 5$, and $z = -4$.

Solution: We first run the forward pass (shown in black) of the network, which computes values from inputs to output. Then, we do the backward pass (blue), which starts at the end of the network and recursively applies the chain rule to compute the gradients. Think of the gradients as flowing backwards through the network. By the time we reach the beginning of

the network in backprop, each gate will has learned about the gradient of its output value on the final output of the entire circuit. There are two take-aways about backprop from this: 1) Backprop can thus be thought of as message passing (via the gradient signal) between gates about whether they want their outputs to increase or decrease (and how strongly), so as to make the final output value higher. 2) Backprop is a local process.



2 Backpropagation Practice

Disclaimer: the notation used in the following problem is different from the notation used in homework 3. In this discussion, like in discussion 0, we refer to $\frac{\partial \ell}{\partial W}$ as the **derivative** of ℓ with respect to W but we use this same notation in homework 3 to refer to the **gradient** instead. Remember that gradients and derivatives are transposes of each other.

(a) Assume that you have functions $f(x_1, x_2, \ldots, x_n)$, and $g_i(w) = x_i$ for $i = 1, \ldots, n$. Sketch out the computation graph for $f(g_1(w), g_2(w), \ldots, g_n(w))$. How would you compute the following derivative?

$$\frac{\mathrm{d}}{\mathrm{d}w} f(g_1(w), g_2(w), \ldots, g_n(w))$$

Solution: We will use the chain rule for multiple variables. In general, we have

$$\frac{\mathrm{d}f}{\mathrm{d}w} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial x_i}{\partial w} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial w}.$$

The computation graph for this function is given in Figure 1.

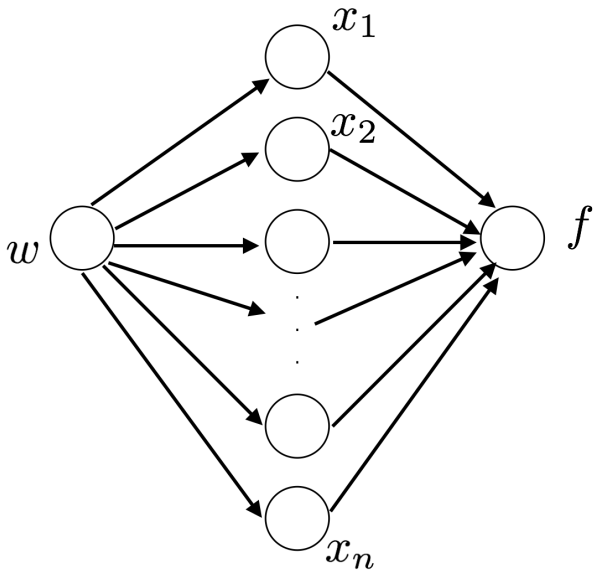


Figure 1: Example function computation graph

(b) Let $w_1, w_2, \ldots, w_n \in \mathbb{R}^d$, and we refer to these weights together as $W \in \mathbb{R}^{n \times d}$. We also have $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$. Consider the function

$$f(W, x, y) = \left(y - \sum_{i=1}^{n} \phi(w_i^\top x + b_i) \right)^2.$$

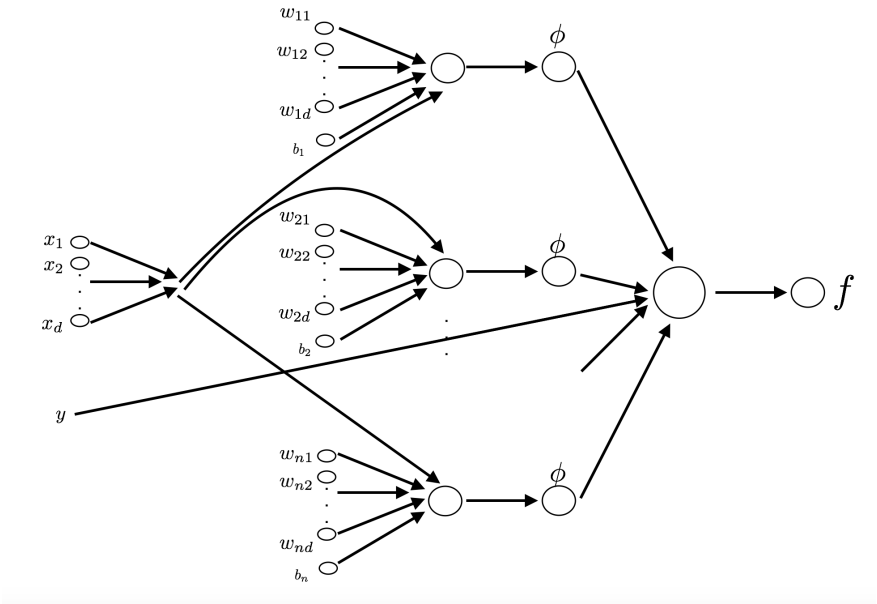


Figure 2: Example function computation graph

Sketch the computation graph for this function.

Solution: See Figure 2.

(c) Suppose $\phi(x)$ (from the previous part) is the sigmoid function, $\sigma(x)$. Compute the derivatives $\frac{\partial f}{\partial w_i}$ and $\frac{\partial f}{\partial b_i}$. Use the computational graph you drew in the previous part to guide you.

Solution: Define $r = y - \sum_{i=1}^{n} \sigma(w_i^\top x + b_i)$ and $z_i = w_i^\top x + b_i$. With these intermediate variables, we will write out the forward pass:

$$f = r^2$$
$$r = y - \sum_{i=1}^{n} \sigma(z_i)$$
$$z_i = w_i^\top x + b_i$$

Now the backward pass:

$$\frac{\partial f}{\partial r} = 2r$$
$$\frac{\partial r}{\partial z_i} = -\sigma(z_i)(1 - \sigma(z_i))$$
$$\frac{\partial z_i}{\partial w_i} = x^\top$$
$$\frac{\partial z_i}{\partial b_i} = 1$$

By applying the chain rule, we get

$$\frac{\partial f}{\partial w_i} = 2\left(\sum_{j=1}^{n} \sigma(w_j^\top x + b_j) - y\right)\sigma(w_i^\top x + b_i)(1 - \sigma(w_i^\top x + b_i))x^\top$$

$$\frac{\partial f}{\partial b_i} = 2\left(\sum_{j=1}^{n} \sigma(w_j^\top x + b_j) - y\right)\sigma(w_i^\top x + b_i)(1 - \sigma(w_i^\top x + b_i))$$

(d) Write down a single gradient descent update for $w_i^{(t+1)}$ and $b_i^{(t+1)}$, assuming step size ϵ. You answer should be in terms of $w_i^{(t)}$, $b_i^{(t)}$, x, and y.

Solution: We transpose the derivatives above to get the gradients. With this, we perform the following gradient descent update:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - 2\epsilon\left(\sum_{j=1}^{n} \sigma(w_j^{(t)\top} x + b_j^{(t)}) - y\right)\sigma(w_i^{(t)\top} x + b_i^{(t)})(1 - \sigma(w_i^{(t)\top} x + b_i^{(t)}))x$$

$$b_i^{(t+1)} \leftarrow b_i^{(t)} - 2\epsilon\left(\sum_{j=1}^{n} \sigma(w_j^{(t)\top} x + b_j^{(t)}) - y\right)\sigma(w_i^{(t)\top} x + b_i^{(t)})(1 - \sigma(w_i^{(t)\top} x + b_i^{(t)}))$$

(e) Define the cost function

$$\ell(x) = \frac{1}{2}\left\|W^{(2)}\Phi\left(W^{(1)}x + b\right) - y\right\|_2^2, \tag{1}$$

where $W^{(1)} \in \mathbb{R}^{d\times d}$, $W^{(2)} \in \mathbb{R}^{d\times d}$, and $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ is some nonlinear transformation. Compute the derivatives $\frac{\partial \ell}{\partial x}, \frac{\partial \ell}{\partial W^{(1)}}, \frac{\partial \ell}{\partial W^{(2)}}$, and $\frac{\partial \ell}{\partial b}$.

Solution: First, we write out the forward pass using intermediate variables:

$$\begin{aligned} x^{(1)} &= W^{(1)}x + b \\ x^{(2)} &= \Phi(x^{(1)}) \\ x^{(3)} &= W^{(2)}x^{(2)} \\ x^{(4)} &= x^{(3)} - y \\ \ell &= \frac{1}{2}\|x^{(4)}\|_2^2. \end{aligned}$$

Remember that the superscripts represent indices, not powers. Next, we write out the backward pass by applying the chain rule repeatedly:

$$\begin{aligned} \frac{\partial \ell}{\partial x^{(4)}} &= x^{(4)\top} \\ \frac{\partial \ell}{\partial x^{(3)}} &= \frac{\partial \ell}{\partial x^{(4)}}\frac{\partial x^{(4)}}{\partial x^{(3)}} = \frac{\partial \ell}{\partial x^{(4)}} \\ \frac{\partial \ell}{\partial x^{(2)}} &= \frac{\partial \ell}{\partial x^{(3)}}\frac{\partial x^{(3)}}{\partial x^{(2)}} = \frac{\partial \ell}{\partial x^{(3)}}W^{(2)} \end{aligned}$$

$$\frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial x^{(3)}} \frac{\partial x^{(3)}}{\partial W^{(2)}} = x^{(2)} \frac{\partial \ell}{\partial x^{(3)}}$$
$$\frac{\partial \ell}{\partial x^{(1)}} = \frac{\partial \ell}{\partial x^{(2)}} \frac{\partial \Phi}{\partial x^{(1)}}$$
$$\frac{\partial \ell}{\partial x} = \frac{\partial \ell}{\partial x^{(1)}} \frac{\partial x^{(1)}}{\partial x} = \frac{\partial \ell}{\partial x^{(1)}} W^{(1)}$$
$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial x^{(1)}} \frac{\partial x^{(1)}}{\partial b} = \frac{\partial \ell}{\partial x^{(1)}}$$
$$\frac{\partial \ell}{\partial W^{(1)}} = \frac{\partial \ell}{\partial x^{(1)}} \frac{\partial x^{(1)}}{\partial W^{(1)}} = x \frac{\partial \ell}{\partial x^{(1)}}.$$

The easy trick for computing these derivative is to "guess an expression" so that the dimensions line up both sides. However, this trick does not always work, especially when you are dealing with more complicated functions.

A more formal way to compute these derivatives requires expanding them out. For example, naively applying the chain rule would yield $\frac{\partial \ell}{\partial W^{(1)}} = \frac{\partial \ell}{\partial x^{(1)}} \frac{\partial x^{(1)}}{\partial W^{(1)}}$ but it is not immediately clear what this expression means: note that the term on the right, $\frac{\partial x^{(1)}}{\partial W^{(1)}}$, is technically a 3D tensor, and we don't know how to manipulate them. So, we need to compute $\frac{\partial \ell}{\partial W^{(1)}}$ manually.

Since $x_k^{(1)} = \sum_l W_{kl}^{(1)} x_l + b_k$, we have

$$\begin{aligned}
\frac{\partial \ell}{\partial W_{ij}^{(1)}} &= \sum_k \frac{\partial \ell}{\partial x_k^{(1)}} \frac{\partial x_k^{(1)}}{\partial W_{ij}^{(1)}} \\
&= \sum_k \frac{\partial \ell}{\partial x_k^{(1)}} \left[\frac{\partial}{\partial W_{ij}^{(1)}} \left(\sum_l W_{kl}^{(1)} x_l + b_k \right) \right] \\
&= \sum_k \sum_l \frac{\partial \ell}{\partial x_k^{(1)}} \left(\delta_{ik} \delta_{jl} x_l \right) \\
&= \frac{\partial \ell}{\partial x_i^{(1)}} x_j
\end{aligned}$$

We can now stack these partial derivatives back into a matrix to get:

$$\left[\frac{\partial \ell}{\partial W^{(1)}} \right]_{ij} = \frac{\partial \ell}{\partial W_{ji}^{(1)}} = x_i \frac{\partial \ell}{\partial x_j^{(1)}} \implies \frac{\partial \ell}{\partial W^{(1)}} = x \frac{\partial \ell}{\partial x^{(1)}}.$$

(f) Suppose Φ is the identity map. Write down a single gradient descent update for $W_{t+1}^{(1)}$ and $W_{t+1}^{(2)}$ assuming step size ϵ. Your answer should be in terms of $W_t^{(1)}$, $W_t^{(2)}$, b_t, x, and y.

Solution:

$$W_{t+1}^{(1)} \leftarrow W_t^{(1)} - \epsilon (W_t^{(2)})^\top \left(W_t^{(2)} \left(W_t^{(1)} x + b_t \right) - y \right) x^\top$$
$$W_{t+1}^{(2)} \leftarrow W_t^{(2)} - \epsilon \left(W_t^{(2)} \left(W_t^{(1)} x + b_t \right) - y \right) (W_t^{(1)} x + b)^\top$$

Side note: The computation complexity of computing the $\frac{\partial \ell}{\partial W}$ for Equation (1) using the analytic derivatives and numerical derivatives is quite different!

For numerical differentiation, what we do is to use the following first order formula

$$\frac{\partial \ell}{\partial W_{ij}} = \frac{\ell\left(W_{ij} + \epsilon, \cdot\right) - \ell\left(W_{ij}, \cdot\right)}{\epsilon}.$$

We need $O(d^4)$ operations in order to compute $\frac{\partial \ell}{\partial W}$. On the other hand, it only takes $O(d^2)$ operations to compute it analytically.

3 Model Intuition

(a) What can go wrong if you just initialize all the weights in a neural network to exactly zero? What about to the same nonzero value?

Solution: The neural network will have an undesirable property: symmetry.

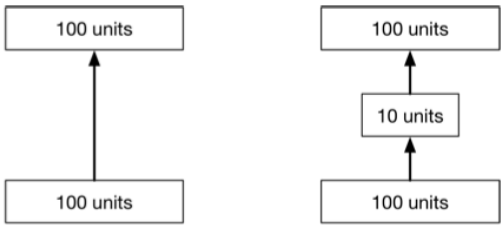
Perhaps the only property known [about initialization] with complete certainty is that the initial parameters need to "break symmetry" between different units. If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters. If they have the same initial parameters, then a deterministic learning algorithm applied to a deterministic cost and model will constantly update both of these units in the same way.

–page 301 of Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville

(b) Adding nodes in the hidden layer gives the neural network more approximation ability, because you are adding more parameters. How many weight parameters are there in a neural network with architecture specified by $d = \left[d^{(0)}, d^{(1)}, ..., d^{(N)}\right]$, a vector giving the number of nodes in each of the N layers? Evaluate your formula for a 2 hidden layer network with 10 nodes in each hidden layer, an input of size 8, and an output of size 3.

Solution: The number of parameters for the connections between two layers is: $d^{(i)} \times d^{(i+1)}$. Note that if we were computing the number of weights and biases, there would be an additional +1 in the formula, which would become: $(d^{(i)} + 1) \times d^{(i+1)}$. The total number of weight parameters in the architecture specified by d is: $\sum_{i=0}^{N-1} d^{(i)} \times d^{(i+1)}$. Applying this formula to the supplied network gives: $8 \times 10 + 10 \times 10 + 10 \times 3 = 210$ weights.

(c) Consider the two networks in the image below, where the added layer in going from Network A to Network B has 10 units with linear activation. Give one advantage of Network A over Network B, and one advantage of Network B over Network A.



Solution: Adding a linear-activated hidden layer does not make a network more powerful. It actually limits the functions that the network can represent.

Possible advantages of Network A over Network B: 1) A is more expressive than B. Specifically, it can learn any function that B can learn, plus some additional functions. 2) A has fewer layers, so it is less susceptible to problems with exploding or vanishing gradients.

Possible advantages of Network B over Network A: 1) B has fewer parameters, so it is less prone to overfitting. 2) B is computationally cheaper because a matrix-vector product of size

10×100 followed by one of size 100×10 requires fewer operations than one of size 100×100.
3) B has an embedding (or bottleneck) layer, so the network is forced to learn a more compact representation.