Topics

- Review of gradient descent
- Loss functions
- Multivariable Chain Rule
- Basic neural networks (preview)
- Backpropagation

These notes are loosely based on ILA Ch 9.2. Reviewing CalcBLUE 2 Ch. 5 on the chain rule is recommended.

Backpropagation – Motivation

Last class, we studied the unconstrained optimization

$$\text{minimize } f(\underline{x}) \qquad (P)$$

over $\underline{x} \in \mathbb{R}^n$, where we look for the $\underline{x} \in \mathbb{R}^n$ that makes the value of the cost function $f: \mathbb{R}^n \to \mathbb{R}$ as small as possible. We saw that one way to find either a local or global minimum $\underline{x}^*$ is gradient descent. Starting at an initial guess $\underline{x}^{(0)}$, we iteratively update our guess via

$$\underline{x}^{(k+1)} = \underline{x}^{(k)} - s \nabla f(\underline{x}^{(k)}), \quad k = 0, 1, 2, \ldots \qquad (GD)$$

where $\nabla f(\underline{x}^{(k)}) \in \mathbb{R}^n$ is the gradient of f evaluated at the current guess, and $s > 0$ is a step size chosen large enough to make progress towards $\underline{x}^*$, but not so big as to overshoot.

Today, we'll focus our attention on optimization problems (P) for which the cost function takes the following special form

$$f(\underline{x}) = \sum_{i=1}^{N} f_i(\underline{x}), \quad (S)$$

i.e, cost functions f that decompose into a sum of N "sub-costs" f_i. Problems with cost functions of the form (S) are particularly common in machine learning.

For example, a typical problem setup in machine learning is as follows (we saw an example of this when we studied least squares for data-fitting). We are given a set training data $(\underline{z}_i, \underline{y}_i)$, $i = 1, \ldots, N$, composed of "inputs" $\underline{z}_i \in \mathbb{R}^p$ "outputs" $\underline{y}_i \in \mathbb{R}^m$. Our goal is to find a set of weights $\underline{x} \in \mathbb{R}^n$ which parameterize a model such that $m(\underline{z}_i; \underline{x}) \approx \underline{y}_i$ on our training data. A common way of doing this is to minimize a loss function of the form

$$\text{loss}((\underline{z}_i, \underline{y}_i); \underline{x}) = \frac{1}{N} \sum_{i=1}^{N} \ell(m(\underline{z}_i; \underline{x}) - \underline{y}_i), \qquad (L)$$

Where each term $\ell(m(\underline{z}_i; \underline{x}) - \underline{y}_i)$ is a term penalizing the difference between our model prediction $m(\underline{z}_i; \underline{x})$ on input $\underline{z}_i$ and the observed output $\underline{y}_i$. In this setting, the loss function (L) takes the form (S), with $f_i = \frac{1}{N} \ell(m(\underline{z}_i; \underline{x}) - \underline{y}_i)$ the error between our prediction $\hat{\underline{y}}_i = m(\underline{z}_i; \underline{x})$ and the true output $\underline{y}_i$.

A common choice for the "sub-loss" function is $\ell(\underline{e}) = \|\underline{e}\|^2$, leading to a least-squares regression problem, but note that most other choices of loss function are compatible with the following discussion.

Now, suppose that we want to implement gradient descent (GD) on the loss function (L). Our first step is to compute the gradient $\nabla_{\underline{x}} \, loss((\underline{z}_i, y_i); \underline{x})$. Because of the sum structure of (L), we have that:

$$\nabla_{\underline{x}} \, loss((\underline{z}_i, y_i); \underline{x}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\underline{x}} \, \ell(m(\underline{z}_i; \underline{x}_i) - y_i),$$

i.e., the gradient of the loss function is the sum of the gradients of the "sub-losses" on each of the $i = 1, \dots, N$ data points.

Our task now is therefore to compute the gradient $\nabla_{\underline{x}} \, \ell(m(\underline{z}_i; \underline{x}_i) - y_i)$. This requires the multivariable chain rule, as $f_i(\underline{x}) = \ell(m(\underline{z}_i; \underline{x}_i) - y_i)$ is a composition of the functions $\ell(\underline{e})$, $\underline{e} = \underline{w} - y_i$, and $\underline{w} = m(\underline{z}_i; \underline{x})$.

<u>The Multivariate Chain Rule</u> (CalcBLUE 2 Ch. 5).

We begin with a reminder of the chain rule for scalar functions. Let $f: \mathbb{R} \to \mathbb{R}$ and $g: \mathbb{R} \to \mathbb{R}$ be differentiable functions. Then for $h(x) = g(f(x))$, we have that

$$h'(x) = g'(f(x)) \, f'(x). \qquad (C1)$$

If we define $g = g(f)$, and $f = f(x)$, then we can rewrite (C1) as $\frac{dh}{dx} = \frac{dg}{df} \cdot \frac{df}{dx}$. This is a useful way of writing things as we can "cancel" df on the RHS to check that our formula is correct.

WARNING: $\frac{dh}{dx} = \frac{dg}{df} \cdot \frac{df}{dx}$ is shorthand for $\frac{dh}{dx}(x) = \frac{dg}{df}(f(x)) \frac{df}{dx}(x)$. The evaluation points matter!

Generalizing slightly, suppose now that $f: \mathbb{R}^n \to \mathbb{R}$ maps a vector $\underline{x} \in \mathbb{R}^n$ to $f(\underline{x}) \in \mathbb{R}$. Then for $h(\underline{x}) = g(f(\underline{x}))$, we have:

$$\nabla_{\underline{x}} h(\underline{x}) = g'(f(\underline{x})) \, \nabla f(\underline{x}), \qquad (C2)$$

which we see is a natural generalization of equation (C1). It will be convenient for us later to define $\frac{df}{d\underline{x}} = \nabla_{\underline{x}} f(\underline{x})^T$ and $\frac{dh}{d\underline{x}} = \nabla_{\underline{x}} h(\underline{x})^T$. Again defining $g = g(f)$ and $f = f(\underline{x})$, we can rewrite (C2) as $\frac{dh}{d\underline{x}} = \frac{dg}{df} \cdot \frac{df}{d\underline{x}}$, which looks exactly the same as before!

WARNING: $\frac{dh}{d\underline{x}} = \frac{dg}{df} \cdot \frac{df}{d\underline{x}}$ is shorthand for $\frac{dh}{d\underline{x}}(\underline{x}) = \frac{dg}{df}(f(\underline{x})) \frac{df}{d\underline{x}}(\underline{x})$. The evaluation points matter!

Now, let's apply these ideas to computing the gradient of $h(\underline{x}) = \ell(m(\underline{z}_i; \underline{x}) - y_i)$, where we'll assume for now that $m(\underline{z}_i; \underline{x}), y_i \in \mathbb{R}$. Applying (C2), we get

$$\nabla_{\underline{x}} h(\underline{x}) = \ell'(m(\underline{z}_i; \underline{x}) - y_i) \cdot \nabla_{\underline{x}} (m(\underline{z}_i; \underline{x}) - y_i) = \ell'(m(\underline{z}_i; \underline{x}) - y_i) \cdot \nabla_{\underline{x}} m(\underline{z}_i; \underline{x})$$

where we use that $D_{\underline{x}} y_i = 0$ (since its a constant). Without knowing more about the functions l and m, this is all we can say.

<u>Example</u>: Suppose $l(e) = \frac{1}{2}e^2$ and $m(\underline{z}_i; \underline{x}) = \underline{x}^T \underline{z}_i$. Then

$$l(m(\underline{z}_i;\underline{x}) - y_i) = \frac{1}{2}(\underline{x}^T\underline{z}_i - y_i)^2 \text{ and } \nabla_{\underline{x}}\, l(m(\underline{z}_i;\underline{x}) - y_i) = \underbrace{(\underline{x}^T\underline{z}_i - y_i)}_{l'(m-y_i)} \cdot \underbrace{\underline{z}_i}_{D_{\underline{x}} m}$$

Next class we will have a brief introduction to deep learning. In deep learning, the function $m(\underline{z}_i;\underline{x})$ is often parameterized as a chain of function compositions:

$$m(\underline{z}_i;\underline{x}) = \underline{m}_L(\underline{m}_{L-1}(\cdots(\underline{m}_2(\underline{m}_1(\underline{z}_i))\cdots)) \qquad (\text{DN})$$
$$= \underline{m}_L \circ \underline{m}_{L-1} \circ \cdots \circ \underline{m}_2 \circ \underline{m}_1(\underline{z}_i).$$

A more suggestive way of writing this parameterization (that also highlights the dependence on $\underline{x}$) is

$$\begin{aligned}
\underline{o}_0 &= \underline{z}_i \\
\underline{o}_1 &= \underline{m}_1(\underline{o}_0; \underline{x}_1) && \underline{o}_1 \in \mathbb{R}^{p_1},\ \underline{o}_0 \in \mathbb{R}^{p_0} \qquad (\text{DNN}) \\
\underline{o}_2 &= \underline{m}_2(\underline{o}_1; \underline{x}_2) && \underline{o}_2 \in \mathbb{R}^{p_2},\ \underline{o}_1 \in \mathbb{R}^{p_1} \\
&\ \ \vdots && \qquad \vdots \\
\underline{o}_L &= \underline{m}_L(\underline{o}_{L-1}; \underline{x}_L) && \underline{o}_L \in \mathbb{R}^{p_L},\ \underline{o}_{L-1} \in \mathbb{R}^{p_{L-1}}
\end{aligned}$$

Here the model parameters $\underline{x} = (\underline{x}_1, \ldots, \underline{x}_L)$ are split across the layers $1, \ldots, L$. The intermediate outputs $\underline{o}_i$ can be of different dimension, as can the layer parameters $\underline{x}_i$. Writing (DN) as (DNN) highlights why these functions are called deep neural networks as the number of layers L increases. Our goal is then to compute $D_{\underline{x}}\, l(m(\underline{z}_i;\underline{x}) - \underline{y}_i)$ for m of the form (DN), and where $m, y_i \in \mathbb{R}^{p_L}$ are now also possibly vector-valued. To do this, we need the fully general multivariable chain rule.

For $h(\underline{x}) = \underline{g}(\underline{f}(\underline{x}))$, with vector-valued $\underline{f}: \mathbb{R}^n \to \mathbb{R}^p$ and $\underline{g}: \mathbb{R}^p \to \mathbb{R}^m$, we need to define the Jacobian matrices for $\underline{f}$ and $\underline{g}$:

$$\frac{d\underline{f}}{d\underline{x}} = \begin{bmatrix} \frac{df_1}{d\underline{x}} \\ \vdots \\ \frac{df_p}{d\underline{x}} \end{bmatrix} = \begin{bmatrix} \partial f_1/\partial x_1 & \cdots & \partial f_1/\partial x_n \\ \vdots & \ddots & \\ \partial f_p/\partial x_1 & \cdots & \partial f_p/\partial x_n \end{bmatrix} \text{ and } \frac{d\underline{g}}{d\underline{f}} = \begin{bmatrix} \frac{dg_1}{d\underline{f}} \\ \vdots \\ \frac{dg_m}{d\underline{f}} \end{bmatrix} = \begin{bmatrix} \partial g_1/\partial f_1 & \cdots & \partial g_1/\partial f_p \\ \vdots & \ddots & \vdots \\ \partial g_m/\partial f_1 & \cdots & \partial g_m/\partial f_p \end{bmatrix} \qquad (\text{J})$$

as the $p \times n$ and $m \times p$ matrices of partial derivatives, respectively.

We'll use our same intuition of "cancelling" to derive the expression:

$$\frac{d\underline{h}}{d\underline{x}} = \frac{d\underline{g}}{\partial \underline{f}} \cdot \frac{d\underline{f}}{\partial \underline{x}} \qquad (C3)$$

Note that (C3) is defined by a matrix-matrix multiplication of an $m \times p$ and $p \times n$ matrix, meaning $\frac{d\underline{h}}{d\underline{x}} \in \mathbb{R}^{m \times n}$. The claim is that $(i,j)^{th}$ entry of $\frac{d\underline{h}}{d\underline{x}}$ is the rate of change of $h_i(\underline{x}) = g_i(f(\underline{x}))$ with respect to x_j. From (J) and (C3), we have

$$\left(\frac{d\underline{h}}{d\underline{x}}\right)_{ij} = \underbrace{\frac{d g_i}{d\underline{f}}}_{i^{th}\text{ row of } \frac{d\underline{g}}{d\underline{f}}} \cdot \underbrace{\begin{bmatrix} \partial f_1 / \partial x_j \\ \vdots \\ \partial f_p / \partial x_j \end{bmatrix}}_{j^{th}\text{ column of } \frac{d\underline{f}}{\partial \underline{x}}} = \frac{d g_i}{\partial f_1}\frac{\partial f_1}{\partial x_j} + \cdots + \frac{\partial g_i}{\partial f_p}\frac{\partial f_p}{\partial x_j},$$

which is precisely the expression we were looking for. The "cancellation rule" tells us each term in the sum is computing the part of $\frac{\partial g_i}{\partial x_j}$ in the "f_i" channel.

We can apply this formula recursively to our function class (DN) to obtain the formula:

$$\frac{d\underline{m}}{d\underline{x}} = \frac{d\underline{m}_L}{d\underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{m}_{L-2}} \cdots \frac{d\underline{m}_1}{d\underline{m}_2} \cdot \frac{d\underline{m}_1}{\partial \underline{x}} \qquad (MC),$$

which is a fully general matrix chain rule. We'll use (MC) next to explore the key ideas behind backpropagation, which has been a key technical enabler of contemporary deep learning.

Backpropagation

We are going to work out how to efficiently compute the gradient of

$$\ell(\underline{m}(\underline{z}_i; \underline{x}) - \underline{y}_i)$$

when $\underline{m}$ takes the form in (DNN). We'll further assume, as is often the case in deep learning, that each layer function $\underline{m}_i$ takes the following form:

$$\underline{m}_i(\underline{o}_{i-1}; \underline{x}_i) = \sigma\left(X_i \begin{bmatrix} \underline{o}_i \\ 1 \end{bmatrix}\right)$$

where X_i is a $\mathbb{R}^{p_i \times (n_i+1)}$ matrix with entries given by $\underline{x}_i \in \mathbb{R}^{p_i(n_i+1)}$, and σ is a pointwise nonlinearity $\sigma(\underline{x}) = (\sigma(x_1), \ldots, \sigma(x_n))$ called an activation function (more on these next class).

Applying our matrix chain rule to $\ell(\underline{m}(\underline{x})-\underline{y}_i)$ (we won't write $\underline{z}_i$ to save space), we get the expression

$$\frac{\partial \ell}{\partial \underline{x}} = \frac{\partial \ell}{\partial \underline{m}} \cdot \frac{\partial \underline{m}}{\partial \underline{x}} = \frac{\partial \ell}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdots \frac{\partial \underline{m}_2}{\partial \underline{m}_1} \cdot \frac{\partial \underline{m}_1}{\partial \underline{x}}.$$

Here, $\frac{\partial \ell}{\partial \underline{m}}$ is a p_L dimensional row vector, and $\frac{\partial \underline{m}_i}{\partial \underline{m}_{i-1}}$ is a $p_i \times p_{i-1}$ matrix.

In modern applications, the layer dimensions, also called layer widths, p_i can be very large (on the order of 100s of thousands or even millions), meaning the $\frac{\partial \underline{m}_i}{\partial \underline{m}_{i-1}}$ matrices are **very very** large! Too large to store in memory actually.

Fortunately, since $\frac{\partial \ell}{\partial \underline{m}}$ is a row vector, we can build $\frac{\partial \ell}{\partial \underline{x}}$ by sequentially computing inner products. For example, if $\frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} = [\underline{a}_1 \cdots \underline{a}_{p_{L-1}}]$

$$\frac{\partial \ell}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} = \underbrace{\left[\text{---} \ \frac{\partial \ell}{\partial \underline{m}_L} \ \text{---} \right]}_{1 \times p_L} \underbrace{\begin{bmatrix} \underline{a}_1 & \cdots & \underline{a}_{p_{L-1}} \end{bmatrix}}_{p_L \times p_{L-1}}$$

$$= \left[\frac{\partial \ell}{\partial \underline{m}_L} \underline{a}_1 \ \cdots \ \frac{\partial \ell}{\partial \underline{m}_L} \underline{a}_{p_{L-1}} \right],$$

meaning we only ever need to store $\frac{\partial \ell}{\partial \underline{m}_L}$ and $\underline{a}_i$ in memory at any given time, which is only $2p_L$ numbers, as opposed to $p_L \times p_{L-1}$ #s! Then once we've computed $\frac{\partial \ell}{\partial \underline{m}_L}\frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}}$, which is now a p_{L-1} dim. row vector, we can continue our way down the chain.

What's left to do is compute the partial derivatives! Let's break down $\frac{\partial \ell}{\partial \underline{x}}$ into partial derivatives with respect to a layer's parameters $\underline{x}_i$. For layer L, we have:

$$\frac{\partial \ell}{\partial \underline{x}_L} = \frac{\partial \ell}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{x}_L} + \frac{\partial \ell}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{x}_L} = \frac{\partial \ell}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{x}_L}$$

Since $\underline{x}_L$ appears in the last layer, it shows up right away in the first term above, which is the derivative of $\underline{m}_L(\underline{m}_{L-1}; \underline{x}_L)$ with respect to $\underline{x}_L$ (the 2nd argument). The second term,

$$\frac{\partial \ell}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{x}_L} = 0$$

which measures how $\underline{m}_L$ changes with respect to changes in $\underline{m}_{L-1}$ caused by changes in $\underline{x}_L$ is zero because $\underline{m}_{L-1}$ does not depend on $\underline{x}_L$ at all! This is a key observation in the back-propagation algorithm!

Let's proceed to computing the derivative with respect to the parameters $\underline{x}_{L-2}$:

$$\frac{\partial l}{\partial \underline{x}_{L-1}} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \left(\frac{\partial \underline{m}_{L-1}}{\partial \underline{x}_{L-1}} + \cancelto{0}{\frac{\partial \underline{m}_{L-1}}{\partial \underline{m}_{L-2}} \cdot \frac{\partial \underline{m}_{L-2}}{\partial \underline{x}_{L-1}}} \right)$$

$$= \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{x}_{L-1}}$$

We see again that we can "stop" once we hit the layer that depends explicitly on $\underline{x}_{L-1}$. In general, we have:

$$\frac{\partial l}{\partial \underline{x}_L} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{x}_L}$$

$$\frac{\partial l}{\partial \underline{x}_{L-1}} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{x}_{L-1}} \qquad \left(\frac{\partial l}{\partial \underline{m}_{L-1}} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \right)$$

$$\frac{\partial l}{\partial \underline{x}_{L-2}} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{m}_{L-2}} \cdot \frac{\partial \underline{m}_{L-2}}{\partial \underline{x}_{L-2}} \qquad \left(\frac{\partial l}{\partial \underline{m}_{L-2}} = \frac{\partial l}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{m}_{L-2}} \right)$$

$$\vdots$$

$$\frac{\partial l}{\partial \underline{x}_j} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{m}_{L-2}} \cdot \frac{\partial \underline{m}_{L-2}}{\partial \underline{x}_{L-2}} \cdots \frac{\partial \underline{m}_{j+1}}{\partial \underline{m}_j} \cdot \frac{\partial \underline{m}_j}{\partial \underline{x}_j} \qquad \left(\frac{\partial l}{\partial \underline{m}_{j-1}} = \frac{\partial l}{\partial \underline{m}_j} \cdot \frac{\partial \underline{m}_j}{\partial \underline{m}_{j-1}} \right)$$

Notice that there is a lot of reuse of expressions, which means we don't have to recompute things over and over. In particular

$$\frac{\partial l}{\partial \underline{m}_{L-1}} = \frac{\partial l}{\partial \underline{m}_L} \cdot \frac{\partial \underline{m}_L}{\partial \underline{m}_{L-1}}, \qquad \frac{\partial l}{\partial \underline{m}_{L-2}} = \frac{\partial l}{\partial \underline{m}_{L-1}} \cdot \frac{\partial \underline{m}_{L-1}}{\partial \underline{m}_{L-2}},$$

and in general

$$\frac{\partial l}{\partial \underline{m}_{j-1}} = \frac{\partial l}{\partial \underline{m}_j} \cdot \frac{\partial \underline{m}_j}{\partial \underline{m}_{j-1}},$$

where $\frac{\partial l}{\partial \underline{m}_j}$ will have been computed at the layer above. This is another key piece of backpropagation!

The only thing left to compute is $\frac{\partial \underline{m}_j}{\partial \underline{x}_j}$ — this is now just an exercise in calculus, so we'll not work it out in class, but the online notes will provide links to pages with further information for those interested

Optional:

We apply our chain rule (with $\underline{w} = X_j \begin{bmatrix} \underline{a}_{j-1} \\ 1 \end{bmatrix}$) to get

$$\frac{\partial \underline{m}_j}{\partial \underline{x}_j} = \frac{\partial}{\partial \underline{x}_j} \underline{\sigma}\left(X_j \begin{bmatrix} \underline{a}_{j-1} \\ 1 \end{bmatrix} \right) = \frac{\partial \underline{\sigma}}{\partial \underline{w}} \cdot \frac{\partial \underline{w}}{\partial \underline{x}_j}.$$

Now for $\underline{\sigma}(\underline{w}) = \begin{bmatrix} \sigma(w_1) \\ \vdots \\ \sigma(w_{p_{j-1}}) \end{bmatrix}$, $\frac{\partial \underline{\sigma}}{\partial \underline{w}} = \begin{bmatrix} \sigma'(w_1) & & \\ & \ddots & \\ & & \sigma'(w_{p_{j-1}}) \end{bmatrix}$. Next, we need to find

$$\frac{\partial \underline{w}}{\partial \underline{x}_j} = \frac{\partial}{\partial \underline{x}_j} \left(X_j \begin{bmatrix} \underline{a}_{j-1} \\ 1 \end{bmatrix} \right).$$ This can be computed using multilinear algebra (tensors).

We won't work it out, but note that it can be found efficiently.