MNIST classification in numpy

We will consider classifying MNIST with a simple NN of the form

$$\hat{z}^i = W^i z^i + b^i \quad , \quad i = 1, \ldots, k-1$$
$$z^i = \sigma(\hat{z}^i) \qquad\quad i = 2, \ldots, k-1$$

where $z_1 = x$, where x is an input to the network, k is # of layers & we let

$$h(x, \theta) := \hat{z}^k \quad \leftarrow \text{ I think we want } h(x,\theta) = SM(\hat{z}^k)$$

Here, $\theta = ((W^i)_i, (b^i)_i)$ represent all params & $x \mapsto h(\cdot, \theta)$ is the input/output map of the network.

Let N = # of classes.

We will classify "goodness" of θ using the cross entropy loss. Given an input/label pair (x, y), $x \in \mathbb{R}^n$, $y \in \{0, \ldots, N\}$, the cross entropy loss is given by

$$CE(h(x,\theta), y) = -\sum_{i=1}^{N} (1_y)_i \ln(h(x,\theta)_i)$$

indicator array $= \begin{cases} 1 & \text{if } i = y \\ 0 & \text{else} \end{cases}$

$$= -\ln(\underbrace{h(x,\theta)_y}_{\text{the } y\text{-th coordinate of } h})$$

We want to optimize θ by using gradient descent on the empirical risk.

Let $D = ((x_i, y_i))_i$ denote the MNIST training set. The empirical risk is given by

$$L(\theta) = \sum_{(x,y)\in D} \underbrace{CE(h(x,\theta), y)}_{=: \ell(\theta, (x,y))}$$

The gradient of L is given by

gradient w.r.t. θ, of course.

$$\nabla L(\theta) = \sum_{(x,y)\in D} \underbrace{\nabla CE(h(x,\theta), y)}_{(*)\ \cancel{=: \ell(\theta,(x,y))}}$$

Computing $(*)$ for each (x,y) is costly. If we choose a random $(\tilde{x},\tilde{y}) \in D$ (uniform random in D) ~~then~~ & let $g = \nabla CE(h(\tilde{x},\theta)\, \tilde{y})$, then $\mathbb{E}\, g = \frac{1}{|D|} \nabla L(\theta)$. We would like to use SGD,

$$\theta_{t+1} = \theta - \alpha g, \qquad g = \nabla CE(\cdot), \ (\tilde{x},\tilde{y}) \sim U(D)$$

However, drawing a random sample from D each iteration of SGD is costly. Instead, we will shuffle D randomly & simply iterate through it:

$$\theta_{t+1} = \theta - \alpha g_i, \qquad g_i = \cancel{\nabla CE(h} \ \nabla \ell(\theta, (x_i, y_i)).$$

we need to compute $\nabla_\theta CE(h(x,\theta), y)$. To make things unambiguous, we'll use the following notation.

$$\frac{\partial}{\partial\theta} f(g(\theta))$$ means the derivative of $h = f\circ g$ taken w.r.t. θ.

$$D_z f(z)$$ means ~~derivative of f~~ Derivative of f in usual sense

Basically, $\frac{\partial}{\partial\theta}$ means we're going to have to use the chain rule to evaluate. D_z means we're not taking the derivative of a composition. They really mean the same thing, but I'm hoping this gives context. We also still use $\frac{\partial}{\partial x_i}$ for standard partials. (... Maybe use ∂_θ to suggest chain rule?)
Want to compute

$$\frac{\partial}{\partial\theta} CE\left(h(x,\theta), y\right) = \frac{\partial}{\partial\theta} -\ln\left[h(x,\theta)\right]_y$$

we'll need to compute ~~the following derivatives~~ derivatives of the following functions.

- σ
- linear
- softmax
- CE.

In all subsequent derivations, we use the following

for $f : \mathbb{R}^n \to \mathbb{R}^m$,

$$D_z f(z)\Big|_{z=w} = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \vdots & & \\ \frac{\partial f_m}{\partial z_1} & \cdots & \frac{\partial f_m}{\partial z_n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

where each partial is evaluated at the pt w

Sometimes, we use the shorthand $D_z f(w)$.
↑ base point

For $f \circ g \circ h$ (composition), the chain rule is given by

$$\frac{\partial}{\partial z} f(g(h(z))) = \cancel{D_w f(g(\ldots))}$$

$$= D_w f(w)\Big|_{w=g(h(z))} D_w g(w)\Big|_{w=h(z)} D_w h(w)\Big|_{w=z}$$

In shorthand:

$$= D f(g(h(z)))\; D g(h(z))\; D h(z)$$

To Do: prove chain rule sometime?

Derivation of σ

In an abuse of notation, we use σ to mean the Scalar Sigmoid function applied elementwise. To clarify, here (and only here) we'll say,

$$\sigma : \mathbb{R} \to (0,1)$$

is

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

and

$$\vec{\sigma} : \mathbb{R}^n \to \mathbb{R}^n$$

is given elementwise by

$$\vec{\sigma}_i(z) = \sigma(z_i).$$

Note

$$\frac{\partial \vec{\sigma}_i(z)}{\partial z_j} = \begin{cases} \sigma'(z_i) & j = i \\ 0 & \text{else} \end{cases}$$

Hence

$$D\vec{\sigma}(z) = \operatorname{diag}(\sigma'(z_1) \ldots, \sigma'(z_n)).$$

To ~~compl~~ be complete (so we can code it) lets compute σ'.

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz} \underbrace{\cancel{\tfrac{1}{1+e^{-z}}}}_{(1+e^{-z})^{-1}} = -\frac{1}{(1+e^{-z})^2}(-e^{-z}) = \frac{e^{z}}{(1+e^{-z})^2}$$

use chain rule.

$$= \underbrace{\frac{e^{z}}{1+e^{-z}}}_{\overset{(*)}{=} 1-\sigma(z)} \; \underbrace{\frac{1}{1+e^{-z}}}_{=\sigma(z)}.$$

(*) Note: $1 - \dfrac{e^{z}}{1+e^{-z}} = \dfrac{1}{1+e^{-z}} = \sigma(z)$

~~$= 1+e^{-z} - e^{-z}$~~

Hence $\dfrac{e^{z}}{1+e^{-z}} = 1-\sigma(z)$

We should already be able to compute σ (stably) from the forward pass. So, now we have a complete $D\vec{\sigma}$.

Derivation of Softmax

$$SM(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Note:

(1) $$\frac{\partial SM(x)_i}{\partial x_l} = \frac{-e^{x_i}}{\left(\sum e^{x_j}\right)^2} e^{x_l} \qquad l \neq i$$

(2) $$\frac{\partial SM(x)_i}{\partial x_i} = \frac{e^{x_i}}{\sum_j e^{x_j}} + -e^{x_i}\left(\sum_j e^{x_j}\right)^2 e^{x_i}$$

Note that (2) corresponds to diagonals of DSM

Let

$$c := \left(\sum_j e^{x_j}\right)^{-2}$$

$$M := \underbrace{\exp(x)\ \exp(x)^T}_{\text{outer product between the vectors}}$$

$$v = \exp(x)\left(\sum e^{x_j}\right)^{-1}$$

Then

$$D\,SM(x) = \cancel{c(M+v)}$$
$$c\left(M + \mathrm{diag}(v)\right)$$

Derivative of cross entropy

$$CE(p, q) = -\sum q_i \ln p_i$$

We want the derivative w.r.t. p.

$$D_p\, CE(p, q) = -\sum_i \frac{q_i}{p_i} \quad \Leftarrow \quad \frac{d}{dx} \ln(x) = \frac{1}{x}$$

Derivative of Linear function

Let ~~$f(x) = W$~~ $f(x, W, b) = Wx + b$

We must compute $D_x f$, $D_W f$, and $D_b f$.

1. $D_x f$

Note that $f(x)_i = W_i^T x + b_i$

$\hat{}$ ith row of W

$$\frac{\partial f(x)_i}{\partial x_j} = W_{ij}$$

$$\Longrightarrow \quad D_x f = W$$

2. $D_W f$

We will treat W as a big column vector, so $D_W f$ takes the form

Let $W \in \mathbb{R}^{m \times n}$

$$D_W f = \begin{bmatrix} \frac{\partial f_1}{\partial W_{11}} & \cdots & \frac{\partial f_1}{\partial W_{1n}} & \cdots & \frac{\partial f_1}{\partial W_{m1}} & \cdots & \frac{\partial f_1}{\partial W_{mn}} \\ \vdots & & & & & & \vdots \\ \frac{\partial f_m}{\partial W_{11}} & & & \cdots & & & \frac{\partial f_1}{\partial W_{mn}} \end{bmatrix}$$

abuse of notation. omit nonactive args.

Note that $f(w)_i = w_i^T x + b_i$.

$$\frac{\partial f(w)_i}{\partial w_{jk}} = \begin{cases} 0 & j \neq i \\ x_k & \text{else} \end{cases}$$

Hence

$$\left[\frac{\partial f(w)_i}{\partial w_{j1}} \cdots \frac{\partial f(w)_i}{\partial w_{jn}} \right] = \begin{cases} 0 & j \neq i \\ x^T & j = i \end{cases}$$

Hence

$$D_w f = \begin{bmatrix} x^T & 0 & \cdots & 0 \\ 0 & x^T & \cdots & 0 \\ 0 & 0 & \cdots 0 & x^T \end{bmatrix}$$

3. $D_b f$

$$f(b)_i = w_i^T x + b_i$$

$$\frac{\partial f(b)_i}{\partial b_j} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases} \implies D_b f = I_m$$

Finally, how do we compute derivatives w.r.t. b^i and W^i.

Recall the previous discussion of reverse-mode autodiff.

Let $J(\cdot)$ denote our cost function (w.r.t. some fixed data).

For concreteness, suppose we have a network w/ 3 layers

$$h(x;\theta) = \underbrace{W^3\sigma(\underbrace{W^2\sigma(\underbrace{W^1x+b^1}_{\hat z^1})+b^2}_{\hat z^2})+b^3}_{\hat z^3}$$

We have

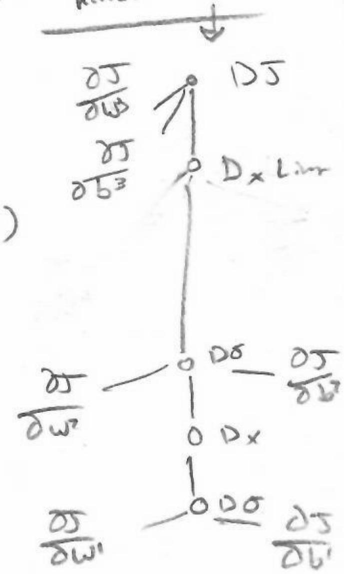
computation of derivative looks kinda like:

$$\frac{\partial J}{\partial W^3} = \underbrace{DJ(\hat z^3)}\, D_W \text{Linear}(\sigma(\hat z^2))$$

$$\frac{\partial J}{\partial W^2} = \underbrace{DJ(\hat z^3)\, D_x \text{Linear}(\sigma(\hat z^2))\, D\sigma(\hat z^2)}_{=:\, Q}\, D_W \text{Linear}(\sigma(\hat z^1))$$

=: Q, for lack of a better name

$$\frac{\partial J}{\partial W^2} = Q\, D_x\text{Linear}(\sigma(\hat z^1))\, D\sigma(\hat z^1)\, D_W\text{Linear}(x)$$



Note how as you work backwards computing the derivatives w.r.t. parameters, you can reuse previous derivative computations. Also, the base points for derivatives, $\hat z^1$, $\sigma(\hat z^1)$, etc. are saved from the forward pass.

Appendix: log sum exp & Stable Softmax

Suppose you wish to compute the softmax of $x \in \mathbb{R}^n$

$$\mathrm{Sm}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

If any elements of x are very large, you will run into overflow when computing e^{x_i} or $\sum_j e^{x_j}$. To fix this, you can use the following trick:

$$\frac{e^{x_i}}{\sum_j e^{x_j}} = \exp\left(\ln\left(\frac{e^{x_i}}{\sum e^{x_j}}\right)\right)$$

$$= \exp\left(x_i - \underbrace{\ln \sum_j e^{x_j}}_{(*)}\right)$$

$(*)$ Can be computed stably by recentering as follows. For any $c \in \mathbb{R}$ we have

$$\ln \sum_j e^{x_j} = \ln e^c \sum e^{x_j - c} = c + \ln \sum e^{x_j - c}$$

If you choose $c = \max\{x_1, \ldots, x_n\}$, this tends to work well. I dont think max vs min vs mean makes a huge difference. Mostly you want to get what goes into the exponent centered close to zero. I think...

Observations:

Final form for numerically stable softmax is

$$\mathrm{Sm}(x)_i = \exp\left(x_i - c - \ln \sum_j e^{x_j - c}\right), \quad c = \max\{x_1, \dots, x_n\}$$

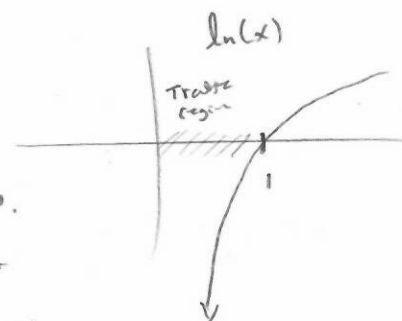
Note 3 things:

- Inside the sum, $e^{x_i - c}$ should be less prone to overflow/underflow because $x_i - c$ should hopefully be close to zero. (Maybe not if $\min x_i \ll 0$ & $\max x_i \gg 0$, but that seems like it shouldn't arise often

- In the outer exp, we have $\exp(\underbrace{x_i - c} + \text{stuff})$

  This should be more numerically stable for the same reason

- In the log we have $\ln \sum_j e^{x_j - c}$.



We would like the argument to $\ln(\cdot)$ to be ≥ 0, to avoid the singularity and instability in log near 0. This is guaranteed by our choice of c, since for $x_k \in \operatorname{argmax}\{x_1, .., x_n\}$ $e^{x_k - c} = 1$ & $e^{x_j - c} > 0$ for all other j.