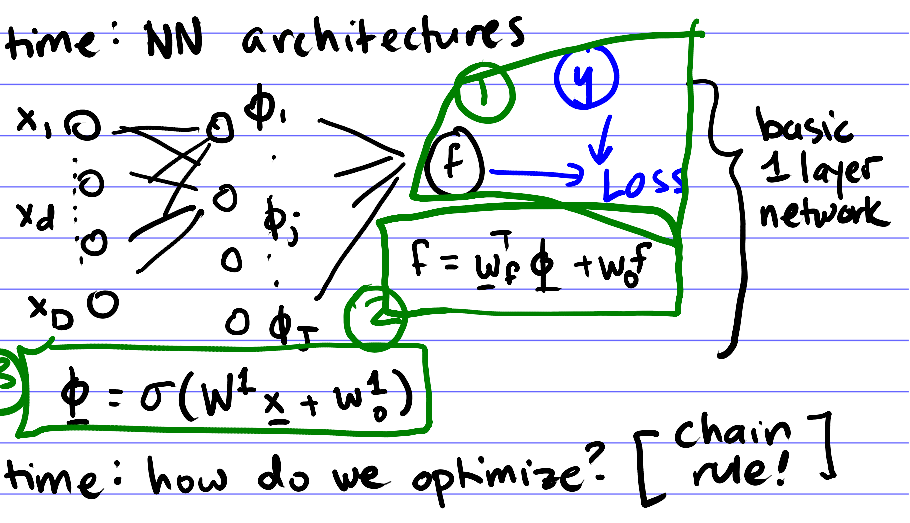
CS181 - Neural Nets, Part II

So far: ✓ supervised learning (basics)
✓ model selection

TODAY * more expressive models (NNs)
more losses

Notes: Book 4.4 (please pull / download)
Also: HW3, Beyond CS181 sec.

Last time: NN architectures



This time: how do we optimize? [chain rule!]

(1) Let's start our opt. w/ the last layer/last output part.

- if regression: $\mathcal{L}(W) = \frac{1}{2}\sum_n (y_n - f_n)^2$

(note: we could choose something else..)

$\frac{\partial \mathcal{L}}{\partial W} =$

chain rule: $\sum_n \underbrace{\frac{\partial \mathcal{L}}{\partial f_n}}_{-(y_n - f_n)} \nabla_W (f_n)$ } only place where the final loss choice appears

- classification: same idea:

$p_1 = p(y=1|x) = \frac{1}{1+\exp\{-f_n\}}$

$p_0 = p(y=0|x) = \frac{1}{1+\exp\{f_n\}}$

$\mathcal{L}(W) = \sum_n y_n \log(1+\exp\{-f_n\}) + (1-y_n)\log(1+\exp\{f_n\})$

(we got this by $-\log p(y|x) = -\log \prod_n p_1^{y_n} p_0^{1-y_n}$)

(note: this could have been hinge, etc.)

$\frac{\partial \mathcal{L}}{\partial W} = \sum_n \frac{\partial \mathcal{L}}{\partial f_n}\frac{\partial f_n}{\partial W}$

$\frac{y_n \exp\{-f_n\}}{1+\exp\{-f_n\}}(-1) + \frac{(1-y_n)\exp\{f_n\}}{1+\exp\{f_n\}}$

We can calculate this! analytic form!
except: depends on f_n.

step 1: given current W, compute f_n's.
step 2: use those f_n's to compute derivatives.

(2) Quick detour on vector/matrix chain rule:

suppose: $y = f(u)$
$u = g(v)$
$v = h(x)$

$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$ ← basic chain rule (if all scalars)

Suppose: x is a vector of size D

$\nabla_x y = \underbrace{\frac{\partial y}{\partial u}}_{\text{scalars}} \underbrace{\frac{\partial u}{\partial v}}_{\text{scalars}} \underbrace{\nabla_x v}_{\text{vector of size D (size of x)}}$

now suppose $\underset{\text{size: } J'}{u}, \underset{J}{v}, \underset{D}{x}$ are vectors:

$\nabla_x y = \underbrace{\nabla_u y}_{\text{vec of size 1 by } J'} \underbrace{J_v[u]}_{\text{matrix } J' \times J} \underbrace{J_x[v]}_{\text{matrix } J \times D}$

Returning to gradient for w_f

$f_n = \underline{w}_f^T \underline{\phi}_n + w_0^f$

$\frac{\partial \mathcal{L}_n}{\partial w_f} = \underbrace{\frac{\partial \mathcal{L}_n}{\partial f_n}}_{\text{already did this!}} \cdot \underbrace{\frac{\partial f_n}{\partial \underline{w}_f}}_{\text{just } \underline{\phi}_n!}$

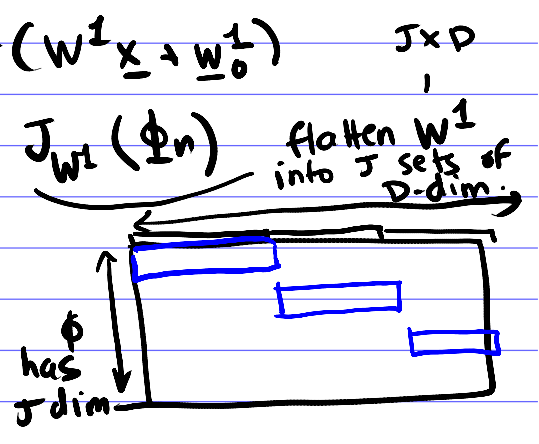
(Note: as before, we need to do a forward pass to compute $\underline{\phi}_n$)

w/ L^2 loss is $\underbrace{-(y_n - f_n)}_{\text{scalar}}$ (e.g. regression,

(3) Now, finally, gradients w.r.t. W^1

recall that $\underline{\phi}_n = \sigma(W^1 \underline{x} + \underline{w}_0^1)$

$\nabla_{W^1} \mathcal{L}_n() = \underbrace{\frac{\partial \mathcal{L}_n}{\partial f_n}}_{\text{scalar (reg. } -(y_n - f_n))} \underbrace{\nabla_{\underline{\phi}_n} f_n}_{w_f} \underbrace{J_{W^1}(\underline{\phi}_n)}_{\text{flatten } W^1 \text{ (J×D) into J sets of D-dim.}}$



$\underline{\phi}_{nj} = \sigma\left(\sum_d W^1_{jd} x_{nd} + w^1_j\right)$

$\frac{\partial \phi_{nj}}{\partial W^1_{jd}}$

ϕ is J dim

So now: what about multiple layers?

1. step 1: forward pass: compute

$\underline{x} \xrightarrow[W^1]{} \underline{\phi}^1 \xrightarrow[W^2]{} \phi^2 \to \cdots \phi^\ell \xrightarrow[W^\ell]{} \cdots \xrightarrow[W^L]{} \phi^L \to f$

2. step 2: backward pass: chain rule!

forward: $\phi^{\ell-1} \to \phi^\ell(\phi^{\ell-1}; W^\ell) \to \phi^\ell$

$\nabla_{\phi^{\ell-1}} \mathcal{L} = \underbrace{\nabla_{\phi^\ell} \mathcal{L} \cdot J_{\phi^{\ell-1}}(\phi^\ell)}_{\text{compute grad to pass back}}$

$\nabla_{\phi^\ell} \mathcal{L}$

$\underbrace{\frac{\partial \mathcal{L}}{\partial W^\ell} = \nabla_{\phi^\ell} \mathcal{L} \cdot J_{W^\ell}(\phi^\ell)}_{\text{get grad for our layer}}$

Notes: 1) above is called reverse-mode diff. (start at f_n, went back) → good for ∇_W, reuses computation ✓)

2) another approach is forward-mode: change in x forward to change in f (still chain rules..) can be useful for a local gradient check/question that doesn't need all ∇_W..

Why does this all work?

- computation (GPUs)
- data (large datasets)
- optimization: overparam makes opt. easier.