Last time: Chain Rule, logistic regression

$$p(y=1 \mid x; w) = \frac{1}{1+\exp\{-f(x,w)\}}$$

$$p(y=0 \mid x; w) = \frac{1}{1+\exp\{f(x,w)\}}$$

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

notice: could plug in other expressions for $p(y=1|x)$, $p(y=0|x)$

wrote loss in terms of f_n,

$$\frac{\partial \mathcal{L}_n}{\partial w} = \frac{\partial \mathcal{L}_n}{\partial f_n}\frac{\partial f_n}{\partial w} \qquad \text{e.g. } f_n = w^T x_n + w_0$$

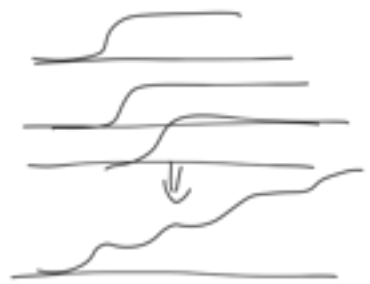
Applied this idea to shallow network (adaptive basis regression)

$$f_n = w^T \phi$$

$$\phi_j = \sigma(w_j^{1T} x + w_j^1) \quad \} \text{"basis"}$$

nonlinearity $\Rightarrow$ sigmoid/logit; ReLu/hinge; tanh

deep networks:



How are we going to fit these models? $(W^\ell, w^\ell, w) \Rightarrow$ lots of params!

problem: very non-convex! (can be a real issue → in practice:
lots of things to make the problem easier ⇒ random restarts,
may use various forms of momentum/stochasticity, overparameterization)

How to opt. NNs with SGD, key thing we need is the gradient of the loss w.r.t. params.

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

$$\frac{\partial}{\partial f_n}\mathcal{L}_n = \boxed{-(y_n - f_n)}$$

Aside: vector-valued chain rule

scalar chain rule: $y = f(u) = f(g(h(x)))$
$u = g(v)$
$v = h(x)$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial g}{\partial v}\frac{\partial h}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$$

now, suppose x was a vector. $\quad \nabla_x f = \underbrace{\frac{\partial f}{\partial u}\frac{\partial u}{\partial v}}\underbrace{\nabla_x v}_{\text{vector}}$

scalars

if v was also a vector:

$$\nabla_x f = \underbrace{\frac{\partial f}{\partial u}}_{\text{scalar}} \underbrace{\nabla_v[u]}_{\substack{\text{vector of} \\ \text{size } J}} \underbrace{J_x[v]}_{\substack{\text{matrix} \\ J \times D}} \qquad x \in \mathbb{R}^D \quad v \in \mathbb{R}^J$$

if u were also a vector:

$$\nabla_x f = \nabla_u f \; J_v[u] \; J_x[v]$$

Now we're ready to start optimizing. Let's start w/ w: $\boxed{x \xrightarrow[W^1]{} \phi \xrightarrow[w]{} f}$

$$\nabla_w \mathcal{L}_n = \underbrace{\frac{\partial \mathcal{L}_n}{\partial f_n}}_{-(y_n - f_n)} \nabla_w f_n \qquad f_n = \boxed{w^T \phi_n + w_0} \quad \nabla_w f_n = \phi_n \qquad = \boxed{\underbrace{-(y_n - f_n)}_{\text{scalar}} \underbrace{\phi_n}_{\substack{\text{vector of} \\ \text{size } J}}}$$

Notice! We need f_n to compute $\nabla_w \mathcal{L}_n$, and f_n depends on w.

1) for our current w, we compute f_n } feed-forward (calc values)
2) which allows us to compute $\nabla_w \mathcal{L}_n$ w.r.t. our current w. } backward pass (calc gradients)

$$\nabla_{W^1} \mathcal{L}_n = \underbrace{\frac{\partial \mathcal{L}_n}{\partial f_n}}_{\substack{= \\ -(y_n - f_n) \\ \text{scalar}}} \underbrace{\nabla_{\phi_n} f_n}_{\substack{w^T \\ \text{vector} \\ \text{of size} \\ J}} \overbrace{J_{W^1}[\phi_n]}^{J \times JD}$$

↳ lets flatten W^1 into W^1_{dj}

$$\phi_{nj} = \sigma\left(\sum_d W^1_{dj} x_d + w^1_j\right)$$

$$\frac{\partial \phi_{nj}}{\partial W^1_{dj'}} = 0 \text{ if } j \neq j'$$

$$= \sigma\left(\sum_d W^1_{dj} x_d + w^1_j\right) \cdot \left(1 - \sigma\left(\sum_d W^1_{dj} x_d + w^1_j\right)\right) \cdot x_d$$

we get a $J \times JD$ matrix

General approach: compute $\boxed{\mathcal{L}(\phi^L(\phi^{L-1}(\cdots \phi^1(x))))}$
for L layers

& store all intermediate values ϕ^ℓ

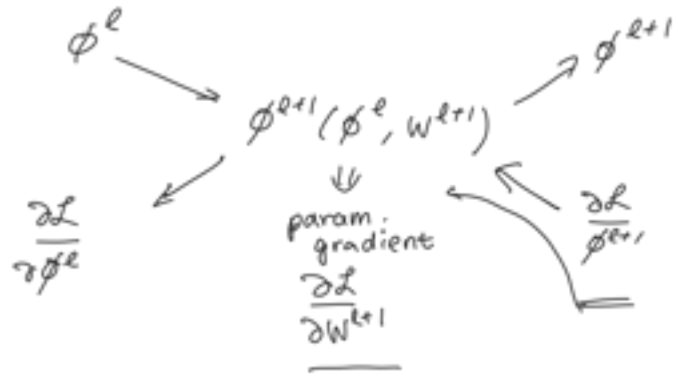
(forward pass)

Backward pass: compute $\frac{\partial \mathcal{L}}{\partial \phi^\ell} = \nabla_{\phi^L} \mathcal{L} \; J_{\phi^{L-1}}[\phi^L] \; J_{\phi^{L-2}}[\phi^{L-1}] \cdots J_{\phi^\ell}[\phi^{\ell+1}]$

if we want $\frac{\partial \mathcal{L}}{\partial \phi^{\ell-1}} = \frac{\partial \mathcal{L}}{\partial \phi^\ell} J_{\phi^{\ell-1}}[\phi^\ell]$

Compute parameter gradients $\frac{\partial \mathcal{L}}{\partial W^\ell} = \nabla_{\phi^L} \mathcal{L} \; J_{\phi^{L-1}}[\phi^L] \cdots J_{W^\ell}[\phi^\ell]$

forward pass



backward pass

computation graph

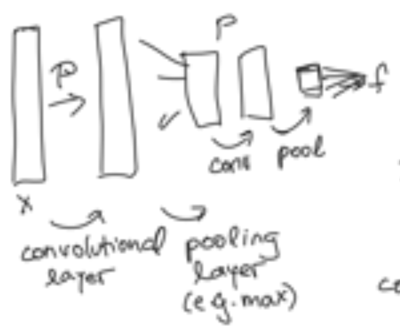
What's special about NNs now?

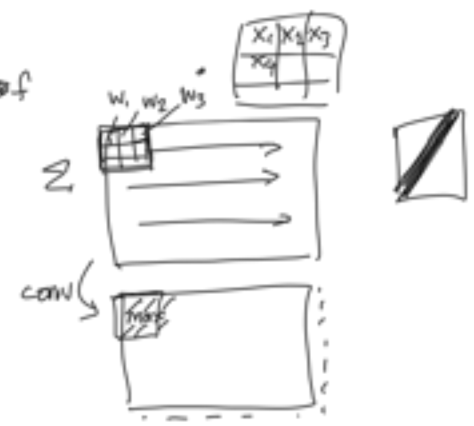
- ⇒ we have more data now
- ⇒ we have GPUs / software tools for autodiff
- ⇒ that said, lots of practicalities
  - ~~speed~~ overfitting: • L1/L2
    - • perturbations on inputs
    - • dropout
  - choices of activation (hinge/tanh, etc.) to avoid saturation
  - SGD: ways of variance reduction / subgradients / ADAM

Different Architectures:

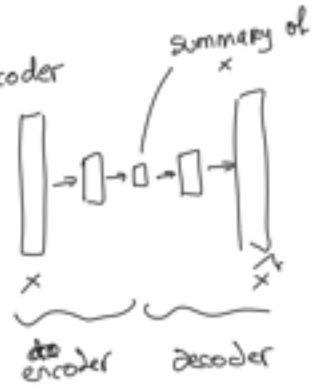
Feed forward Network, Multi-layer Perceptron

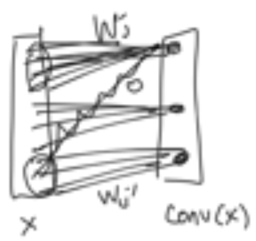
Convolutional NN (parameter tying)



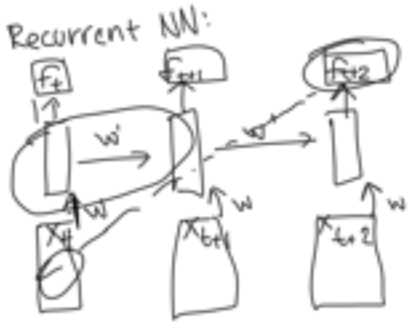


Autoencoder





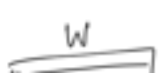
RNN: Recurrent NN:



(variant: long short-term memory - LSTM)

gated recurrent units GRU

Concept Exercise:



Neural networks 2

Suppose we have an image that is D 

1) how many params would be rqd in FF NN? Assume L layers, size M nodes, single output.

$$(DW)(M) + M^2(L-1) + M$$

2) conv net w/ patch size P, L layers (fully connected single output)

$$LP^2 + \underbrace{(W-LP)(D-LP)}_{\text{ignore for now}} / \text{some pooling factor}$$

3) RNN that travels from $d=0$ to $d=D$ w/ single output, H dim Hidden layer?