ML and Opt Lect 19

- Backprop for MLPs
- Automatic differentiation & ML frameworks
- Autoencoders
- Example of using Pytorch to learn a 2-layer classifier for Fashion MNIST

Let's give the relevant backprop formulas for the MLP case

$$o^l = \sigma(a^l) = \sigma(w^l o^{l-1} + b^l)$$ ← use this to relate $\nabla_{o^l} f$ to $\nabla_{w^l} f$ and $\nabla_{b^l} f$

$$o^{l+1} = \sigma(a^{l+1}) = \sigma(w^{l+1} o^l + b^{l+1})$$ ← use this to relate $\nabla_{o^l} f$ to $\nabla_{o^{l+1}} f$

We have, by the chain rule,

$$J_{o^{l+1}}(o^l) = \operatorname{diag}\left(\sigma'(a^{l+1})\right) \cdot w^{l+1} \in \mathbb{R}^{n_{l+1} \times n_l}$$

$$J_{o^l}(b^l) = \operatorname{diag}\left(\sigma'(a^l)\right) \in \mathbb{R}^{n_l \times n_l}$$

and $J_{o^l}(w^l) \in \mathbb{R}^{n_l \times (n_l \times n_{l-1})}$ satisfies

$$\left[J_{o^l}(w^l)\right]_{i,:} = \left[\frac{\partial (o^l)_i}{\partial w^l}\right] \in \mathbb{R}^{n_l \times n_{l-1}}$$

$$= \operatorname{diag}\left(\sigma'(a^l)\right) e_i (o^{l-1})^T$$

so we can show that for any vector $v \in \mathbb{R}^{n_l}$,

$$J_{o^l}(w^l)^T v = \text{diag}\left(\sigma'(a^l)\right) v \, (o^{l-1})^T \in \mathbb{R}^{n_l \times n_{l-1}}$$

Putting these pieces together

## Backprop for MLPs

$\text{temp} \leftarrow \nabla_{o^L} f$

$\nabla_{b^L} f = \text{diag}(\sigma'(a^L))\,\text{temp} + \lambda \nabla_{b^L} R_L$

$\nabla_{W^L} f = \text{diag}(\sigma'(a^L))\,\text{temp}\,(o^{L-1})^T + \lambda \nabla_{W^L} R_L$

for $l = L-1, \dots, 1$

$\text{temp} \leftarrow (W^{l+1})^T \text{diag}(\sigma'(a^{l+1}))\,\text{temp}$

$\nabla_{b^l} f \leftarrow \text{diag}(\sigma'(a^l))\,\text{temp} + \lambda \nabla_{b^l} R_l$

$\nabla_{W^l} f \leftarrow \text{diag}(\sigma'(a^l))\,\text{temp}\,(o^{l-1})^T + \lambda \nabla_{W^l} R_l$

Notes: (i) in practice we cache the preactivations $a^l$ during the forward pass

(ii) cost of forward & backward passes similar

(iii) costs of passes dominated by lin algebra ops (mat vec prods)

BP is a core computational tool for working with NNs. Become familiar with it — we will see that it gives rise to some important optimization phenomena when training NNs.

- We can implement BP for any computational DAG with (sub) differentiable neurons. In practice doing this by hand is tedious, error-prone, and requires careful optimizations for efficiency.

- Now we have available several high-quality ML frameworks whose main focus is to automate BP as much as possible: you specify how $f(x)$ is computed in a forward pass, then they deduce the backward pass using **autodifferentiation**. They also optimize the computations and support GPGPUs for much faster computation.

Common ML Frameworks:

- Pytorch (we use in this class, more popular now in research)

  2016 Facebook Research

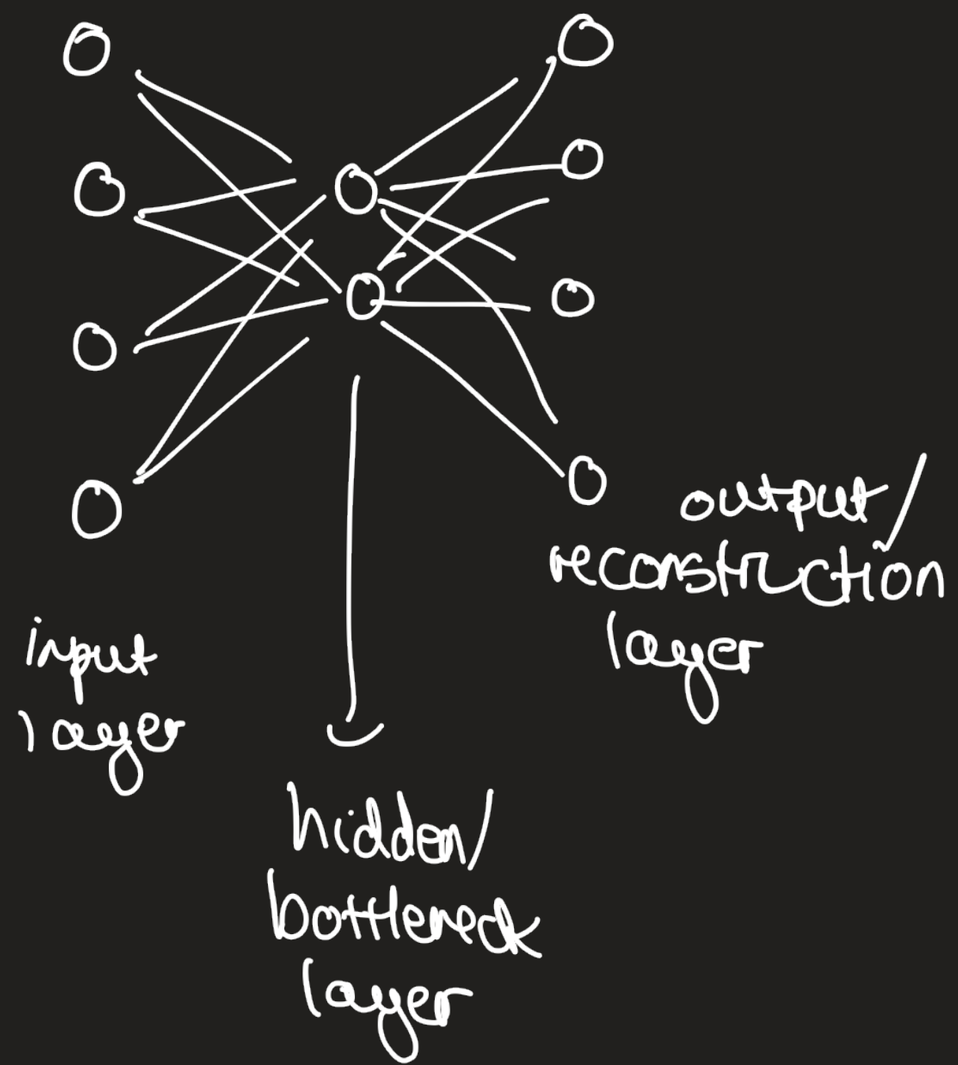
- Tensorflow (used the last time course was taught; more used in production)

  2015 Google Brain

# Autoencoders

- Nonlinear generalization of PCA & SVD

- Typically autoencoders are used to create useful features for a downstream task, in an unsupervised way




Goal: learn a (compressed) representation that does a good job of reconstructing the input

Ex $l_2$-loss $l(v,v) = \|v-v\|_2^2$

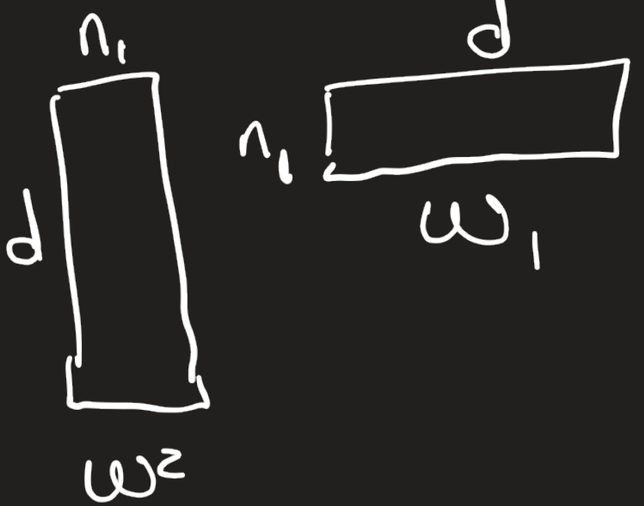
and $\sigma(x) = x$

$$\hat{y}(x) = o^2 = \sigma(w^2 o^1 + b^2)$$
$$= \sigma(w^2 \sigma(w^1 x + b^1) + b^2)$$
$$= w^2 w^1 x + w^2 b^1 + b^2)$$

fix $b^1 = b^2 = 0$ then

$$\hat{y}(x) = w^2 w^1 x$$

where




so the autoencoder learns $w^2, w^1$

to minimize $f(w^1, w^2) = \frac{1}{n} \sum_{i=1}^{n} \|\hat{y}(x_i) - x_i\|_2^2$

$$= \frac{1}{n} \sum_{i=1}^{n} \|w^1 w^2 x_i - x_i\|_2^2 = \frac{1}{n} \|w^1 w^2 X - X\|_F^2$$

where $X = [x_1 | \cdots | x_n]$

so fitting an autoencoder with $\ell(v,u) = \|v-u\|_2^2$ and $b^1 = b^2 = 0$ is equivalent to solving

$$W^2_*, W^1_* = \underset{\substack{W^2 \in \mathbb{R}^{n_2 \times d} \\ W^1 \in \mathbb{R}^{d \times n_1}}}{\operatorname{argmin}} \|W^2 W^1 X - X\|_F^2$$

this is the SVD!

Clearly autoencoders are a more flexible method of dim reduction than SVD because one can take a nonlinear $\sigma$ and nonzero biases.

# Deep autoencoders :