Last time

- Abridged history of deep learning
  - Charted the convergence of the key innovations that led to AlexNet in 2012
    - ⚠ only 13 years ago
      - model development (F)
      - backpropagation (opt)
      - GPUs (opt)
      - ImageNet (D)

Today:

- statistical learning framework
  - F (search space)
  - L (search criterion)
  - opt (search algorithm)
    - gradient descent

Def Statistical Learning

→ a principled process, informed by data, for the optimal selection of a function ($\hat{f}$) to accomplish a task (T).

Def Deep Learning

→ statistical learning with the set of candidate functions ($\mathcal{F}$) being neural networks

Statistical Learning Process

Step1: Selection of Search Space ($\mathcal{F}$)

→ First, we need to define which functions (f) we are choosing between.

→ Restrict the search to a certain set of functions $f \in \mathcal{F}$

e.g. T = regression : $\mathcal{F} = \{ \beta_0 + \beta_1 X : \beta_0, \beta_1 \in \mathbb{R} \}$

e.g. T = inference : $\mathcal{F} = \left\{ \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2} \left(\frac{x_i - \mu}{\sigma} \right)^2 \right\} : \mu \in \mathbb{R}, \sigma^2 \in \mathbb{R}^+ \right\}$

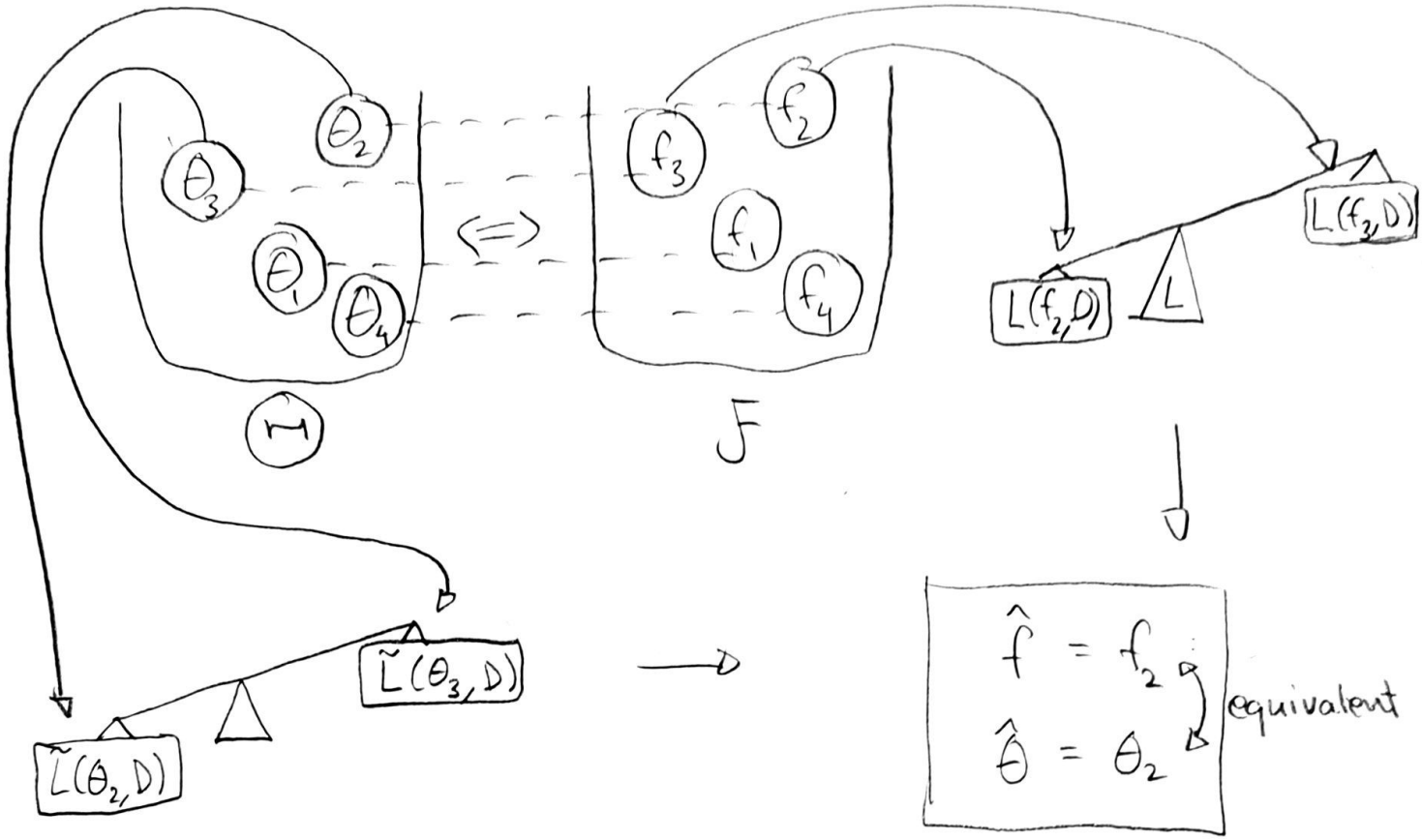
Step 2: Selection of Search Criterion (L)

- want to choose the best f among $f \in \mathcal{F}$
- need a way to evaluate task performance of each $f \in \mathcal{F}$
- Define a loss function $L(f, D)$ that scores a given $f \in \mathcal{F}$ on how (badly) it accomplishes the task on the observed data (D).
- Then search for the best (as measured by L) $f \in \mathcal{F}$. Ideally obtaining:

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; L(f, D)$$

$$\Leftrightarrow \; f_{\hat{\theta}} = \underset{\theta \in \Theta}{\operatorname{argmin}} \; L(f_\theta, D)$$

$$\Leftrightarrow \; \hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \; \tilde{L}(\theta, D)$$



Challenges

1. How to encode performance on task T into a loss function?
2. How to conduct the search of $\mathcal{F}$? (i.e. optimization)

① Loss functions (the creative bit)

→ Task: regression

→ mean squared error: $L(f, D) = \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2$

→ mean zero-one loss: $L(f, D) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{f(X_i) = Y_i\}$

→ Task: classification

→ cross entropy loss: $L(f, D) = \frac{1}{n} \sum_{i=1}^{n} - \sum_{c=1}^{C} \log(f_c(X_i)) \mathbb{I}\{Y_i = c\}$

→ misclassification rate: $L(f, D) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{\underset{c \in \{1, \dots C\}}{\text{argmax}} f_c(X_i) \neq Y_i\}$

Different choices for L will affect how each candidate is scored and may thus determine different models to be optimal.

② Optimization

Our task now becomes solving:

$$\hat{\theta} = \underset{\theta \in \mathcal{H}}{\text{argmin}} \; L(\theta, D)$$

using some algorithm, call it "opt", so

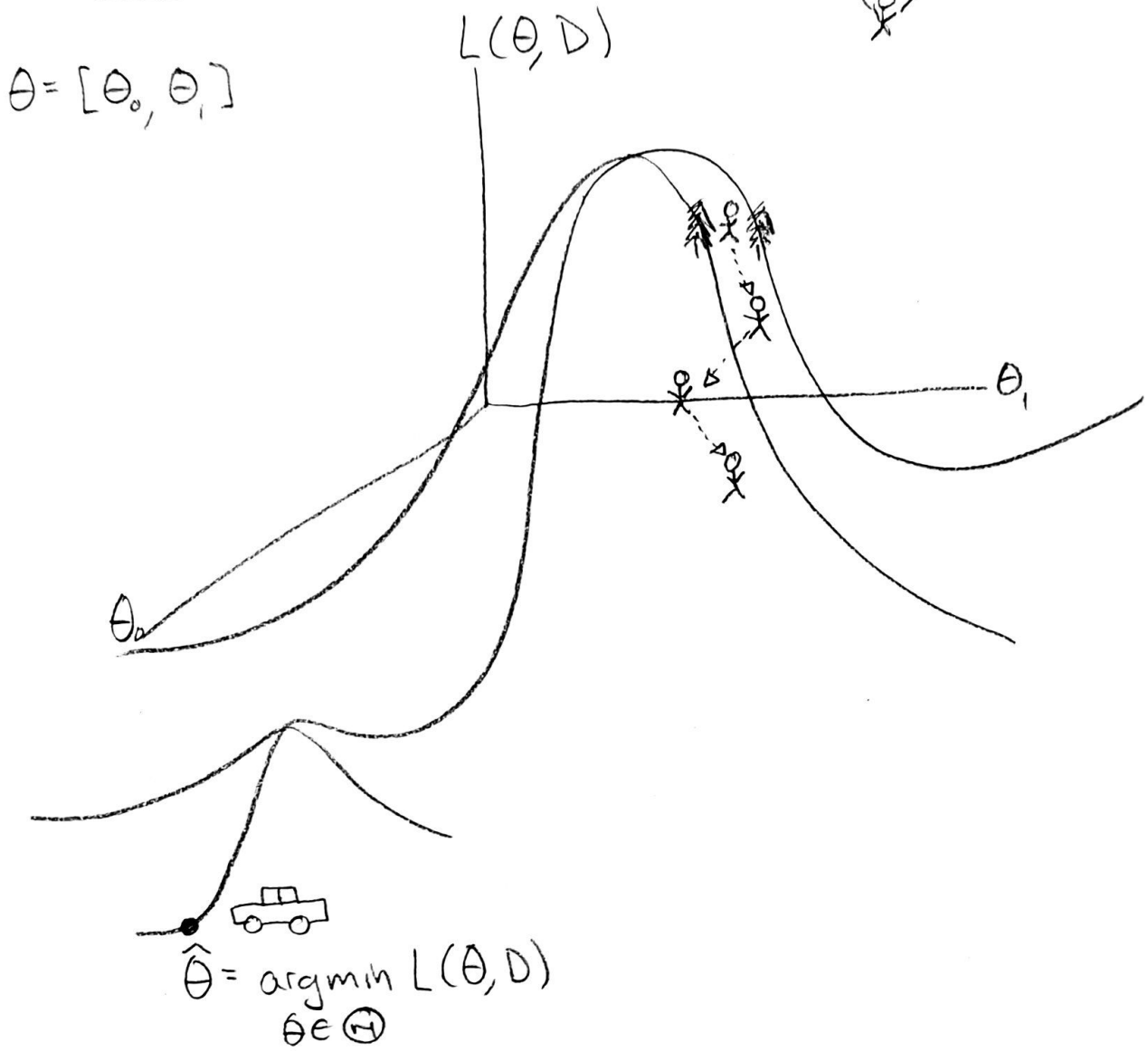
$$\hat{\hat{\theta}} = \underset{\theta \in \mathcal{H}}{\text{opt}} \; L(\theta, D)$$

where we would like $|L(\hat{\theta}, D) - L(\hat{\hat{\theta}}, D)|$ to be small.

Some approaches:

- Don't search $\mathcal{H}$, just pick a $\theta \in \mathcal{H}$ at random
- Exhaustive search: evaluate $L(\theta, D)$ $\forall \theta \in \mathcal{H}$
- Analytical solution (calculus)
- Random Search: evaluate $L(\theta, D)$ $\forall \theta \in \mathcal{H}, C \mathcal{H}$
- Gradient Descent & Variants
  - Stochastic Gradient Descent (SGD)
  - SGD + Momentum
  - RMSprop
  - Adam (SGD + momentum + RMSprop)

Gradient Descent



<u>Idea</u>: Imagine the loss function as a surface. You are standing on the surface and want to walk to the lowest point, but you can't see very far. Look down, determine in which direction the ground is sloped most steeply (the gradient), walk a step in this direction, then repeat.

Algorithm: given the function $L(\theta)$, with Jacobian (ie multivariate derivative) $\nabla_\theta L(\theta) = \frac{\partial L(\theta)}{\partial \theta}$, then $\theta^{(t+1)}$, the $(t+1)^{th}$ estimate of $\underset{\theta \in \Theta}{\operatorname{argmin}} L(\theta)$, is given as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_t \left[\nabla_\theta L(\theta)\right]\Big|_{\theta = \theta^{(t)}}, \quad \forall t = 0, \ldots T-1$$

where $\alpha_t > 0$ is the "step size" or "learning rate"

→ $\theta^{(0)}$ can be initialized randomly or using prior knowledge

→ T, the number of steps, can be chosen beforehand or can be a function of the algorithm's progress (ie a stopping rule)