Lecture Plan

* Overview of weakly supervised learning: a bridge between supervised and unsupervised learning
* Multi-task learning
  - Hard parameter sharing
  - Soft parameter sharing
* Transfer learning
  - Fine tuning

So far in the class, we have been studying supervised learning. This makes several assumptions:

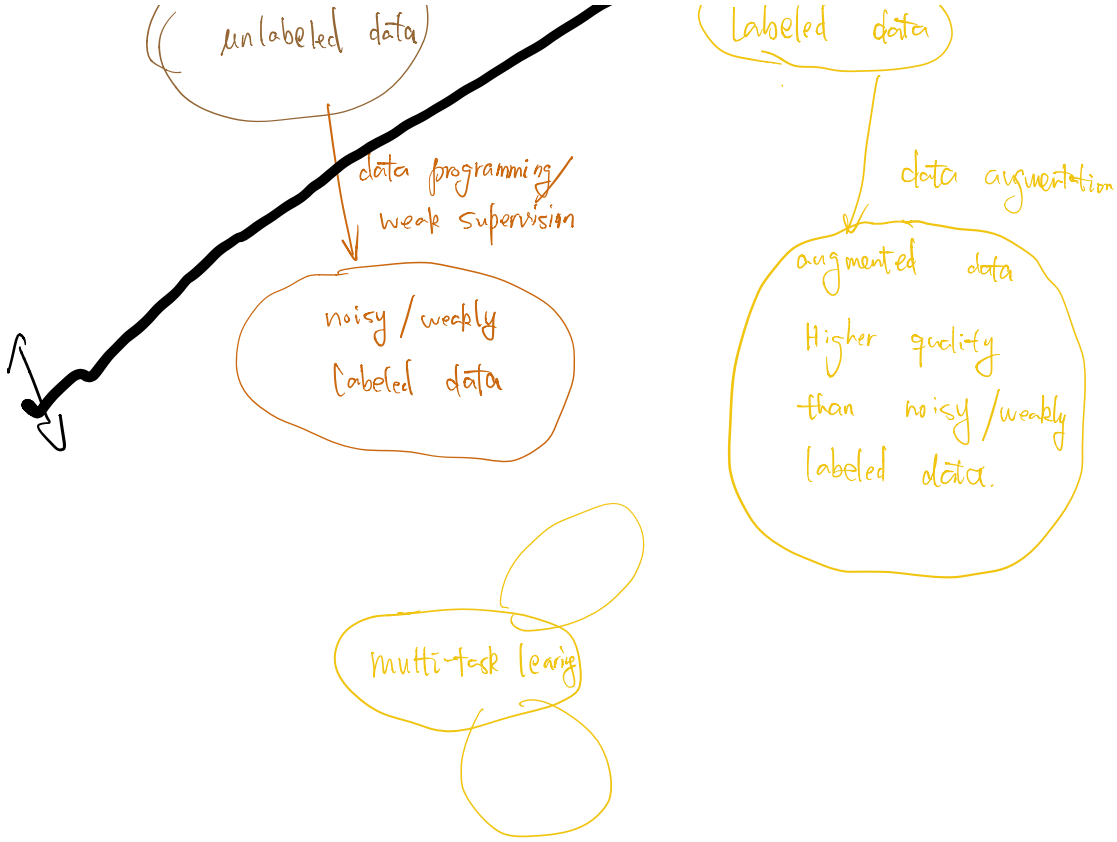
① * We have lots of labeled data already (Neural net training requires lots of labeled data)

② * The training data distribution and the test data distribution are the same.

The goal of weakly supervised learning is to address problem ①.

The goal of transfer learning or domain adaptation is to address problem ②.

A picture of weakly supervised learning



Crossing the boundary from supervised to unsupervised learning is a challenging task.

—— Self-supervised learning

Ex. Noisy student (Xie et al. CVPR 2020)

~ Imagine that we only have a limited amount of labeled data.

~ Bootstrap:

(i) we learn a neural net using the labeled data.

(ii) use the learned neural net to infer pseudo-labels over unlabeled data

(iii) re-train using the labeled data + pseudo-labeled data.

Ex. Masked next word prediction (BERT)

— Take the entire Wikipedia corpus (?!)

— Train a multi-layer (12/24) transformer that predicts the next word.

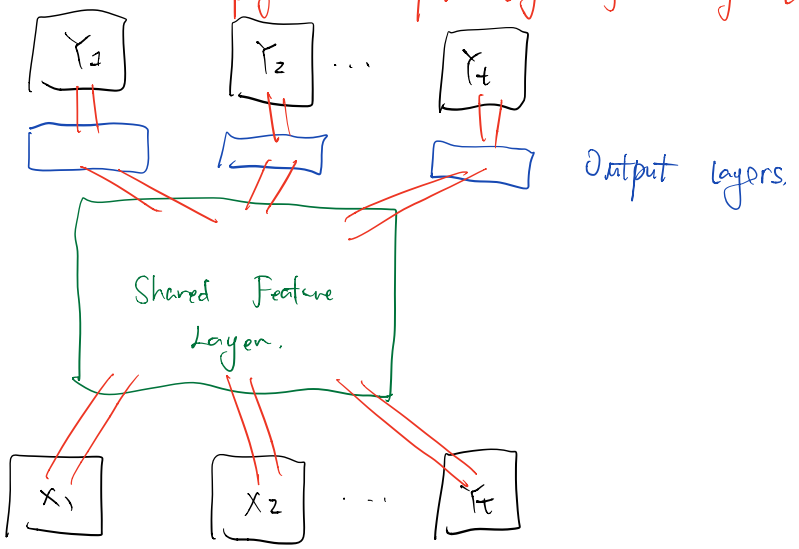
In both of these examples, the model is trained using zero or very little labeled data.

Multi-task learning : There are two approaches in neural nets

— Hard parameter sharing

— Soft parameter sharing

* Hard parameter sharing is generally applied by sharing the feature layers between all tasks while keeping an output layer for every task.



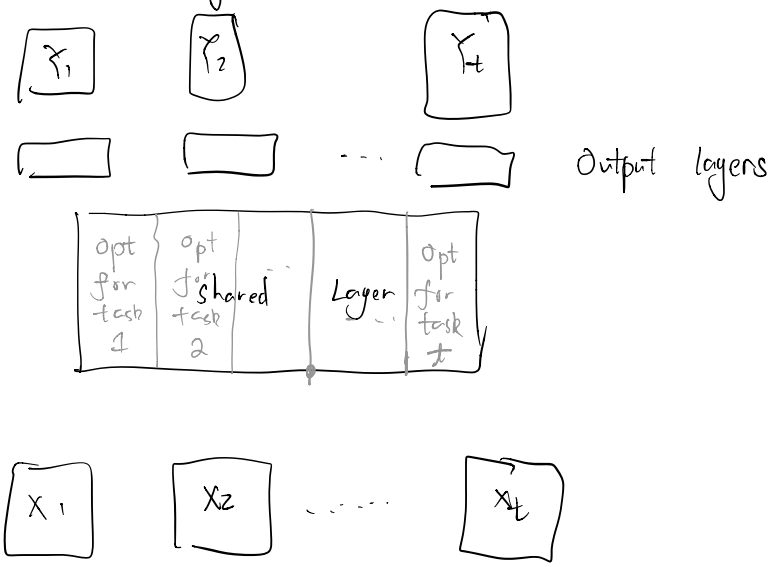
Ex. For text classification tasks, a commonly used shared feature layer is BERT.

Why does hard parameter sharing work ?

— The first reason is regularization.

Claim. By restricting the shared layer's size, one can reduce overfitting.

Setting. Consider a two-layer neural network:



Observation. If the shared layer is too large, then it can fit the optimal solution of every single task.

On the other hand, by restricting the shared layer' size, we encourage information sharing !

— The second reason is adding inductive bias from multiple tasks.

Question: the inductive bias depends on datasets' properties such as sample sizes, covariance matrices, and so on. How are they related ?

Claim. Hard parameter sharing helps by reducing variance, but hurts by increasing bias.

Ex. (Multi-label regression).

- We have t linear regression tasks.

Task 1: $x_1, \quad y_1^{(1)} = x_1^T \cdot \beta^{(1)} + \varepsilon_1^{(1)} \to N(0, b^2)$

$x_2, \quad y_2^{(1)} = x_2^T \cdot \beta^{(1)} + \varepsilon_2^{(1)}$

$\vdots$

$x_n, \quad y_n^{(1)} = x_n^T \cdot \beta^{(1)} + \varepsilon_n^{(1)}$

$\mathbb{E}[x_i x_i^T] = \Sigma$

In matrix notation, we write

$$Y^{(1)} = X \cdot \beta^{(1)} + \varepsilon^{(1)}$$

Task 2: $Y^{(2)} = X \cdot \beta^{(2)} + \varepsilon^{(2)}$

$\vdots$

Task t: $Y^{(t)} = X \cdot \beta^{(t)} + \varepsilon^{(t)}$

Recall: for single-task learning, we can solve a linear regression problem, and get that

$$\hat{\beta}_i = (X^T X)^{-1} X^T Y^{(i)}$$

and the test loss of $\hat{\beta}_i$ is

$$L(\hat{\beta}_i) = b^2 \cdot \mathrm{Tr}\left[\Sigma \cdot (X^T X)^{-1}\right], \text{ where } \Sigma \text{ is the population covariance matrix.}$$

Problem formulation for hard parameter sharing

$$f(A,B) = \sum_{i=1}^{t} \left\| X \cdot B \cdot A_i - Y^{(i)} \right\|^2$$

B: shared layer; A_i: output layer

Let $(\hat{A}, \hat{B})$ denote the global minimum of $f(A,B)$.

Hence the hard parameter sharing predictor for task i is:

$$\hat{\beta}_i^{HPS} = \hat{B} \cdot \hat{A}_i$$

Claim. The test loss of $\hat{\beta}_i^{HPS}$ is equal to

$$\mathcal{L}(B^* \cdot a_i^*) + \sigma^2 \cdot \|a_i^*\|^2 \cdot \mathrm{Tr}\left[\Sigma \cdot (X^T X)^{-1}\right].$$

where: $B^* = [\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(t)}]$.

$$A^* = \underset{\mathrm{rank}(U) = r}{\operatorname{argmin}} \langle UU^T, B^{*T} \Sigma B^* \rangle$$

$a_i^* \longrightarrow$ i-th column of $A^* A^{*T}$

$\mathcal{L}(B^* a_i^*)$ = test loss of $B^* \cdot a_i^*$.

Remark. This is a bias-variance decomposition of HPS!!

Bias: $\mathcal{L}(B^* \cdot a_i^*)$ does not depend on noise.

Variance: $\sigma^2 \cdot \|a_i^*\|^2 \cdot \mathrm{Tr}\left[\Sigma \cdot (X^T X)^{-1}\right]$

Ex. $\|a_i^*\|^2 \le 1$

$\leq \sigma^2 \cdot \mathrm{Tr}\left[\Sigma \cdot (X^T X)^{-1}\right]$

because A^* is a projection matrix

Hence, the bias of HPS is always larger than STA, the variance of HPS is always smaller than STA.

Proof sketch. $f(A,B) = \sum_{i=1}^{t} \| XBA_i - A^{(i)} \|^2$

Step 1. Derive the global minimum of $f(A,B)$:

- Using local optimality condition: $\frac{\partial f(A,B)}{\partial B} = 0$, we get B
- Applying B to $f(A,B)$, we get A.

$\Rightarrow \hat{\beta}_i^{HPS}$ predictor

Step 2. $\mathbb{E}_{X, \xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(n)}} \left[L(\hat{\beta}_i^{HPS}) \right]$

$= n \cdot \left\| \Sigma^{1/2} B^* \left(\underbrace{A^T (AA^T)^\dagger A - Id}_{} \right) \right\|_F^2 + \sigma^2 \cdot (\text{int.} \cdot \text{pr.})$.

The minimizer of this equation is precisely A^*.

Geometric intuition. HPS essentially finds a rank-r approximation of all the tasks!

Recap. In a simplified linear model, we have shown that

(i) HPS' inductive bias comes from reducing variance

(ii) also introduces bias

Explaining negative transfer.

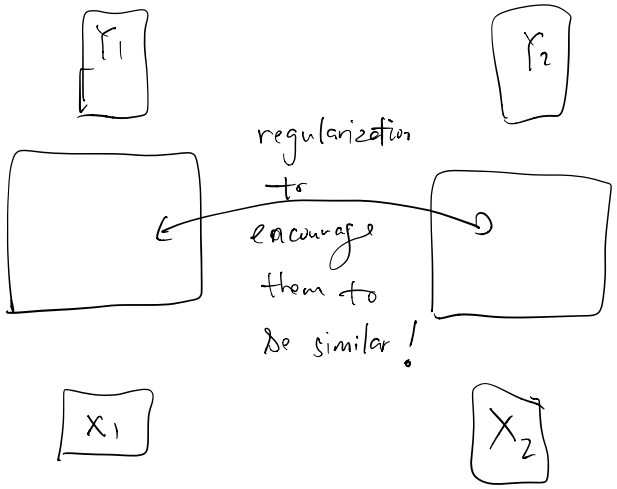
Research question: design optimization schedules for multi-task learning.

- Curriculum learning: start from easy examples, then go to more difficult examples!

Challenges: different sample sizes, covariates shifts, lots of room for improvement!

- Soft parameter sharing:

Ex.



- Transfer learning: In TL, you train a model on one task, and hope that the model can help learn the other task.

Fine-tuning:

- Train a model on the source task: $\hat{\theta}^{(S)}$
- On the target task,
  - Initialize from $\hat{\theta}^{(S)}$
  - Run SGD
- Early stopping / Checkpointing is crucial in the use of fine-tuning (T5, google)

Many exciting work in this direction:

- Meta learning. Can a model learn to adapt to new environments quickly? What happens to a new environment with little labeled data?
- Few shot learning. Suppose we have a tiny amount of data from various tasks but lots of data from some other sources. How can we use those data to predict the "small data" tasks?