Multi-Task & Meta-Learning Basics

CS 330
Logistics

Homework 1 posted today, due **Wednesday, October 9**

Fill out **paper preferences** by tomorrow.

TensorFlow review session **tomorrow, 4:30 pm in Gates B03**
Plan for Today

Multi-Task Learning
- Models & training
- Challenges
- Case study of real-world multi-task learning

— short break —

Meta-Learning
- Problem formulation
- General recipe of meta-learning algorithms
- Black-box adaptation approaches

} Topic of Homework 1!
Multi-Task Learning Basics
Some notation

Typical loss: negative log likelihood
\[ \mathcal{L}(\theta, \mathcal{D}) = -\mathbb{E}_{(x,y) \sim \mathcal{D}}[\log f_\theta(y \mid x)] \]

Single-task learning:
\[ \mathcal{D} = \{(x, y)_k\} \]

\[ \min_{\theta} \mathcal{L}(\theta, \mathcal{D}) \]

What is a task? (more formally this time)

A task:
\[ \mathcal{T}_i \triangleq \{p_i(x), p_i(y \mid x), \mathcal{L}_i\} \]

Corresponding datasets:
\[ \mathcal{D}_i^{tr} \quad \mathcal{D}_i^{test} \]

will use \( \mathcal{D}_i \) as shorthand for \( \mathcal{D}_i^{tr} \):
Examples of Tasks

A task: \( \mathcal{T}_i \triangleq \{p_i(x), p_i(y | x), \mathcal{L}_i\} \)

- data generating distributions

Corresponding datasets: \( \mathcal{D}_i^{tr} \quad \mathcal{D}_i^{test} \)

- will use \( \mathcal{D}_i \) as shorthand for \( \mathcal{D}_i^{tr} \).

- Multi-task classification: \( \mathcal{L}_i \) same across all tasks
  - e.g. per-language handwriting recognition
  - e.g. personalized spam filter

- Multi-label learning: \( \mathcal{L}_i, p_i(x) \) same across all tasks
  - e.g. CelebA attribute recognition
  - e.g. scene understanding

\[ L_{\text{tot}} = w_{\text{depth}} L_{\text{depth}} + w_{\text{kpt}} L_{\text{kpt}} + w_{\text{normals}} L_{\text{normals}} \]

When might \( \mathcal{L}_i \) vary across tasks?

- mixed discrete, continuous labels across tasks
- if you care more about one task than another
Objective: \[ \min_{\theta} \sum_{i=1}^{T} \mathcal{L}_i(\theta, \mathcal{D}_i) \]

A model decision and an algorithm decision:

How should we condition on \( z_i \)?

How to optimize our objective?

e.g. one-hot encoding of the task index

or, whatever meta-data you have

- personalization: user features/attributes
- language description of the task
- formal specifications of the task
Conditioning on the task

Let’s assume $z_i$ is the task index.

**Question:** How should you condition on the task in order to share as little as possible?
Conditioning on the task

\[ y = \sum_{j} 1(z_i = j) y_j \]

\[ \rightarrow \text{independent training within a single network!} \]

\[ \text{with no shared parameters} \]
The other extreme

Concatenate $\mathbf{z}_i$ with input and/or activations

all parameters are shared except the parameters directly following $\mathbf{z}_i$
An Alternative View on the Multi-Task Objective

Split $\theta$ into shared parameters $\theta^{sh}$ and task-specific parameters $\theta^i$

Then, our objective is:

$$\min_{\theta^{sh}, \theta^1, \ldots, \theta^T} \sum_{i=1}^{T} \mathcal{L}_i(\{\theta^{sh}, \theta^i\}, \mathcal{D}_i)$$

Choosing how to condition on $z_i$ is equivalent to choosing how & where to share parameters
Conditioning: Some Common Choices

1. **Concatenation-based** conditioning

Concatenation-based conditioning simply concatenates the conditioning representation to the input. The result is passed through a linear layer to produce the output.

2. **Additive** conditioning

Additive conditioning first maps the conditioning representation to a bias vector. The bias vector is then added to the input.

These are actually the same!

Conditioning: Some Common Choices

3. Multi-head architecture

Why might multiplicative conditioning be a good idea?
- more expressive
- recall: multiplicative gating

4. Multiplicative conditioning

Multiplicative conditioning generalizes independent networks and independent heads.

Conditioning: More Complex Choices

Cross-Stitch Networks. Misra, Shrivastava, Gupta, Hebert ’16

Multi-Task Attention Network. Liu, Johns, Davison ’18

Deep Relation Networks. Long, Wang ’15

Sluice Networks. Ruder, Bingel, Augenstein, Sogaard ’17
Unfortunately, these design decisions are like neural network architecture tuning:

- **problem dependent**
- largely guided by **intuition** or **knowledge** of the problem
- currently more of an **art** than a **science**
Optimizing the objective

Objective: \( \min_{\theta} \sum_{i=1}^{T} \mathcal{L}_i(\theta, \mathcal{D}_i) \)

Basic Version:

1. Sample mini-batch of tasks \( \mathcal{B} \sim \{\mathcal{T}_i\} \)
2. Sample mini-batch datapoints for each task \( \mathcal{D}_i^b \sim \mathcal{D}_i \)
3. Compute loss on the mini-batch: \( \hat{\mathcal{L}}(\theta, \mathcal{B}) = \sum_{\mathcal{T}_k \in \mathcal{B}} \mathcal{L}_k(\theta, \mathcal{D}_k^b) \)
4. Backpropagate loss to compute gradient \( \nabla_{\theta} \hat{\mathcal{L}} \)
5. Apply gradient with your favorite neural net optimizer (e.g. Adam)

Note: This ensures that tasks are sampled uniformly, regardless of data quantities.

Tip: For regression problems, make sure your task labels are on the same scale!
Challenges
Challenge #1: Negative transfer

Negative transfer: Sometimes independent networks work the best.

<table>
<thead>
<tr>
<th>Multi-Task CIFAR-100 state-of-the-art approaches</th>
<th>% accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>task specific-1-fc (Rosenbaum et al., 2018)</td>
<td>42</td>
</tr>
<tr>
<td>task specific-all-fc (Rosenbaum et al., 2018)</td>
<td>49</td>
</tr>
<tr>
<td>cross stitch-all-fc (Misra et al., 2016b)</td>
<td>53</td>
</tr>
<tr>
<td>routing-all-fc + WPL (Rosenbaum et al., 2019)</td>
<td>64.1</td>
</tr>
<tr>
<td>independent</td>
<td>64.3</td>
</tr>
</tbody>
</table>

Why?
- optimization challenges
  - caused by cross-task interference
  - tasks may learn at different rates
- limited representational capacity
  - multi-task networks often need to be much larger than their single-task counterparts
If you have negative transfer, **share less** across tasks.

It’s not just a binary decision!

\[
\min_{\theta^h, \theta^1, \ldots, \theta^T} \sum_{i=1}^{T} \mathcal{L}_i(\{\theta^h, \theta^i\}, \mathcal{D}_i) + \sum_{t' = 1}^{T} \|\theta^t - \theta^{t'}\|
\]

"soft parameter sharing"

+ allows for more fluid degrees of parameter sharing
- yet another set of design decisions / hyperparameters
Challenge #2: Overfitting

You may not be sharing enough!

Multi-task learning <-> a form of regularization

Solution: Share more.
Case study

**Recommending What Video to Watch Next: A Multitask Ranking System**

Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, Ed Chi
Google, Inc.
{zhezhao,lichan,liwei,jilinc,aniruddhnath,shawnandrews,aditeek,nlogn,xinyang,edchi}@google.com

**Goal:** Make recommendations for YouTube

![YouTube recommendation screen](Figure 4: Recommending what to watch next on YouTube.)
Case study

**Recommending What Video to Watch Next: A Multitask Ranking System**

Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, Ed Chi
Google, Inc.
{zhezhao,lichan,liwei,jilinc,aniruddhnath,shawnandrews,aditeek,nlogn,xinyang,edchi}@google.com

*Goal:* Make recommendations for YouTube

*Conflicting objectives:*
- videos that users will rate highly
- videos that users they will share
- videos that user will watch

*implicit bias caused by feedback:*
user may have watched it because it was recommended!
Framework Set-Up

**Input:** what the user is currently watching (query video) + user features

1. Generate a few hundred of candidate videos
2. Rank candidates
3. Serve top ranking videos to the user

**Candidate videos:** pool videos from multiple candidate generation algorithms
- matching topics of query video
- videos most frequently watched with query video
- And others

**Ranking:** central topic of this paper
The Ranking Problem

Input: query video, candidate video, user & context features

Model output: engagement and satisfaction with candidate video

Engagement:
- binary classification tasks like clicks
- regression tasks for tasks related to time spent

Satisfaction:
- binary classification tasks like clicking “like”
- regression tasks for tasks such as rating

Weighted combination of engagement & satisfaction predictions -> ranking score
score weights manually tuned
The Architecture

Basic option: “Shared-Bottom Model"
(i.e. multi-head architecture)

-> harm learning when correlation between tasks is low
Instead: use a form of soft-parameter sharing “Multi-gate Mixture-of-Experts (MMoE)”

Allow different parts of the network to “specialize” expert neural networks $f_i(x)$

Decide which expert to use for input $x$, task $k$:

$$g^k(x) = \text{softmax}(W_g^k x)$$

Compute features from selected expert:

$$f^k(x) = \sum_{i=1}^{n} g^k_{(i)}(x) f_i(x)$$

Compute output:

$$y_k = h^k(f^k(x)),$$
Experiments

Set-Up

- Implementation in TensorFlow, TPUs
- Train in temporal order, running training continuously to consume newly arriving data
- Offline AUC & squared error metrics
- Online A/B testing in comparison to production system
  - live metrics based on time spent, survey responses, rate of dismissals
- Model computational efficiency matters

Results

<table>
<thead>
<tr>
<th>Model Architecture</th>
<th>Number of Multiplications</th>
<th>Engagement Metric</th>
<th>Satisfaction Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared-Bottom</td>
<td>3.7M</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Shared-Bottom</td>
<td>6.1M</td>
<td>+0.1%</td>
<td>+1.89%</td>
</tr>
<tr>
<td>MMOE (4 experts)</td>
<td>3.7M</td>
<td>+0.20%</td>
<td>+1.22%</td>
</tr>
<tr>
<td>MMOE (8 experts)</td>
<td>6.1M</td>
<td>+0.45%</td>
<td>+3.07%</td>
</tr>
</tbody>
</table>

Table 1: YouTube live experiment results for MMOE.

Expert Utilization for Multiple Tasks

Found 20% chance of gating polarization during distributed training -> use drop-out on experts
Plan for Today

Multi-Task Learning
- Models & training
- Challenges
- Case study of real-world multi-task learning

— short break —

Meta-Learning
- Problem formulation
- General recipe of meta-learning algorithms
- Black-box adaptation approaches

} Topic of Homework 1!
Meta-Learning Basics
Two ways to view meta-learning algorithms

Mechanistic view
- Deep neural network model that can read in an entire dataset and make predictions for new datapoints
- Training this network uses a meta-dataset, which itself consists of many datasets, each for a different task
- This view makes it easier to implement meta-learning algorithms

Probabilistic view
- Extract prior information from a set of (meta-training) tasks that allows efficient learning of new tasks
- Learning a new task uses this prior and (small) training set to infer most likely posterior parameters
- This view makes it easier to understand meta-learning algorithms
Problem definitions

supervised learning:

$$\arg \max_{\phi} \log p(\phi | \mathcal{D})$$

$$= \arg \max_{\phi} \log p(\mathcal{D} | \phi) + \log p(\phi)$$

$$= \arg \max_{\phi} \sum_{i} \log p(y_i | x_i, \phi) + \log p(\phi)$$

$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$

input (e.g., image) \quad \text{label}

What is wrong with this?

➢ The most powerful models typically require large amounts of labeled data
➢ Labeled data for some tasks may be very limited
Problem definitions

supervised learning:
\[
\arg \max_\phi \log p(\phi|\mathcal{D})
\]

can we incorporate additional data?
\[
\arg \max_\phi \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})
\]

\[
\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}
\]

\[
\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}
\]

\[
\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}
\]

Image adapted from Ravi & Larochelle
The meta-learning problem

meta-learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x^i_1, y^i_1), \ldots, (x^i_k, y^i_k)\}$$

what if we don’t want to keep $\mathcal{D}_{\text{meta-train}}$ around forever?

learn meta-parameters $\theta$: $p(\theta|\mathcal{D}_{\text{meta-train}})$

whatever we need to know about $\mathcal{D}_{\text{meta-train}}$ to solve new tasks

log $p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) = \log \int_\Theta p(\phi|\mathcal{D}, \theta)p(\theta|\mathcal{D}_{\text{meta-train}})d\theta$

$$\approx \log p(\phi|\mathcal{D}, \theta^*) + \log p(\theta^*|\mathcal{D}_{\text{meta-train}})$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \approx \arg\max_{\phi} \log p(\phi|\mathcal{D}, \theta^*)$$

this is the meta-learning problem

$$\theta^* = \arg\max_{\theta} \log p(\theta|\mathcal{D}_{\text{meta-train}})$$
A Quick Example

meta-learning: $\theta^* = \arg\max_{\theta} \log p(\theta|D_{\text{meta-train}})$

adaptation: $\phi^* = \arg\max_{\phi} \log p(\phi|D, \theta^*)$

$D = \{(x_1, y_1), \ldots, (x_k, y_k)\}$

$D_{\text{meta-train}} = \{D_1, \ldots, D_n\}$

$D_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$
How do we train this thing?

**meta-learning**: \( \theta^* = \arg \max_{\theta} \log p(\theta|D_{\text{meta-train}}) \)

adaptation: \( \phi^* = \arg \max_{\phi} \log p(\phi|D, \theta^*) \)

Key idea:
“our training procedure is based on a simple machine learning principle: test and train conditions must match”

Vinyals et al., Matching Networks for One-Shot Learning

\[
D = \{(x_1, y_1), \ldots, (x_k, y_k)\} \\
D_{\text{meta-train}} = \{D_1, \ldots, D_n\} \\
D_i = \{(x^i_1, y^i_1), \ldots, (x^i_k, y^i_k)\}
\]
How do we train this thing?

**meta-learning:** \( \theta^* = \arg \max_{\theta} \log p(\theta | D_{\text{meta-train}}) \)

**adaptation:** \( \phi^* = \arg \max_{\phi} \log p(\phi | D, \theta^*) \)

(meta) test-time

\( (x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \)

\( D \)

\( y^{ts} \)

(test label)

\( x^{ts} \)

(test input)

(meta) training-time

\( D = \{(x_1, y_1), \ldots, (x_k, y_k)\} \)

\( D_{\text{meta-train}} = \{D_1, \ldots, D_n\} \)

\( D_i = \{(x^i_1, y^i_1), \ldots, (x^i_k, y^i_k)\} \)

Key idea:

“our training procedure is based on a simple machine learning principle: test and train conditions must match”

Vinyals et al., Matching Networks for One-Shot Learning
Reserve a test set for each task!

Key idea:
“our training procedure is based on a simple machine learning principle: test and train conditions must match”

Vinyals et al., Matching Networks for One-Shot Learning
The complete meta-learning optimization

meta-learning: \( \theta^* = \arg \max_\theta \log p(\theta|D_{\text{meta-train}}) \)

adaptation: \( \phi^* = \arg \max_\phi \log p(\phi|D^{tr}, \theta^*) \)

\[
\phi^* = f_{\theta^*}(D^{tr})
\]

learn \( \theta \) such that \( \phi = f_\theta(D^{tr}_i) \) is good for \( D^{ts}_i \)

\[
\theta^* = \max_\theta \sum_{i=1}^n \log p(\phi_i|D^{ts}_i)
\]

where \( \phi_i = f_\theta(D^{tr}_i) \)

\[\begin{align*}
D_{\text{meta-train}} &= \{(D^{tr}_1, D^{ts}_1), \ldots, (D^{tr}_n, D^{ts}_n)\} \\
D^{tr}_i &= \{(x^i_1, y^i_1), \ldots, (x^i_k, y^i_k)\} \\
D^{ts}_i &= \{(x^i_1, y^i_1), \ldots, (x^i_i, y^i_i)\}
\end{align*}\]
Some meta-learning terminology

learn $\theta$ such that $\phi_i = f_\theta(D^\text{tr}_i)$ is good for $D^\text{ts}_i$

$$\theta^* = \arg\max_\theta \sum_{i=1}^n \log p(\phi_i | D^\text{ts}_i)$$

where $\phi_i = f_\theta(D^\text{tr}_i)$

$D_{\text{meta-train}} = \{(D^\text{tr}_1, D^\text{ts}_1), \ldots, (D^\text{tr}_n, D^\text{ts}_n)\}$

$T_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$

$D^\text{tr}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$

$D^\text{ts}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$

(meta-training) task $T_i$

(i.e., k-shot, 5-shot)

image credit: Ravi & Larochelle '17
Closely related problem settings

meta-learning:

$$\theta^* = \max_{\theta} \sum_{i=1}^{n} \log p(\phi_i|D_{i}^{ts})$$

where \(\phi_i = f_\theta(D_{i}^{tr})\)

\(D_{\text{meta-train}} = \{ (D_{1}^{tr}, D_{1}^{ts}), \ldots, (D_{n}^{tr}, D_{n}^{ts}) \}\)

\(D_{i}^{tr} = \{ (x_1^i, y_1^i), \ldots, (x_k^i, y_k^i) \}\)

\(D_{i}^{ts} = \{ (x_1^i, y_1^i), \ldots, (x_k^i, y_k^i) \}\)

multi-task learning: learn model with parameters \(\theta^*\) that solves multiple tasks \(\theta^* = \arg\max_{\theta} \sum_{i=1}^{n} \log p(\theta|D_{i})\)

can be seen as special case where \(\phi_i = \theta\) (i.e., \(f_\theta(D_{i}) = \theta\))

hyperparameter optimization & auto-ML: can be cast as meta-learning

hyperparameter optimization: \(\theta =\) hyperparameters, \(\phi =\) network weights

architecture search: \(\theta =\) architecture, \(\phi =\) network weights

very active area of research! but outside the scope of this course
Plan for Today

Multi-Task Learning
- Models & training
- Challenges
- Case study of real-world multi-task learning

— short break —

Meta-Learning
- Problem formulation
- General recipe of meta-learning algorithms
- Black-box adaptation approaches

} Topic of Homework 1!

Rest will be covered next time.
Reminders

Homework 1 posted today, due Wednesday, October 9
Fill out paper preferences by tomorrow.
TensorFlow review session tomorrow, 4:30 pm in Gates B03