Topics:
- (Continue) Low-label ML Formulations
Administrative

• Projects!
  – Poster details out on piazza
  – Note: No late days for anything project related!
  – Also note: Keep track of your GCP usage and costs! Set limits on spending
Meta-Learning for Few-Shot Recognition

- **Key idea:** We want to learn from a few examples (called the **support set**) to make predictions on **query set** for **novel** classes
  - Assume: We have larger labeled dataset for a different set of categories (**base classes**)

- **How do we test this?**
  - N-way k-shot test
  - k: Number of examples in **support set**
  - N: Number of “confusers” that we have to choose target class among

(C) Dhruv Batra & Zsolt Kira
Normal Approach

- Do what we always do: Fine-tuning
  - Train classifier on base classes
  - Freeze features
  - Learn classifier weights for new classes using few amounts of labeled data (during “inference” time!)

A Closer Look at Few-shot Classification, Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, Jia-Bin Huang
Cons of Normal Approach

• The training we do on the base classes does not factor the task into account

• No notion that we will be performing a bunch of N-way tests

• Idea: simulate what we will see during test time
Meta-Training Approach

- Set up a set of smaller tasks *during training* which simulates what we will be doing during testing
  
  - Can optionally pre-train features on held-out base classes (not typical)

- Testing stage is now the same, but with new classes
Meta-Learning Approaches

- Learning a model conditioned on support set $M(\cdot | S)$
Meta-Learner

• How to parametrize learning algorithms?

• Two approaches to defining a meta-learner
  – Take inspiration from a known learning algorithm
    • kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    • Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    • Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017), MAML (Finn et al. 2017)
  – Derive it from a black box neural network
    • MANN (Santoro et al. 2016)
    • SNAIL (Mishra et al. 2018)
Meta-Learner

• How to parametrize learning algorithms?

• Two approaches to defining a meta-learner
  – Take inspiration from a known learning algorithm
    • kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    • Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    • Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017), MAML (Finn et al. 2017)
  – Derive it from a black box neural network
    • MANN (Santoro et al. 2016)
    • SNAIL (Mishra et al. 2018)
Matching Networks

- Training a **pattern matcher**

\[ \hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i \]

\[
a(\hat{x}, x_i) = \frac{e^{c(f(\hat{x}), g(x_i))}}{\sum_{j=1}^{k} e^{c(f(\hat{x}), g(x_j))}}
\]

- Matching networks for one shot learning (2016)
  Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra
Prototypical Networks

• Training a “prototype extractor”

\[ p_\phi(y = k \mid x) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_{k'} \exp(-d(f_\phi(x), c_{k'}))} \]

\[ c_k = \frac{1}{|S_k|} \sum_{(x_i, y_i) \in S_k} f_\phi(x_i) \]

\[ S_k = \{(x_i, y_i) \mid y_i = k, (x_i, y_i) \in D_{train}\} \]

\[ \phi \equiv \Theta \]

• Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel
Prototypical Networks

- Training a "prototype extractor"
  - distance function $d(\cdot, \cdot)$ can be anything (euclidean squared, negative cosine)
  - if distance is euclidean squared, equivalent to learning an embedding network $f_{\phi}(\cdot)$ such that a Gaussian classifier works well
  - prototype vectors are equivalent to output weights of a neural network
  - Snell et al. find that using more classes in the meta-training episodes compared to meta-testing works better

- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel
More Sophisticated Meta-Learning Approaches

• Learn gradient descent:
  – Parameter initialization and update rules

• Learn just an initialization and use normal gradient descent (MAML)
Meta-Learner LSTM

- Training a "gradient descent procedure" applied on some learner $M$
  - gradient descent starts from some initial parameters $\theta_0$ and then performs the following updates:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

- Optimization as a Model for Few-Shot Learning (2017)
  *Sachin Ravi and Hugo Larochelle*
Meta-Learner LSTM

- Training a "gradient descent procedure" applied on some learner $M$
  - gradient descent starts from some initial parameters $\theta_0$ and then performs the following updates:
    \[
    \theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t
    \]
  - this is quite similar to LSTM cell state updates:
    \[
    c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t
    \]
    - state $c_t$ is model $M$'s parameter space $\theta_t$
    - state update $\tilde{c}_t$ is the negative gradient $-\nabla_{\theta_{t-1}} \mathcal{L}_t$
    - $f_t$ and $i_t$ are LSTM gates:
      \[
      i_t = \sigma \left( W_I \cdot \left[ \nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1} \right] + b_I \right)
      \]
      \[
      f_t = \sigma \left( W_F \cdot \left[ \nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1} \right] + b_F \right)
      \]
- Optimization as a Model for Few-Shot Learning (2017)

*Sachin Ravi and Hugo Larochelle*
Meta-Learner LSTM

- Training a "gradient descent procedure" applied on some learner $M$
  
  - gradient descent starts from some initial parameters $\theta_0$ and then performs the following updates:
    \[
    \theta_t = \theta_{t-1} - \alpha_t \nabla \theta_{t-1} L_t
    \]
  
  - this is quite similar to LSTM cell state updates:
    \[
    c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t
    \]
    - state $c_t$ is model $M$'s parameter space $\theta_t$ \(\Leftarrow\) $c_0$ becomes a learned initialization
    - state update $\tilde{c}_t$ is the negative gradient $-\nabla \theta_{t-1} L_t$
    - $f_t$ and $i_t$ are LSTM gates:
      \[
      i_t = \sigma (W_1 \cdot [\nabla \theta_{t-1} L_t, L_t, \theta_{t-1}, i_{t-1}] + b_1)
      \]  \(\Leftarrow\) adaptive learning rate
      \[
      f_t = \sigma (W_F \cdot [\nabla \theta_{t-1} L_t, L_t, \theta_{t-1}, f_{t-1}] + b_F)
      \]  \(\Leftarrow\) adaptive weight decay

- Optimization as a Model for Few-Shot Learning (2017)
  
  Sachin Ravi and Hugo Larochelle

(C) Dhruv Batra & Zsolt Kira  Slide Credit: Hugo Larochelle
Meta-Learner LSTM

- Training a “gradient descent procedure” applied on some learner $M$

- Optimization as a Model for Few-Shot Learning (2017)
  
  Sachin Ravi and Hugo Larochelle

(C) Dhruv Batra & Zsolt Kira

Slide Credit: Hugo Larochelle
Algorithm 1 Train Meta-Learner

Input: Meta-training set $D_{meta-train}$, Learner $M$ with parameters $\theta$, Meta-Learner $R$ with parameters $\Theta$.

1: $\Theta_0 \leftarrow$ random initialization
2: 
3: for $d = 1, n$ do
4: $D_{train}, D_{test} \leftarrow$ random dataset from $D_{meta-train}$
5: $\theta_0 \leftarrow c_0$ \hspace{1cm} $\triangleright$ Initialize learner parameters
6: 
7: for $t = 1, T$ do
8: $X_t, Y_t \leftarrow$ random batch from $D_{train}$
9: $\mathcal{L}_t \leftarrow \mathcal{L}(M(X_t; \theta_{t-1}), Y_t)$ \hspace{1cm} $\triangleright$ Get loss of learner on train batch
10: $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ \hspace{1cm} $\triangleright$ Get output of meta-learner using Equation 2
11: $\theta_t \leftarrow c_t$ \hspace{1cm} $\triangleright$ Update learner parameters
12: end for
13: 
14: $X, Y \leftarrow D_{test}$
15: $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(X; \theta_T), Y)$ \hspace{1cm} $\triangleright$ Get loss of learner on test batch
16: Update $\Theta_d$ using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ \hspace{1cm} $\triangleright$ Update meta-learner parameters
17: 
18: end for
Meta-Learner LSTM

- Training a "gradient descent procedure" applied on some learner $M$
  - LSTM parameters are shared across $M$'s parameters (i.e. treated like a large minibatch)
  - can ignore (stop) gradients through the inputs of the LSTM
  - gradient (and loss) inputs to the Meta-LSTM preprocessed as proposed by Andrychowicz et al. (2016)
    \[
    \nabla^k \rightarrow \begin{cases} 
    \left( \frac{\log(|\nabla|)}{p}, \text{sgn}(\nabla) \right) & \text{if } |\nabla| \geq e^{-p} \\
    (-1, e^p \nabla) & \text{otherwise}
    \end{cases}
    \]
  - we are careful to avoid "leakage" from batchnorm statistics between meta-train / meta-test sets (sometimes referred to as the "transductive setting")

- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle
Model-Agnostic Meta-Learning (MAML)

- Training a "gradient descent procedure" applied on some learner $M$
  - MAML proposes not to bother with training an LSTM for the gradient descent updates and constant step-size updates

  Chelsea Finn, Pieter Abbeel and Sergey Levine
Model-Agnostic Meta-Learning (MAML)

**Algorithm 1** Model-Agnostic Meta-Learning

**Require:** $p(\mathcal{T})$: distribution over tasks

**Require:** $\alpha, \beta$: step size hyperparameters

1: randomly initialize $\theta$

2: while not done do

3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$

4: for all $\mathcal{T}_i$ do

5: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples

6: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$

7: end for **Note:** the meta-update is using different set of data.

8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$

9: end while
Model-Agnostic Meta-Learning (MAML)

a general recipe:

\[ \theta \leftarrow \theta - \beta \sum_i \nabla_\theta \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_{train}^i), \mathcal{D}_{test}^i) \]

"meta-loss" for task \( i \)

* in general, can take more than one gradient step here
** we often use 4 – 10 steps

Finn et al., “Model-Agnostic Meta-Learning”
Model-Agnostic Meta-Learning (MAML)

supervised learning: \( f(x) \rightarrow y \)

supervised meta-learning: \( f(D_{\text{train}}, x) \rightarrow y \)

model-agnostic meta-learning: \( f_{\text{MAML}}(D_{\text{train}}, x) \rightarrow y \)

\[
f_{\text{MAML}}(D_{\text{train}}, x) = f_{\theta'}(x)
\]

\[
\theta' = \theta - \alpha \sum_{(x,y) \in D_{\text{train}}} \nabla_{\theta} \mathcal{L}(f_{\theta}(x), y)
\]

Just another computation graph...

Can implement with any autodiff package (e.g., TensorFlow)
### Comparison

**RNN-based meta-learning**

- **y_{test}** ← test label
- **x_{test}** ← test input
- This implements the “learned learning algorithm”

- Does it converge?
  - Kind of?
- What does it converge to?
  - Who knows...
- What to do if it’s not good enough?
  - Nothing...

**MAML**

\[
\theta \xrightarrow{\nabla_\theta L} \theta'
\]

- Does it converge?
  - Yes (it’s gradient descent...)
- What does it converge to?
  - A local optimum (it’s gradient descent...)
- What to do if it’s not good enough?
  - Keep taking gradient steps (it’s gradient descent...)

---

(C) Dhruv Batra & Zsolt Kira  
Slide Credit: Sergey Levine
Meta-Learner

• How to parametrize learning algorithms?

• Two approaches to defining a meta-learner
  – Take inspiration from a known learning algorithm
    • kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    • Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    • Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017), MAML (Finn et al. 2017)
  – Derive it from a black box neural network
    • MANN (Santoro et al. 2016)
    • SNAIL (Mishra et al. 2018)
Experiments

- **Mini-ImageNet** (split used in Ravi & Larochelle, 2017)
  - random subset of 100 classes (64 training, 16 validation, 20 testing)
  - random sets $D_{\text{train}}$ are generated by randomly picking 5 classes from class subset

<table>
<thead>
<tr>
<th>Model</th>
<th>5-class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
</tr>
<tr>
<td>Prototypical Nets (Snell et al.)</td>
<td>49.42% ± 0.78%</td>
</tr>
<tr>
<td>MAML (Finn et al.)</td>
<td>48.70% ± 1.84%</td>
</tr>
<tr>
<td>SNAIL (Mishra et al.)</td>
<td>55.71% ± 0.99%</td>
</tr>
<tr>
<td>Matching Network FCE</td>
<td>43.56% ± 0.84%</td>
</tr>
<tr>
<td>Meta-Learner LSTM (OURS)</td>
<td>43.44% ± 0.77%</td>
</tr>
</tbody>
</table>
Memory-Augmented Neural Network

- Training a **neural Turing machine** to learn a learning algorithm

- One-shot learning with memory-augmented neural networks (2016)
  
  *Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap*
But beware

A Closer Look at Few-shot Classification,
Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, Jia-Bin Huang
Table 2: Few-shot classification results for both the mini-ImageNet and CUB datasets. The Baseline++ consistently improves the Baseline model by a large margin and is competitive with the state-of-the-art meta-learning methods. All experiments are from 5-way classification with a Conv-4 backbone and data augmentation.

<table>
<thead>
<tr>
<th>Method</th>
<th>CUB 1-shot</th>
<th>CUB 5-shot</th>
<th>mini-ImageNet 1-shot</th>
<th>mini-ImageNet 5-shot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>47.12 ± 0.74</td>
<td>64.16 ± 0.71</td>
<td>42.11 ± 0.71</td>
<td>62.53 ± 0.69</td>
</tr>
<tr>
<td>Baseline++</td>
<td>60.53 ± 0.83</td>
<td>79.34 ± 0.61</td>
<td>48.24 ± 0.75</td>
<td>66.43 ± 0.63</td>
</tr>
<tr>
<td>MatchingNet</td>
<td>61.16 ± 0.89</td>
<td>72.86 ± 0.70</td>
<td>48.14 ± 0.78</td>
<td>63.48 ± 0.66</td>
</tr>
<tr>
<td>ProtoNet</td>
<td>51.31 ± 0.91</td>
<td>70.77 ± 0.69</td>
<td>44.42 ± 0.84</td>
<td>64.24 ± 0.72</td>
</tr>
<tr>
<td>MAML</td>
<td>55.92 ± 0.95</td>
<td>72.09 ± 0.76</td>
<td>46.47 ± 0.82</td>
<td>62.71 ± 0.71</td>
</tr>
<tr>
<td>RelationNet</td>
<td>62.45 ± 0.98</td>
<td>76.11 ± 0.69</td>
<td>49.31 ± 0.85</td>
<td>66.60 ± 0.69</td>
</tr>
</tbody>
</table>
Distribution Shift

• What if there is a distribution shift (cross-domain)?

• **Lesson:** Methods that are successful *within-domain* might be worse *across domains!*

<table>
<thead>
<tr>
<th></th>
<th>mini-ImageNet → CUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>65.57 ± 0.70</td>
</tr>
<tr>
<td>Baseline++</td>
<td>62.04 ± 0.76</td>
</tr>
<tr>
<td>MatchingNet</td>
<td>53.07 ± 0.74</td>
</tr>
<tr>
<td>ProtoNet</td>
<td>62.02 ± 0.70</td>
</tr>
<tr>
<td>MAML</td>
<td>51.34 ± 0.72</td>
</tr>
<tr>
<td>RelationNet</td>
<td>57.71 ± 0.73</td>
</tr>
</tbody>
</table>

Table 3: 5-shot accuracy under the cross-domain scenario with a ResNet-18 backbone. Baseline outperforms all other methods under this scenario.
Distribution Shift

Let’s Talk about Meta-Overfitting

- Meta learning requires task distributions
- When there are too few meta-training tasks, we can *meta-overfit*
- Specifying task distributions is hard, especially for meta-RL!
- Can we propose tasks *automatically?*
Random Task Proposals

- Use randomly initialize discriminators for reward functions

\[ R(s, z) = \log p_D(z|s) \]

- \( D \rightarrow \) randomly initialized network

- Important: Random functions over state space, **not** random policies
Does it Work?

2D Navigation

Cheetah

Ant

Meta-test performance with rewards

Discussions

• What is the right definition of distributions over problems?
  – varying number of classes / examples per class (meta-training vs. meta-testing) ?
  – semantic differences between meta-training vs. meta-testing classes ?
  – overlap in meta-training vs. meta-testing classes (see recent “low-shot” literature) ?

• Move from static to interactive learning
  – how should this impact how we generate episodes ?
  – meta-active learning ? (few successes so far)