Week 18: Trends of deep learning

Instructor: Ruixuan Wang
wangruix5@mail.sysu.edu.cn

School of Data and Computer Science
Sun Yat-Sen University

27 June, 2019
1. Few-shot learning

2. Lifelong learning
Limitation of deep learning

Deep learning works well...
Deep learning works well...

when large training dataset is available!
Few-shot learning

- Few-shot learning: learning with a few training data per class
Few-shot learning

- Few-shot learning: learning with a few training data per class
- Traditionally, KNN or kernel density estimation (KDE)
- Traditionally, feature extraction was pre-designed
Few-shot learning

- Few-shot learning: learning with a few training data per class
- Traditionally, KNN or kernel density estimation (KDE)
- Traditionally, feature extraction was pre-designed
- With deep learning, any way to learn feature representation?
- Or: how to train a DL classifier with just a few data?
Few-shot learning: learning with a few training data per class

Traditionally, KNN or kernel density estimation (KDE)

Traditionally, feature extraction was pre-designed

With deep learning, any way to learn feature representation?

Or: how to train a DL classifier with just a few data?

Impossible?!
Few-shot learning: matching network

- But: may train a meta-classifier with large ‘meta-dataset’!
- Meta-classifier: input is a dataset; output is a classifier
Few-shot learning: matching network

- But: may train a meta-classifier with large ‘meta-dataset’!
- Meta-classifier: input is a dataset; output is a classifier
- How to represent the output (i.e., a classifier)?

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

where \( \{(x_i, y_i)\} \) are small dataset as input to meta-classifier, and \( a(\cdot) \) could be considered as an attention model

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

where \( f(\cdot), g(\cdot) \): feature extractors; \( c(\cdot) \): similarity measure
Few-shot learning: matching network

- But: may train a meta-classifier with large ‘meta-dataset’!
- Meta-classifier: input is a dataset; output is a classifier.
- How to represent the output (i.e., a classifier)?

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

where \( \{(x_i, y_i)\} \) are small dataset as input to meta-classifier, and \( a(\cdot) \) could be considered as an attention model

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

where \( f(\cdot), g(\cdot) \): feature extractors; \( c(\cdot) \): similarity measure.

- Meta-classifier training: using many sets of small datasets to learn to find the optimal \( f(\cdot) \) and \( g(\cdot) \).
Few-shot learning: matching network

- Traditional classifier training: train by comparing the difference between predicted and ground-truth output.
- But what is the ground-truth output for a meta-classifier?
Few-shot learning: matching network

- Traditional classifier training: train by comparing the difference between predicted and ground-truth output.
- But what is the ground-truth output for a meta-classifier?
- No ‘ground-truth classifier’ for output of a meta-classifier!
Few-shot learning: matching network

- Traditional classifier training: train by comparing the difference between predicted and ground-truth output.
- But what is the ground-truth output for a meta-classifier?
- No ‘ground-truth classifier’ for output of a meta-classifier!
- Training: given a small set $\{(x_i, y_i)\}$, use another small set $\{ (\tilde{x}_j, \tilde{y}_j) \}$ to evaluate goodness of meta-classifier output:

$$
\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i
$$
Few-shot learning: matching network

- Traditional classifier training: train by comparing the difference between predicted and ground-truth output.
- But what is the ground-truth output for a meta-classifier?
- No ‘ground-truth classifier’ for output of a meta-classifier!
- Training: given a small set \( \{(x_i, y_i)\} \), use another small set \( \{(	ilde{x}_j, \tilde{y}_j)\} \) to evaluate goodness of meta-classifier output:

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

- So in each training iteration, training set consists of two small subsets \( \{(x_i, y_i)\} \) and \( \{(	ilde{x}_j, \tilde{y}_j)\} \).
- Over iterations: training sets may be from different classes.
So meta-classifier training is to find the optimal $f(\cdot)$ and $g(\cdot)$ by minimizing the prediction error of the classifier

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

on training set $\{(x_i, y_i)\}, \{({\hat{x}}_j, {\hat{y}}_j)\}$ over iterations.
Once the meta-classifier is trained, then given a small training dataset for certain number of new classes, the meta-classifier would output a new classifier for the new classes!
Once the meta-classifier is trained, then given a small training dataset for certain number of new classes, the meta-classifier would output a new classifier for the new classes!

The method learned better feature extractor $f(\cdot)$ and $g(\cdot)$ compared to using pretrained CNN as feature extractor:
The proposed method outperforms all others on Omniglot (below) and mini-ImageNet (not shown)!

<table>
<thead>
<tr>
<th>Model</th>
<th>Matching Fn</th>
<th>Fine Tune</th>
<th>5-way Acc</th>
<th>20-way Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1-shot</td>
<td>5-shot</td>
</tr>
<tr>
<td>Pixels</td>
<td>Cosine</td>
<td>N</td>
<td>41.7%</td>
<td>63.2%</td>
</tr>
<tr>
<td>Baseline Classifier</td>
<td>Cosine</td>
<td>N</td>
<td>80.0%</td>
<td>95.0%</td>
</tr>
<tr>
<td>Baseline Classifier</td>
<td>Cosine</td>
<td>Y</td>
<td>82.3%</td>
<td>98.4%</td>
</tr>
<tr>
<td>Baseline Classifier</td>
<td>Softmax</td>
<td>Y</td>
<td>86.0%</td>
<td>97.6%</td>
</tr>
<tr>
<td>MANN (No Conv) [21]</td>
<td>Cosine</td>
<td>N</td>
<td>82.8%</td>
<td>94.9%</td>
</tr>
<tr>
<td>Convolutional Siamese Net</td>
<td>Cosine</td>
<td>N</td>
<td>96.7%</td>
<td>98.4%</td>
</tr>
<tr>
<td>Convolutional Siamese Net</td>
<td>Cosine</td>
<td>Y</td>
<td>97.3%</td>
<td>98.4%</td>
</tr>
<tr>
<td>Matching Nets (Ours)</td>
<td>Cosine</td>
<td>N</td>
<td>98.1%</td>
<td>98.9%</td>
</tr>
<tr>
<td>Matching Nets (Ours)</td>
<td>Cosine</td>
<td>Y</td>
<td>97.9%</td>
<td>98.7%</td>
</tr>
</tbody>
</table>

Note: ‘Baseline classifier’: trained on all training data, then extract feature from last conv layer for attention module.
Another idea: train a model that can **quickly adapt** to a new task using only a few data points and training iterations!
Another idea: train a model that can **quickly adapt** to a new task using only a few data points and training iterations!

Consider adapting model $f_\theta$ to a new task $\mathcal{T}_i$, with $\theta$ updated to $\theta'_i$ by (1 or few iters) gradient descent of loss on task $\mathcal{T}_i$

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$$
Few-shot learning: modal-agnostic meta-learning (MAML)

- Another idea: train a model that can **quickly adapt** to a new task using only a few data points and training iterations!

- Consider adapting model $f_\theta$ to a new task $\mathcal{T}_i$, with $\theta$ updated to $\theta'_i$ by (1 or few iters) gradient descent of loss on task $\mathcal{T}_i$

  $$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

- Better model $f_\theta$ means less loss $\mathcal{L}_{\mathcal{T}_i}(f'_\theta)$ on new tasks after one/few (so ‘quick adapt’) update of model parameter to $\theta'_i$.

  $$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f'_\theta) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_\theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_\theta))$$

One task: one ‘training data’ for meta-learning!
Few-shot learning: modal-agnostic meta-learning (MAML)

- Another idea: train a model that can **quickly adapt** to a new task using only a few data points and training iterations!

- Consider adapting model $f_\theta$ to a new task $T_i$, with $\theta$ updated to $\theta'_i$ by (1 or few iters) gradient descent of loss on task $T_i$

$$
\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)
$$

- Better model $f_\theta$ means less loss $\mathcal{L}_{T_i}(f_{\theta'_i})$ on new tasks after one/few (so ‘quick adapt’) update of model parameter to $\theta'_i$.

$$
\min_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'_i}) = \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)})
$$

One task: one ‘training data’ for meta-learning!

- Note: meta-optimization is performed over model parameters $\theta$, but loss is computed using updated parameters $\theta'_i$. 
MAML (cont’)

- Meta-optimization over tasks (‘training data’) to update model param $\theta$

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$
MAML (cont’)

- Meta-optimization over tasks (‘training data’) to update model param $\theta$

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

- Meta-gradient update involves a gradient through gradient

---

**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
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
MAML: result

- MAML works for any differentiable objective, including those of regression and reinforcement learning!
- Matching network learns feature embedding, while MAML learns good model initialization for multiple tasks.
MAML: result

- MAML works for any differentiable objective, including those of regression and reinforcement learning!
- Matching network learns feature embedding, while MAML learns good model initialization for multiple tasks.
- Classification: MAML outperforms matching networks.

<table>
<thead>
<tr>
<th>MiniImagenet (Ravi &amp; Larochelle, 2017)</th>
<th>5-way Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
</tr>
<tr>
<td>fine-tuning baseline</td>
<td>28.86 ± 0.54%</td>
</tr>
<tr>
<td>nearest neighbor baseline</td>
<td>41.08 ± 0.70%</td>
</tr>
<tr>
<td>matching nets (Vinyals et al., 2016)</td>
<td>43.56 ± 0.84%</td>
</tr>
<tr>
<td>meta-learner LSTM (Ravi &amp; Larochelle, 2017)</td>
<td>43.44 ± 0.77%</td>
</tr>
<tr>
<td>MAML, first order approx. (ours)</td>
<td>48.07 ± 1.75%</td>
</tr>
<tr>
<td>MAML (ours)</td>
<td>48.70 ± 1.84%</td>
</tr>
</tbody>
</table>
Lifelong learning: another limitation

We learn new knowledge without forgetting old!

But AI catastrophically forgets old!
Lifelong learning: elastic weight consolidation (EWC)

- EWC idea: when learning a new task, do not change weights too much which are important to previous tasks.
Lifelong learning: elastic weight consolidation (EWC)

- EWC idea: when learning a new task, do not change weights too much which are important to previous tasks.
- Fisher information matrix $\mathbf{F}$: importance of model params.
Lifelong learning: elastic weight consolidation (EWC)

- EWC idea: when learning a new task, do not change weights too much which are important to previous tasks.
- Fisher information matrix $\mathbf{F}$: importance of model params.
- Can overcome catastrophic forgetting by minimizing loss

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta_{A,i}^*)^2$$

- Fisher-weighted regularization helps update model parameters (red arrow) good for both previous task A and new task B.
EWC: result

- On MNIST, with EWC: classifier does not degrade on current and previous tasks
- Blue curve: updating model by just focusing on current task
Memory aware synapse

- EWC: estimate parameter importance based on sensitivity of loss function to changes in parameters.
- Another idea: estimate parameter importance based on sensitivity of network output to changes in parameters.
Memory aware synapse

- EWC: estimate parameter importance based on sensitivity of loss function to changes in parameters.
- Another idea: estimate parameter importance based on sensitivity of network output to changes in parameters.
- Output change with a small change $\delta$ in parameters:
  \[
  F(x_1; \theta + \delta) - F(x_1; \theta) \approx \sum_{i,j} g_{ij}(x_1) \delta_{ij}
  \]
  where $g_{ij}$ is the partial derivative of network output $F$ w.r.t. parameter $\theta_{i,j}$ at data point $x_1$. 

Memory aware synapse

- EWC: estimate parameter importance based on sensitivity of loss function to changes in parameters.
- Another idea: estimate parameter importance based on sensitivity of network output to changes in parameters.
- Output change with a small change $\delta$ in parameters:
  \[ F(x_1; \theta + \delta) - F(x_1; \theta) \approx \sum_{i,j} g_{ij}(x_1) \delta_{ij} \]
  where $g_{ij}$ is the partial derivative of network output $F$ w.r.t. parameter $\theta_{i,j}$ at data point $x_1$.
- Importance of parameter $\theta_{i,j}$ can be estimated by accumulating $g_{ij}$ over all available data points:
  \[ \Omega_{ij} = \frac{1}{N} \sum_{k=1}^{N} \| g_{ij}(x_k) \| \]
Loss is similar to EWC, except the importance parameter

\[ L(\theta) = L_{\text{new}}(\theta) + \frac{\lambda}{2} \sum_{i,j} \Omega_{ij} (\theta_{ij} - \theta^*_{ij})^2 \]

Data label is not necessary when computing \( \Omega_{ij} \), so \( \Omega_{ij} \) can be updated on any available data (without corresponding labels).

Both this method and EWC focus on model parameters.

Another idea: somehow get ‘data’ of previous tasks!
Continual learning with deep generative replay

- Idea: generate realistic synthetic data for previous tasks
Continual learning with deep generative replay

● Idea: generate realistic synthetic data for previous tasks
● Solution: using GAN!
Continual learning with deep generative replay

- Idea: generate realistic synthetic data for previous tasks
- Solution: using GAN!
- Dual model ‘scholar’: (GAN, Solver); Solver, e.g., classifier
Continual learning with deep generative replay

- Idea: generate realistic synthetic data for previous tasks
- Solution: using GAN!
- Dual model ‘scholar’: (GAN, Solver); Solver, e.g., classifier
- Train GAN: with GAN-generated data and new task’s data
- Train Solver: with new task’s (data, labels) and old scholar’s (generated data, predicted labels)
Continual learning with deep generative replay: result

- On MNIST, 5 tasks, continuously learning to recognize new classes of digits; test on all tasks’ (test) data
- Similar performance between ER and GR

![Graph showing performance over iterations](image)

- ER: using exact past real data with predicted labels for replay
- GR (proposed): using realistic synthetic data for replay
- ‘Noise’: using un-realistic synthetic data for replay
More trends and limitations of deep learning or AI

- Learn from experience: deep reinforcement learning
- Learn from partially labelled data: semi-supervised
- Learn from unlabelled data: unsupervised learning
- Learn from multi-modality data
- ...

So far, mostly perceptual AI! Need cognitive AI!

Current deep learning depends on gradient descent. But human brains probably does not use gradient descent. Learning and inference by reasoning!
More trends and limitations of deep learning or AI

- Learn from experience: deep reinforcement learning
- Learn from partially labelled data: semi-supervised
- Learn from unlabelled data: unsupervised learning
- Learn from multi-modality data
- ...

So far, mostly perceptual AI! Need cognitive AI!
More trends and limitations of deep learning or AI

- Learn from experience: deep reinforcement learning
- Learn from partially labelled data: semi-supervised
- Learn from unlabelled data: unsupervised learning
- Learn from multi-modality data
- ...

So far, mostly perceptual AI! Need cognitive AI!

- Current deep learning depends on gradient descent.
- But human brains probably does not use gradient descent.
More trends and limitations of deep learning or AI

- Learn from experience: deep reinforcement learning
- Learn from partially labelled data: semi-supervised
- Learn from unlabelled data: unsupervised learning
- Learn from multi-modality data
- ...

So far, mostly perceptual AI! Need cognitive AI!

- Current deep learning depends on gradient descent.
- But human brains probably does not use gradient descent.
- Learning and inference by **reasoning**!
  e.g., deep learning + graphical model
**Project reports**

Course project report:
- **Title; Team members**
- **Abstract**: problem, difficulty, method idea, key result.
- **Introduction**: application background, research problem, related existing methods, implemented methods, main results including team ranking (e.g., ranked 5th over 120 teams).
- **Problem formulation**: formally describe the research problem, better with math representation.
- **Method**: the basic ideas, model structures, etc.
- **Experiments**: all experiments, including worse and better results, better explaining why.
- **Conclusion**: very short summary, conclusion from experimental evaluation, future work.
- **Source code!

No plagiarism!!
Lab project report:

- **Title; authors; your name.**
- **Abstract:** problem, difficulty, idea, your key result.
- **Introduction:** application background, research problem, related existing methods, the paper’s idea, your key results.
- **Problem formulation:** formally describe the research problem.
- **Method:** the basic idea, model structure.
- **Implementation:** what you have done, difficulties & solutions.
- **Experiments:** all tests, including worse and better results.
- **Conclusion:** conclusion from experimental evaluation.
- **Source code!**

No plagiarism!!