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

Limitation of deep learning

Deep learning works well...

Limitation of deep learning

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

- 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

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

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 $\{(\tilde{x}_j, \tilde{y}_j)\}$ to evaluate goodness of meta-classifier output:

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

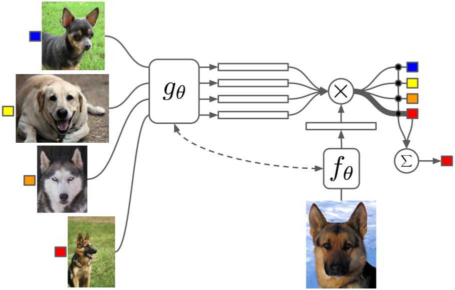
- So in each training iteration, training set consists of two small subsets $\{(x_i, y_i)\}$ and $\{(\tilde{x}_j, \tilde{y}_j)\}$.
- Over iterations: training sets may be from different classes.

Few-shot learning: matching network

- 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.

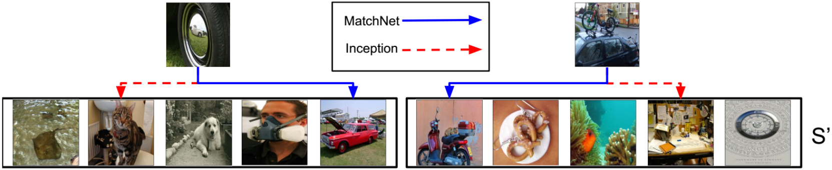


Few-shot learning: matching network

- 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!

Few-shot learning: matching network

- 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:



Matching network: result

- The proposed method outperforms all others on Omniglot (below) and mini-ImageNet (not shown)!

Model	Matching Fn	Fine Tune	5-way Acc		20-way Acc	
			1-shot	5-shot	1-shot	5-shot
PIXELS	Cosine	N	41.7%	63.2%	26.7%	42.6%
BASELINE CLASSIFIER	Cosine	N	80.0%	95.0%	69.5%	89.1%
BASELINE CLASSIFIER	Cosine	Y	82.3%	98.4%	70.6%	92.0%
BASELINE CLASSIFIER	Softmax	Y	86.0%	97.6%	72.9%	92.3%
MANN (NO CONV) [21]	Cosine	N	82.8%	94.9%	–	–
CONVOLUTIONAL SIAMESE NET [11]	Cosine	N	96.7%	98.4%	88.0%	96.5%
CONVOLUTIONAL SIAMESE NET [11]	Cosine	Y	97.3%	98.4%	88.1%	97.0%
MATCHING NETS (OURS)	Cosine	N	**98.1%**	**98.9%**	**93.8%**	98.5%
MATCHING NETS (OURS)	Cosine	Y	97.9%	98.7%	93.5%	**98.7%**

Note: ‘Baseline classifier’: trained on all training data, then extract feature from last conv layer for attention module.

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!

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_θ to a new task $\mathcal{T}_i$, with θ udpated to θ_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_θ to a new task $\mathcal{T}_i$, with θ udpated to θ_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_θ means less loss $\mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ on new tasks after one/few (so 'quick adapt') update of model parameter to θ_i'.

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

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_θ to a new task $\mathcal{T}_i$, with θ udpated to θ'_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_θ means less loss $\mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ on new tasks after one/few (so 'quick adapt') update of model parameter to θ'_i.

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

  One task: one 'training data' for meta-learning!
- Note: meta-optimization is performed over model parameters θ, but loss is computed using updated parameters θ'_i.

MAML (cont')

- Meta-optimization over tasks ('training data') to update model param θ

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

MAML (cont')

- Meta-optimization over tasks ('training data') to update model param θ

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

- Meta-gradient update involves a gradient through gradient

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        Evaluate ∇_θ L_{T_i}(f_θ) with respect to K examples
        Compute adapted parameters with gradient descent: θ'_i = θ − α∇_θ L_{T_i}(f_θ)
    end for
    Update θ ← θ − β∇_θ Σ_{T_i~p(T)} L_{T_i}(f_{θ'_i})
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.

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy 1-shot	5-way Accuracy 5-shot
fine-tuning baseline	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
nearest neighbor baseline	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
matching nets (Vinyals et al., 2016)	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
meta-learner LSTM (Ravi & Larochelle, 2017)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML, first order approx. (ours)	$\mathbf{48.07} \pm \mathbf{1.75\%}$	$\mathbf{63.15} \pm \mathbf{0.91\%}$
MAML (ours)	$\mathbf{48.70} \pm \mathbf{1.84\%}$	$\mathbf{63.11} \pm \mathbf{0.92\%}$

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)

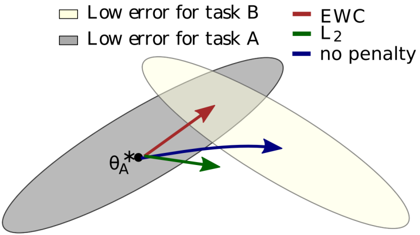
- EWC idea: when learning a new task, do not change weights too much which are important to previous tasks.
- Fisher information matrix $\mathbf{F}$: importance of model params.

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

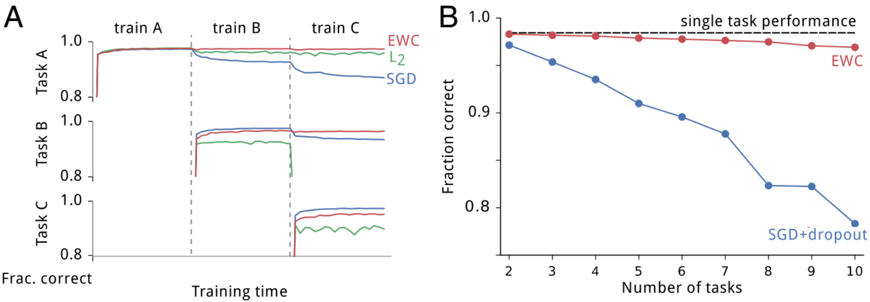
$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta^*_{A,i})^2$$

- 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 focuing 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 δ 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 δ 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

$$\Omega_{ij} = \frac{1}{N} \sum_{k=1}^{N} || \, g_{ij}(x_k) \, ||$$

Memory aware synapse

- Loss is similar to EWC, except the importance parameter

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

- Data label is not necessary when computing Ω_{ij}, so Ω_{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

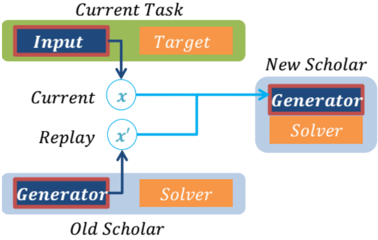
- Idea: generate realistic synthetic data for previous tasks
- Solution: using GAN!

Continual learning with deep generative replay

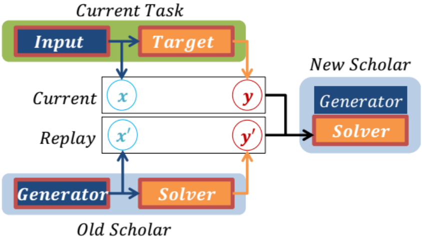
- Idea: generate realistic synthetic data for previous tasks
- Solution: using GAN!
- Dual model 'scholar': (GAN, Solver); Solver, e.g., classifier

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)



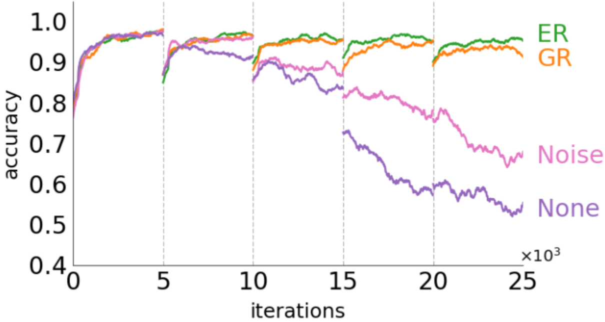
Training Generator



Training Solver

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



- 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
- ...

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!!

Project reports

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!!