Continual Zero-Shot Learning

Ivan Skorokhodov

May 12, 2020

What is Continual Learning?

- Modern neural networks are prone to *catastrophic forgetting*: they forget previous tasks while are learning new ones.
- Continual Learning tries to find ways to make model learn skills one by one in such a way that we do not forget previous skills
  - Example 1: a robot that travels the world and learn new skills. We want it not to forget previous skills while he is acquiring new ones.
  - Example 2: a classification model is learning datasets one by one: we do not want its performance on previously learned datasets to decrease.

Modern Continual Learning techniques

Modern CL techniques can be divided into three groups:

- Regularization-based ([4], [1], etc): detect the weights which are important for previous tasks and do not change them much in the future.
- Rehearsal-based ([2], [6], etc): store a part of previous data to replay it in the future.
- Component-based ([5], [3], etc): divide your network into components, and let future tasks not to break components which are important for previous tasks.

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^{\mathrm{s}}, Y^{\mathrm{s}}, A^{\mathrm{s}}\}$
  - Unseen dataset: $D^u = \{X^{\mathrm{u}}, Y^{\mathrm{u}}, A^{\mathrm{u}}\}$
- During training we have an access only to seen dataset D^s.
  - Our goal is to learn to match images with class descriptions
  - I.e. model learns to detect "blue tails", "large heads", "short beaks", etc and not only concrete bird species
- At test time we evaluate model performance on unseen dataset D^u
- Using the knowledge about how inputs and attributes correspond to each other we can detect birds that we have not seen before just based on their class description a_c.

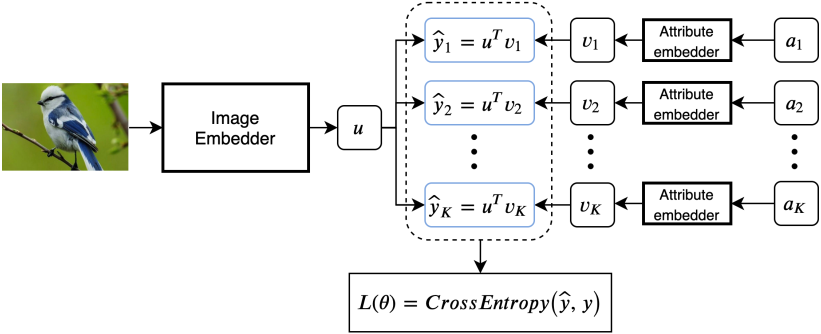
Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based
  - Train a conditional generative model that will learn to generate images based on class descriptions
  - I.e. GAN model that can generate a pigeon given description "grey feather, white head, short legs, etc"
  - At test time generate a lot of synthetic images, then train a classifier based on this synthetic dataset
  - Challenges: how to train a good conditional generative model?
  - Currently performs better than embedding-based approaches

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.
  - And it shouldn't forget previously seen objects while doing so...
  - And the set of attribute descriptions can grow over time...
  - In some sense, it is “the next” step of ZSL
- I.e. build a *zero-shot* model that is trained in a *continual learning* fashion
- Semantic guidance should help to alleviate forgetting without additional regularization and tricks

Baseline model



The approach is similar in spirit to metric learning:

- Image embedder produces $u = f_\theta(x)$
- Attribute embedder produces $v = g_\phi(a)$
- We want the distance $d(u, v)$ to be low for proper pairs x, a and large for improper ones.

“Improved” model

Let’s look closer at logits:

1. We compute c-th logit as $y_c = u^\top v_c$
2. (1) is very similar to $y_c = u^\top V_c u$ for diagonal matrix V_c with $\mathrm{diag}(V_c) = v_c$.
3. (2) is a special case of $(u - \mu_c)^\top V_c (u - \mu_c)$ (Maholonobis distance squared)
4. (3) is a part of gaussian log-density $\mathcal{N}(u|\mu_c, \Sigma_c)$

Building a generative model

- So, let's define a generative model $p(u)$ as a GMM:

$$p(u) = \sum_c \alpha_c \cdot \mathcal{N}(u|\mu_c, \Sigma_c)$$

- Since our classes are balanced, we have $\alpha_c = P(y = c) = \frac{1}{K}$
- Then, we can obtain $P(y_c|u)$ as:

$$P(y_c|u) = \frac{p(u|y_c)P(y_c)}{p(u)} = \frac{\mathcal{N}(u|\mu_c, \Sigma_c) \cdot \frac{1}{K}}{\sum_c \frac{1}{K}\mathcal{N}(u|\mu_c, \Sigma_c)} = \frac{\mathcal{N}(u|\mu_c, \Sigma_c)}{\sum_c \mathcal{N}(u|\mu_c, \Sigma_c)}$$

- This gives us a way to do log-likelihood maximization for $P(y_c|u)$
- The only change is that we now compute logits as $\log \mathcal{N}(u|\mu_c, \Sigma_c)$:

$$\log p(u|y_c) = \log \left[(2\pi)^{-\frac{k}{2}} \det(\Sigma_c)^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c)} \right]$$

How to compute μ_c and Σ_c?

- For attribute a_c let's set $\mu_c = W_\mu a_c$ and $\Sigma_c = \mathbb{W}_\Sigma a_c$
- But Σ_c can be large, so $\mathbb{W}$ will be very expensive to use.
- Solution: let's use low-rank + diagonal approximation for Σ_c:

  $$\Sigma_c = A \times B + \Lambda,$$

  where A, B are low-rank matrices and Λ is diagonal (to fix the rank).
- A problem: in density computation for $\mathcal{N}(u|\mu_c, \Sigma_c)$ we need to compute $\det \Sigma_c$ and Σ_c^{-1}.
- Solution: let's output Σ_c^{-1} directly as LL^T (Cholesky decomposition) and use the identity $\det \Sigma_c = 1/\det(\Sigma_c^{-1})$.

Changes summary

Old logit computation:

$$\log p(u|y_c) = \log u^\top v_c$$

New logit computation:

$$\log p(u|y_c) = -\frac{1}{2}\log\det(\Sigma_c) - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_c)^\top \Sigma_c^{-1}(\mathbf{x} - \boldsymbol{\mu}_c)$$

GMM differs from the baseline in 3 ways:

1. We use full-fletched covariance Σ_c: helps catching more entangled relationships
2. We take into account the shift μ_c
3. We use determinant regularization for Σ_c

Additional benefit: we now have a principled generative model.

Experiments

- Datasets
  - CUB (200 classes): 20 tasks, 10 classes each
  - AwA2 (50 classes): 10 tasks, 10 classes each
  - NABirds (in progress)
- Only a single epoch per task, i.e. we see each example only once per lifetime
- Hyperparameter search:
  - Validation sequence: find the best hyperparams for a model on the first 3 tasks, then train the model on the rest with the best hyperparams.
  - Use the same hyperparameters grid for both models
- Metrics: accuracy on seen, accuracy on unseen, harmonic mean, AUSUC, forgetting, etc.

The problem

The problem is that it does not quite work in practice...

- Baseline model has normalize+scale trick which is *crucial* to achieve good performance
- However, for the GMM this trick is not applicable
- How can we do normalize+scale for the GMM?

References

- Rahaf Aljundi et al. "Memory Aware Synapses: Learning what (not) to forget". In: *CoRR* abs/1711.09601 (2017).
- Arslan Chaudhry et al. "Efficient Lifelong Learning with A-GEM". In: *International Conference on Learning Representations*. 2019.
- Chrisantha Fernando et al. "PathNet: Evolution Channels Gradient Descent in Super Neural Networks". In: *CoRR* abs/1701.08734 (2017).
- James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526.
- Joan Serra et al. "Overcoming Catastrophic Forgetting with Hard Attention to the Task". In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 4548–4557.
- Chenshen Wu et al. "Memory Replay GANs: Learning to Generate New Categories without Forgetting". In: *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, pp. 5962–5972.