Overview of Few-shot Learning

Qinyuan Ye
qinyuany@usc.edu

1 Few-shot Learning

Problem Statement. In few-shot classification, we have three datasets: a training set, a support set and a query set. The support set and the query set share the same label space, but the training set has its own label space that is disjoint with support/query set. If the support set contains K labeled examples for each C unique classes, the target few-shot problem is called C-way K-shot.

Benchmarks. The Omniglot dataset [1] contains 50 examples of 1623 handwritten characters from 50 writing systems. It is considered "transpose" of MNIST dataset, because the number of classes is large in Omniglot, but the examples per class is small. miniImageNet [2] provides 100 classes and 600 examples for each class. Standard setting set aside 20 classes for few-shot testing.

Figure 1: Omniglot dataset. **Right**: An example of 20-way one-shot classification task. The classes in support/query set should not appear in training set.

Discussion. A straight-forward way is to train the classifier on the support set. However, due to lack of labeled samples in each class, the performance is not satisfactory. One effective way to exploit training set is to mimic few-shot learning via episode-based training. In each episode, we sample C classes from training set and sample K examples from each of them. This gives us a *train* set. Also we sample several examples from the same C classes for a *test* set. These train/test sets align with support/query sets. We train the model to learn well on these train and test sets, so that it can generalize to support/query set. This is also the basic idea of meta-learning.

2 Siamese Network [3]

High-level idea. (1) Train a model to discriminate whether input pairs are from the same class or different class. This task is also called *verification* task. (2) Re-used the network for one-shot learning without any re-training.

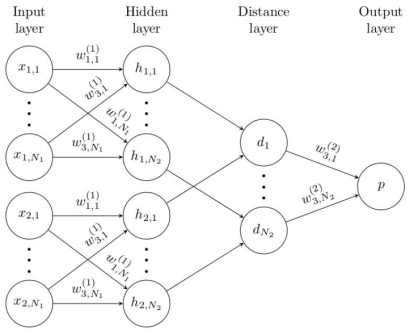

Figure 2: Siamese Network with 2 hidden layer for binary classification with logistic prediction p.

Verification. Use some off-the-shelf encoder which outputs representation $\mathbf{h}_1$ and $\mathbf{h}_2$ for the two inputs $\mathbf{x}_1$ and $\mathbf{x}_2$. Use logistic regression to output the probability p that they belong to the same class.

$$p = \sigma(\sum_j \alpha_j \left|\mathbf{h}_1^{(j)} - \mathbf{h}_2^{(j)}\right|) \tag{1}$$

where σ is the sigmoid function, and α_j are learned during training.

One-shot Prediction. Query the learned verification network with $\hat{x}$ in the query set, and each $x_c, c \in \{1, 2, ..., C\}$ in the support set. The prediction is the class with maximum similarity p.

3 Matching Networks [2]

3.1 Model Architecture

Matching networks are expected to produce sensible test labels for unobserved classes without any changes to the network. That is, we wish to map from a small set of k examples of image-label pairs $S = \{(x_i, y_i)\}_{i=1}^k$ to a classifier $c_S(\hat{x})$, which gives a distribution over outputs $\hat{y}$, given a test example $\hat{x}$. The mapping $S \rightarrow c_S(\hat{x})$ is parameterized by a neural network $P(\hat{y}|\hat{x}, S)$.

In one-shot evaluation, we feed the new support set S' to the learned neural network P, so that we can make predictions about the appropriate label $\hat{y}$ for each query example $\hat{x}$ with $P(\hat{y}|\hat{x}, S')$.

Matching Networks in its simplest form computes $\hat{y}$ as follows:

$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i \tag{2}$$

where x_i and y_i are instances in the support set, and a is an attention mechanism.

Ways to Interpret Eq. (2) The prediction in Eq. (2) subsumes kernel density estimator and kNN methods. Where the attention mechanism a is a kernel on $X \times X$, then Eq. 2 is akin to a kernel density estimator. Where $a = 0$ for $d(\hat{x}, x_i) > b$ and $a = 1$ for $d(\hat{x}, x_i) < b$, Eq. 2 is the same as $k - b$ nearest neighbor. From another perspective, a acts as an attention mechanism and y_i acts as the memories bound to the corresponding x_i. With this, given a input, we "point" to the corresponding example in the support set, retrieving its label.

The Attention Kernel. The attention kernel a, in its most general form, can be written as,

$$a(\hat{x}, x_i) = \frac{\exp(c(f(\hat{x}), g(x_i)))}{\sum_{j=1}^{k} \exp(c(f(\hat{x}), g(x_j)))} \tag{3}$$

where $c(.,.)$ measures cosine distance[1], $f(.)$ and $g(.)$ are encoders to embed $\hat{x}$ and x_j (potentially $f = g$). f and g can be deep CNNs for image tasks, or simple word embeddings for language tasks.

[1] In the Prototypical Network paper, the authors found using square Euclidean distance yields better results.

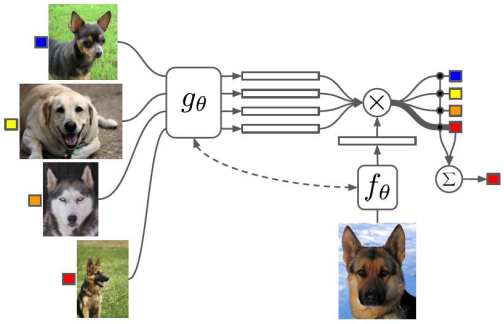

Figure 3: Matching Networks Architecture

Full Context Embeddings. Though the classification strategy is fully conditioned on the whole support set, the encoding process of x_j is myopic, in the sense each x_j is encoded independently of other elements in support set S. Furthermore, $\hat{x}$ should also be encoded with the knowledge of the support set S.

In light of this, we introduce Full Context Embeddings, *i.e.*, $g(x_j)$ become $g(x_j, S)$, where S can influence how we encode x_j. We use a bidirectional LSTM to encode x_i in the context of S considered as a sequence. That is, $g(x_i, S) = g'(x_i) + \overrightarrow{h}_i + \overleftarrow{h}_i$, with:

$$\overrightarrow{h}_i, \overrightarrow{c}_i = \text{LSTM}(g'(x_i), \overrightarrow{h}_{i-1}, \overrightarrow{c}_{i-1})$$
$$\overleftarrow{h}_i, \overleftarrow{c}_i = \text{LSTM}(g'(x_i), \overleftarrow{h}_{i-1}, \overleftarrow{c}_{i-1})$$

As for $f(.)$, $f(\hat{x})$ becomes $f(\hat{x}, S)$ with read-attention over the whole set S.

$$f(\hat{x}, S) = \text{attLSTM}(f'(\hat{x}), g(S), K) \tag{4}$$

where $f'(\hat{x})$ is the original $f(\hat{x})$ provided by the encoder, and is used as the input to LSTM (kept constant at each timestamp). $g(S)$ is the set over which we attend, embedded with g. K is the unrolling steps of the LSTM.

3.2 Training Strategy

The training procedure has to be chosen carefully so as to match inference at test time. In each "episode", we sample L in all possible labels (e.g., {cats, dogs} for two-way classification), then we sample instances of L into a support set S and a query set B (both contain examples of cats and dogs).[2] More precisely, the Matching Networks' training objective is,

$$\theta^* = \underset{\theta}{\text{argmax}}\, E_{L\sim T}\left[E_{S\sim L, B\sim L}\left[\sum_{(x,y)\in B} \log P_\theta(y|x, S)\right]\right] \tag{5}$$

The model works well when sampling $S' \sim T'$ from a different distribution of novel labels. The model does not need any fine-tuning on the classes it has never seen due to its non-parametric nature.

4 Prototypical Networks [4]

High-level idea. There exists an embedding in which points cluster around a single prototype representation for each class. We want to the learn the metric space in which classification can be performed by computing distances to prototype representations of each class.

[2]This is, in fact, formulating the problem into meta-learning. The training procedure learns to learn from support set S to minimize a loss over query set B.

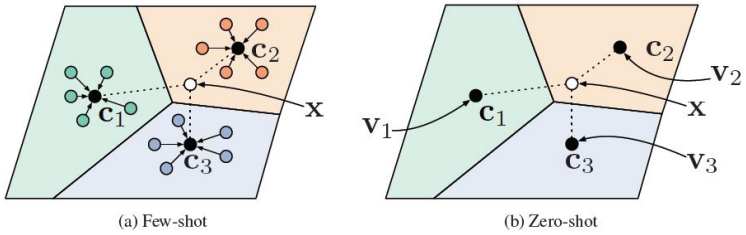

Figure 4: Prototypical Networks in few-shot and zero-shot scenarios.

4.1 Notation

We have a support set of N labeled examples $S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N)\}$, where each $\mathbf{x}_i \in \mathcal{R}^D$ is a D-dimensional feature vector of the example and $y_i \in \{1, 2, ..., K\}$ is the corresponding label. S_k denotes the examples labeled with class k.

4.2 Model

Prototypical Networks compute an M-dimensional representation $\mathbf{c}_k \in \mathcal{R}^M$, called prototype, of each class, through an embedding function $f_\phi : \mathcal{R}^D \rightarrow \mathcal{R}^M$. Each prototype is the mean vector of the embedded support points belonging to its class:

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i) \tag{6}$$

Given a distance function $d : \mathcal{R}^M \times \mathcal{R}^M \rightarrow [0, +\infty)$, Prototypical Networks makes the prediction as follows:

$$p_\phi(y = k|\mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}'_k))} \tag{7}$$

The optimization objective is negative log-probability of true class k, $J(\phi) = -\log p_\phi(y = k|\mathbf{x})$, and is optimized with standard SGD.

4.3 Interpretation

Mixture Density Estimation. For a particular class of distance functions, known as *regular Bregman divergences*, the Prototypical Networks algorithm is equivalent to performing mixture density estimation on the support set with an exponential family density. [3]

For Bregman divergence that the cluster representative achieving minimal distance to its assigned points is the cluster mean. Thus Eq. (6) yields the optimal cluster representatives given the support set labels when a Bregman divergence is used.

For any exponential family distribution $p_\phi(\mathbf{z}|\theta)$ with parameters θ and cumulant function ψ can be written in terms of a uniquely determined regular Bregman divergence.[4]

$$p_\psi(\mathbf{z}|\theta) = \exp\{-d_\phi(\mathbf{z}, \mu(\theta)) - g_\phi(\mathbf{z})\} \tag{8}$$

If our prediction model is a regular exponential family mixture model, with parameters $\mathbf{\Gamma} = \{\theta_k, \pi_k\}_{k=1}^K$, it can be expressed with

$$p(\mathbf{z}|\mathbf{\Gamma}) = \sum_{k=1}^K \pi_k p_\phi(\mathbf{z}|\theta_k) = \sum_{k=1}^K \pi_k \exp(-d_\phi(\mathbf{z}, \mu(\theta_\mathbf{k})) - g_\phi(\mathbf{z})) \tag{9}$$

[3] An example of regular Bregman divergence is Euclidean distance. An example of exponential family distribution is normal (Gaussian) distribution.

[4] There is a one-to-one mapping between member of exponential family and unique Bregman Divergence. Here, the cumulant function ψ maps to a function θ, so that the distribution fits in the Bregman divergence form.

So that during inference, we assign label k to unlabeled point $\mathbf{z}$ with probability $p(y = k|\mathbf{z})$.

$$p(y = k|\mathbf{z}) = \frac{\pi_k \exp(-d_\phi(\mathbf{z}, \mu(\theta_k))}{\sum_{k'} \pi_{k'} \exp(-d_\phi(\mathbf{z}, \mu(\theta_{k'}))} \tag{10}$$

Comparing Eq. (7) to Eq. (10), we found that Eq. (7) is a equally-weighted mixture model with one cluster per class. Prediction with Eq. (7) is the same as cluster assignment with Eq. (10) with $f_\phi(\mathbf{x}) = \mathbf{z}$ and $\mathbf{c}_k = \mu(\theta_k)$.

Above all, we can interpret Prototypical Networks as doing clustering with mixture density estimation, which an exponential family distribution determined by d_ϕ.

Linear Model. When we use Euclidean distance $d(\mathbf{z}, \mathbf{z}') = ||\mathbf{z} - \mathbf{z}'||^2$, the model in Eq. (7) is equivalent to a linear model, We can expand the terms in Eq. (7) with

$$-||f_\phi(\mathbf{x}) - \mathbf{c}_k||^2 = -f_\phi(\mathbf{x})^T f_\phi(\mathbf{x}) + 2\mathbf{c}_k^T f_\phi(\mathbf{x}) - \mathbf{c}_k^T \mathbf{c}_k \tag{11}$$

The first term is constant and won't affect softmax probabilities. The remaining parts are

$$2\mathbf{c}_k^T f_\phi(\mathbf{x}) - \mathbf{c}_k^T \mathbf{c}_k = \mathbf{w}_k^T f_\phi(\mathbf{x}) + b_k \tag{12}$$

where $\mathbf{w}_k = 2\mathbf{c}_k$ and $b_k = -\mathbf{c}_k^T \mathbf{c}_k$. In their experiments, they found Euclidean distance is an effective choice despite the equivalence to a linear model. The authors hypothesize this is because of all the required non-linearity are alreay learned within embedding function f_ϕ.

Comparison to Matching Networks. The two models are the same in one-shot scenario when both are using Euclidean distance, because the average of one point is itself, *i.e.*, $\mathbf{c}_k = \mathbf{x}_k$.

4.4 Episode Composition

In previous work, people usually choose N_c classes and N_S support points per class in order to match the expected situation at test-time. The authors have found, however, it can be extremely beneficial to train with a higher N_c, than will be used at test-time; but it's usually best to train and test with the same N_s.

5 Relation Networks [5]

In previous work we're using cosine similarity or Euclidean distance. In relation network we aim learn a transferrable deep metric for comparing the relation between images (few-shot learning) or between image and class descriptions (zero-shot learning). The Relation Networks has two modules. An embedding module generates representations of the query and training images; a relation module compares the embeddings and determines if they're from matching categories or not.

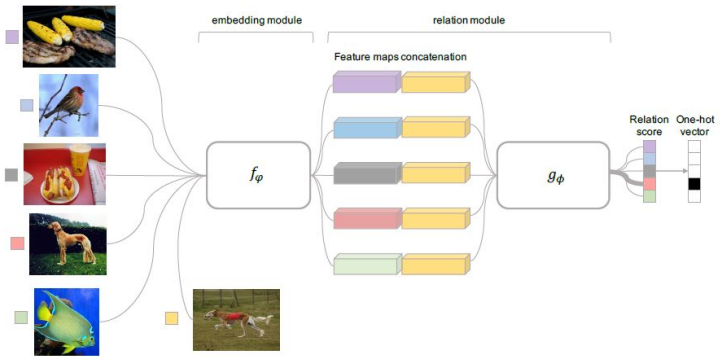

Figure 5: Relation Network architecture for a 5-way 1-shot problem with one query example.

5.1 Model Design

One-shot. In the C-way one-shot setting, we generate C relation scores $r_{i,j}$ for the relation between one query input x_j and training sample set examples x_i,

$$r_{i,j} = g_\psi(\mathcal{C}(f_\phi(x_i), f_\phi(x_j)), \qquad i = 1, 2, ..., C \tag{13}$$

where f_ϕ is the embedding module, g_ψ is the relation module, $\mathcal{C}$ means concatenation.

Specifically, f_ϕ consists of four convolutional blocks, and g_ψ consists of two convolutional blocks and two fully connected layers. One convolutional block contains a 64 filter 3×3 convolution, followed by batch normalization, ReLU non-linearity and 2×2 max pooling.

K-shot. We element-wise sum over the embedding module outputs of all samples from the same class to form this class' feature map. Then this pooled class-level feature is combined with query image feature map to calculate a score, as in Eq. (13).

Objective. Mean square error (MSE) is used because we can consider it as a regression problem on predicting relation "scores".

5.2 Analysis

Why is relation module helpful? In previous work cosine similarity and Euclidean distance are used. These metrics solely do comparison in an element-wise way. Also, prototypical network assumes linear separability after the embedding, which in fact strongly depend on the ability of embedding module. By deep learning a non-linear similarity metric jointly with the embedding, Relation Network can better identify matching/mismatching pairs.

References

[1] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[2] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[3] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

[4] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4080–4090. Curran Associates Inc., 2017.

[5] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.