Meta Networks

Munkhdalai and Yu

5.16.17

Summary by Justin Chen

Summary:

- Meta Networks address having to collect larges amounts of labeled data by demonstrating that meta information can be used for one-shot learning, continual learning, reverse transfer learning, and rapid generalization
- Meta Networks is composed of two neural networks, the meta learner and the base learner, and an external memory
- Meta network learns generic knowledge across different tasks and transfers what it learned to the base network which generalizes to that specific task
- Learning occurs in both the meta space and task space
    - Base network operates in input task space
    - Meta networks operates in task-agnostic meta space
- Weights of Meta Network change at different rates
    - Slow weights - updated with backprop to capture meta information
        - Task-agnostic
    - Fast weights - updated for each task
        - Task-dependent
- Used gradient of base network as meta information
    - Other information could also be used as meta information in lieu of base network's gradient
    - However, meta information must reflect base network so that meta network can help the base network learn faster
- **Slow weights**
    - Batch update parameters regardless of current task
- **Fast weights**
    - Batch size = 1, normal SGD

$$\mathcal{L}_t = loss_u(u(Q, x_t), y_t)$$
$$\nabla_t = \nabla_Q \mathcal{L}_t$$
$$Q^* = m^{LSTM}(G, \nabla_t)$$

    - Sample T examples from support set to calculate meta information
    - Loss function can be different from the base network's, but should allow the meta network to learn a good representation cost
        - When dealing with more than one example per class, contrastive loss is suggested

- One-shot learning experiments have an extra dataset called the “support set”, which contains a single example for each class, which is called the “support example”
    - Assuming want to do classification, not sure what the support set would contain for regression or generative tasks
- **Base Learner**
    - Given input vector, and fast weights
    - Neural network parameterized by both slow weights and fast weights
    - Performs one-shot learning given fast weights and current input vector
      $$P(y_j|x_j, W, W_j^*) = b(W, W_j^*, x_j)$$
    - Uses label set to train and minimize cost function to update the slow weights
- **Label Set**
    - Separate labeled dataset for minimizing cost function use for one-shot
- **Meta Information**
    - Meta information obtained by generating gradient with a single forward and backwards pass to get:
      $$\mathcal{L}_i = loss_b(b(W, x_i), y_i)$$
      $$\nabla_i = \nabla_W \mathcal{L}_i$$
        - Loss for i^{th} input support example label pair
        - Loss function can be any function appropriate for task
        - Product of loss and gradient w.r.t. slow weights
        - Loss is given output of base network, which is given slow weights and support set input and label
- **Meta Learner**
    - Given parameters Z and **meta loss gradient**
    - Transforms meta loss gradients for each support sample into a set of fast weights corresponding to each support example
    - Fast weights stored in memory
    - Does this for each task
- **Memory**
    - Indexed with **task-dependent representations**, set R, corresponding to support set
        - Each support example has a r_i, which is the input vector from the label set embedded into the **task space**
        - Set R obtained with dynamic key embedding neural network
    - Fast weights are read from memory with attention mechanism
        - Given **task-dependent representation** set R and a specified index r_j to get attention vector a_j
        - Fast weight corresponding to attention vector is computed as softmax transpose of attention vector times the memory matrix
            - Basically the attention vector selects the row in the matrix by masking out all other rows

- **Dynamic Key Embedding Neural Network**
    - Given fast weights, slow weights, and support input vector
    - Generates the **task-dependent representations** for indexing fast weights in memory
    - Fast and slow weights are individually transformed with ReLU, concatenated, and split apart continuously
    - After repeating the transformations in the previous bullet, the concatenation of both the fast and slow weights are concatenated and passed through a softmax layer
    - Fast and slow weights in this networks are used as **feature detectors** each in a different domain
        - Nonlinearity maps both into the same domain (x-axis) so that they can be processed together as if a single weight vector
- Losses for dynamic key embedding function and for base network are not used for parameter network updates
- **Training:**
    - Dynamic embedding network and base network use same architecture:
        - 5 conv layers, 3x3x64, ReLU, 2x2 max-pooling, 2 FC layers, and then softmax
    - Meta learner m^{LSTM}:
        - Single-layer LSTM, 20 units wide using ReLU
        - Normalized gradients
    - Memory indexer networks:
        - 3-layer MLP 20 units wide using ReLU
        - Normalized gradients
    - ADAM with learning rate of 10e-3
    - Randomly initialized uniformly over [-0.1, 0.1)
- **Results:**
    - Demonstrates one-shot learning, continual learning, reverse transfer learning, and rapid parameterized
    - In rapid generalization test, replace base learner with no learned weights and parameterize it with the fast weights from the meta learner and have it bias weight values to current task
    - Neatest result was that if trained for N-way classification, can easily generalize to K classes. If K > N, accuracy degrades at K increase. If K < N, accuracy increases as K decreases.
        - Kinda obvious property since the probability is less spread out limiting its choices so it gains more confidence or loses confidence as the number of classes changes.

Questions:

1. Are slow weights batch updates?
2. How is this one-shot learning if it needs an additional L dataset for minimizing the cost function?

3. Why does it need the augmented layer?
4. Why does the augmented layer concatenate the fast and slow weights?
5. Why does are the weights transformed in the augmented layer?
6. Would increased training with more labeled data increase one-shot accuracy?
7. What are other potential meta information that can be used?
8. Is it possible to have multiple parallel meta networks or multiple parallel base networks to learn to multitask as a single end-to-end network?