Continual learning with Hypernetworks

January 9, 2020

Model description

Model description

- A hypernetwork (meta-model) is just a model which produces the weights for some other model.

Model description

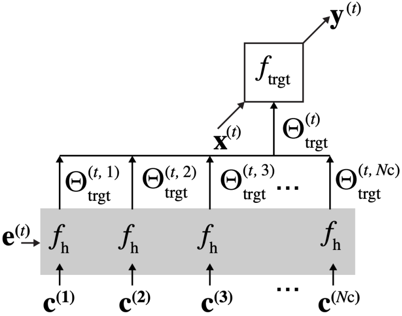
- A hypernetwork (meta-model) is just a model which produces the weights for some other model.
- What is an input to a meta-model? Task input data, task embeddings, random noise, etc.

Model description

- A hypernetwork (meta-model) is just a model which produces the weights for some other model.
- What is an input to a meta-model? Task input data, task embeddings, random noise, etc.
- A meta-model can have less parameters than a target model!

Model description

- A hypernetwork (meta-model) is just a model which produces the weights for some other model.
- What is an input to a meta-model? Task input data, task embeddings, random noise, etc.
- A meta-model can have less parameters than a target model!
- A good way to reduce number of parameters for a meta-model is to condition it on a layer embedding.



How to use hypernetworks not to forget previous tasks

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.
- Add a regularization term that ensures that its outputs on previously learned task embeddings do not change:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{cls}}^{T}(\theta) + \beta \sum_{t=1}^{T-1} \|h(\theta_{\text{old}}, e_t) - h(\theta + \Delta\theta, e_t)\|_2^2,$$

where:

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.
- Add a regularization term that ensures that its outputs on previously learned task embeddings do not change:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{cls}}^{T}(\theta) + \beta \sum_{t=1}^{T-1} \| h(\theta_{\text{old}}, e_t) - h(\theta + \Delta\theta, e_t) \|_2^2,$$

where:

- $\mathcal{L}_{\text{cls}}$ is our classification loss at the current task $t = T$;

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.
- Add a regularization term that ensures that its outputs on previously learned task embeddings do not change:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathsf{cls}}^{T}(\theta) + \beta \sum_{t=1}^{T-1} \| h(\theta_{\mathsf{old}}, e_t) - h(\theta + \Delta\theta, e_t)\|_2^2,$$

where:

- $\mathcal{L}_{\mathsf{cls}}$ is our classification loss at the current task $t = T$;
- h is our hypernetwork;

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.
- Add a regularization term that ensures that its outputs on previously learned task embeddings do not change:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{cls}}^{T}(\theta) + \beta \sum_{t=1}^{T-1} \| h(\theta_{\text{old}}, e_t) - h(\theta + \Delta\theta, e_t) \|_2^2,$$

where:

- $\mathcal{L}_{\text{cls}}$ is our classification loss at the current task $t = T$;
- h is our hypernetwork;
- θ is the current parameter vector of h;

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.
- Add a regularization term that ensures that its outputs on previously learned task embeddings do not change:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{cls}}^{T}(\theta) + \beta \sum_{t=1}^{T-1} \| h(\theta_{\text{old}}, e_t) - h(\theta + \Delta\theta, e_t) \|_2^2,$$

where:

- $\mathcal{L}_{\text{cls}}$ is our classification loss at the current task $t = T$;
- h is our hypernetwork;
- θ is the current parameter vector of h;
- $\Delta\theta$ is the change in parameters produced by Adam optimizer to minimize $\mathcal{L}_{\text{cls}}$.

How to use hypernetworks not to forget previous tasks

- Condition a hypernetwork on task embedding e_t.
- Add a regularization term that ensures that its outputs on previously learned task embeddings do not change:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathsf{cls}}^{T}(\theta) + \beta \sum_{t=1}^{T-1} \| h(\theta_{\mathsf{old}}, e_t) - h(\theta + \Delta\theta, e_t) \|_2^2,$$

where:

- $\mathcal{L}_{\mathsf{cls}}$ is our classification loss at the current task $t = T$;
- h is our hypernetwork;
- θ is the current parameter vector of h;
- $\Delta\theta$ is the change in parameters produced by Adam optimizer to minimize $\mathcal{L}_{\mathsf{cls}}$.
- θ_{old} — parameters snapshot before starting to learn the current task.

CL setups

Authors consider 3 different scenarios for continual learning:

CL setups

Authors consider 3 different scenarios for continual learning:

- With knowing task identities;

CL setups

Authors consider 3 different scenarios for continual learning:

- With knowing task identities;
- Without knowing task identities with a multi-headed model;

CL setups

Authors consider 3 different scenarios for continual learning:

- With knowing task identities;
- Without knowing task identities with a multi-headed model;
- Without knowing task identities with a single-headed model.

And they propose two ways to infer a task identity:

CL setups

Authors consider 3 different scenarios for continual learning:

- With knowing task identities;
- Without knowing task identities with a multi-headed model;
- Without knowing task identities with a single-headed model.

And they propose two ways to infer a task identity:

- Check which task embedding gives the lowest entropy;

CL setups

Authors consider 3 different scenarios for continual learning:

- With knowing task identities;
- Without knowing task identities with a multi-headed model;
- Without knowing task identities with a single-headed model.

And they propose two ways to infer a task identity:

- Check which task embedding gives the lowest entropy;
- Train a generative memory model and be ready to classify without task identity (in a multi-headed variant);

CL setups

Authors consider 3 different scenarios for continual learning:

- With knowing task identities;
- Without knowing task identities with a multi-headed model;
- Without knowing task identities with a single-headed model.

And they propose two ways to infer a task identity:

- Check which task embedding gives the lowest entropy;
- Train a generative memory model and be ready to classify without task identity (in a multi-headed variant);
- Train a generative memory and a model that predicts task identity by an input (in a single-headed variant).

Details, results and discussion

Details, results and discussion

- Learned task embeddings tend to have a good performance for all the previous tasks as well.

Details, results and discussion

- Learned task embeddings tend to have a good performance for all the previous tasks as well.
- Authors use a VAE model for their generative memory model (people usually use GANs).

Details, results and discussion

- Learned task embeddings tend to have a good performance for all the previous tasks as well.
- Authors use a VAE model for their generative memory model (people usually use GANs).
- They compare the approach to EWC, SI and DGR (these are from 2017), beat them and claim SotA.

Details, results and discussion

- Learned task embeddings tend to have a good performance for all the previous tasks as well.
- Authors use a VAE model for their generative memory model (people usually use GANs).
- They compare the approach to EWC, SI and DGR (these are from 2017), beat them and claim SotA.
- A drawback of the approach is the necessity to recompute parameters for all the previous target models on each iteration.