Online Structured Laplace Approximations for Overcoming Catastrophic Forgetting

Problem Statement

Online learning – learning algorithm, when we get new tasks, but don't retrain on previous tasks. During such learning appears problem of catastrophic forgetting: when we forget information gained during previous tasks.

Elastic weight consolidation

Main formula:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i \cdot (\theta - \theta^*_A)^2,$$

where F_i – Fisher matrix, θ – weight, θ^* – optimal corresponding weight obtained at the previous task, $\mathcal{L}_B(\theta)$ – loss function at the current task.

Fisher matrix – negative expected Hessian of log likelihood of the model. We can use it to approximate precision of parameters distribution

$$F = \frac{1}{N} \sum_{i=1}^{N} \nabla \log p(x_i|\theta) \nabla \log p(x_i|\theta)^T$$

EWC – approximation of Hessian with Fisher matrix diagonal.

Code documentation

In "EWC loss experiment.ipynb" file there is an experiment of comparing online learning with EWC and without.

Data.py file generates task_number permuted MNIST datasets (each dataset consists of MNIST images permuted with the same transformation)

EWC training cylce consists of base training cycle that used in training model without EWC, and consolidation part – re-estimation of Fisher matrix for current train dataset.

Kronecker factored approximation

Loss:

$$\mathcal{L}_{1:t+1}(\theta) = \mathcal{L}_{t+1}(\theta) + \frac{1}{2}(\theta - \theta^*_t)^T \Lambda_t (\theta - \theta^*_t)$$

where Λ – precision matrix of Gaussin distribution.

After each task we update optimal parametyrs and precision: $\Lambda_{t+1} = H_{t+1}(\theta^*_{t+1}) + \Lambda_t$

Method of Kronecker factored approximation – approximation of Hessian $H_{t+1}(\theta^*_{t+1})$ with block-diagonal matrix, where each block corresponds to layers of nerual network (a_{l-1} – input of layer, W_l – weight matrix of layer, $h_l = W_l a_{l-1}$, f_l – activation function of the linear layer). Block H_l can be presented in Kronecker factored form:

$$H_l = \mathcal{Q}_l \otimes \mathcal{H}_l,$$

where

$$Q_l = a_{l-1}a_{l-1}^T, \mathcal{H}_l = B_l W_{l+1}^T \mathcal{H}_{l+1} W_{l+1} B_l, B_l = diag(f_l'(h_l))$$

where recursion of $\mathcal{H}_l$ initialized with $\mathcal{H}_L = \frac{\partial^2 E(target, model\_output)}{\partial h_L \partial h_L}$.

Code documentation

Training cycle consists of two stages – model weights update and factors updates.

To update $\mathcal{Q}$ we need to iterate through all data and store $a_{l-1}a_{l-1}^T$ for each layer (self.Q_factors) – at each layer we sum values of $\mathcal{Q}_l$ and in the end devide by dataset size. To calculate recursivly $\mathcal{H}_l$ we first during iteration through dataset store $B_l W_{l+1}^T$ in self.WB_factors in the same manner as $\mathcal{Q}_l$. After training through one task we calculate recursivly $\mathcal{H}_l$ using self.WB_factors. In the end we sum up current $\mathcal{Q}_l, \mathcal{H}_l$ with previous $\mathcal{Q}_{l+1}, \mathcal{H}_{l+1}$ with coefficient λ.

Model used in experiment – linear model with 2 hidden layers and ReLU activations.