Deep Learning Chapter 7

Jee Dong Jun

23.4.2020

Parameter Tying

- From prior knowledge, we know there exists some dependencies between model parameters.
- Certain parameters should be close to one another

Example

Two models performing similar task, model A with parameters $\boldsymbol{w}^{(A)}$and model B with parameters $\boldsymbol{w}^{(B)}$
Parameter norm penalty of form $||\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}||^2$

Parameter Sharing

- Parameter sharing goes further and force sets of parameters to be same
- Various models or model components as sharing a unique set of parameters.
- Only need subset of parameters to be stored

Sparse Representations

- Weight decay imposes penalty on model parameters. Instead we can imposes penalty on output of hidden layers
- This imposes indirect penalty on parameters

Representations

To extend linear model, we consider transformed input $\phi(\boldsymbol{x})$. We can think of neural network (with linear output layer) as $y = \boldsymbol{W}^T \phi(\boldsymbol{x}; \Theta)$. Therefore, we can think of output of hidden layers as some representations of data $\boldsymbol{x}$

Sparse Representations

- We regularize the representation by using penalty $\Omega(\boldsymbol{h})$

$$\tilde{J}(\boldsymbol{\Theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\Theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

- Sparse representation means many elements of vector $\boldsymbol{h}$ are zero.
- We saw that L1 penalty on parameters induce sparsity.
- Similarly, we use L1 penalty on representations to induce sparsity

Sparse Representations

- Orthogonal matching pursuit encodes an input $\boldsymbol{x}$ with the representation $\boldsymbol{h}$ by solving

$$\underset{\boldsymbol{h}, ||\boldsymbol{h}||_0 < k}{\operatorname{argmin}} ||\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}||^2$$

- Kind of feature learning algorithm which we can use with other algorithm.(?)

Bagging and other ensemble method

- Technique of combining several models are called ensemble methods (model averaging)
- Each member of the ensemble could be formed by training a completely different kind of model
- For bagging, we use models trained with different train sets.
- We create k different datasets by sampling with replacement. Then models are trained on each sets.

Boosting

- Boosting builds ensemble by adding neural networks to the ensemble.
- At each step t, we train neural network h_t using $D_t(i)$ as weight of dataset ($D_1(i) = \frac{1}{N}$)
- calculate error ϵ_t and let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$, update D_{t+1} using β_t and D_t
- decision rule is based on sum of $log 1/\beta_t$

Dropout

- Bagging involves training multiple models and evaluating multiple models
- Generally, these ensemble methods are computationally expensive
- Dropout is cheap approximation for a bagged ensemble.
- Dropout trains the ensemble consisting of all neural network that can be obtained by removing any number of nonoutput units from an base network

Description of Dropout

- Minibatch-based learning algorithm such as stochastic gradient descent.
- Each time we load an example into a minibtach, we randomly sample a different binary mask μ. (equivalent to choosing one model from all the subnetwork)
- We then run forward propagation, and update as usual.

What Dropout Training is doing?

- $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model
- Dropout training consists of minimizing $E_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$
- Let batch size be k, then we are calculating $J(x^{(i)}; \boldsymbol{\theta}, \boldsymbol{\mu}^{(i)})$ for $i = 1, ..., k$

Dropout Training in terms of ensemble method

- We are taking ensemble of all subnetwork of base network.
- Instead of training all subnetworks to convergence, we exploits parameter sharing.
- We think of each subnetworks as sharing parameters. Therefore, we only train each submodel for a single step.
- The training set encountered by each subnetwork is sampled with replacement. (?)

Making inference

- In order to make inference, ensemble must accumulate votes from all models
- Suppose model output is probability distribution $p^{(i)}(y|x)$
- Then,we take arithmetic mean of all these distributions as prediction of the ensemble.

$$\frac{1}{k}\sum p^{(i)}(y|x)$$

- For dropout we need to calculate

$$\Sigma p(\boldsymbol{\mu})p(y|x,\boldsymbol{\mu})$$

Making inference

- This sum requires us to evaluate exponential number of models.
- Approximate this by sampling μ 10 or 20 times and take average.
- Instead we can use geometric mean

$$\tilde{p}_{ensemble}(y|x) = (\Pi p(y|x,\mu))^{1/(2^d)}$$

- We need to normalize this, as geometric mean does not guarantee that it is probability distribution.

$$p_{ensemble}(y|x) = \frac{\tilde{p}_{ensemble}(y|x)}{\sum_{y'} \tilde{p}_{ensemble}(y'|x)}$$

Making inference

- We can use the weight scaling inference rule.
- Approximate $p_{ensemble}(y|x)$ by evaluating $p(y|x)$ in one model: the model with all units but with the weights going out of unit i multiplied by the probability of including unit i
- This approximation is exact in many classes of models that do not have nonlinear hidden units.

Why dropout?

- Computationally cheap ensemble method
- It works well with any type of model or training procedure.
- The cost of using dropout in a complete system can be significant. (Larger model+many iterations)
- Stocahsticity not required (it is just process of estimating submodels)
- Hidden units become indepedent
- Clever destruction of extracted features
- multiplicative noise

Adversarial Training

- We observe examples that neural network misclassify.
- Adversarial Examples are constructed on purpose by using an optimization procedure such that network misclassify.
- We can reduce the error rate by training on adversarial examples. (Adversarial Training)
- Main cause is excessive linearity. If each input is changed by ϵ, then a linear function can change as $\epsilon||\boldsymbol{w}||_1$
- Adversarial training encourages local constancy

Tangent Distance,Prop and manifold tangent classifier

- Manifold hypothesis assumes that the data lies near a low-dimensional manifold
- Tangent distance algorithm is nearest neighbour algorithm with manifold distance as metric. (estimated by tangent plane)
- This requires one to specify the tangent vectors.
- Tangent prop algorithm trains a neural net classifier to be locally invariant.

Tangent Prop Algorithm

- Local invariance is achieved by $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ is orthogonal to the known manifold tangent vectors $\boldsymbol{v}^{(i)}$ or equivalently, direction derivative is zero
- This can be achieved by adding a regularization penalty

$$\Omega(f) = \sum_i (((\nabla_{\boldsymbol{x}} f(\boldsymbol{x}))^T \boldsymbol{v}^{(i)})^2$$

- This approach poses difficulties for models based on rectified linear units. (?)
- $\boldsymbol{v}^{(i)}$ are prior knowledge. The manifold tangent classifier eliminates the need to know the tangent vectors a priori