Chapter 7: Regularization (Part 2)

Dong-Ho Lee
University of Southern California
dongho.lee@usc.edu

September 17, 2019

1 Parameter Tying and Parameter Sharing

We have discussed regularization with respect to a fixed region or point so far. For example, L2 regularization penalizes model parameters for deviating from the fixed value of zero, but close to zero. Sometimes we need other ways to express prior knowledge of parameters. we might not know what values the parameter should take. However, we may know from domain and model architecture that there should be some dependencies between model parameters.

1.1 Parameter Tying

Assume that we have two models A and B with parameters W^A and W^B. If the tasks are similar enough that the both model parameters should be close to each other, W_i^A should be close to W_i^B for all i. We can leverage this information via regularization. Specifically, we can use a parameter norm penalty of the form $\Omega(W^A, W^B) = \| W^A - W^B \|_2^2$. Here we used an L2 penalty, but other choices are also possible.

One example of this approach is hybrid of discriminative and generative models. (Lasserre, 2006) First, train one model on small amount of labelled data as supervised paradigm. Then make the model parameters to be close to the parameters of another model which learns joint probability of an unlabelled data as unsupervised paradigm.

$$p\left(c|x, W^A\right) \rightarrow p\left(c, x|W^B\right)$$

1.2 Parameter Sharing

Parameter sharing imposes much stronger assumptions on parameters through forcing the parameter sets to be equal. A significant advantage of parameter sharing is that only a subset of the parameters needs to be stored in memory.

Convolutional Neural Network As shown in figure 1, Convolutional Neural Network (CNN) shares kernel parameters across multiple image locations. It enabled CNN to dramatically lower the number of unique parameters and to significantly increase network sizes without requiring corresponding increase in training data.

Siamese Network (Koch et al, 2015) Siamese Network propagates both inputs through identical twin neural nets with shared parameters. Then, use the absolute difference as the input to a linear classifier for learning similarity function.

2 Sparse Representations

Inducing sparse representations by penalization can be categorized into parameter sparsity and representational sparsity.

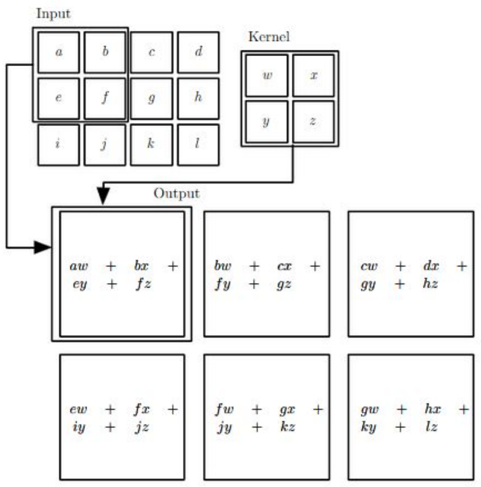

Figure 1: Parameter Sharing of CNN

Parameter Sparsity L1 penalization induces a parameter sparsity, meaning that many of the parameters corresponding to useless features become zero. (or close to zero)

$$\underbrace{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}}_{\boldsymbol{\theta} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}_{\boldsymbol{x} \in \mathbb{R}^n}$$

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \Omega(\boldsymbol{\theta})$$
$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \textstyle\sum_i |\theta_i|$$

Representational Sparsity Another strategy is to place penalty on the activation of the units in the neural network. Representational sparsity describes a representation where many of the elements of the representation become zero. (or close to zero). Below is linear regression with a sparse representation h of the data x.

$$\underbrace{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}}_{\boldsymbol{\theta} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}_{\boldsymbol{h} \in \mathbb{R}^n}$$

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha \Omega(\boldsymbol{h})$$
$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \textstyle\sum_i |h_i|$$

Orthogonal matching pursuit Orthogonal matching pursuit is an approach that obtain representational sparsity with a hard constraint on the activation values. It encodes an input x with the representation h that solves below constrained optimization problem:

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

where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of h. This method is often called OMP-k.

3 Bagging and other Ensemble Methods

Ensemble methods are techniques employing strategies such as model averaging and bagging. The idea is to train several different models separately, then all the models vote on the output for test examples. The reason that ensemble methods work is that different models will usually not make all the same errors on the test set.

Consider a set of k regression models and each model makes an error ϵ_i on each example. With the errors drawn from a zero-mean multivariate normal distribution with $\mathbb{E}\left[\epsilon_i^2\right] = v$ and covariances $\mathbb{E}\left[\epsilon_i \epsilon_j\right] = c$, error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.

Then, the expected squared error of the ensemble predictor is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

If errors are perfectly correlated ($c = v$), then mean squared error reduces to v, so model averaging does not help. However, if errors are perfectly uncorrelated ($c = 0$), expected squared error of ensemble is only $\frac{1}{k}v$. It means that if the members make independent errors, the ensemble will perform significantly better than its members.

Model Averaging Model averaging is the simplest form of ensemble learning. Multiple models are trained on the same dataset, and during prediction, take the average over multiple models.

Bagging Bagging (short for bootstrap aggregating) constructs k different datasets. Each dataset has the same number of instances as the original dataset, but each dataset is constructed by sampling with replacement from the original dataset. Each model is then trained on each dataset.

4 Dropout

Ensemble seems impractical when each model is a large neural network. However, dropout provides an inexpensive approximation to train and evaluate a bagged ensemble of exponentially many neural networks. Specifically, dropout trains the ensemble consisting of all subnetworks that can be formed by removing non-output units from an underlying base network as Figure 2. We can effectively remove units by multiplying its output value by binary mask μ. Dropout training consists of minimizing the cost of model defined by parameter θ and mask μ.

$$\text{minimize } \mathbb{E}_\mu J(\boldsymbol{\theta}, \boldsymbol{\mu})$$

To train with dropout, we use mini-batch based algorithm that takes small steps such as SGD. At each step, we randomly sample a different binary mask to apply to all the input and hidden units in the network.

4.1 Difference between Bagging and Dropout

Training In the case of bagging, the models are all independent. However, in the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing allows an exponential number of models with a tractable amount of memory.

Prediction The prediction of the ensemble is given by the arithmetic of all probability distributions which are produced by each model i.

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y|\boldsymbol{x})$$

In the case of dropout, each submodel defined by mask vector μ defines a probability distribution $p(y|\boldsymbol{x}, \boldsymbol{\mu})$. The arithmetic mean over all masks is given by

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

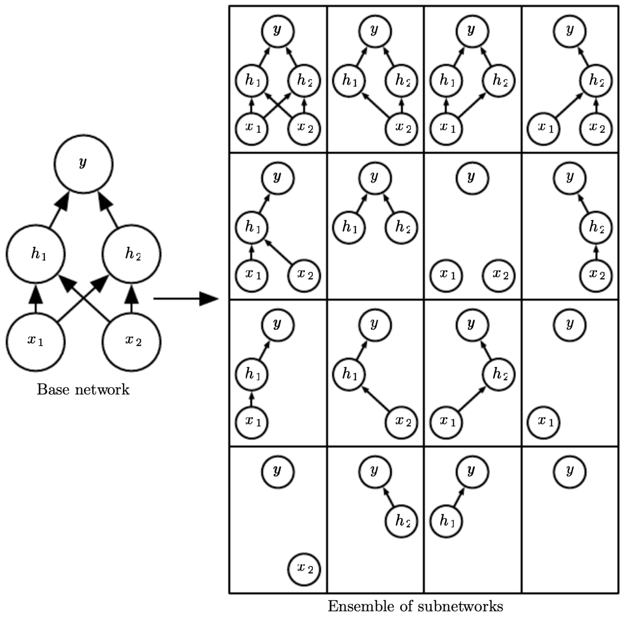

Figure 2: Dropout trains ensemble consisting of all subnetworks

where $p(\boldsymbol{\mu})$ is the probability distribution that was used to sample $\boldsymbol{\mu}$ at training time. Due to an exponential number of terms, it is intractable to predict. Thus we can approximate inference using mask sampling. Even 10-20 masks are sufficient for good performance.

4.2 Better approaches for the dropout

Geometric mean The geometric mean of multiple probability distributions is not guaranteed to be a probability distribution. To guarantee that the result is a probability distribution, we impose the requirement that none of the submodels assigns probability 0 to any event, and we renormalize the resulting distribution. The unnormalized probability distribution defined directly by the geometric mean is given by

$$\tilde{p}_{\text{ensemble}}(y|\boldsymbol{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y|\boldsymbol{x}, \boldsymbol{\mu})}$$

where d is the number of units that may be dropped. Here we use a uniform distribution over μ to simplify the presentation, but nonuniform distributions are also possible. To make predictions we must renormalize the ensemble:

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

Weights Scaling Inference Rule At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in Figure 3. This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time. By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time. (Srivastava, 2014) found that training a network with dropout and using this approximate averaging method at test time leads to significantly lower generalization error on a wide variety of classification problems compared to training with other regularization methods.

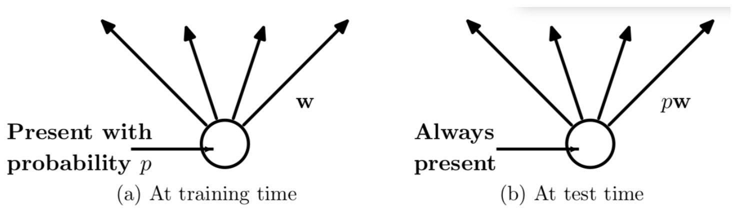

Figure 3: Weight Scaling Inference Rule

5 Adversarial Training

Szegedy et al.(2014b) found that even neural networks that perform at human level accuracy have a nearly 100 percent error rate on examples that are intentionally constructed by using an optimization procedure to search for an input $x^{'}$ near a data point x such that the model output is very different at $x^{'}$. Those examples are called adversarial examples.

Adversarial training refers to training on images which are adversarially generated and it has been shown to reduce the error rate. One conjecture on main factor of adversarial behavior is the linearity of the model. Thus, a small change on the input causes a drastic change on the output. The idea of adversarial training is to avoid this jumping and induce the model to be locally constant in the neighborhood of the training data.

5.1 Consistency Regularization

Adversarial training can be applied to semi-supervised learning. One of semi-supervised settings is consistency regularization. First, Given an observed example, create a perturbed version of it. Then, Enforce the model predictions on the two examples(original,perturbed) to be similar. Thus, it can regularize the model to be less sensitive to small perturbations.

Virtual Adversarial Training (Miyato, 2018) Smoothing via random noise or random data augmentation often leaves the predictor particularly vulnerable to a small perturbations in a specific direction, that is, the adversarial direction. It tries to giving perturbation through adversarial training.

Unsupervised Data Augmentation (Qizhe, 2019) Using state-of-the-art data augmentation methods for each task found in supervised learning as the perturbation function.

6 Tangent Propagation

We would like to find ways to force the neural network to be invariant to variations such as translation and rotation, without discarding valuable information. One of approaches is data augmentation with modified patterns with desired invariances. However, in the limit for infinite set of variations, it is equivalent with tangent propagation.

Consider the case of continuous transformation parameterized by ξ applied to x_n sweeps a manifold $\mathcal{M}$ in the input space, and let the vector resulting from action on x_n by this transformation be denoted by $\mathrm{s}\left(\boldsymbol{x}_n, \xi\right)$. Then the tangent to the curve $\mathcal{M}$ is given by the directional derivative $\tau = \delta \mathrm{s} / \delta \xi$ and the tangent vector at point x_n is given by

$$\tau_n = \left.\frac{\partial s(x_n,\xi)}{\partial \xi}\right|_{\xi=0}$$

Under a transformation s of input vector, the network output vector will change. Then, we can express as below:

$$\left.\frac{\partial y_k}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} \left.\frac{\partial y_k}{\partial x_i}\frac{\partial x_i}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} J_{ki}\tau_i$$

This result is used to modify error function:

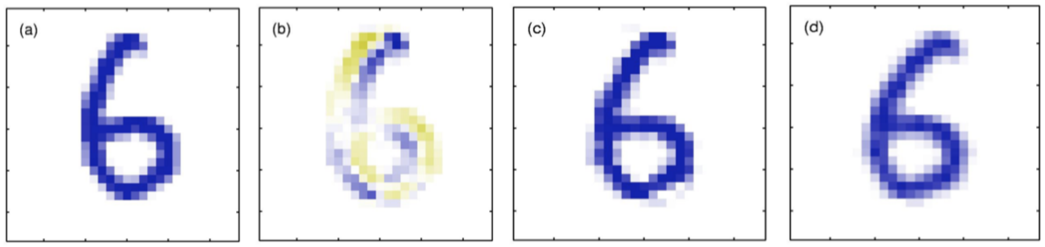

Figure 4: (a) original image (b) tangent vector τ corresponding to small clockwise rotation (c) adding small contribution from tangent vector to image $x + \xi\tau$ (d) true image rotated for comparison

$$\tilde{E} = E + \lambda\Omega$$

$$\Omega = \frac{1}{2}\sum_n \sum_k \left(\frac{\partial y_{nk}}{\partial \xi}\bigg|_{\xi=0} \right)^2 = \frac{1}{2}\sum_n \sum_k \left(\sum_{i=1}^{D} J_{nki}\tau_{ni} \right)^2$$

References

[1] Lasserre, Julia A., Christopher M. Bishop, and Thomas P. Minka. "Principled hybrids of generative and discriminative models." 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). Vol. 1. IEEE, 2006.

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

[3] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

[4] Miyato, Takeru, et al. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning." IEEE transactions on pattern analysis and machine intelligence 41.8 (2018): 1979-1993.

[5] Xie, Qizhe, et al. "Unsupervised data augmentation." arXiv preprint arXiv:1904.12848 (2019).