Lecture 3: Regularization For Deep Models

Ali Harakeh

University of Waterloo

WAVE Lab
ali.harakeh@uwaterloo.ca

May 23, 2017

Overview

1. Regularization: A Motivation
2. Regularization Strategies: Parameter Norm Penalties
3. Regularization Strategies: Dataset Augmentation
4. Regularization Strategies: Noise Robustness
5. Regularization Strategies: Early Stopping
6. Regularization Strategies: Parameter Tying and Parameter Sharing
7. Regularization Strategies: Multitask Learning
8. Regularization Strategies: Bagging and Other Ensemble Methods
9. Regularization Strategies: Dropout
10. Regularization Strategies: Adversarial Training

Section 1

Regularization: A Motivation

Introduction

- **Regularization** is any modification made to the learning algorithm with an intention to lower the generalization error but not the training error.
- Many standard regularization concepts from machine learning can be readily extended to deep models.

Introduction

- In context of deep learning, most regularization strategies are based on **regularizing estimator**s. This is done through reducing variance at the expense of increasing the bias of the estimator.
- An effective regularizer is one that decreases the variance significantly while not overly increasing the bias.

Introduction

- We discussed three regimes concerning the capacity of models where the model either:
  - Excludes the true data generating process which induces bias (underfitting).
  - Matches the true data generating process.
  - Includes the true data generating process, but also includes many other possible candidates, which results in variance dominating the estimation error (overfitting).
- The goal of regularization is to take the model from the **third** to the **second** regime.

Motivation

- In practice, we never have access to the true data generating distribution. This is a direct result of the extremely complicated domains (images, text and audio sequences) we work with when applying deep learning algorithms.
- In most applications of deep learning, the data generating process is almost certainly **outside the chosen model family**.

Motivation

- All of the above implies that controlling the complexity of the model is **not** a simple matter of finding the right model size and the right number of parameters.
- Instead, deep learning relies on finding the best fitting model as a large model that has been **regularized** properly.

Section 2

Regularization Strategies: Parameter Norm Penalties

Parameter Norm Penalties

- The most traditional form of regularization applicable to deep learning is the concept of **parameter norm penalties**.
- This approach limits the capacity of the model by adding the penalty $\Omega(\theta)$ to the objective function resulting in:

$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

- $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty to the value of the objective function.

Parameter Norm Penalties

- When the optimization procedure tries to minimize the objective function, it will also decrease some measure of **size** of the parameters θ.
- *Note*: The bias terms in the affine transformations of deep models usually require less data to be fit and are usually left unregularized.
- Without loss of generality, we will assume we will be regularizing only the weights $\mathbf{w}$.

L2 Norm Parameter Regularization

- The L2 parameter norm penalty, also known as **weight decay** drives $\mathbf{w}$ closer to the origin by adding the regularization term:

$$\Omega(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_2^2 = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

- For now, assume there is no bias parameters, only weights.

L2 Norm Parameter Regularization

- The update rule of gradient decent using L2 norm penalty is:

$$\mathbf{w} \leftarrow (1 - \epsilon\alpha)\mathbf{w} - \epsilon\nabla_w J(\mathbf{w})$$

- The weights **multiplicatively shrink** by a constant factor at each step.

L1 Norm Parameter Regularization

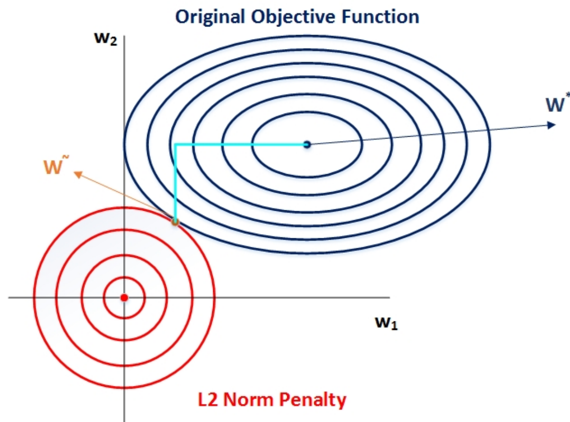
- L1 norm is another option that can be used to penalize the size of model parameters.
- L1 regularization on the model parameters **w** is:

$$\Omega(\theta) = ||\mathbf{w}|| = \sum_i |w_i|$$

- What is the difference between L2 and L1 norm penalty when applied to machine learning models ? But what happens over the entire course of training in both?

L2 Norm Parameter Regularization

- The L2 Norm penalty decays the components of the vector **w** that do not contribute much to reducing the objective function.



L1 Norm Parameter Regularization

- On the other hand, the L1 norm penalty provides solutions that are **sparse**.
- This **sparsity** property can be thought of as a **feature selection** mechanism.

Conclusion

- **L2 norm** penalty can be interpreted as a MAP Bayesian Inference with a **Gaussian** prior on the weights.
- On the other hand, **L1 norm** penalty can be interpreted as a MAP Bayesian Inference with a **Isotropic Laplace Distribution** prior on the weights.

Section 3

Regularization Strategies: Dataset Augmentation

Dataset Augmentation

- We have seen that for consistent estimators, the best way to get better generalization is to train on more data.
- The problem is that under most circumstances, data is limited. Furthermore, labelling is an extremely tedious task.
- **Dataset Augmentation** provides a cheap and easy way to increase the amount of your training data.
- Certain tasks such as steering angle regression require dataset augmentation to perform well.

Dataset Augmentation: Color jitter

- **Color jitter** is a very effective method to augment datasets. It is also extremely easy to apply.
- **Fancy PCA** was proposed by Krizhevsky et al. in the famous Alex net paper. It is a way to perform color jitter on images.
- Fancy PCA Algorithm:
  - Perform PCA on the three color channels of your entire dataset.
  - From the covariance matrix provided by PCA, extract the eigenvalues $\lambda_1, \lambda_2, \lambda_3$ and their corresponding eigenvectors p_1, p_2, p_3.
  - Add $p_i[a_1\lambda_1, a_2\lambda_2, a_3\lambda_3]^T$ to the ith color channel. $a_1 ... a_3$ are random variables sampled for each augmented image from a zero mean Gaussian distribution with a variance of 0.1.

Dataset Augmentation: Color jitter

Dataset Augmentation: Horizontal Flipping

- **Horizontal Flipping** is applied on data that exhibit horizontal asymmetry.
- Care must be taken to propagate the labels through this transformation.
- Horizontal flipping can be applied to natural images and point clouds. Essentially, one can double the amount of data through horizontal flipping.

Dataset Augmentation: Horizontal Flipping

Dataset Augmentation: Conclusion

- Many other task specific dataset augmentation algorithms exist. It is highly advised to always use dataset augmentation.
- However, be careful not to alter the correct output!
- Example: b and d, horizontal flipping.
- Furthermore, when comparing two machine learning algorithms train both with either augmented or non-augmented dataset. Otherwise, no subjective decision can be made on which algorithm performed better.

Section 4

Regularization Strategies: Noise Robustness

Noise Robustness

- **Noise Injection** can be thought of as a form of regularization. The addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop, 1995).
- Noise can be injected at different levels of deep models.

Noise Robustness : Noise Injection on Weights

- Noise added to weights can be interpreted as a more traditional form of regularization.
- This form of regularization encourages the parameters to go to regions of parameter space where small perturbations of the weights have a relatively small influence on the output.
- In other words, it pushes the model into regions where the model is relatively insensitive to small variations in the weights, finding points that are not merely minima, but minima surrounded by flat regions (Hochreiter and Schmidhuber, 1995).

Noise Robustness : Noise Injection on Outputs

- Most datasets have some amount (A LOT!) of mistakes in the y labels. Minimizing our cost function on wrong labels can be extremely harmful.
- One way to remedy this is to explicitly model the noise on labels. This is done through setting a probability ϵ for which we think the labels are correct.
- This probability is easily incorporated into the cross entropy cost function **analytically**.
- An example is **label smoothing**.

Noise Robustness : Label Smoothing

- Usually, we have output vectors provided to us as $y_{label} = [1, 0, 0, 0...0]$.
- Softmax output is usually of the form $y_{out} = [0.87, 0.001, 0.04, 0.1, ...0.03]$.
- Maximum likelihood learning with a softmax classifier and hard targets may actually never converge, the softmax can never predict a probability of exactly 0 or exactly 1, so it will continue to learn larger and larger weights, making more extreme predictions. forever

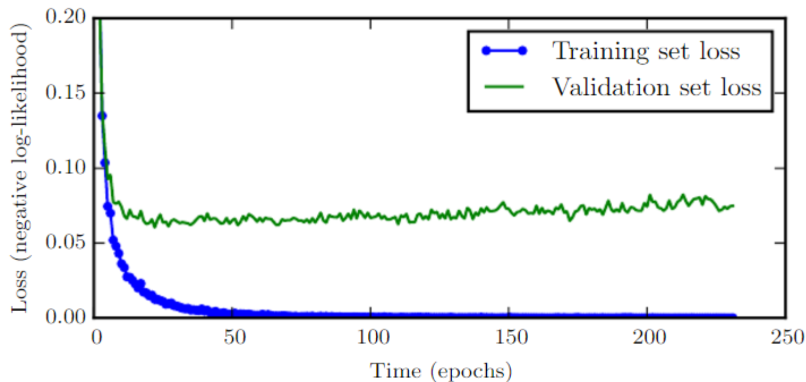
Noise Robustness : Label Smoothing

- **Label smoothing** replaces the label vector with $y_{label} = [1 - \epsilon, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1} \cdots \frac{\epsilon}{K-1}]$.
- The above representation has the advantage of preventing the pursuit of hard probabilities without discouraging correct classification.

Section 5

Regularization Strategies: Early Stopping

Motivation



Early Stopping: Motivation

- When training models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, while the error on the validation set begins to rise again.
- The occurrence of this behaviour in the scope of our applications is almost certain.
- This means we can obtain a model with better validation set error (and thus,hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.
- This is termed **Early Stopping**.

Early Stopping: Meta-Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Early Stopping: Practical Issues

- Early Stopping is probably one of the most used regularization strategies in deep learning.
- Early stopping can be thought of as a hyperparameter selection method, where **training time** is the hyperparameter to be chosen.
- Choosing the training time automatically can be done through a single run through the training phase, the only addition being the evaluation of the validation set error at every n iterations. This is usually done on a second GPU.
- Overhead for writing parameters to disk is negligible.

Early Stopping: Practical Issues

- Early Stopping is probably one of the most used regularization strategies in deep learning.
- Early stopping can be thought of as a hyperparameter selection method, where **training time** is the hyperparameter to be chosen.
- However, a portion of data should be reserved for validation.

Early Stopping: Exploiting The Validation Data

- To exploit all of our precious training data we can:
  - Employ early stopping as described above.
  - Retrain using all of the data up to the point that was determined during early stopping.
- Some subtleties arise regarding the definition of **point**.
- Do we train for the same number of parameter updates or for the same number of epochs(passes through training data) ?

Early Stopping: Exploiting The Validation Data

- A second strategy to exploit the full training dataset would be to:
  - Employ early stopping as described above.
  - Continue training with the parameters determined by early stopping, using the validation set data.
- This strategy avoids the high cost of retraining the model from scratch, but is not well-behaved.
- Since we no longer have a validation set, we cannot know if generalization error is improving or not. Our best bet is to stop training when the training error is not decreasing much any more.

Section 6

Regularization Strategies: Parameter Tying and Parameter Sharing

Parameter Sharing

- So far, we have discussed regularization as adding constraints or penalties to the parameters with respect to a **fixed region**.
- However, we might want to express priors on parameters in other ways. Specifically, we might not know which region the parameters would lie in, but rather that there is some dependencies between them.
- Most common type of dependency: Some parameters should be close to each other.

Parameter Tying

- **Parameter Tying** refers to explicitly forcing the parameters of two models to be close to each other, through the norm penalty:

$$||\mathbf{w}^{(A)} - \mathbf{w}^{(B)}||$$

- Here, $\mathbf{w}^{(A)}$ refers to the weights of the first model while $\mathbf{w}^{(B)}$ refers to those of the second one.

Parameter Sharing

- **Parameter Sharing** imposes much stronger assumptions on parameters through forcing the parameter sets to be **equal**.
- Examples would be Siamese networks, convolution operators, and multitask learning.

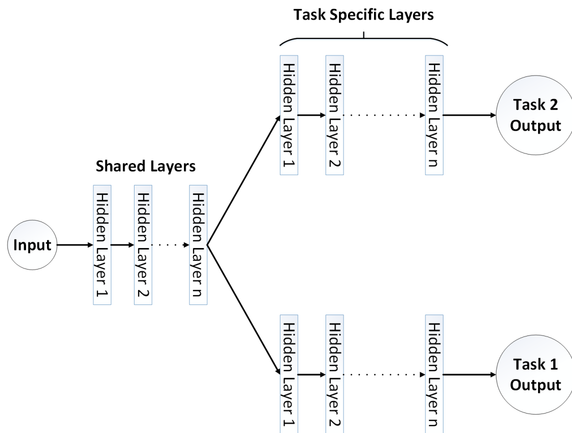
Section 7

Regularization Strategies: Multitask Learning

Multitask Learning

- **Multitask Learning** is a way to improve generalization by pooling the examples arising out of several tasks.
- Usually, the most common form of multitask learning is performed through an architecture which is devided to two parts:
  - Task-specific parameters (which only benefit from the examples of their task to achieve good generalization).
  - Generic parameters, shared across all the tasks (which benefit from the pooled data of all the tasks).
- Multitask learning is a form of parameter sharing.

Multitask Learning



Multitask Learning

- Improved generalization and generalization error bounds (Baxter, 1995) can be achieved because of the shared parameters, for which statistical strength can be greatly improved in proportion with the increased number of examples for the shared parameters, compared to the scenario of single-task models.
- Intuitively, the additional task imposes constraints on the parameters in the shared layers, preventing overfitting.
- Improvement in generalization only occurs when there is something shared across the tasks at hand.

Section 8

Regularization Strategies: Bagging and Other Ensemble Methods

Bagging

- **Bagging** (short for **bootstrap aggregating**) is a technique for reducing generalization error through combining several models (Breiman, 1994).
- Bagging is defined as follows:
  - Train k different models on k different subsets of training data, constructed to have the same number of examples as the original dataset through random sampling from that dataset **with replacement**.
  - Have all of the models **vote** on the output for test examples.
- Techniques employing bagging are called ensemble models.

Bagging

- The reason that Bagging works is that different models will usually not all make the same errors on the test set.
- This is a direct results of training on k different subsets of the training data, where each subset is missing some of the examples from the original dataset.
- Other factors such as differences in random initialization, random selection of mini-batches,differences in hyperparameters, or different outcomes of non-deterministic implementations of neural networks are often enough to cause different members of the ensemble to make **partially independent errors**.

Bagging

- The reason that Bagging works is that different models will usually not all make the same errors on the test set.
- This is a direct results of training on k different subsets of the training data, where each subset is missing some of the examples from the original dataset.
- Other factors such as differences in random initialization, random selection of mini-batches,differences in hyperparameters, or different outcomes of non-deterministic implementations of neural networks are often enough to cause different members of the ensemble to make **partially independent errors**.

Ensemble Models

- On average, the ensemble will perform at least as well as any of its members, and if the members make independent errors, the ensemble will perform significantly better than its members.
- Proof ?
- The only disadvantage of ensemble models is that they do not provide us with a **scalable** way to improve performance. Usually, ensemble models of more than 2-3 networks become too tedious to train and handle.

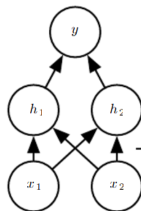
Section 9

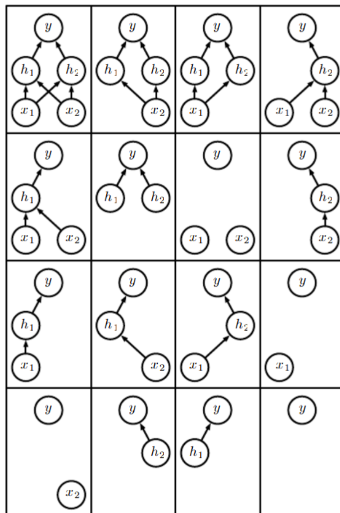
Regularization Strategies: Dropout

Dropout

- Dropout provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.
- Specifically, dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.

Dropout1





Training with Dropout

- To train with dropout,we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- Typically, the probability of including a hidden unit is 0.5, while the probability of including an input unit is 0.8.

Dropout

- Dropout allows us to represent an exponential number of models with a tractable amount of memory.
- Furthermore, Dropout removes the need to accumulate model votes at the inference stage.
- Dropout can intuitively be explained as forcing the model to learn with missing input and hidden units.

Dropout Training

- Dropout training has some intricacies we need to be wary of.
- At training time, we are **required** to divide the output of each unit by the probability of that unit's dropout mask.
- The goal is to make sure that the expected total input to a unit at test time is roughly the same as the expected total input to that unit at train time, even though half the units at train time are missing on average.
- No theoretically satisfying basis for the accuracy of this approximate training rule in deep non linear networks, but empirically it performs very well.

Conclusion

- Dropout is that it is very computationally cheap, using dropout during training requires only $O(n)$ computation per example per update,to generate n random binary numbers and multiply them by the state.
- Dropout does not significantly limit the type of model or training procedure that can be used. It works well with nearly any model that uses a distributed representation and can be trained with stochastic gradient descent.

Conclusion

- Though the cost per-step of applying dropout to a specific model is negligible,the cost of using dropout in a complete system can be significant.
- Applying Dropout indirectly requires us to design larger systems to preserve capacity. Larger systems usually are slower at inference time.
- Practitioners have to keep in mind that for very large datasets, regularization confers little reduction in generalization error. In these cases, the computational cost of using dropout and larger models may outweigh the benefit of regularization.

Section 10

Regularization Strategies: Adversarial Training

Adversarial Training

- In many cases, neural networks have begun to reach human performance when evaluated on an i.i.d. test set.
- However, szegedy et al.(2014) found that even networks that have achieved human accuracy, have a 100% error rate on examples that have been intentionally constructed to "fool" the network.
- In many cases, the modified example is so similar to the original one, human observers cannot tell the difference.
- These examples are called **adversarial examples**.

Adversarial Examples

$+ \ .007 \times$

$=$

$\boldsymbol{x}$	$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon\, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda" w/ 57.7% confidence	"nematode" w/ 8.2% confidence	"gibbon" w/ 99.3 % confidence

Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d.test set via **adversarial training** - training on adversarially perturbed examples from the training set.
- Adversarial training discourages highly sensitive linear behaviour through explicitly introducing a local constancy prior into supervised neural nets.

Next Lecture

- Next Lecture, Optimization.