CIE 555
Neural Networks and Deep Learning

Regularization for Deep Learning

1

Regularization for Deep Learning

- A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs.
- Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as regularization.
- Topics in this lecture:
  - Parameter Norm Penalties (Section 7.1)
    - L^2 *Regularization* (7.1.1)
    - L^1 *Regularization* (7.1.2)
  - Dataset Augmentation (7.4)
  - Noise Robustness (7.5)
  - Early Stopping (7.8)
  - Dropout (7.12)

2

Regularization for Deep Learning

- In practice, controlling the complexity of the model (i.e. finding the model of the right size, with the right number of parameters) is not a simple matter.
- Instead, we might find—and indeed in practical deep learning scenarios, we almost always do find—that the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately.
- In the context of deep learning, most regularization strategies are based on regularizing estimators.
- Regularization of an estimator works by trading increased bias for reduced variance.
- An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias.

Goodfellow, Bengio, Courville 2016

3

Norm Penalties

4

Parameter Norm Penalties

- Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J.

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

- A Norm is a measure the size of a vector (functions mapping vectors to non-negative values).
- The L^p norm is given by:

$$L^p = \|x\|_p = \left(\sum_i |x_i|^p\right)^{1/p}$$

Goodfellow, Bengio, Courville 2016

5

Regularized objective function by $\tilde{J}$

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

- $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function *J*.
- Setting $\alpha \; to \; 0$ results in no regularization. Larger values of α correspond to more regularization.
- When the training algorithm minimizes the regularized objective function $\tilde{J}$ it will decrease both the original objective J on the training data and some measure of the size of the parameters θ (or some subset of the parameters).
- Different choices for the parameter norm Ω can result in different solutions being preferred as we will discuss.

Goodfellow, Bengio, Courville 2016

6

L^2 Parameter Regularization (weight decay)

- This regularization strategy drives the weights closer to the origin by adding a regularization term $\Omega(\theta)$ to the objective function, where

$$\Omega(\theta) = \frac{1}{2}\|w\|_2^2.$$

w: weights of the linear transformation

θ consisting of ***w*** and *bias* ***b***.

- L^2 Parameter Regularization is also known as ridge regression or Tikhonov regularization.

Goodfellow, Bengio, Courville 2016

7

Impact of L^2 Regularization

- To simplify the presentation, we assume no bias parameter, so θ is just w.
- Such a model has the following total objective function:

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^\top w + J(w; X, y)$$

with the corresponding parameter gradient

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

To take a single gradient step to update the weights, we perform this update:

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

OR

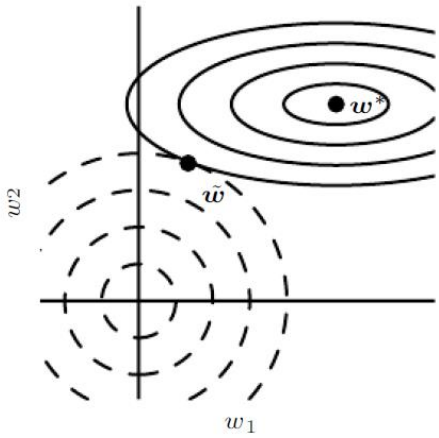
$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y))$$

- the addition of the weight decay term has modified the learning rule to multiplicatively shrink the weight vector by a constant factor on each step just before performing the usual gradient update.

Goodfellow, Bengio, Courville 2016

8

L^2 Parameter Regularization



- The solid ellipses represent contours of equal value of the unregularized objective.
- The dotted circles represent contours of equal value of the L^2 regularizer.
- At the point $\tilde{w}$, these competing objectives reach an equilibrium.

Goodfellow, Bengio, Courville 2016

9

L^1 Parameter Regularization

- Formally, L^1 regularization on the model parameter *w* is defined as:

$$\Omega(\theta) = \|w\|_1 = \sum_i |w_i|$$

(the sum of absolute values of the individual parameters)

Goodfellow, Bengio, Courville 2016

10

Impact of L^1 Regularization

- As with L^2 weight decay, L^1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α.

$$\tilde{J}(\theta; X, y) = \alpha \|w\|_1 + J(\theta; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha sign(w) + \nabla_w J(w; X, y)$$

- The regularization contribution to the gradient no longer scales linearly with each w_i; instead, it is a constant factor with a sign equal to sign(w_i).

Goodfellow, Bengio, Courville 2016

11

L^1 and L^2 Regularization

- In comparison to L^2 regularization, L^1 regularization results in a solution that is more **sparse** (more parameters will have an optimal value of zero).
- The sparsity property induced by L^1 regularization has been used extensively as a **feature selection** mechanism (choosing which subset of the available features should be used).
- The L^1 penalty causes a subset of the weights to become zero, suggesting that the corresponding features may safely be discarded.

Goodfellow, Bengio, Courville 2016

12

Dataset Augmentation

13

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- Of course, in practice, the amount of data we have is limited.
- One way to get around this problem is to *create fake data* and add it to the training set.
- This approach is easiest for classification. A classifier needs to take a complicated, high dimensional input *x* and summarize it with a single category identity *y*.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the *x* inputs in our training set.
- For other tasks, such as a density estimation task, data augmentation is difficult.

Goodfellow, Bengio, Courville 2016

14

Dataset Augmentation – An example application

- Dataset augmentation has been a particularly effective technique for a specific classification problem: *object recognition*.
- Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated.
- Operations like translating the training images a few pixels in each direction can often greatly improve generalization, even if the model has already been designed to be partially translation invariant by using the convolution and pooling techniques described later.
- Many other operations such as rotating the image or scaling the image have also proven quite effective.
- One must be careful not to apply transformations that would change the correct class. For example, optical character recognition tasks require recognizing the difference between 'b' and 'd' and the difference between '6' and '9', so horizontal flips and 180◦ rotations are not appropriate ways of augmenting datasets for these tasks.

Goodfellow, Bengio, Courville 2016

15

Noise Robustness

16

Injecting Noise to inputs

- Neural networks prove not to be very robust to noise (Tang and Eliasmith, 2010)
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Noise injection also works when the noise is applied to the hidden units, which can be seen as doing dataset augmentation at multiple levels of abstraction.
- Poole *et al.* (2014) showed that this approach can be highly effective *provided that the magnitude of the noise is carefully tuned.*
- Dropout, can be seen as a process of constructing new inputs by *multiplying* by noise as we will dicuss later.
- Injecting noise in the input to a neural network can also be seen as a form of data augmentation.

Goodfellow, Bengio, Courville 2016

17

Adding noise to the weights (a prior distribution of the weights)

- Another way that noise has been used in the service of regularizing models is by *adding it to the weights*.
- This technique has been used primarily in the context of ***recurrent neural networks (Jim et al., 1996; Graves, 2011).***
- This can be interpreted as a stochastic implementation of Bayesian inference over the weights.
- The Bayesian treatment of learning would consider the model weights to be uncertain and representable via a probability distribution that reflects this uncertainty.
- Adding noise to the weights is a practical, stochastic way to reflect this uncertainty.

Goodfellow, Bengio, Courville 2016

18

Injecting Noise at the Output Targets

- Most datasets have some amount of mistakes in the y labels. It can be harmful to maximize $\log p(y|x)$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels. For example, we can assume that for some small constant ϵ, the training set label *y* is correct with probability $1 - \epsilon$, and otherwise any of the other possible labels might be correct.
- This assumption is easy to incorporate into the cost function analytically, rather than by explicitly drawing noise samples. For example, **label smoothing** regularizes a model based on a softmax with *k* output values by replacing the hard 0 and 1 classification targets with targets of $\frac{\epsilon}{k-1}$ and $1 - \epsilon$, respectively.

Goodfellow, Bengio, Courville 2016

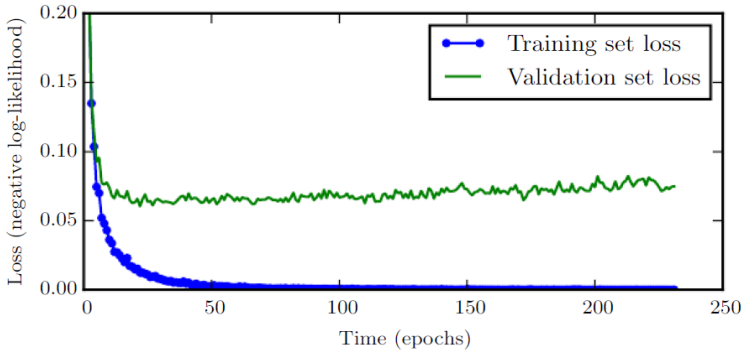
19

Early Stopping

20

Early Stopping

- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again.



Goodfellow, Bengio, Courville 2016

21

Early Stopping – Main idea

- 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.
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest parameters.
- The algorithm terminates when no parameters have improved over the best recorded validation error for some pre-specified number of iterations.

Goodfellow, Bengio, Courville 2016

22

Early Stopping

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^*

Goodfellow, Bengio, Courville 2016

23

Early Stopping – advantages

- The most commonly used form of regularization in deep learning for the following reasons:
  1. Simple and effective.
  2. Early stopping reduces the computational cost of the training procedure. Besides the obvious reduction in cost due to limiting the number of training iterations, it also has the benefit of providing regularization without requiring the addition of penalty terms to the cost function or the computation of the gradients of such additional terms.
  3. It requires almost no change in the underlying training procedure, the objective function, or the set of allowable parameter values. This means that it is easy to use early stopping without damaging the learning dynamics. This is in contrast to weight decay, where one must be careful not to use too much weight decay and trap the network in a bad local minimum corresponding to a solution with pathologically small weights.

Goodfellow, Bengio, Courville 2016

24

Early Stopping – advantages

4. While most hyperparameters are chosen using an expensive guess and check process, where we set a hyperparameter at the start of training, then run training for several steps to see its effect, we can try many values of the "training time" hyperparameter in a single run.
5. Early stopping may be used either alone or in conjunction with other regularization strategies. Even when using regularization strategies that modify the objective function to encourage better generalization, it is rare for the best generalization to occur at a local minimum of the training objective.

25

Early stopping – additional costs

- The only significant cost to choosing this hyperparameter automatically via early stopping is running the validation set evaluation periodically during training.
- Ideally, this is done in parallel to the training process on a separate machine, separate CPU, or separate GPU from the main training process.
- An additional cost to early stopping is the need to maintain a copy of the best parameters. This cost is generally negligible, because it is acceptable to store these parameters in a slower and larger form of memory (for example, training in GPU memory, but storing the optimal parameters in host memory or on a disk drive).

Goodfellow, Bengio, Courville 2016

26

Early stopping – how to exploit the validation set

- Early stopping requires a validation set, which means some training data is not fed to the model. To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed. In the second, extra training step, all of the training data is included.
- There are two basic strategies one can use for this second training procedure:

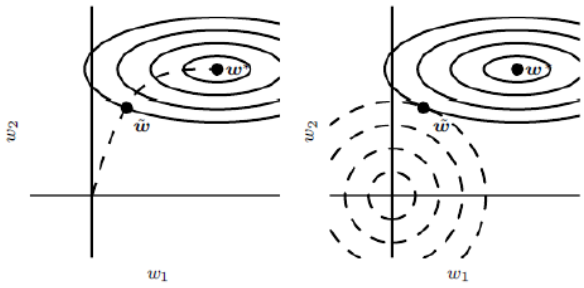
1. One strategy is to initialize the model again and retrain on all of the data. In this second training pass, we train for the same number of steps as the early stopping procedure determined was optimal in the first pass. Shall we use the same number of parameter updates or the same number of passes through the dataset.
2. Another strategy for using all of the data is to keep the parameters obtained from the first round of training and then *continue* training but now using all of the data. When shall we stop? (after how many steps) . One way is to monitor the average loss function on the validation set, and continue training until it falls below the value of the training set objective at which the early stopping procedure halted (no guarantee that the objective on the validation set will ever reach the target value)

Goodfellow, Bengio, Courville 2016

27

How does early stopping act as a regularizer?

- Bishop (1995a) and Sjöberg and Ljung (1995) argued that early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter value θ_0



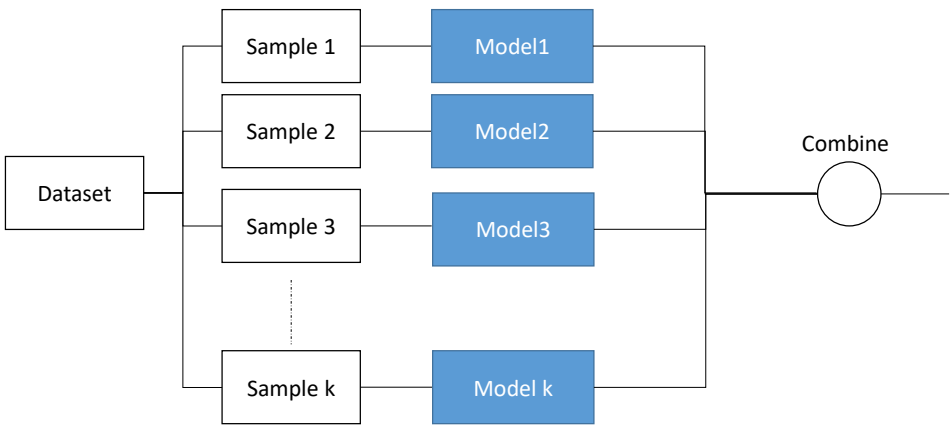
Goodfellow, Bengio, Courville 2016

28

Dropout

29

Review: Bagging (model averaging)



30

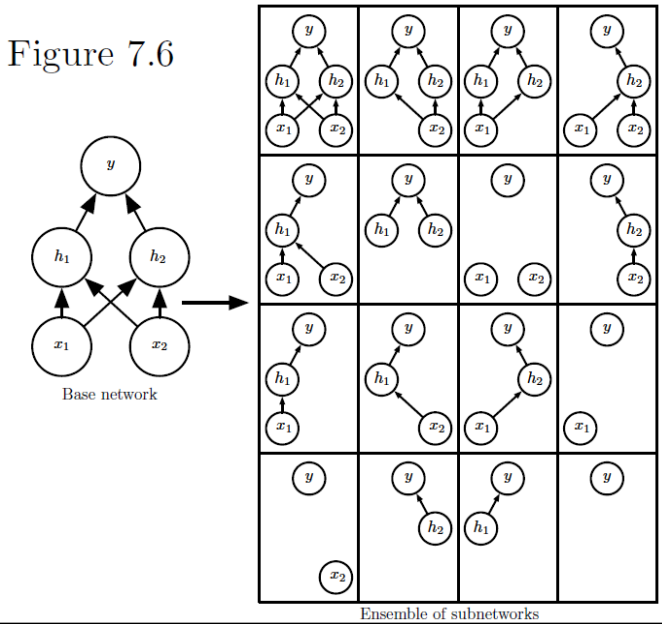
Dropout

- An effective simple and regularization technique by Srivastava et al. (Dropout: A Simple Way to Prevent Neural Networks from Overfitting)
- Implemented by activating each neuron with some probability p (a hyperparameter), or setting it to zero otherwise.
- Readings:
  - Dropout: A Simple Way to Prevent Neural Networks from Overfitting. By Srivastava et al
  - Dropout Training as Adaptive Regularization by Stefan Wager et al.

Dropout

- Bagging seems impractical when each model is a large neural network? why?
- It is common to use ensembles of five to ten neural networks—Szegedy *et al.* (2014a) used six to win the ILSVRC but more than this rapidly becomes unwieldy
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.
- Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network
- We can effectively remove a unit from a network by multiplying its output value by zero.

Dropout



Goodfellow, Bengio, Courville 2016

33

Dropout -Procedure

- Use a minibatch-based learning algorithm that makes small steps.
- 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. The probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- Run forward propagation, back-propagation, and the learning update as usual.

Goodfellow, Bengio, Courville 2016

34

Dropout and bagging

- Dropout training is not quite the same as bagging training.
- In the case of bagging, the models are all independent. 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 makes it possible to represent an exponential number of models with a tractable amount of memory.
- In the case of bagging, each model is trained to convergence on its respective training set.
- In the case of dropout, typically most models are not explicitly trained at all. Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters.

Goodfellow, Bengio, Courville 2016

35