ECBM E6040 Neural Networks and Deep Learning

Regularization of Deep or Distributed Models (cont'd)

Nikul H. Ukani
Instructor: Aurel A. Lazar

Department of Electrical Engineering
Columbia University

March 8, 2016

Outline

Outline

Outline

Outline

- Regularization is any component of the model, training process or prediction procedure which is included to account for limitations of the training data, including its finiteness.
- Most classical 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 $\mathbf{\Omega}(\theta)$ to the objective function J. We denote the regularized objective function by $\tilde{J}$:

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

- For L^2 regression, $\mathbf{\Omega}(\theta) = \frac{1}{2}|\mathbf{w}\|_2^2$
- For L^1 regression, $\mathbf{\Omega}(\theta) = |\mathbf{w}\|_1$

Remark

L^1 regularization prefers sparser solutions i.e it prefers some parameters to have an optimal of zero

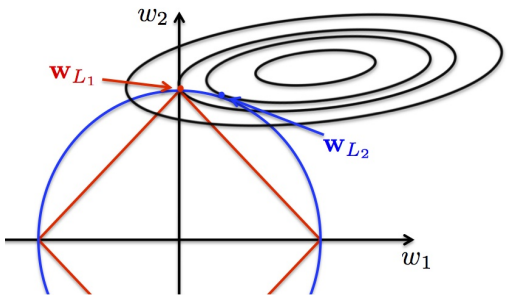


Figure credit: http://g2pi.tsc.uc3m.es/en/Primal-sparse-SVM

- We can also constrain the norm to be smaller than some value, rather than imposing a penalty on it. This is a case of constrained optimization.
- Such constrained optimization can be solved by projecting the weight vector on the constraint space after every update. This can allow us to have a higher learning rate and train networks faster.
- In the case of least squares linear regression, if the system is underconstrained, the matrix $\mathbf{X}^\top\mathbf{X}$ w ill not be invertible. However, the matrix $\mathbf{X}^\top\mathbf{X} + \alpha\mathbf{I}$ will always be invertible

Outline

Regularization via Injecting Noise

- Neural networks can be sensitive to noisy inputs
- It is possible to use noise as part of a regularization strategy
- Under certain assumptions on the noise model and certain approximations, we can show such regularization strategies as implementing a penalty based regularization
- We can either add noise to the inputs during training or inject noise into the weights during training

Injecting noise at the inputs

- With each input presentation to the model, we also include a random perturbation, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \nu\mathbf{I})$
- We will analyze injecting noise at the inputs in the context of regression, where we are interested in learning a model $\hat{y}(\mathbf{x})$
- The cost function then becomes

$$\begin{aligned}\tilde{J}_{\mathrm{x}} &= \mathbb{E}_{p(\mathrm{x},y,\boldsymbol{\epsilon})}[(\hat{y}(\mathbf{x}+\boldsymbol{\epsilon}) - y)^2] \\ &= \mathbb{E}_{p(\mathrm{x},\boldsymbol{\epsilon})}[\hat{y}^2(\mathbf{x}+\boldsymbol{\epsilon})] - 2\mathbb{E}_{p(\mathrm{x},\boldsymbol{\epsilon})}[y\hat{y}(\mathbf{x}+\boldsymbol{\epsilon})] + \mathbb{E}_{p(y)}[y^2]\end{aligned}$$

- Assuming that the noise is small, we can model its effect using the Taylor series expansion

$$\hat{y}(\mathbf{x}+\boldsymbol{\epsilon}) = \hat{y}(\mathbf{x}) + \boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}\hat{y}(x) + \frac{1}{2}\boldsymbol{\epsilon}^T\nabla^2_{\mathrm{x}}\hat{y}(x)\boldsymbol{\epsilon} + \mathcal{O}(\boldsymbol{\epsilon}^3)$$

Injecting noise at the inputs

- Plugging the taylor series we get

$$\tilde{J}_{\mathrm{x}} \approx \mathbb{E}_{p(\mathrm{x},\boldsymbol{\epsilon})}\left[\left(\hat{y}(\mathbf{x})+\boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}\hat{y}(x)+\frac{1}{2}\boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}^2\hat{y}(x)\boldsymbol{\epsilon}\right)^2\right]$$

$$-2\mathbb{E}_{p(\mathrm{x},\boldsymbol{\epsilon})}\left[y\left(\hat{y}(\mathbf{x}+\boldsymbol{\epsilon})=\hat{y}(\mathbf{x})+\boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}\hat{y}(x)+\frac{1}{2}\boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}^2\hat{y}(x)\boldsymbol{\epsilon}\right)\right]+\mathbb{E}_{p(y)}[y^2]$$

$$=\mathbb{E}_{p(\curvearrowleft,y)}[(\hat{y}(x)-y)^2]+\mathbb{E}_{p(\mathrm{x},\boldsymbol{\epsilon})}\left[\hat{y}(\mathbf{x})\boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}^2\hat{y}(x)\boldsymbol{\epsilon}+\left(epsilon^T\nabla_{\mathrm{x}}\hat{y}(x)\right)^2+\mathcal{O}(\boldsymbol{\epsilon}^3\right.$$

$$-\mathbb{E}_{p(\mathrm{x},y,\boldsymbol{\epsilon})}\left[y\boldsymbol{\epsilon}^T\nabla_{\mathrm{x}}^2\hat{y}(x)\boldsymbol{\epsilon}\right]$$

$$=J+\nu\mathbb{E}_{p(\mathrm{x},y)}\left[(\hat{y}(\mathbf{x})-y)\nabla_{\mathrm{x}}^2\hat{y}(x)\right]+\nu\mathbb{E}_{p(\mathrm{x},y)}\left[\|\nabla_{\mathrm{x}}\hat{y}(x)\|^2\right]$$

Injecting noise at the inputs

- If we minimize this objective function, by taking the functional gradient of $\hat{y}(x)$ and setting it to zero, we get

$$\hat{y}(\mathbf{x}) = \mathbb{E}_{p(y|\mathrm{x})}[y] + \mathcal{O}(\nu)$$

- This implies that $\nu \mathbb{E}_{p(\mathrm{x},y)}\left[(\hat{y}(\mathbf{x}) - y)\nabla_{\mathrm{x}}^2 \hat{y}(x)\right]$ reduces to $\mathcal{O}(\nu^2)$
- Therefore,

$$\tilde{J}_{\mathrm{x}} \approx J + \nu \mathbb{E}_{p(\mathrm{x},y)}\left[\|\nabla_{\mathrm{x}} \hat{y}(x)\|^2\right] + \mathcal{O}(\nu^2)$$

- This regularization term has the effect of penalizing large gradients of the function $\hat{y}(\mathbf{x})$. That is, it has the effect of reducing the sensitivity of the output of the network with respect to small variations in its input $\mathbf{x}$.
- We can interpret this as attempting to build in some local robustness into the model and thereby promote generalization. We note also that for linear networks, this regularization term reduces to simple weight decay

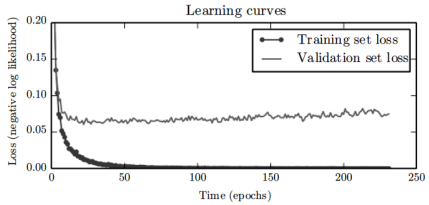
Injecting noise at the Weights

- Through similar analysis, we can approximate the cost function in the case where we add noise $\boldsymbol{\epsilon}_{\mathrm{w}} \sim \mathcal{N}(\mathbf{0}, \eta \mathbf{I})$ at the weights by the following

$$\tilde{J}_{\mathrm{w}} \approx J + \eta \mathbb{E}_{p(\mathrm{x},y)} \left[\|\nabla_{\mathrm{w}} \hat{y}(x)\|^2 \right] + \mathcal{O}(\eta^2)$$

- 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.

Early Stopping



- When training large models with large enough capacity, we often observe that training error decreases steadily over time, but validation set error begins to rise again.
- Instead of running our optimization algorithm until we reach a (local) minimum of validation error, we run it until the error on the validation set has not improved for some amount of time.
- Every time the error on the validation set improves, we store a copy of the model parameters.

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^*

Early Stopping

- Consider the quadratic approximation of the cost function in the neighbourhood of the empirically optimal value of the weights $\mathbf{w}^*$

$$\hat{J}(\mathbf{w}) = J(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*\mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

- Thus,

$$\nabla_w \hat{J}(\mathbf{w}) = \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

- For simplicity, we assume $\mathbf{w}^{(0)} = \mathbf{0}$
- Then, the gradient descent updates can be expressed as

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau)} - \eta\mathbf{H}(\mathbf{w}^{(\tau-1)} - \mathbf{w}^*)$$
$$\mathbf{w}^{(\tau)} - \mathbf{w}^* = (\mathbf{I} - \eta\mathbf{H})(\mathbf{w}^{(\tau-1)} - \mathbf{w}^*)$$

Early Stopping

- With $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\top}$, we rewrite the above as

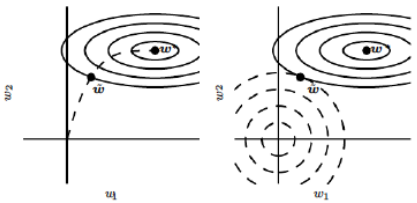
$$\mathbf{Q}^{\top}(\mathbf{w}^{(\tau)} - \mathbf{w}^{*}) = (\mathbf{I} - \eta\mathbf{\Lambda})\mathbf{Q}^{\top}(\mathbf{w}^{(\tau-1)} - \mathbf{w}^{*})$$

- Assuming $\mathbf{w}^{(}0) = \mathbf{0}$

$$\mathbf{Q}^{\top}\mathbf{w}^{(\tau)} = \left(\mathbf{I} - (\mathbf{I} - \eta\mathbf{\Lambda})^{\tau}\right)\mathbf{Q}^{\top}\mathbf{w}^{*}$$

- The closed form solution of L^2 regularization can be written as

$$\mathbf{Q}^{\top}\tilde{w} = \left(\mathbf{I} - (\mathbf{I} + \alpha\mathbf{\Lambda})^{-1}\alpha\right)\mathbf{Q}^{\top}\mathbf{w}^{*}$$



Dataset Augmentation

- More complex models require more training data and training data is always finite
- It may not always be possible to have large datasets
- For certain tasks, especially image classification, it may be possible to generate fake data
- For example, operations like translating the training images a few pixels in each direction can often greatly improve generalization
- 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^o rotations are not appropriate ways of augmenting datasets for these tasks.

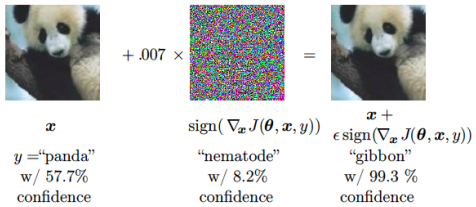
Adversarial Training

- Even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimization procedure to search for an input $\mathbf{x}'$ near a data point $\mathbf{x}$ such that the model output is very different at $\mathbf{x}'$. In many case, $\mathbf{x}'$ can be so similar to x that a human observer cannot tell the difference between the original example and the adversarial example.
- Neural networks are built out of primarily linear building blocks. In some experiments the overall function they implement proves to be highly linear as a result.
- To understand the implications of it, let's consider a linear mapping,

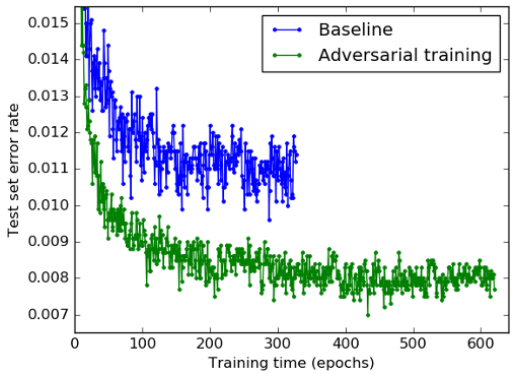
$$\hat{y} = \mathbf{w}^\top \mathbf{x}$$

- If we add a small $\boldsymbol{\epsilon}$ to the input in the linear case we get $\hat{y} = \mathbf{w}^\top \mathbf{x} + \mathbf{w}^\top \boldsymbol{\epsilon}$. We can maximize this increase by assigning $\boldsymbol{\epsilon} = \epsilon \ \text{sign}(\mathbf{w})$. When $\mathbf{w}$ is high dimensional, this can result in a very large change in the output.

Adversarial Training



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



Parameter Sharing

- Another way to regularize or reduce the complexity of a deep model is through parameter sharing.
- Various components of the model are made to share a unique set of parameters
- The most popular and extensive use of parameter sharing occurs in convolutional neural networks (CNNs) (To be covered later in class) applied to computer vision

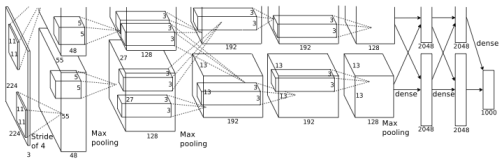
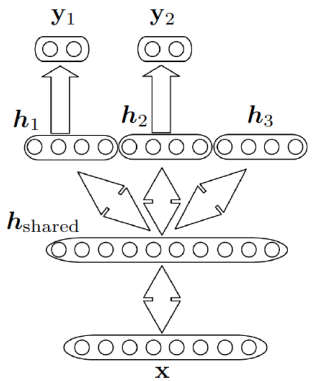


Figure Credit: Krizhevsky et al., NIPS 2012

Multi-Task Learning

- Parameter sharing restricts different components of the model to share parameters for a particular task. On the other hand, in multi-task learning, part of the model is shared across various tasks.
- Biological vision systems share initial processing across all tasks as well
- Assuming that sharing is justified, when part of a model is shared across tasks, that part of the model is more constrained towards "good "(in the context of generalization) values.

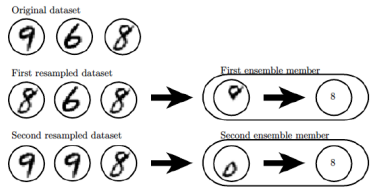


Bootstrap Aggregating (Bagging)

- Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models
- Consider for example a set of k regression models, eaxh of which makes an error, ϵ_i on each example. Let $\mathbb{E}(\epsilon_i) = 0, \mathbb{E}(\epsilon_i^2) = \nu, \mathbb{E}(\epsilon_i \epsilon_i) = c$. Then

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k}\nu + \frac{k-1}{k}c$$

- Bagging involves constructing the k models by training them on k different datasets obtained by the process of bootstrapping. Bootstrapping involves uniformly sampling from the training dataset without replacement.



Dropout

- Dropout can be thought of as a method of making bagging practical for large neural networks.
- Dropout trains the ensemble consisting of all sub-networks that can be formed by removing units from an underlying base network.
- Dropout can be more effective than other standard computationally inexpensive regularizers, such as weight decay

Outline

Let $\mathbf{A} \in \mathbb{R}^{2\times 2}$ denote the matrix that reflects any vector $\mathbf{y} \in \mathbb{R}^2$ about the line $\mathbf{a}^T\mathbf{x} = 0$, where $\mathbf{a} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$.

1. What are are the eigenvalues of $\mathbf{A}$
2. Find $\mathbf{A}^2$
3. It can be seen intuitively that the operation performed by $\mathbf{A}$ preserves norms (lengths). Thus, it represents an orthogonal transformation and $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. Combined with the knowledge gained by solving question 2, we see that $\mathbf{A}$ is a symmetric matrix. Find the eigenvectors of A and using the form of eigenvalue decomposition of symmetric matrices, find $\mathbf{A}$

$\mathbf{A} \in \mathbb{R}^{5\times 5}$ is a symmetric matrix with eigenvalues $-4, -2, 1, 2, 3$
Let $\mathbf{B} = \mathbf{A}^2 + 3\mathbf{A}$, $\mathbf{C} = \mathbf{A}^2 - 3\mathbf{A}$

1. Find $\det(\mathbf{C})$
2. Is $\mathbf{B}$ a symmetric matrix?
3. Find $\|B\|_F^2$