Lecture 5
Regularization in Deep Neural Networks

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

April 10, 2017

- Things we will look at today
  - Norm Penalties (weight decay)

- Things we will look at today
  - Norm Penalties (weight decay)
  - Early Stopping as a form of Regularization

- Things we will look at today
  - Norm Penalties (weight decay)
  - Early Stopping as a form of Regularization
  - Dropout

- Things we will look at today
  - Norm Penalties (weight decay)
  - Early Stopping as a form of Regularization
  - Dropout
  - Other Approaches that have a regularizing effect

Housekeeping

- Quiz scores will be uploaded after class
- Projects:
  - One page proposal due 19 April 23:59
  - Summarize the task of interest and why is it of interest to you
  - Describe the dataset intended for use
  - Roughly: What model do you want to use?
  - What framework do you plan to use?
- Mid Term dates will be announced on Wednesday

Regularization

- Introduce additional information to solve an *ill posed* inverse problem

Regularization

- Introduce additional information to solve an *ill posed* inverse problem
- A practical way to impose Occam's Razor on the solution

Regularization

- Introduce additional information to solve an *ill posed* inverse problem
- A practical way to impose Occam's Razor on the solution
- We already looked at (Regularized Risk Minimization):

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) + \Omega(\theta)$$

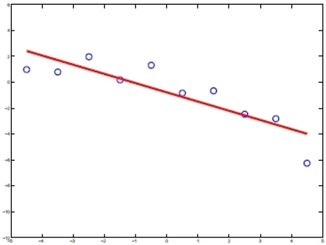
Regularization

- Introduce additional information to solve an *ill posed* inverse problem
- A practical way to impose Occam's Razor on the solution
- We already looked at (Regularized Risk Minimization):

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) + \Omega(\theta)$$

- **More generally:** Any modification to a learning algorithm intended to reduce its generalization error but not its training error

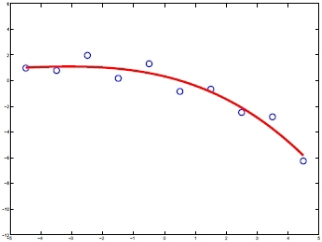
Training Regimes



$$m = 1$$

- **Regime 1 in training:** Model family excludes the true generation process (underfitting, high bias)

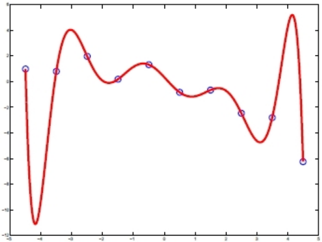
Training Regimes



$$m = 3$$

- **Regime 2 in training:** Model family matches the true generative process

Training Regimes



$$m = 10$$

- **Regime 3 in training:** The generative process is included but many other generating processes as well (overfitting!)

Regularization

Goal of Regularization: Take a model from the third regime to second regime

In Deep Learning

A trend: Use extremely large models (high capacity) and then regularize strongly (try to limit capacity)

In Deep Learning

A trend: Use extremely large models (high capacity) and then regularize strongly (try to limit capacity)

"If you are not in the small-data regime, you should just use a bigger model so that you are in the small-data regime. You should always be in the small-data regime." – David Belanger

Yet another quote

Geoffrey Hinton: "The brain has about 10^{14} synapses and we live for about 10^{9} seconds. So we have a lot more parameters than data."

- Note: $\#$ synapses $\equiv \#$ parameters is problematic, so is the use of seconds as a unit. But the point remains: *Large looking model, small data*

- Open area of research: How do deep learning models generalize with such large models that can "memorize" the data

- Open area of research: How do deep learning models generalize with such large models that can "memorize" the data
- Personal "Conjecture" (feel free to ignore!): A reasonable upper bound on the Kolmogorov Complexity of models with good generalization performance will turn out to be small i.e. they are essentially simple models, not as complex as they seem. Generalization in this case is a result of parsimony.

Parameter Norm Penalties

Parameter Norm Penalties

- Recall the regularized objective function:

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

Parameter Norm Penalties

- Recall the regularized objective function:

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

- Note: J can be the loss or the likelihood function, so we will call it *cost* interchangeably

Parameter Norm Penalties

- Recall the regularized objective function:

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

- Note: J can be the loss or the likelihood function, so we will call it *cost* interchangeably
- α is the tradeoff parameter

Parameter Norm Penalties

- Recall the regularized objective function:

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

- Note: J can be the loss or the likelihood function, so we will call it *cost* interchangeably
- α is the tradeoff parameter
  - $\alpha = 0$ implies no regularization
  - High value of α implies strong regularization

Penalizing the $L2$ norm

- Let's fold in all parameters θ i.e weight matrices, biases etc. into $\mathbf{w}$ (although biases are usually not regularized)

$$\tilde{J}(\mathbf{w}; X, y) = J(\mathbf{w}; X, y) + \frac{\alpha}{2}\|\mathbf{w}\|_2^2$$

Penalizing the $L2$ norm

- Let's fold in all parameters θ i.e weight matrices, biases etc. into $\mathbf{w}$ (although biases are usually not regularized)

$$\tilde{J}(\mathbf{w}; X, y) = J(\mathbf{w}; X, y) + \frac{\alpha}{2}\|\mathbf{w}\|_2^2$$

- The corresponding gradient then is:

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

Penalizing the $L2$ norm

- Familiar gradient update:

$$\mathbf{w} := \mathbf{w} - \epsilon(\alpha \mathbf{w} + \nabla_{\mathbf{w}} J(\mathbf{w}; X, y))$$

Penalizing the $L2$ norm

- Familiar gradient update:

$$\mathbf{w} := \mathbf{w} - \epsilon(\alpha\mathbf{w} + \nabla_{\mathbf{w}} J(\mathbf{w}; X, y))$$

- Let's re-write it:

$$\mathbf{w} := (1 - \epsilon\alpha)\mathbf{w} - \epsilon\nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

Update Rule

- Old update rule (without the penalty; seen before!):

$$\mathbf{w} := \mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

Update Rule

- Old update rule (without the penalty; seen before!):

$$\mathbf{w} := \mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

- New update rule:

$$\mathbf{w} := (1 - \epsilon\alpha)\mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

Update Rule

- Old update rule (without the penalty; seen before!):

$$\mathbf{w} := \mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

- New update rule:

$$\mathbf{w} := (1 - \epsilon\alpha)\mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

- **Interpretation:** Multiplicatively *shrink* weight vector by a constant factor before performing the usual gradient update

Update Rule

- Old update rule (without the penalty; seen before!):

$$\mathbf{w} := \mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

- New update rule:

$$\mathbf{w} := (1 - \epsilon\alpha)\mathbf{w} - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; X, y)$$

- **Interpretation:** Multiplicatively *shrink* weight vector by a constant factor before performing the usual gradient update
- This is the origin of the terminology *weight decay*

Update Rule for One Weight Matrix

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nn} \end{bmatrix} = (1 - \epsilon\alpha) \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nn} \end{bmatrix} - \epsilon \nabla_W J(\mathbf{w}; X, y)$$

$\epsilon \nabla_W J(\mathbf{w}; X, y)$ is also a $n \times n$ matrix

A Simple Analysis

$L2$ Penalty: Analysis

- Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$ (i.e. weights that attain optimal training cost on the unregularized objective)

$L2$ Penalty: Analysis

- Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$ (i.e. weights that attain optimal training cost on the unregularized objective)
- Consider a quadratic approximation to J evaluated at $\mathbf{w}^*$

$L2$ Penalty: Analysis

- Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$ (i.e. weights that attain optimal training cost on the unregularized objective)
- Consider a quadratic approximation to J evaluated at $\mathbf{w}^*$

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

$L2$ Penalty: Analysis

- Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$ (i.e. weights that attain optimal training cost on the unregularized objective)
- Consider a quadratic approximation to J evaluated at $\mathbf{w}^*$

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

- Since $\mathbf{w}^*$ is a minimum, $\nabla_{\mathbf{w}} J(\mathbf{w}^*) = 0$, then

$L2$ Penalty: Analysis

- Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$ (i.e. weights that attain optimal training cost on the unregularized objective)
- Consider a quadratic approximation to J evaluated at $\mathbf{w}^*$

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

- Since $\mathbf{w}^*$ is a minimum, $\nabla_{\mathbf{w}} J(\mathbf{w}^*) = 0$, then

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

$L2$ Penalty: Analysis

- Let $\mathbf{w}^* = \arg\min_{\mathbf{w}} J(\mathbf{w})$ (i.e. weights that attain optimal training cost on the unregularized objective)
- Consider a quadratic approximation to J evaluated at $\mathbf{w}^*$

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

- Since $\mathbf{w}^*$ is a minimum, $\nabla_{\mathbf{w}} J(\mathbf{w}^*) = 0$, then

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

- Note that $\nabla_{\mathbf{w}} J(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$ (just differentiate the quadratic approximation)

$L2$ Penalty: Analysis

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

$L2$ Penalty: Analysis

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

- Since $\mathbf{w}^*$ is a minimum: $\nabla_{\mathbf{w}} J(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*) = 0$

$L2$ Penalty: Analysis

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

- Since $\mathbf{w}^*$ is a minimum: $\nabla_{\mathbf{w}} J(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*) = 0$
- To understand what weight decay does, modify above by adding weight decay gradient:

$$\alpha\tilde{\mathbf{w}} + H(\tilde{\mathbf{w}} - \mathbf{w}^*) = 0$$

$L2$ Penalty: Analysis

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

- Since $\mathbf{w}^*$ is a minimum: $\nabla_{\mathbf{w}} J(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*) = 0$
- To understand what weight decay does, modify above by adding weight decay gradient:

$$\alpha \tilde{\mathbf{w}} + H(\tilde{\mathbf{w}} - \mathbf{w}^*) = 0$$

- Rearranging, we have:

$$\tilde{\mathbf{w}} = (H + \alpha I)^{-1} H \mathbf{w}^*$$

$L2$ Penalty: Analysis

$$\tilde{\mathbf{w}} = (H + \alpha I)^{-1} H \mathbf{w}^*$$

$L2$ Penalty: Analysis

$$\tilde{\mathbf{w}} = (H + \alpha I)^{-1} H \mathbf{w}^*$$

- As $\alpha \to 0$, $\tilde{\mathbf{w}} \to \mathbf{w}^*$ i.e. regularized solution approaches the unregularized solution

$L2$ Penalty: Analysis

$$\tilde{\mathbf{w}} = (H + \alpha I)^{-1} H \mathbf{w}^*$$

- As $\alpha \to 0$, $\tilde{\mathbf{w}} \to \mathbf{w}^*$ i.e. regularized solution approaches the unregularized solution
- Now H is real and symmetric $\implies H = Q\Lambda Q^T$

$L2$ Penalty: Analysis

$$\tilde{\mathbf{w}} = (H + \alpha I)^{-1} H \mathbf{w}^*$$

- As $\alpha \to 0$, $\tilde{\mathbf{w}} \to \mathbf{w}^*$ i.e. regularized solution approaches the unregularized solution
- Now H is real and symmetric $\implies H = Q \Lambda Q^T$
- Q are the eigenvectors and Λ is a diagonal matrix of eigenvalues

$L2$ Penalty: Analysis

$$\tilde{\mathbf{w}} = (H + \alpha I)^{-1} H \mathbf{w}^*$$

- As $\alpha \to 0$, $\tilde{\mathbf{w}} \to \mathbf{w}^*$ i.e. regularized solution approaches the unregularized solution
- Now H is real and symmetric $\implies H = Q \Lambda Q^T$
- Q are the eigenvectors and Λ is a diagonal matrix of eigenvalues
- Plug decomposition in above equation and rearrange:

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

$L2$ Penalty: Analysis

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

- What is the interpretation of this?

$L2$ Penalty: Analysis

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

- What is the interpretation of this?
- Effect of weight decay: Rescale $\mathbf{w}^*$ (the optimal solution for the unregularized objective) along axes defined by the H

$L2$ Penalty: Analysis

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

- What is the interpretation of this?
- Effect of weight decay: Rescale $\mathbf{w}^*$ (the optimal solution for the unregularized objective) along axes defined by the H
- Coordinate of $\mathbf{w}^*$ that is aligned with the ith eigenvector of H is rescaled by $\frac{\lambda_i}{\lambda_i + \alpha}$

$L2$ Penalty: Analysis

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

- Along eigenvectors of H that have large eigenvalues $\lambda_i \gg \alpha$, effect of regularization is small

$L2$ Penalty: Analysis

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

- Along eigenvectors of H that have large eigenvalues $\lambda_i \gg \alpha$, effect of regularization is small
- Directions for which $\lambda_i \ll \alpha$ the $\mathbf{w}_i^*$ coordinate will shrink to nearly zero

$L2$ Penalty: Analysis

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

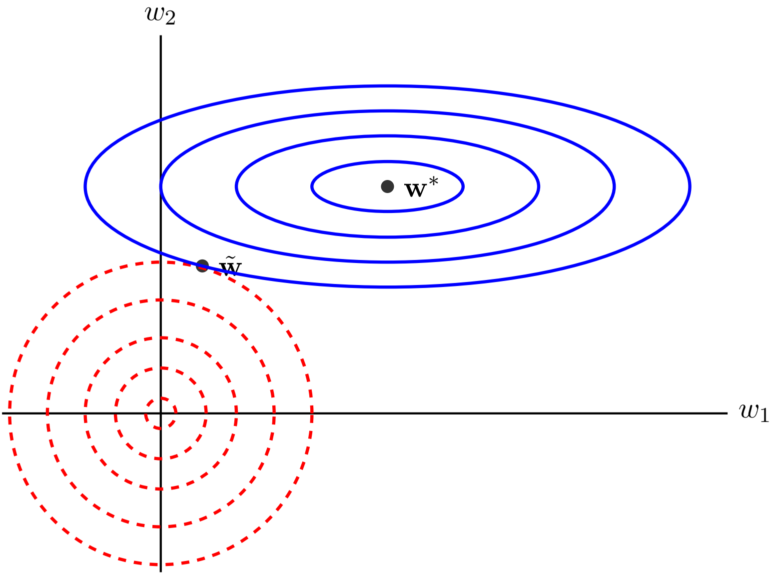
- Along eigenvectors of H that have large eigenvalues $\lambda_i \gg \alpha$, effect of regularization is small
- Directions for which $\lambda_i \ll \alpha$ the $\mathbf{w}_i^*$ coordinate will shrink to nearly zero
- In English:
    - Directions which contribute significantly to reducing the objective function value are kept relatively intact

$L2$ Penalty: Analysis

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

- Along eigenvectors of H that have large eigenvalues $\lambda_i \gg \alpha$, effect of regularization is small
- Directions for which $\lambda_i \ll \alpha$ the $\mathbf{w}_i^*$ coordinate will shrink to nearly zero
- In English:
  - Directions which contribute significantly to reducing the objective function value are kept relatively intact
  - Directions that make little contribution to reducing the objective function value are killed off

$L2$ Penalty: Analysis



$L1$ Weight Decay

$L1$ Regularization

- Recall the regularized objective function:

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

$L1$ Regularization

- Recall the regularized objective function:

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

- Again, fold in all the parameters θ (weight matrices, biases etc) into $\mathbf{w}$ and penalize $L1$ norm

$L1$ Regularization

- Recall the regularized objective function:

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

- Again, fold in all the parameters θ (weight matrices, biases etc) into $\mathbf{w}$ and penalize $L1$ norm

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

$L1$ Regularization

- Recall the regularized objective function:

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

- Again, fold in all the parameters θ (weight matrices, biases etc) into $\mathbf{w}$ and penalize $L1$ norm

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

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

$L1$ Regularization

- Recall the regularized objective function:

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

- Again, fold in all the parameters θ (weight matrices, biases etc) into $\mathbf{w}$ and penalize $L1$ norm

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

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

- We penalize the absolute value of parameters

$L1$ Regularization

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

$L1$ Regularization

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

- Corresponding gradient (sign applied element-wise)

$L1$ Regularization

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

- Corresponding gradient (sign applied element-wise)

$$\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}; X, y) =$$

$L1$ Regularization

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

- Corresponding gradient (sign applied element-wise)

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

$L1$ Regularization

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

- Corresponding gradient (sign applied element-wise)

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

- Recall the gradient for the $L2$ penalty:

$L1$ Regularization

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

- Corresponding gradient (sign applied element-wise)

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

- Recall the gradient for the $L2$ penalty:

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

Update Rule

- For $L2$ penalty

Update Rule

- For $L2$ penalty

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha \mathbf{w})$$

Update Rule

- For $L2$ penalty

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha \mathbf{w})$$

- For $L1$ penalty:

Update Rule

- For $L2$ penalty

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha \mathbf{w})$$

- For $L1$ penalty:

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha sign(\mathbf{w}))$$

Update Rule

- For $L2$ penalty

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha \mathbf{w})$$

- For $L1$ penalty:

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha sign(\mathbf{w}))$$

- Easy to implement, but effect of penalty is very different

Update Rule

- For $L2$ penalty

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha \mathbf{w})$$

- For $L1$ penalty:

$$\mathbf{w} := \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; X, y) + \alpha sign(\mathbf{w}))$$

- Easy to implement, but effect of penalty is very different
- Regularization contribution is only a constant α with $sign$ equal to $sign(w_i)$

Simple Analysis

- Write the Taylor series expansion as before

Simple Analysis

- Write the Taylor series expansion as before
- Gradient is again: $\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$

Simple Analysis

- Write the Taylor series expansion as before
- Gradient is again: $\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$
- H is the Hessian of the unregularized objective J w.r.t $\mathbf{w}$ evaluated at $\mathbf{w}^*$

Simple Analysis

- Write the Taylor series expansion as before
- Gradient is again: $\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$
- H is the Hessian of the unregularized objective J w.r.t $\mathbf{w}$ evaluated at $\mathbf{w}^*$
- For simplicity of analysis assume that Hessian is diagonal

Simple Analysis

- Write the Taylor series expansion as before
- Gradient is again: $\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$
- H is the Hessian of the unregularized objective J w.r.t $\mathbf{w}$ evaluated at $\mathbf{w}^*$
- For simplicity of analysis assume that Hessian is diagonal

$$H = diag([H_{1,1}, \ldots, H_{n,n}]) \text{ with } H_{i,i} > 0$$

Simple Analysis

- Write the Taylor series expansion as before
- Gradient is again: $\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$
- H is the Hessian of the unregularized objective J w.r.t $\mathbf{w}$ evaluated at $\mathbf{w}^*$
- For simplicity of analysis assume that Hessian is diagonal

$$H = diag([H_{1,1}, \ldots, H_{n,n}]) \text{ with } H_{i,i} > 0$$

- Skipping some steps, minimizing the approximate cost function has an analytical solution:

Simple Analysis

- Write the Taylor series expansion as before
- Gradient is again: $\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*)$
- H is the Hessian of the unregularized objective J w.r.t $\mathbf{w}$ evaluated at $\mathbf{w}^*$
- For simplicity of analysis assume that Hessian is diagonal

$$H = diag([H_{1,1}, \ldots, H_{n,n}]) \text{ with } H_{i,i} > 0$$

- Skipping some steps, minimizing the approximate cost function has an analytical solution:

$$w_i = sign(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

Simple Analysis

$$w_i = sign(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

Simple Analysis

$$w_i = sign(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

- Consider the case when $w_i^* > 0 \forall i$

Simple Analysis

$$w_i = sign(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the case when $w_i^* > 0 \forall i$
  - When $w_i^* < \frac{\alpha}{H_{i,i}}$, value of regularized objective $w_i = 0$

Simple Analysis

$$w_i = sign(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

- Consider the case when $w_i^* > 0 \forall i$
  - When $w_i^* < \frac{\alpha}{H_{i,i}}$, value of regularized objective $w_i = 0$
  - When $w_i^* > \frac{\alpha}{H_{i,i}}$, value of w_i is shifted towards zero by $\frac{\alpha}{H_{i,i}}$

Simple Analysis

$$w_i = sign(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the case when $w_i^* > 0 \forall i$
  - When $w_i^* < \frac{\alpha}{H_{i,i}}$, value of regularized objective $w_i = 0$
  - When $w_i^* > \frac{\alpha}{H_{i,i}}$, value of w_i is shifted towards zero by $\frac{\alpha}{H_{i,i}}$
- Similar behaviour when $w_i^* < 0$ with w_i either zero, or becoming less negative by $\frac{\alpha}{H_{i,i}}$

Simple Analysis

$$w_i = sign(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

- Consider the case when $w_i^* > 0 \forall i$
  - When $w_i^* < \frac{\alpha}{H_{i,i}}$, value of regularized objective $w_i = 0$
  - When $w_i^* > \frac{\alpha}{H_{i,i}}$, value of w_i is shifted towards zero by $\frac{\alpha}{H_{i,i}}$
- Similar behaviour when $w_i^* < 0$ with w_i either zero, or becoming less negative by $\frac{\alpha}{H_{i,i}}$
- Conclusion: $L1$ results in a sparser solution (as compared to $L2$)

Early Stopping

Bagging

Model Averaging

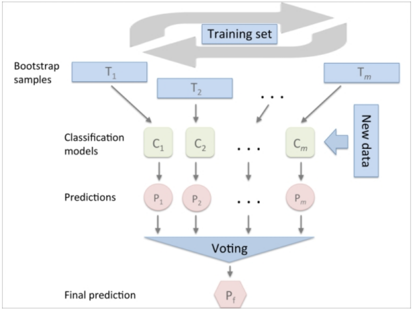
- Bootstrap AGGregatING: Train several *diverse* models separately and average them

Model Averaging

- Bootstrap AGGregatING: Train several *diverse* models separately and average them
- But you have only one training set (bootstrapping)

Model Averaging

- Bootstrap AGGregatING: Train several *diverse* models separately and average them
- But you have only one training set (bootstrapping)



Why does Bagging work?

- Suppose you have k regression models

Why does Bagging work?

- Suppose you have k regression models
- Suppose each model makes an error ϵ_i on each example, with errors drawn from a multivariate gaussian

Why does Bagging work?

- Suppose you have k regression models
- Suppose each model makes an error ϵ_i on each example, with errors drawn from a multivariate gaussian
- Let the variances be $\mathbb{E}[\epsilon_i^2] = v$ and covariances $\mathbb{E}[\epsilon_i \epsilon_j] = c$

Why does Bagging work?

- Suppose you have k regression models
- Suppose each model makes an error ϵ_i on each example, with errors drawn from a multivariate gaussian
- Let the variances be $\mathbb{E}[\epsilon_i^2] = v$ and covariances $\mathbb{E}[\epsilon_i \epsilon_j] = c$
- The average prediction of the k predictors:

$$\frac{1}{k} \sum_i \epsilon_i$$

Why does Bagging work?

- The expected squared error of the ensemble:

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

Why does Bagging work?

- The expected squared error of the ensemble:

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

$$= \frac{1}{k}v + \frac{k-1}{k}c$$

- When errors are perfectly correlated i.e. $c = v$ error reduces to v (averaging does not help)

Why does Bagging work?

- The expected squared error of the ensemble:

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

$$= \frac{1}{k}v + \frac{k-1}{k}c$$

- When errors are perfectly correlated i.e. $c = v$ error reduces to v (averaging does not help)
- When perfectly uncorrelated i.e. $c = 0$ error is $\frac{1}{k}v$

Why does Bagging work?

- The expected squared error of the ensemble:

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

$$= \frac{1}{k}v + \frac{k-1}{k}c$$

- When errors are perfectly correlated i.e. $c = v$ error reduces to v (averaging does not help)
- When perfectly uncorrelated i.e. $c = 0$ error is $\frac{1}{k}v$
- Error decreases linearly with ensemble size

Why does Bagging work?

- The expected squared error of the ensemble:

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

- When errors are perfectly correlated i.e. $c = v$ error reduces to v (averaging does not help)
- When perfectly uncorrelated i.e. $c = 0$ error is $\frac{1}{k}v$
- Error decreases linearly with ensemble size
- On average the ensemble performs atleast as well as any of its members

Reference

"Bagging Regularizes", Tomaso Poggio, Ryan Rifkin, Sayan Mukherjee, Alex Rakhlin, 2002

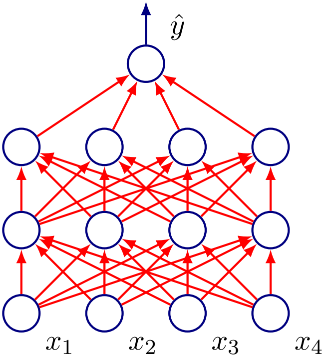
Dropout

Dropout

- A more exotic regularization technique introduced in 2012

Dropout

- A more exotic regularization technique introduced in 2012

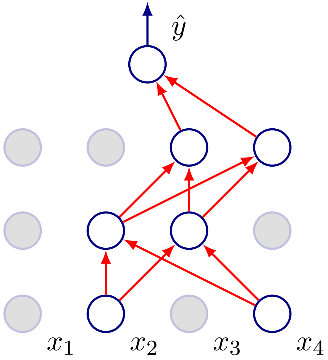


Dropout

- During training, each sample is processed by a decimated network

Dropout

- During training, each sample is processed by a decimated network

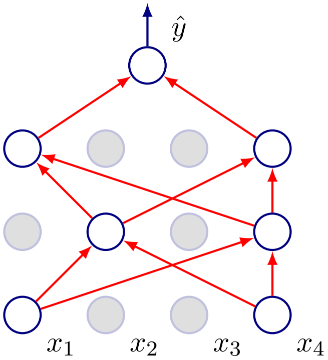


Dropout

- During training the feedforward operation and backpropagation is only done on the decimated network

Dropout

- During training the feedforward operation and backpropagation is only done on the decimated network



Dropout

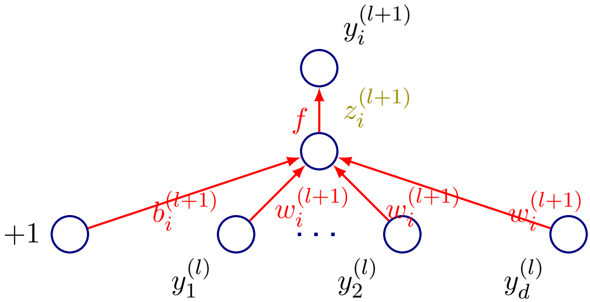
- The decimated network is generated by killing off units with a certain probability

Dropout

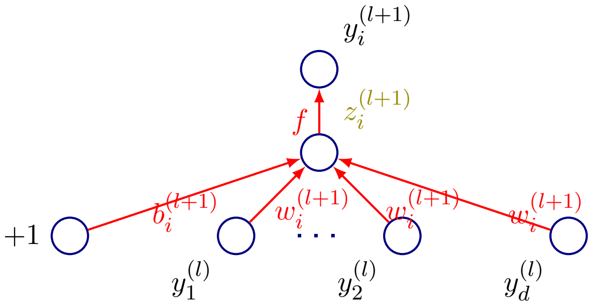
- The decimated network is generated by killing off units with a certain probability
- Consider a hidden node in a network

Dropout

- The decimated network is generated by killing off units with a certain probability
- Consider a hidden node in a network

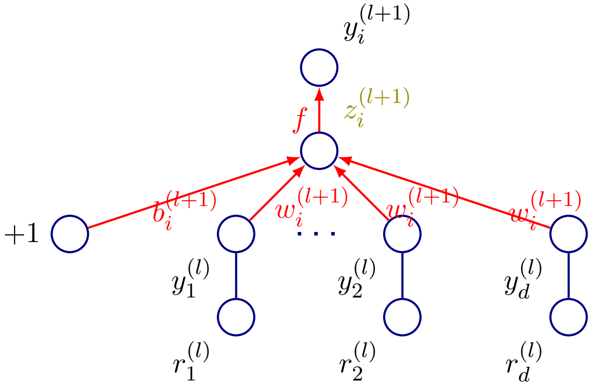


Without Dropout



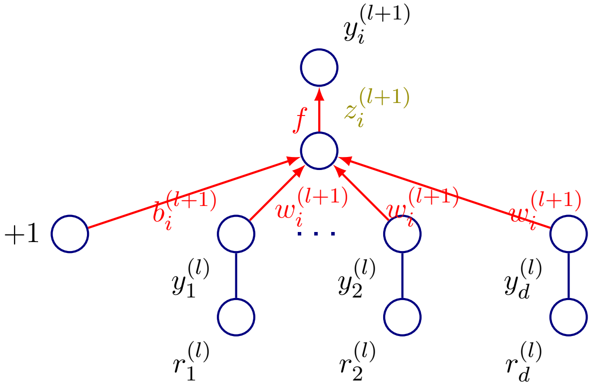
- $z_i^{(l)} = \mathbf{w}^{(l+1)T}\mathbf{y}^{(l)} + b_i^{(l+1)}$ and $y_i^{(l+1)} = f(z_i^{(l+1)})$

With Dropout



- Sample $r_i^{(l)} \sim \text{Bernoulli}(p)$, then $\tilde{y}^{(l)} = r_i^{(l)} * y^{(l)}$

With Dropout



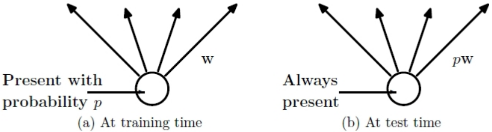
- Sample $r_i^{(l)} \sim \text{Bernoulli}(p)$, then $\tilde{y}^{(l)} = r_i^{(l)} * y^{(l)}$
- Then $z_i^{(l+1)} = \mathbf{w}_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(l)}$ and $y_i^{(l+1)} = f(z_i^{(l+1)})$

At Test Time

- Use a single neural network with weights scaled down

At Test Time

- Use a single neural network with weights scaled down
- What is the point of doing this?



Dropout

- During presentation of each input, the thinned network might be completely different

Dropout

- During presentation of each input, the thinned network might be completely different
- We are effectively sampling from 2^n possible networks

Dropout

- During presentation of each input, the thinned network might be completely different
- We are effectively sampling from 2^n possible networks
- Each network may get trained on only one example!

Dropout

- During presentation of each input, the thinned network might be completely different
- We are effectively sampling from 2^n possible networks
- Each network may get trained on only one example!
- We minimize the loss function stochastically under a noise distribution: Minimizing an expected loss function

Dropout

- During presentation of each input, the thinned network might be completely different
- We are effectively sampling from 2^n possible networks
- Each network may get trained on only one example!
- We minimize the loss function stochastically under a noise distribution: Minimizing an expected loss function
- During test time, we only want the expected output of each neuron, so weights are scaled down by p

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them
  - Later in class we will see a way around this (Dark Knowledge, Distillation)

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them
  - Later in class we will see a way around this (Dark Knowledge, Distillation)
- In Dropout:

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them
  - Later in class we will see a way around this (Dark Knowledge, Distillation)
- In Dropout:
  - Exponential number of models, but they *share* parameters

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them
  - Later in class we will see a way around this (Dark Knowledge, Distillation)
- In Dropout:
  - Exponential number of models, but they *share* parameters
  - Not feasible to explicitly average an exponential number of models

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them
  - Later in class we will see a way around this (Dark Knowledge, Distillation)
- In Dropout:
  - Exponential number of models, but they *share* parameters
  - Not feasible to explicitly average an exponential number of models
  - Scale down weights by p to get an approximate average

Extreme Form of Bagging

- In Bagging:
  - Train many *independent* models to convergence and average them
  - Usually prohibitively expensive to store them
  - Later in class we will see a way around this (Dark Knowledge, Distillation)
- In Dropout:
  - Exponential number of models, but they *share* parameters
  - Not feasible to explicitly average an exponential number of models
  - Scale down weights by p to get an approximate average
  - Can be thought of as an extreme form of bagging

Why Else does Dropout work?

- Noise injection at input, hidden layers

Why Else does Dropout work?

- Noise injection at input, hidden layers
- Can also cause shrinkage: Let's see a toy example

Dropout for Linear Regression

- Objective: $\|\mathbf{y} - X\mathbf{w}\|_2^2$

Dropout for Linear Regression

- Objective: $\|\mathbf{y} - X\mathbf{w}\|_2^2$
- Let $R \in \{0,1\}^{N \times D}$ be a random matrix with $R_{ij} \sim \text{Bernoulli}(p)$

Dropout for Linear Regression

- Objective: $\|\mathbf{y} - X\mathbf{w}\|_2^2$
- Let $R \in \{0,1\}^{N \times D}$ be a random matrix with $R_{ij} \sim \text{Bernoulli}(p)$
- Input is then expressed as $R \odot X$

Dropout for Linear Regression

- Objective: $\|\mathbf{y} - X\mathbf{w}\|_2^2$
- Let $R \in \{0,1\}^{N \times D}$ be a random matrix with $R_{ij} \sim \text{Bernoulli}(p)$
- Input is then expressed as $R \odot X$
- We now have a expected loss function:

$$\min_{\mathbf{w}} \mathbb{E}_{R \sim \text{Bernoulli}(p)} \|\mathbf{y} - (R \odot X)\mathbf{w}\|_2^2$$

Dropout for Linear Regression

- After some basic manipulation:

Dropout for Linear Regression

- After some basic manipulation:

$$\min_{\mathbf{w}} \mathbb{E}_{R \sim \text{Bernoulli}(p)} \|\mathbf{y} - (R \odot X)\mathbf{w}\|_2^2$$
$$= \min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma \mathbf{w}\|_2^2$$

Dropout for Linear Regression

- After some basic manipulation:

$$\min_{\mathbf{w}} \mathbb{E}_{R\sim \text{Bernoulli}(p)} \|\mathbf{y} - (R \odot X)\mathbf{w}\|_2^2 = \min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma\mathbf{w}\|_2^2$$

- Where $\Gamma = (diag(X^T X))^{1/2}$

Dropout for Linear Regression

- After some basic manipulation:

$$\min_{\mathbf{w}} \mathbb{E}_{R\sim\text{Bernoulli}(p)} \|\mathbf{y} - (R \odot X)\mathbf{w}\|_2^2$$
$$= \min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma\mathbf{w}\|_2^2$$

- Where $\Gamma = (diag(X^T X))^{1/2}$
- In expectation, dropout with linear regression is equivalent to ridge regression with a particular form for Γ

Dropout for Linear Regression

- After some basic manipulation:

$$\min_{\mathbf{w}} \mathbb{E}_{R\sim\text{Bernoulli}(p)} \|\mathbf{y} - (R \odot X)\mathbf{w}\|_2^2$$
$$= \min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma\mathbf{w}\|_2^2$$

- Where $\Gamma = (diag(X^T X))^{1/2}$
- In expectation, dropout with linear regression is equivalent to ridge regression with a particular form for Γ
- Γ scales down weight cost for each w_i by the standard deviation of ith dimension of data

Dropout for Linear Regression

$$\min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma\mathbf{w}\|_2^2$$

Dropout for Linear Regression

$$\min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma\mathbf{w}\|_2^2$$

- This can be equivalently viewed as:

$$\min_{\mathbf{w}} \|\mathbf{y} - X\tilde{\mathbf{w}}\|_2^2 + \frac{(1-p)}{p}\|\Gamma\tilde{\mathbf{w}}\|_2^2 \text{ with } \tilde{\mathbf{w}} = p\mathbf{w}$$

Dropout for Linear Regression

$$\min_{\mathbf{w}} \|\mathbf{y} - pX\mathbf{w}\|_2^2 + p(1-p)\|\Gamma\mathbf{w}\|_2^2$$

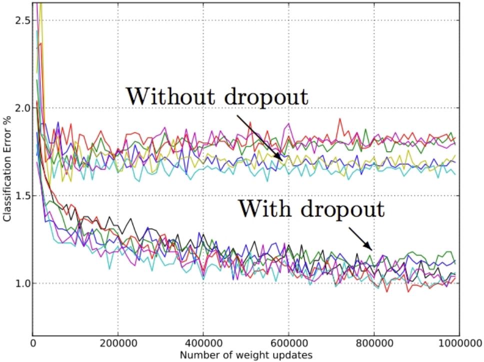
- This can be equivalently viewed as:

$$\min_{\mathbf{w}} \|\mathbf{y} - X\tilde{\mathbf{w}}\|_2^2 + \frac{(1-p)}{p}\|\Gamma\tilde{\mathbf{w}}\|_2^2 \text{ with } \tilde{\mathbf{w}} = p\mathbf{w}$$

- Another interpretation: When p is close to one all inputs are retained and regularization constant is small

Dropout: Performance

These architectures have 2 to 4 hidden layers with 1024 to 2048 hidden units

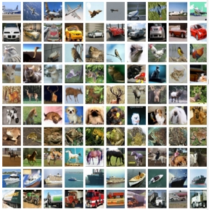


Dropout: Performance

(a) Street View House Numbers (SVHN)

Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.



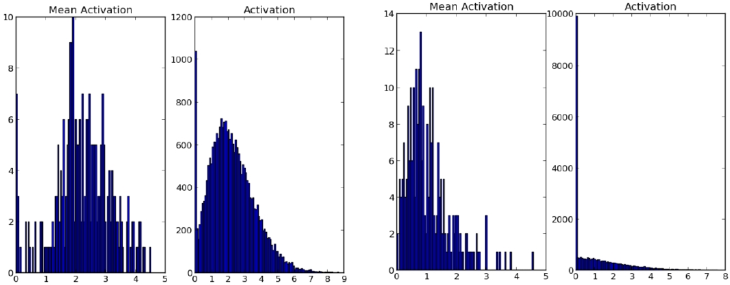
(b) CIFAR-10

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

Dropout: A simple way to prevent neural networks from overfitting, N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov, JMLR 2014

Dropout: Effect on Sparsity



(a) Without dropout (b) Dropout with $p = 0.5$.

Dropout: A simple way to prevent neural networks from overfitting, N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov, JMLR 2014

Why does this make sense?

- We saw three clear reasons:

Why does this make sense?

- We saw three clear reasons:
  - Noise injection and robustification

Why does this make sense?

- We saw three clear reasons:
  - Noise injection and robustification
  - Bagging

Why does this make sense?

- We saw three clear reasons:
  - Noise injection and robustification
  - Bagging
  - Shrinkage
- Some motivations:

Motivation

- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"

Motivation

- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"
- **Motivation 2:** Comes from a theory for the superiority of sexual reproduction in evolution (Livnat, Papadimitriou, PNAS, 2010).

Motivation

- **Motivation 1:** Ten conspiracies each involving five people is probably a better way to wreak havoc than a conspiracy involving 50 people. If conditions don't change (stationary) and plenty of time for rehearsal, a big conspiracy can work well, but otherwise will "overfit"
- **Motivation 2:** Comes from a theory for the superiority of sexual reproduction in evolution (Livnat, Papadimitriou, PNAS, 2010).
- Criterion for natural selection may not be individual fitness but mixability. Thus role of sexual reproduction is not just to allow useful new genes to propagate but also to ensure that complex coadaptations between genes are broken.

Dataset Augmentation

Dataset Augmentation

- Sure-shot way to generalize better: Get more data!

Dataset Augmentation

- Sure-shot way to generalize better: Get more data!
- What if your training data is limited?

Dataset Augmentation

- Sure-shot way to generalize better: Get more data!
- What if your training data is limited?
- In some cases (e.g. object recog.) easy to generate fake data

Dataset Augmentation

- Sure-shot way to generalize better: Get more data!
- What if your training data is limited?
- In some cases (e.g. object recog.) easy to generate fake data

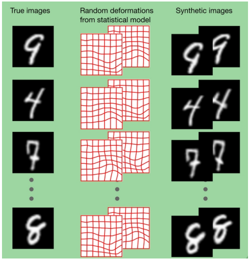


Image Credit: Søren Hauberg

- Warning: Be careful in what transformations you apply to your data

Noise Injection

Noise Injection

- Adding noise to data/input and using them for training is a form of dataset augmentation

Noise Injection

- Adding noise to data/input and using them for training is a form of dataset augmentation
- Generally: Noise injection can be much more powerful than penalizing the parameters

Noise Injection

- Adding noise to data/input and using them for training is a form of dataset augmentation
- Generally: Noise injection can be much more powerful than penalizing the parameters
- In what other ways can we add noise ?

Injecting noise to weights

- To understand how injecting noise to weights might help, consider the least squares cost function

Injecting noise to weights

- To understand how injecting noise to weights might help, consider the least squares cost function

$$J = \mathbb{E}_{p(\mathbf{x},y)}[(\hat{y} - y)^2]$$

Injecting noise to weights

- To understand how injecting noise to weights might help, consider the least squares cost function

$$J = \mathbb{E}_{p(\mathbf{x},y)}[(\hat{y} - y)^2]$$

- Assume that during training, with each example $\mathbf{x}, y$, we also randomly perturb the weights by $\epsilon_W \sim \mathcal{N}(\epsilon; 0, \eta I)$

Injecting noise to weights

- To understand how injecting noise to weights might help, consider the least squares cost function

$$J = \mathbb{E}_{p(\mathbf{x},y)}[(\hat{y} - y)^2]$$

- Assume that during training, with each example $\mathbf{x}, y$, we also randomly perturb the weights by $\epsilon_W \sim \mathcal{N}(\epsilon; 0, \eta I)$
- Let the perturbed model be denoted as: $\hat{y}_{\epsilon_W}$

Injecting noise to weights

- We still care about minimizing the squared error

Injecting noise to weights

- We still care about minimizing the squared error

$$\tilde{J}_W =$$

Injecting noise to weights

- We still care about minimizing the squared error

$$\tilde{J}_W = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[(\hat{y}_{\epsilon_W} - y)^2] =$$

Injecting noise to weights

- We still care about minimizing the squared error

$$\tilde{J}_W = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[(\hat{y}_{\epsilon_W} - y)^2] = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[\hat{y}_{\epsilon_W}^2 - 2y\hat{y}_{\epsilon_W} + y^2]$$

Injecting noise to weights

- We still care about minimizing the squared error

$$\tilde{J}_W = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[(\hat{y}_{\epsilon_W} - y)^2] = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[\hat{y}^2_{\epsilon_W} - 2y\hat{y}_{\epsilon_W} + y^2]$$

- Write out the gradient and update

Injecting noise to weights

- We still care about minimizing the squared error

$$\tilde{J}_W = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[(\hat{y}_{\epsilon_W} - y)^2] = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[\hat{y}^2_{\epsilon_W} - 2y\hat{y}_{\epsilon_W} + y^2]$$

- Write out the gradient and update
- Observation: For small η, minimization of J with added noise, is equivalent to minimization of J with an extra regularization term $\eta \mathbb{E}_{p(\mathbf{x},y}[||\nabla_W \hat{y}||]$

Injecting noise to weights

- We still care about minimizing the squared error

$$\tilde{J}_W = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[(\hat{y}_{\epsilon_W} - y)^2] = \mathbb{E}_{p(\mathbf{x},y),\epsilon_W}[\hat{y}^2_{\epsilon_W} - 2y\hat{y}_{\epsilon_W} + y^2]$$

- Write out the gradient and update
- Observation: For small η, minimization of J with added noise, is equivalent to minimization of J with an extra regularization term $\eta\mathbb{E}_{p(\mathbf{x},y}[\|\nabla_W \hat{y}\|]$
- This automatically pushes the model into regions where it is relatively insensitive to perturbations in the weights (Hochreiter and Schmidhuber, 1995)

Injecting noise into outputs

- What if your dataset has wrongly labeled examples?

Injecting noise into outputs

- What if your dataset has wrongly labeled examples?
- Maximizing likelihood $\log p(y|\mathbf{x})$ with mistakes in labels can be problematic

Injecting noise into outputs

- What if your dataset has wrongly labeled examples?
- Maximizing likelihood $\log p(y|\mathbf{x})$ with mistakes in labels can be problematic
- How can this problem be solved?

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output
- For example, in MNIST for digit 6:

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output
- For example, in MNIST for digit 6:

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.05 & 0 & 0 & 0.02 & 0.08 & 0.05 & 0.8 & 0 & 0 & 0 \end{bmatrix}$$

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output
- For example, in MNIST for digit 6:

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.05 & 0 & 0 & 0.02 & 0.08 & 0.05 & 0.8 & 0 & 0 & 0 \end{bmatrix}$$

- If you knew your data was mislabeled, and you knew that the training set label y was correct with probability $1 - \epsilon$

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output
- For example, in MNIST for digit 6:

$$\mathbf{y} = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}$$

$$\hat{\mathbf{y}} = \begin{bmatrix}0.05 & 0 & 0 & 0.02 & 0.08 & 0.05 & 0.8 & 0 & 0 & 0\end{bmatrix}$$

- If you knew your data was mislabeled, and you knew that the training set label y was correct with probability $1 - \epsilon$
- Instead of a one hot encoding for $\mathbf{y}$, you would replace 0 with $\frac{\epsilon}{k-1}$ and 1 with $1 - \epsilon$

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output
- For example, in MNIST for digit 6:

$$\mathbf{y} = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix}$$

$$\hat{\mathbf{y}} = \begin{bmatrix}0.05 & 0 & 0 & 0.02 & 0.08 & 0.05 & 0.8 & 0 & 0 & 0\end{bmatrix}$$

- If you knew your data was mislabeled, and you knew that the training set label y was correct with probability $1 - \epsilon$
- Instead of a one hot encoding for $\mathbf{y}$, you would replace 0 with $\frac{\epsilon}{k-1}$ and 1 with $1 - \epsilon$
- Use cross entropy on this instead

Solution: Smooth Labels

- Recall cross entropy is done between a vector of predictions and a one hot encoding of the right output
- For example, in MNIST for digit 6:

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\hat{\mathbf{y}} = \begin{bmatrix} 0.05 & 0 & 0 & 0.02 & 0.08 & 0.05 & 0.8 & 0 & 0 & 0 \end{bmatrix}$$

- If you knew your data was mislabeled, and you knew that the training set label y was correct with probability $1 - \epsilon$
- Instead of a one hot encoding for $\mathbf{y}$, you would replace 0 with $\frac{\epsilon}{k-1}$ and 1 with $1 - \epsilon$
- Use cross entropy on this instead
- This is an old idea in Machine Learning going to the early 80s

Multi-Task Learning

Multi-Task Learning

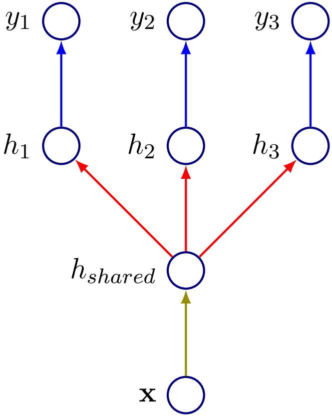
- Improves generalization by pooling examples across tasks

Multi-Task Learning

- Improves generalization by pooling examples across tasks
- Pooling examples $\implies$ soft constraint on the parameters

Multi-Task Learning

- Improves generalization by pooling examples across tasks
- Pooling examples $\implies$ soft constraint on the parameters



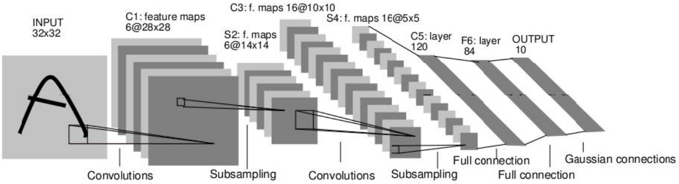
Parameter Sharing

Parameter Sharing

- Sometimes we can use our prior knowledge to impose constraints or dependencies amongst model parameters

Parameter Sharing

- Sometimes we can use our prior knowledge to impose constraints or dependencies amongst model parameters
- Popular way to use constraints: Force sets of parameters to be equal



Representational Sparsity

Recap

- Recall parameter penalized objective from earlier:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i; \theta), y_i) + \Omega(\theta)$$

Recap

- Recall parameter penalized objective from earlier:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) + \Omega(\theta)$$

- For the $L1$ penalty on the *parameters*:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) + \|\theta\|_1$$

Recap

- Recall parameter penalized objective from earlier:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) + \Omega(\theta)$$

- For the $L1$ penalty on the *parameters*:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i;\theta), y_i) + \|\theta\|_1$$

- We saw that the $L1$ penalty encouraged parameters to be *sparse*

Representational Sparsity

- Force the *representation* instead of the *parameters* to be sparse

Representational Sparsity

- Force the *representation* instead of the *parameters* to be sparse
- In other words: Only allow a small number of hidden neurons per layer to fire

Representational Sparsity

- Force the *representation* instead of the *parameters* to be sparse
- In other words: Only allow a small number of hidden neurons per layer to fire
- Has a similar regularizing effect!

Representational Sparsity

- Force the *representation* instead of the *parameters* to be sparse
- In other words: Only allow a small number of hidden neurons per layer to fire
- Has a similar regularizing effect!
- How can we do this?

Representational Sparsity

- Force the *representation* instead of the *parameters* to be sparse
- In other words: Only allow a small number of hidden neurons per layer to fire
- Has a similar regularizing effect!
- How can we do this?
- Add a $L1$ penalty on the representation:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i; \theta), y_i) + \|h\|_1$$

Representational Sparsity

- Force the *representation* instead of the *parameters* to be sparse
- In other words: Only allow a small number of hidden neurons per layer to fire
- Has a similar regularizing effect!
- How can we do this?
- Add a $L1$ penalty on the representation:

$$J(\theta) = \sum_{i=1}^{N} L(f(x_i; \theta), y_i) + \|h\|_1$$

- h represents the hidden unit activations

Illustration: Parameter Sparsity

$$\underbrace{\begin{bmatrix} 7 \\ -43 \\ 3 \\ -74 \\ 29 \\ -15 \end{bmatrix}}_{\text{Output, } \mathbf{y}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 4 \\ 0 & -9 & 0 & 0 & 0 & -3 \\ -1 & 0 & 0 & 0 & -7 & 0 \\ 0 & 0 & 0 & 5 & 5 & -1 \\ 0 & 0 & 0 & -9 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{Parameters, } W} \underbrace{\begin{bmatrix} -1 \\ 5 \\ -7 \\ -4 \\ 6 \\ 3 \end{bmatrix}}_{\text{Input, } \mathbf{x} \text{ or } \mathbf{h}}$$

- Sparsity in parameters (possibly induced by a $L1$ penalty on parameters)

Illustration: Representational Sparsity

$$\underbrace{\begin{bmatrix} -13 \\ -43 \\ 5 \\ -61 \\ 11 \\ -16 \end{bmatrix}}_{\text{Output, } \mathbf{y}} = \underbrace{\begin{bmatrix} 3 & 4 & -5 & 6 & 8 & -7 \\ 1 & -3 & 1 & 6 & -9 & 0 \\ 4 & 6 & 7 & 8 & 1 & -1 \\ 2 & 6 & -1 & 9 & -1 & 3 \\ 1 & -1 & -2 & 2 & 4 & 5 \\ -1 & 3 & 8 & 2 & 5 & 6 \end{bmatrix}}_{\text{Parameters, } W} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ -7 \\ 1 \\ 0 \end{bmatrix}}_{\text{Input, } \mathbf{h}}$$

- Sparsity in representation (possibly induced by a $L1$ penalty on activations)

Representational Sparsity

- Another approach: Put a hard constraint on the activations

Representational Sparsity

- Another approach: Put a hard constraint on the activations
- Example: Orthogonal Matching Pursuit

$$\arg \min_{h, \|h\|_0 \leq k} \|x - Wh\|^2$$

Representational Sparsity

- Another approach: Put a hard constraint on the activations
- Example: Orthogonal Matching Pursuit

$$\arg \min_{h, \|h\|_0 \leq k} \|x - Wh\|^2$$

- $\|h\|_0$ is number of non-zero entries of h

Representational Sparsity

- Another approach: Put a hard constraint on the activations
- Example: Orthogonal Matching Pursuit

$$\arg \min_{h, \|h\|_0 \leq k} \|x - Wh\|^2$$

- $\|h\|_0$ is number of non-zero entries of h
- Encodes the input x with a representation h when at most k of its entries are allowed to be non-zero

Representational Sparsity

- Another approach: Put a hard constraint on the activations
- Example: Orthogonal Matching Pursuit

$$\arg \min_{h, \|h\|_0 \leq k} \|x - Wh\|^2$$

- $\|h\|_0$ is number of non-zero entries of h
- Encodes the input x with a representation h when at most k of its entries are allowed to be non-zero
- Efficiently solvable when W is constrained to be orthogonal

Representational Sparsity

- Another approach: Put a hard constraint on the activations
- Example: Orthogonal Matching Pursuit

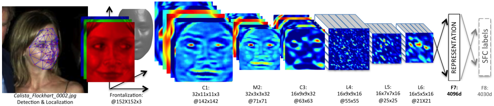
$$\arg \min_{h, \|h\|_0 \leq k} \|x - Wh\|^2$$

- $\|h\|_0$ is number of non-zero entries of h
- Encodes the input x with a representation h when at most k of its entries are allowed to be non-zero
- Efficiently solvable when W is constrained to be orthogonal
- People who have seen sparse coding will recognize this!

Adversarial Training

Motivation

- Since 2014 or so, Deep Neural Networks have matched human performance on some specific tasks:
  - Face recognition (Taigman *et al.*, CVPR 2014)



  - Reading addresses
  - Solving Captchas
  - ...

But do they really “understand”?

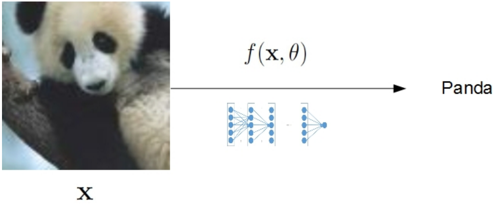
- An interesting phenomenon: Adversarial Examples

But do they really "understand"?

- An interesting phenomenon: Adversarial Examples
- Consider an image classification task with example $\mathbf{x}$ **correctly** classified as y by a network $f(\mathbf{x}, \theta)$

But do they really "understand"?

- An interesting phenomenon: Adversarial Examples
- Consider an image classification task with example $\mathbf{x}$ **correctly** classified as y by a network $f(\mathbf{x}, \theta)$



Adversarial Examples

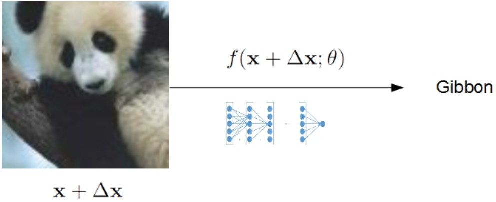
- Suppose we want to *attack* this network into predicting $\mathbf{x}$ to a goal class y_g Gibbon

Adversarial Examples

- Suppose we want to *attack* this network into predicting $\mathbf{x}$ to a goal class y_g Gibbon
- We want to do this by adding to $\mathbf{x}$ a very small perturbation $\Delta\mathbf{x}$ imperceptible to the human eye

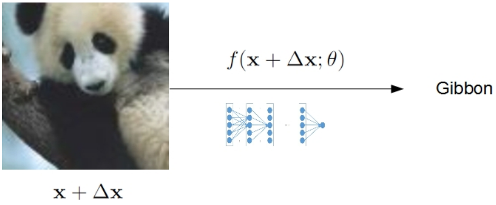
Adversarial Examples

- Suppose we want to *attack* this network into predicting $\mathbf{x}$ to a goal class y_g Gibbon
- We want to do this by adding to $\mathbf{x}$ a very small perturbation $\Delta\mathbf{x}$ imperceptible to the human eye



Adversarial Examples

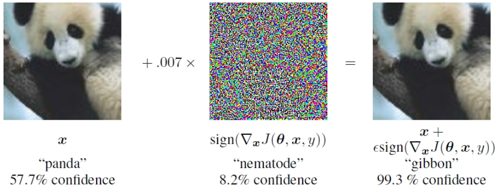
- Suppose we want to *attack* this network into predicting $\mathbf{x}$ to a goal class y_g Gibbon
- We want to do this by adding to $\mathbf{x}$ a very small perturbation $\Delta\mathbf{x}$ imperceptible to the human eye



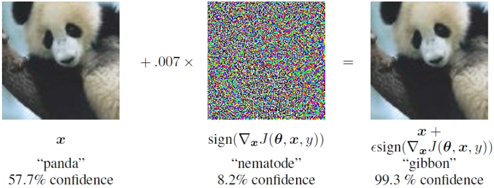
- Obvious optimization problem:

$$\arg\min_{\Delta\mathbf{x}} \|\Delta\mathbf{x}\| \text{ s.t. } f(\mathbf{x} + \Delta\mathbf{x}; \theta) = y_g$$

Adversarial Examples

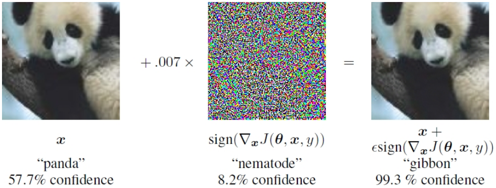


Adversarial Examples



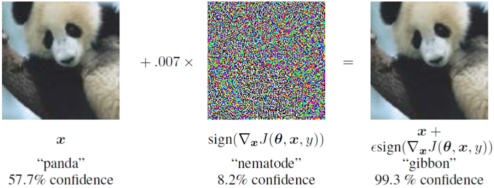
- Such examples are called adversarial examples

Adversarial Examples



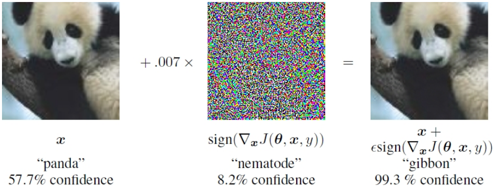
- Such examples are called adversarial examples
- Interesting properties: Such examples often generalize across datasets and models (implications for computer security!)

Adversarial Examples



- Such examples are called adversarial examples
- Interesting properties: Such examples often generalize across datasets and models (implications for computer security!)
- Not specific to deep networks!

Adversarial Examples



- Such examples are called adversarial examples
- Interesting properties: Such examples often generalize across datasets and models (implications for computer security!)
- Not specific to deep networks!
- Designing networks resistant to adversarial attacks is a very active (and important) area of research

Adversarial Training

- Recall the optimization problem

$$\arg\min_{\Delta \mathbf{x}} \|\Delta \mathbf{x}\| \textbf{ s.t. } f(\mathbf{x} + \Delta \mathbf{x}; \theta) = y_g$$

Adversarial Training

- Recall the optimization problem

$$\arg\min_{\Delta\mathbf{x}} \|\Delta\mathbf{x}\| \textbf{ s.t. } f(\mathbf{x}+\Delta\mathbf{x};\theta) = y_g$$

- In general the optimization can be complicated, but adversarial perturbations can also be generated in closed form (Goodfellow *et al.*, ICLR 2015)

Adversarial Training

- Recall the optimization problem

$$\arg\min_{\Delta\mathbf{x}} \|\Delta\mathbf{x}\| \ \textbf{s.t.} \ f(\mathbf{x} + \Delta\mathbf{x}; \theta) = y_g$$

- In general the optimization can be complicated, but adversarial perturbations can also be generated in closed form (Goodfellow *et al.*, ICLR 2015)

Adversarial Training

- Adversarial Training:
  - For correctly classified examples generate adversarial perturbations by either solving an optimization problem or a closed form method (e.g. fast gradient sign method)

Adversarial Training

- Adversarial Training:
  - For correctly classified examples generate adversarial perturbations by either solving an optimization problem or a closed form method (e.g. fast gradient sign method)
  - Add to the original training set these adversarial examples and force the network to correctly classify them

Adversarial Training

- Adversarial Training:
  - For correctly classified examples generate adversarial perturbations by either solving an optimization problem or a closed form method (e.g. fast gradient sign method)
  - Add to the original training set these adversarial examples and force the network to correctly classify them
- Makes the network more robust. But what does this have to do with regularization?

Adversarial Training

- Adversarial Training:
  - For correctly classified examples generate adversarial perturbations by either solving an optimization problem or a closed form method (e.g. fast gradient sign method)
  - Add to the original training set these adversarial examples and force the network to correctly classify them
- Makes the network more robust. But what does this have to do with regularization?
- A form of regularization like dataset augmentation, robustifying the network to perturbations

Next time

- Optimization Methods for Deep Neural Networks