# Deep Learning

## Regularization

Gunhee Kim

Computer Science and Engineering

서울대학교

SEOUL NATIONAL UNIVERSITY

# Outline

- Parameter Norm Penalties
    - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

# Regularization

*Any modification we make to a ML algorithm that is intended to reduce its generalization error but not its training error*

Several strategy

- Extra constraints on a machine learning algorithm
- Extra terms in the objective (based on prior knowledge)
- Ensemble methods (combine multiple hypotheses that explain the training data)

# Parameter Norm Penalties

## Basic form

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

- $J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$: original objective
- $\Omega(\boldsymbol{\theta})$: a parameter norm penalty
- $\alpha \in [0, \infty)$ a hyperparameter that weights the relative contribution of the penalty

## Meaning

- Decreasing $\tilde{J}$: decrease both the original objective $J$ on the training data + some measure of the size of parameter $\boldsymbol{\theta}$
- Different choice of $\Omega(\boldsymbol{\theta})$ results in a different solution

# L2 Regularization

Drive the weights closer to the origin by adding the term

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

- Also known as *ridge regression* or *Tikhonov regularization*

Then the objective $\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$

- Then the gradient is

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

- A single gradient step

$$\boldsymbol{w} \longleftarrow \boldsymbol{w} \ - \epsilon(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}))$$

$$\boldsymbol{w} \longleftarrow \underline{(1 - \epsilon\alpha)}\boldsymbol{w} \ - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$

Multiplicatively shrink the weight vector by a constant factor on each step

# More Analysis for L2 Regularization

Quadratic approximation to the objective in the neighborhood of the unregularized optimal weight (i.e. $\boldsymbol{w}^* = \operatorname{argmin}_{\boldsymbol{w}} J(\boldsymbol{w})$)

$$\hat{J}(\boldsymbol{w}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w}^*) + \cancel{g(\boldsymbol{w} - \boldsymbol{w}^*)} + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T\boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

$g = 0$ at min

- The minimum $\hat{J}(\boldsymbol{w})$ occurs at

$$\nabla_{\boldsymbol{w}}\hat{J}(\boldsymbol{w}) = \alpha\widetilde{\boldsymbol{w}} + \boldsymbol{H}(\widetilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$

$$(\boldsymbol{H} + \alpha\boldsymbol{I})\widetilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$

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

# More Analysis for L2 Regularization

Since $\boldsymbol{H}$ is real and symmetric, $\boldsymbol{H} = \boldsymbol{Q\Lambda Q}^T$

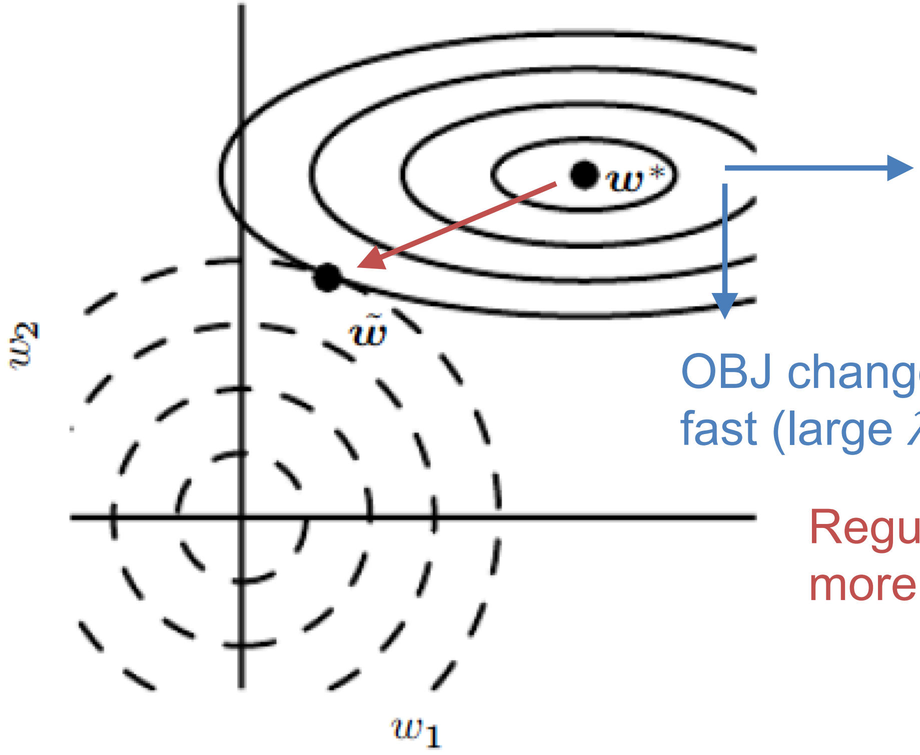
- $\boldsymbol{\Lambda}$ is a diagonal eigenvalue matrix, and $\boldsymbol{Q}$ is engenvectors

$$\begin{aligned}\widetilde{\boldsymbol{w}} &= (\boldsymbol{Q\Lambda Q}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q\Lambda Q}^T\boldsymbol{w}^* \\ &= (\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^T)^{-1}\boldsymbol{Q\Lambda Q}^T\boldsymbol{w}^* \\ &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I}\ )^{-1}\boldsymbol{\Lambda Q}^T\boldsymbol{w}^*\end{aligned}$$

- The component of $\boldsymbol{w}^*$aligned with the i-th engienvector of $\boldsymbol{H}$ is rescaled by a factor of $\lambda_i/(\lambda_i + \alpha)$
- $\lambda_i \gg \alpha$ (the direction where the eigenvalues of $\boldsymbol{H}$ are large) $\Longrightarrow$ the effect of regularization is relatively small
- $\lambda_i \ll \alpha \Longrightarrow$ the component is shrunk to have nearly zero

# Effect of L2 Regularization

$\tilde{\boldsymbol{w}}$ is an equilibrium btw the optimal of objective $\boldsymbol{w}^*$ and weight decay regularization



OBJ change is slow (small $\lambda_1$)

OBJ change is fast (large $\lambda_2$)

Regularizer pulls $w_i$ more to zero

# L1 Regularization

Another option: Penalize the nonzero elements

$$\Omega(\boldsymbol{\theta}) = \alpha||\boldsymbol{w}||_1$$

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

- Then the gradient is

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

- A regularization effect to each component is a constant factor $\alpha$ with a $\mathrm{sign}(w_i)$, instead of scaling with each $w_i$ in L2

# More Analysis for L1 Regularization

Suppose $\boldsymbol{H}$ is positive diagonal (i.e. $H_{ii} > 0$)

- Because the Algebraic form is messy for general $\boldsymbol{H}$

$$\hat{J}(\boldsymbol{w}) = \alpha||\boldsymbol{w}||_1 \ + J(\boldsymbol{w}^*) + \cancel{g(\boldsymbol{w} - \boldsymbol{w}^*)} + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

$g = 0$ at min

$$= J(\boldsymbol{w}^*) + \sum\nolimits_i [\frac{1}{2} H_{ii}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha|\boldsymbol{w}_i|\,]$$

- The analytic solution for each component $i$ is

$$w_i = \operatorname{sign}(w_i^*) \max\{|w_i^*| - \frac{\alpha}{H_{ii}}, 0\}$$

- For all $w_i^* > 0$, if $w_i^* \leq \frac{\alpha}{H_{ii}}$, simply $w_i = 0$.

  Otherwise ($w_i^* > \frac{\alpha}{H_{ii}}$), $w_i$ shrinks by $\frac{\alpha}{H_{ii}}$

- Similar process happens for all $w_i^* < 0$

# Under-constrained Problems

Many algorithms require $\boldsymbol{X}^T\boldsymbol{X}$ to be invertible

- e.g. solution of linear regression: $\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

However, often $\boldsymbol{X}^T\boldsymbol{X}$ is singular

- Whenever the data truly has no variance in some components
- There are fewer examples than the dimension size

Inverting regularized $(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})$ instead

- The regularized matrix is guarantee to be invertible
- e.g. solution of linear regression: $\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

# Under-constrained Problems

Logistic regression classifier for linearly separable problem

- If $\boldsymbol{w}$ achieves perfect classification, then $2\boldsymbol{w}$ does as well with higher likelihood
- An iterative optimization (e.g. SGD) may never halt
- Weight decay regularization can quit it when the slope of likelihood is equal to the weight decay coefficient

Moore-Penrose pseudoinverse

- A generalization inverse matrix

$$\boldsymbol{X}^{+} = \lim_{\alpha \to 0} (\boldsymbol{X}^{T}\boldsymbol{X} + \alpha \boldsymbol{I})^{-1}\boldsymbol{X}^{T}$$

- Use this for undermined linear equations

# Norm Penalties for Neural Networks

L2 regularization (weight decay) is a common practice

Constraining the norm of each column of the weight matrix of each layer

- Not for the entire matrix
- Prevents any one hidden unit from having very large weights
- A separate $\alpha$ for each column

L1 regularization is only used if having a strong reason

# Parameter Typing or Sharing

Sometimes from knowledge of domain, certain parameters should be close to another

Use norm penalty to encourage this

$$\Omega(\boldsymbol{\theta}) = \left|\left|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\right|\right|_2^2$$

Parameter sharing: force sets of parameters to be equal

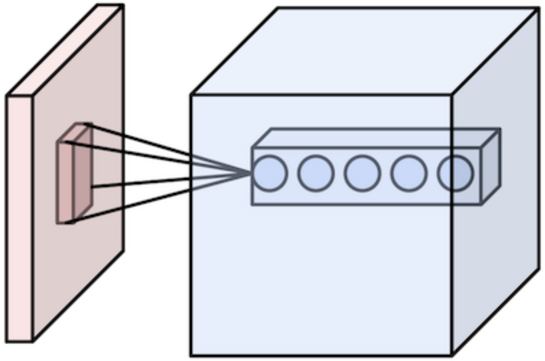
- Popular in practice (especially CNNs)
- Model components share a unique set of parameters

# Parameter Sharing in CNNs

Significantly reduction in memory footprint (i.e. model parameters)

Parameter sharing constraints the neurons in each depth slide to use the same weights

- Total number of weights: 30 x 75 = **2,250**



30 Examples of trained weights of size [5x5x3]

Use the same filter across all spatial location!

# Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

# Data Augmentation

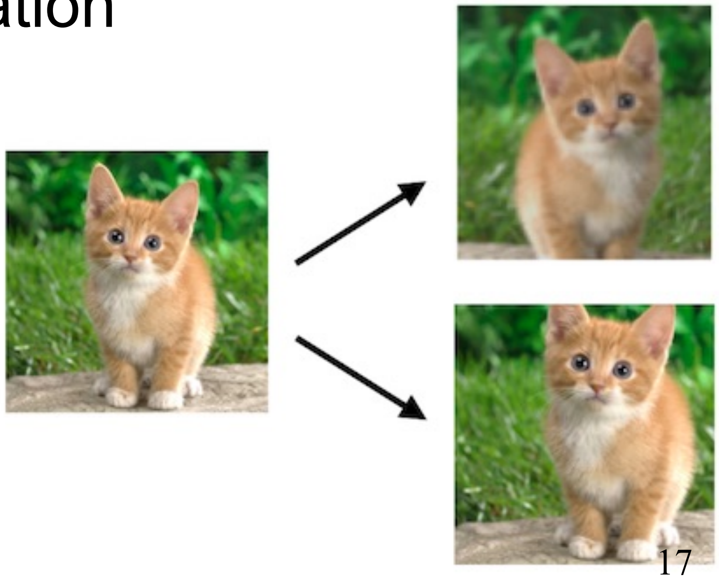
The best way to make ML model generalize better:
More training data

Create fake data from training data!

- For a training sample $(\boldsymbol{x}, y)$, make some transformation $(\boldsymbol{x}', y)$

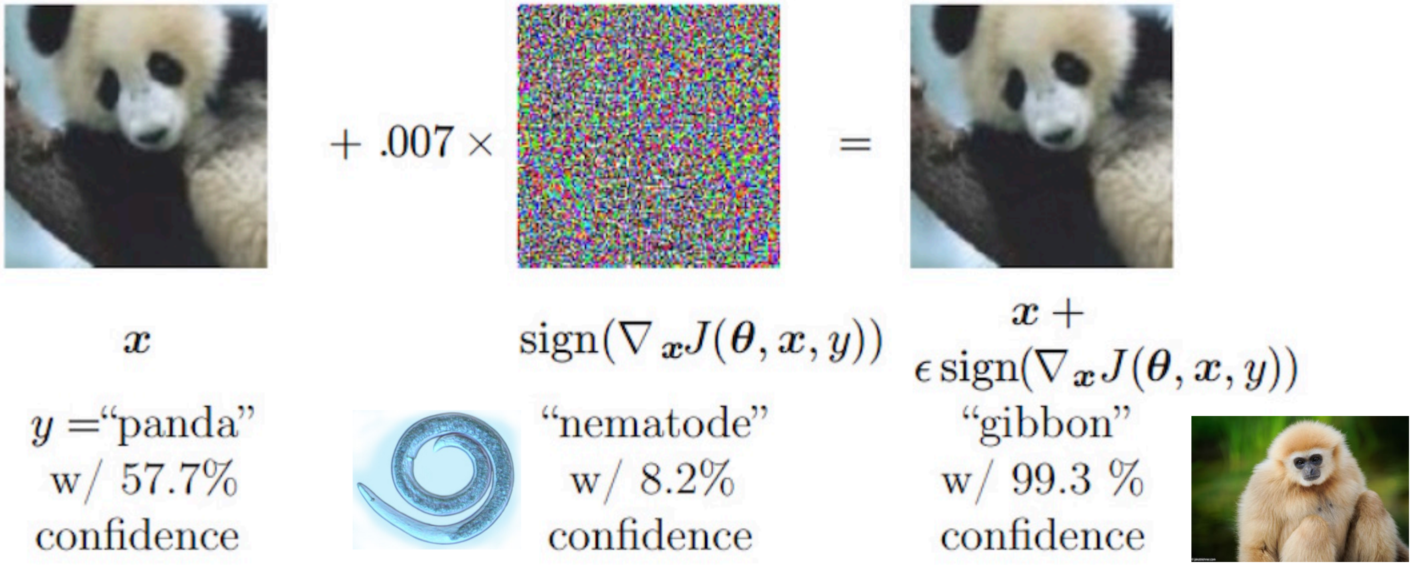
Very effective for image classification

- Images are high-dimensional and have an enormous variety of factors of variation
- (Partial) translating by a few pixels, rotating, scaling, and flipping



# Noise injection

Neural Networks are not robust against noise


$+ .007 \times$
$=$
$\boldsymbol{x}$
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$\boldsymbol{x} + \epsilon\, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda" w/ 57.7% confidence
"nematode" w/ 8.2% confidence
"gibbon" w/ 99.3 % confidence

# Noise injection

Injecting noise in the input can be seen a form of data augmentation

- Highly effective
- Can be seen as regularization

Different ways of noise injection

- A random noise to input
- Noise injection to parameters (or models) (~ Tikhonov regularization)
- Noise at the output targets (e.g. label smoothing)

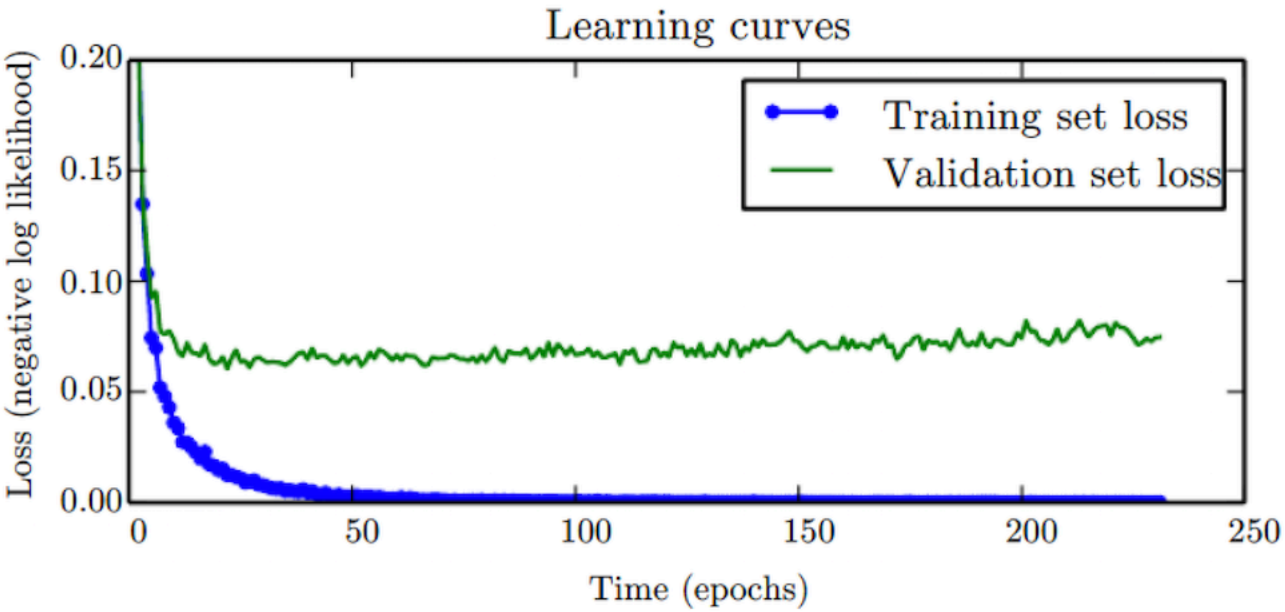
# Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

# Early Stopping

A conventional learning cover (negative log-likelihood)

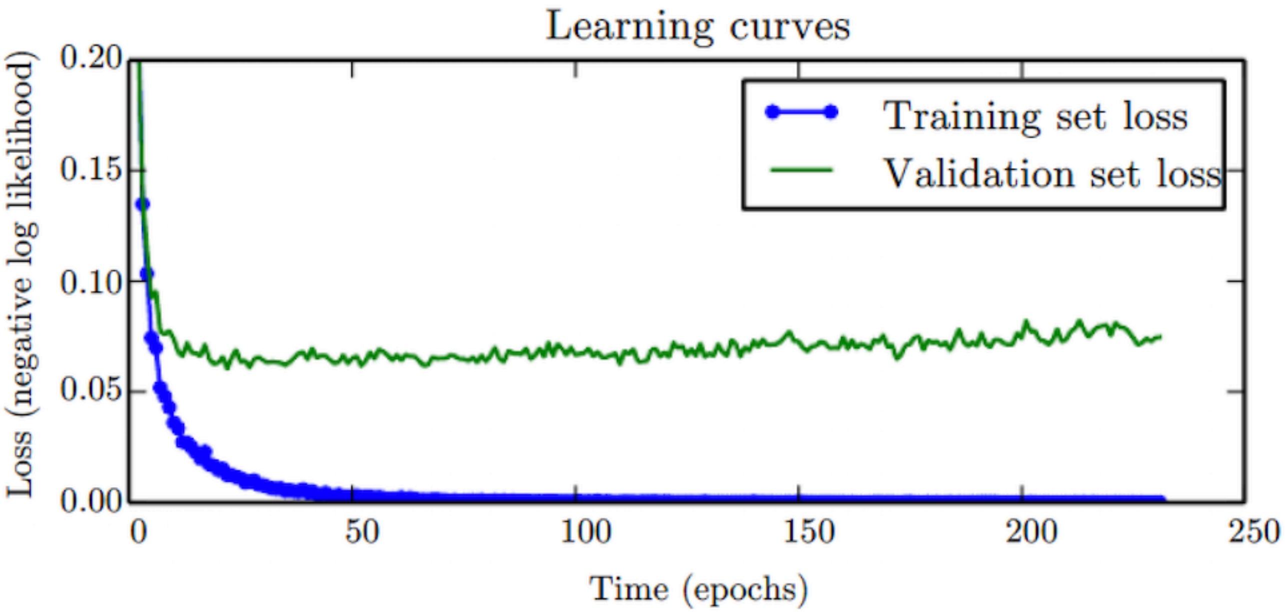
- e.g. Maxout network on MNIST
- The training objective decreases consistently over time
- The validation loss forms an asymmetric U-shaped curve



# Early Stopping

Idea: use the parameters at the minimum validation error

- Maintain a copy of model parameters every iteration
- Additional cost is negligible (+ occasional slow writes)
- A kind of hyperparameter selection algorithm (training time)



# Properties of Early Stopping

A very unobstrusive form of regularization

- No change in training procedure, objective, etc.
- Can be used jointly with other regularization

Another strategy for using all of the training data

- Early stopping requires a validation set (could seem wasteful)
- Learning the model with training data w/o validation set first + Continue training using all data until the validation loss falls below the training loss

Regularization effect (decreasing generalization error) + reduction of training time

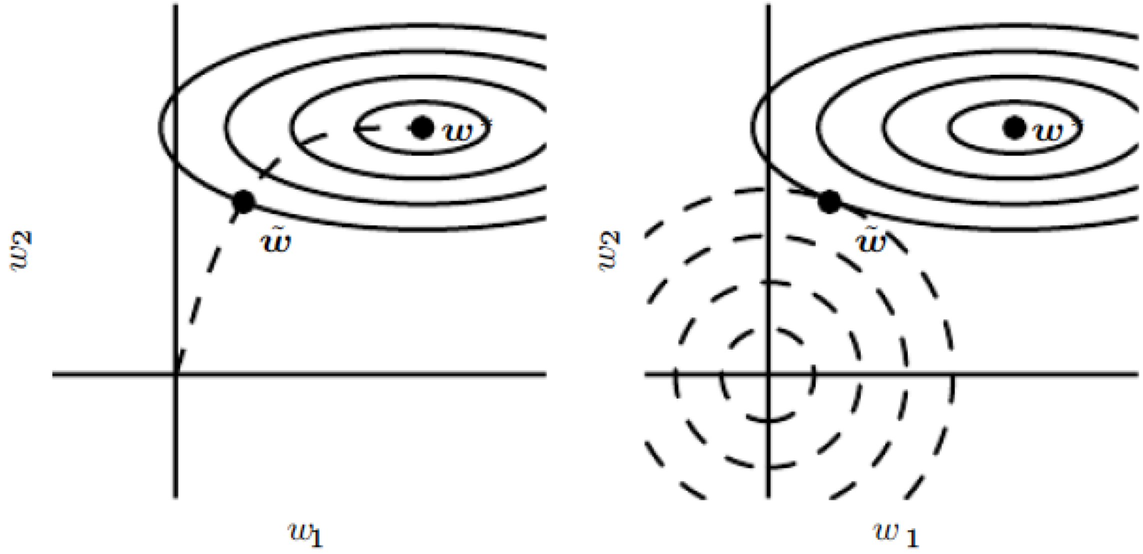
# Early Stopping as a Regularizer

Restrict an optimization procedure to a relatively small volume of parameter space from initial parameter $\boldsymbol{\theta}_0$

- A SGD with a learning rate $\epsilon$ and $\tau$ training iterations
- $\epsilon\tau$: the volume of parameter space reachable from $\boldsymbol{\theta}_0$

The trajectory taken by SGD beginning from the origin

- Stop at $\tilde{\boldsymbol{w}}$ rather than stopping real minimum $\boldsymbol{w}^*$



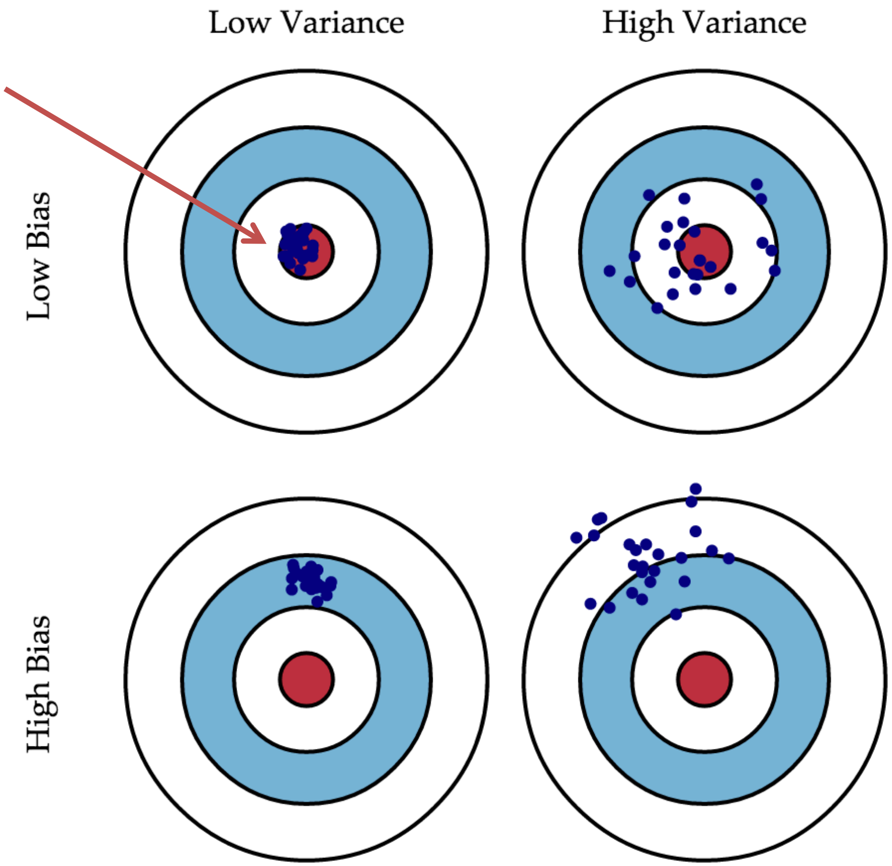
# Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

# Bias and Variance

Two sources of error in an estimator: bias and variance

(Test Error) = (Bias) + (Variance)



Correct prediction

Low bias: predicted well

Low variance: Stable

# Bagging (<u>B</u>ootstrap <u>Agg</u>regat<u>ing</u>)

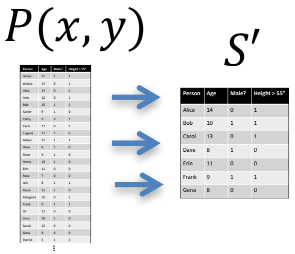
Goal: reduce variance

Variance reduces linearly
Bias unchanged

Expected Error = **Variance** + Bias

Ideal setting: many training sets $S'$

- Train model using each $S'$
- Average predictions


$P(x,y)$

| Person | Age | Male? | Height > 55" |
|---|---|---|---|
| James | 11 | 1 | 1 |
| Jessica | 14 | 0 | 1 |
| Alice | 14 | 0 | 1 |
| Amy | 12 | 0 | 1 |
| Bob | 10 | 1 | 1 |
| Xavier | 9 | 1 | 0 |
| Cathy | 9 | 0 | 1 |
| Carol | 13 | 0 | 1 |
| Eugene | 13 | 1 | 0 |
| Rafael | 12 | 1 | 1 |
| Dave | 8 | 1 | 0 |
| Peter | 9 | 1 | 0 |
| Henry | 13 | 1 | 0 |
| Erin | 11 | 0 | 0 |
| Rose | 7 | 0 | 0 |
| Iain | 8 | 1 | 1 |
| Paulo | 12 | 1 | 0 |
| Margaret | 10 | 0 | 1 |
| Frank | 9 | 1 | 1 |
| Jill | 13 | 0 | 0 |
| Leon | 10 | 1 | 0 |
| Sarah | 12 | 0 | 0 |
| Gena | 8 | 0 | 0 |
| Patrick | 5 | 1 | 1 |
| ⋮ | | | |

$S'$

| Person | Age | Male? | Height > 55" |
|---|---|---|---|
| Alice | 14 | 0 | 1 |
| Bob | 10 | 1 | 1 |
| Carol | 13 | 0 | 1 |
| Dave | 8 | 1 | 0 |
| Erin | 11 | 0 | 0 |
| Frank | 9 | 1 | 1 |
| Gena | 8 | 0 | 0 |

sampled independently

Bootstrapping

In practice: Resample $S'$ with replacement ($|S'| = S$)

- cf. Jackknife: Given a sample of size $N$, generate $N - 1$ sets by ignoring one observation at each time

"Bagging Predictors" [Leo Breiman, 1994]
http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

# Bagging (Bootstrap Aggregation)

**Given:** Training Set $S$

**Bagging:** Generate many bootstrap samples $S'$

Repeat $M$ times

- Sampled with replacement from $S$ ($|S'| = S$)
- Train minimally regularized classifier (ex. DT) on $S'$

**Final predictor**: Combine $M$ predictors

- Voting for classification problems
- Averaging for estimation problems
- Averaging reduces variance

# Ensemble Methods for Neural Networks

Highly recommended!

- Model averaging is an extremely powerful and reliable for reducing generalization error

NNs reach a wide variety of solution points

- Never obtain the same solution at each running
- random initialization, random selection of minibatches, different hyperparameter setting, …

# Outline

- Parameter Norm Penalties
  - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

# Dropout

A method of making bagging practical for ensembles of very many large NNs

- An inexpensive approximation to training and evaluating a bagged ensemble of exponentially many NNs

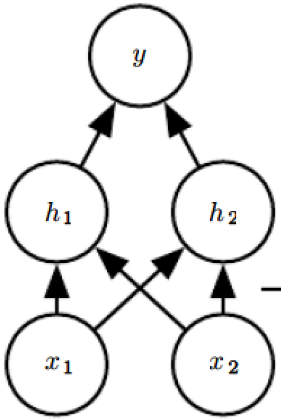
Suppose that we have a big NN model and optimize it with a minibatch-based learning algorithm (e.g. SGD)

- For each minibatch, randomly sample a different binary mask to apply to all of the input and hidden units in the network
- In other words, randomly drop the output of units to zero
- e.g. dropout rate 0.5 for hidden units and 0.2 for input nodes
- Then use learning algorithm as usual

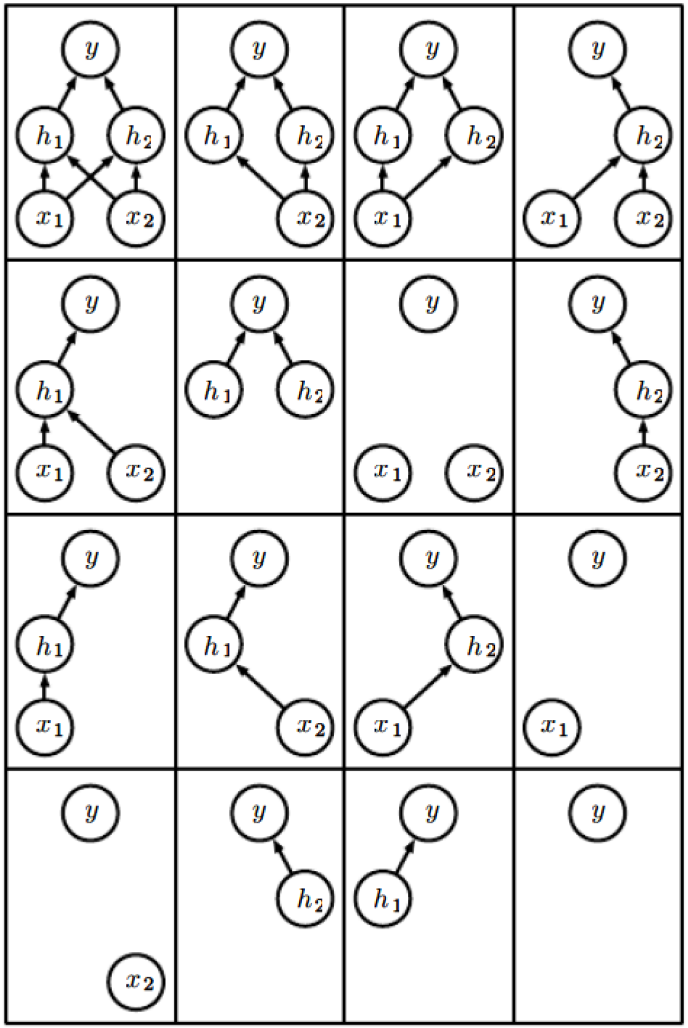
# Dropout

A base network with two input and two hidden units

16 possible subsets



Base network



Ensemble of Sub-Networks

Ignore the ones with no input or no paths to output

# Properties of Dropout

Comparison with Bagging

- In dropout, the models share parameters
- With each minibatch, a different subset of parameter is updated
- In bagging, the models are all independent

Dropout is very effective and highly recommended

- More effective than other regularizers
  (e.g. weight decay, sparsity)

Computationally very cheap

No limitation on model types or training procedures (e.g. CNNs, RNNs, and RBMs)

# Properties of Dropout

Increase the size of the model to offset the regularizer

- Dropout reduces the effective capacity of a model

Do not use Dropout with very few training examples

Key insights

- Training a network with stochastic behavior and making predictions by averaging over multiple stochastic decisions
- Dropout power arises from that masking noise is applied to hidden units

Other relatives: dropconnect, batch normalization

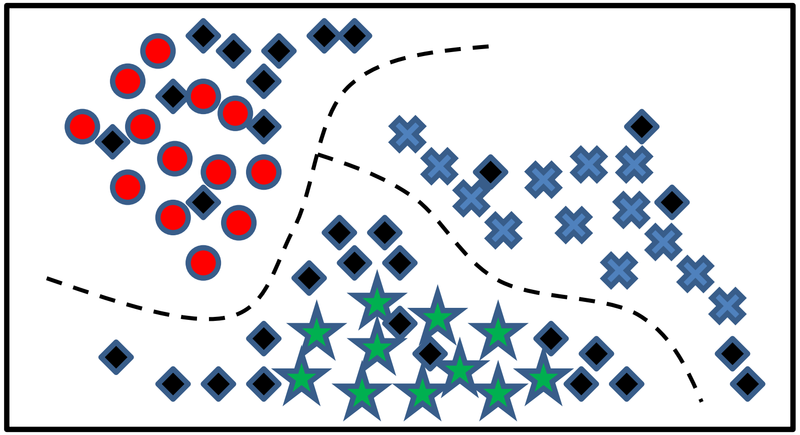
# Outline

- Parameter Norm Penalties
    - L2/L1 Regularization
- Data augmentation
- Early Stopping
- Ensemble Methods
- Dropout
- Meta-learning Frameworks

# Semi-supervised Learning

Use unlabeled data to augment a small labeled sample to improve learning

- Labeled data can be rare or expensive, while unlabeled data is much cheaper
- Model the generative process too
- One of active research areas



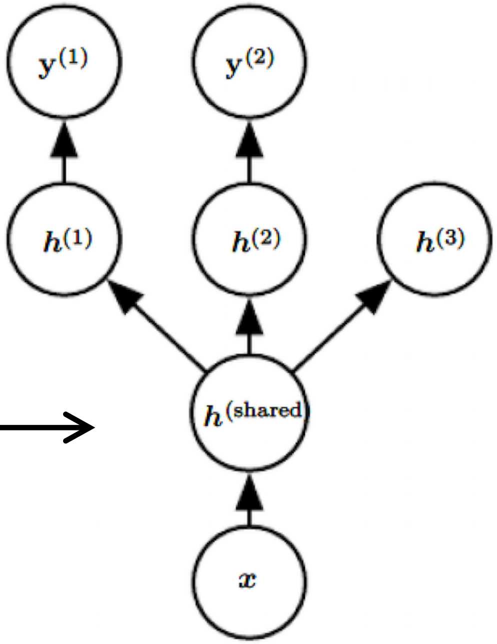
Semi-supervised learning

# Multi-task Learning

Learn multiple-related problems together at the same time

- e.g. Image classification and object detection

Two different supervised tasks (predicting $\boldsymbol{y}^{(i)}$), while sharing input $\boldsymbol{x}$ and some mid-level representation $\boldsymbol{h}^{(\text{shared})}$



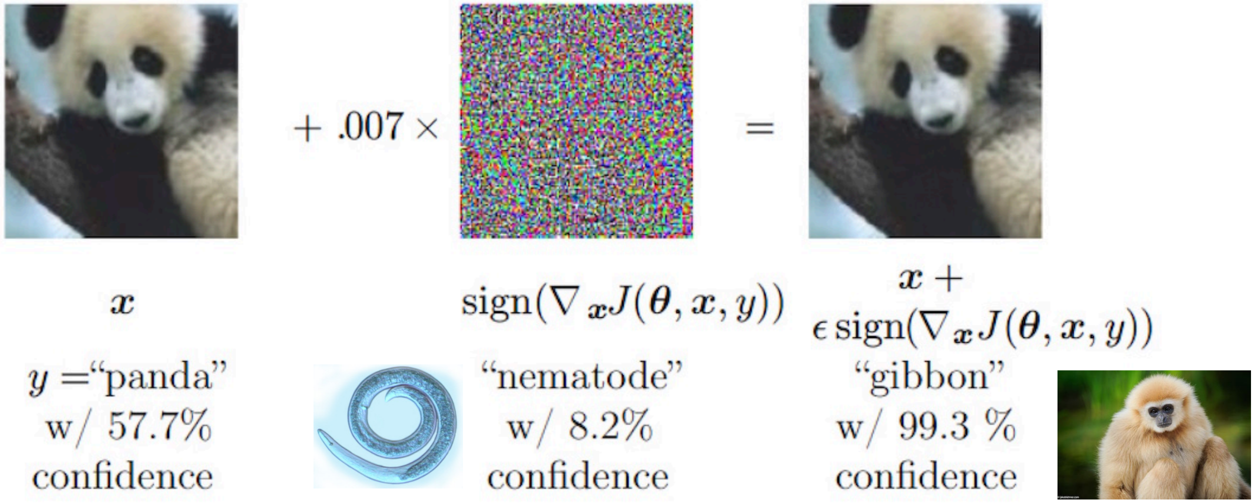
Task-specific parameters

Generic parameters

# Adversarial Training

Adversarial examples

- Human cannot tell the difference with the original example
- However, the network can make highly different predictions



Adversarial training: training on adversarially perturbed examples

- Purely linear models do not resist them