Setup and Regularization for training

Most of this material is from Prof. Andrew Ng'and Chang's slides.

Contents

- Setup for training
- Bias & Variance
- Regularization for training effectiveness

Applied ML is a highly iterative process

- Making good choices in how you set up your training, development, and test sets can make a huge difference for you to find a high performance neural network quickly

Applied ML is a highly iterative process

- When training a neural network, you have to make a lot of decisions:
  - # of layers
  - # of hidden units
  - learning rates
  - activation functions
  - ...

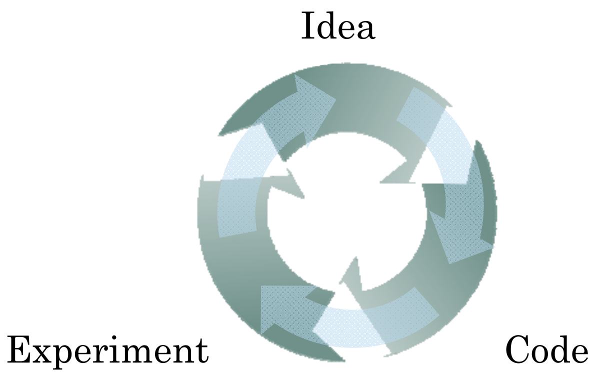
Applied ML is a highly iterative process

- When training a neural network you have to make a lot of decisions:
  - # of layers
  - # of hidden units
  - learning rates
  - activation functions
  - ...
- It is almost impossible to correctly guess the right values for hyperparameters

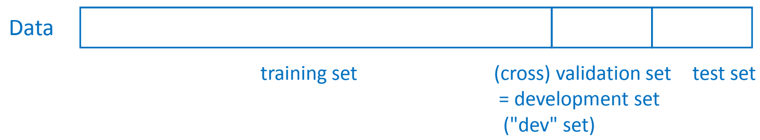
Applied ML is a highly iterative process

- When training a neural network you have to make a lot of decisions:
  - # of layers
  - # of hidden units
  - learning rates
  - activation functions
  - ...
- It is almost impossible to correctly guess the right values for hyperparameters
- Therefore, in practice, applied machine learning is a highly iterative process

Applied ML is a highly iterative process



Train/Dev/Test Sets



Train/Dev/Test Sets

Data

training set | - hold-out cross validation set - development set ("dev" set) | test set

rule of thumb for splitting data

Previous era (# of samples ≒ 100 or 1,000 or 10,000) :

training/test = 70/30 % training/dev/test = 60/20/20 %

Train/Dev/Test Sets

Data

training set | - hold-out cross validation set - development set ("dev" set) | test set

rule of thumb for splitting data

Previous era (# of samples ≒ 100 or 1,000 or 10,000) :

training/test = 70/30 % training/dev/test = 60/20/20 %

Big data era (# of samples ≒ 1,000,000) :

training/dev/test = 99/1/1 %

Mismatched train/test distribution

Goal: Cat Detector

Training set:
(high-density) cat pictures
from webpages

Dev/test sets:
(low-density) cat pictures
from users using your app

Mismatched train/test distribution

Goal: Cat Detector

Training set:
(high-density) cat pictures
from webpages

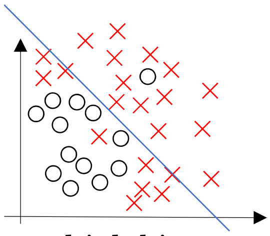
Dev/test sets:
(low-density) cat pictures
from users using your app

rule of thumb for splitting data:

- Make sure dev and test sets come from same distribution
- Even Not having a test set might be okay (only having dev set) for better training

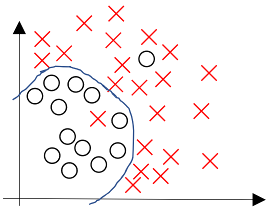
Bias and Variance

- Bias and variance tradeoff is a very important problem in traditional machine learning
- But, in deep learning era, it's importance has been somewhat reduced

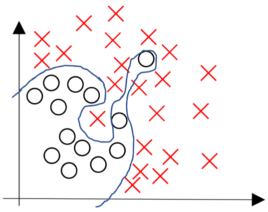


high bias

- underfitting
- too simple classifier



"just right"



high variance

- overfitting
- too complex classifier

Bias and Variance

Cat classification

y = 1

y = 0

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias & low variance

Human's classification error ≈ 0%

Bias and Variance

Cat classification

y = 1

y = 0

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	~~high bias~~	high bias & high variance	low bias & low variance

- If human's error ≈ 15%, then the second case does not have the high bias problem

Bias and Variance

Cat classification

y = 1

y = 0

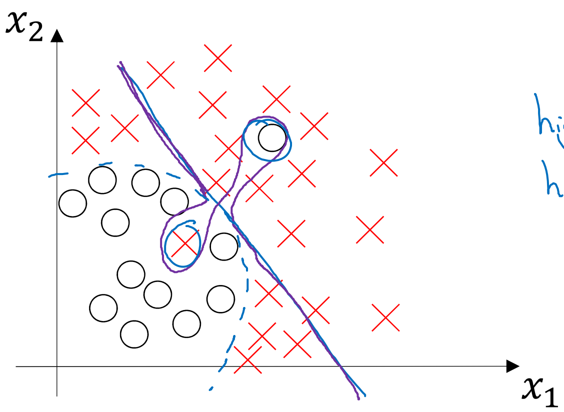
bias

Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	high variance	high bias	high bias & high variance	low bias & low variance

variance
= DS error – TS error

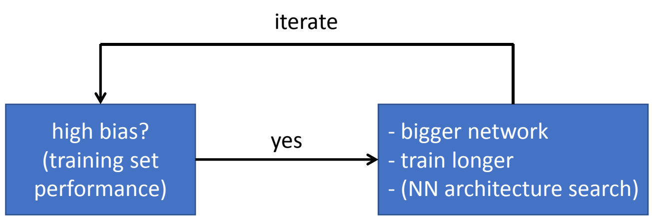
Human's classification error ≈ 0%

High bias and high variance

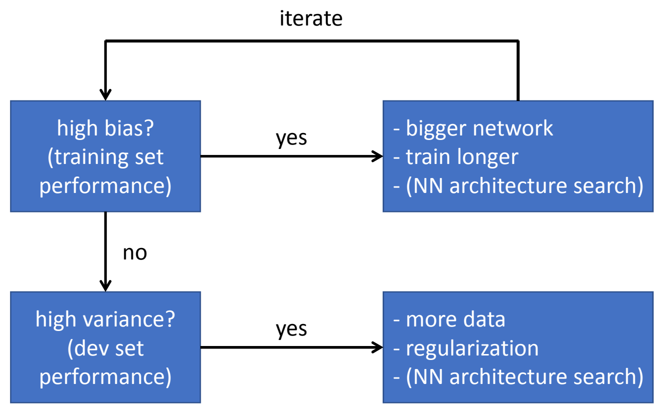


high bias
high varian

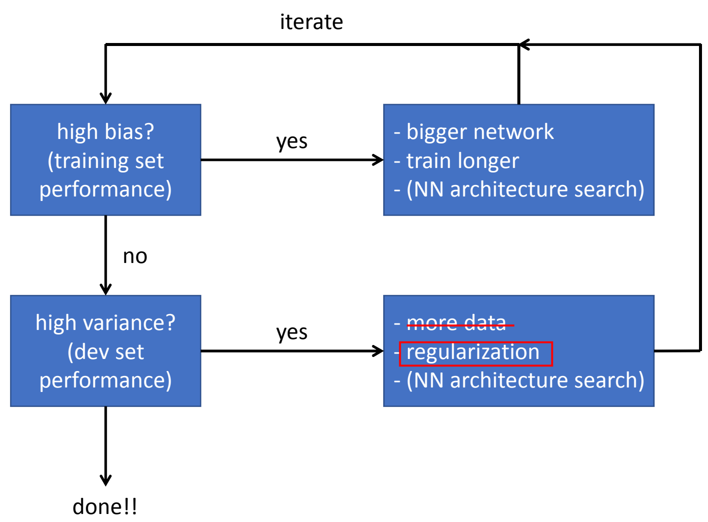
Basic recipe for machine learning



Basic recipe for machine learning



Basic recipe for machine learning



Regularization

- Regularization helps to prevent overfitting, or to reduce the variance of your network

Regularization for Logistic Regression

$$\min_{w,b} J(w,b) \qquad w \in \mathbb{R}^n, b \in \mathbb{R}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\big(\hat{y}^{(i)}, y^{(i)}\big)$$

Regularization for Logistic Regression

$$\min_{w,b} J(w,b) \qquad w \in \mathbb{R}^n,\ b \in \mathbb{R}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\|_2^2 \quad \cancel{+\frac{\lambda}{2m}b^2}$$

usually omitted

L_2 regularization: $\|w\|_2^2 = \sum_{j=1}^{n} w_j^2 = w^T w$

Regularization for Logistic Regression

$$\min_{w,b} J(w,b) \qquad w \in \mathbb{R}^n,\ b \in \mathbb{R}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\|_2^2 \quad \cancel{+\frac{\lambda}{2m}b^2}$$

usually omitted

L_2 regularization: $\|w\|_2^2 = \sum_{j=1}^{n} w_j^2 = w^T w$

L_1 regularization: $\|w\|_1 = \sum_{j=1}^{n} |w_j|$ $\longrightarrow$ sparse w

Regularization for Logistic Regression

$$\min_{w,b} J(w,b) \qquad w \in \mathbb{R}^n,\, b \in \mathbb{R}$$

λ: regularization parameter

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\|w\|_2^2 \cancel{+ \frac{\lambda}{2m} b^2}$$

usually omitted

L2 regularization: $\|w\|_2^2 = \sum_{j=1}^{n} w_j^2 = w^T w$

L1 regularization: $\|w\|_1 = \sum_{j=1}^{n} |w_j| \longrightarrow$ sparse w

Regularization for Neural Network

$$J\left(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}\right) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\|W^{[l]}\right\|_F^2$$

Frobenius norm: $\left\|W^{[l]}\right\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} \left(w_{ij}^{[l]}\right)^2 \qquad W^{[l]}: \left(n^{[l]}, n^{[l-1]}\right)$

Regularization for Neural Network

$$J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\|W^{[l]}\right\|_F^2$$

Frobenius norm: $\left\|W^{[l]}\right\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} \left(w_{ij}^{[l]}\right)^2$ $\quad W^{[l]}: (n^{[l]}, n^{[l-1]})$

Gradient descent: $dW^{[l]} = \text{(from backprop)}$

$W^{[l]} := W^{[l]} - \alpha \cdot dW^{[l]}$ $\quad \left(dW^{[l]} = \frac{\partial J}{\partial W^{[l]}}\right)$

Regularization for Neural Network

$$J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\|W^{[l]}\right\|_F^2$$

Frobenius norm: $\left\|W^{[l]}\right\|_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} \left(w_{ij}^{[l]}\right)^2$ $\quad W^{[l]}: (n^{[l]}, n^{[l-1]})$

Gradient descent: $dW^{[l]} = \text{(from backprop)} + \frac{\lambda}{m} W^{[l]}$ $\quad \left(\because \frac{\partial \left\|W^{[l]}\right\|_F^2}{\partial W^{[l]}} = 2W^{[l]}\right)$

$W^{[l]} := W^{[l]} - \alpha \cdot dW^{[l]}$

Regularization for Neural Network

$$J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|W^{[l]}\|_F^2$$

Frobenius norm: $\|W^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}\left(w_{ij}^{[l]}\right)^2$ $\quad W^{[l]}: (n^{[l]}, n^{[l-1]})$

Gradient descent: $dW^{[l]} = \boxed{(\text{from backprop}) + \frac{\lambda}{m}W^{[l]}}$ $\left(\because \frac{\partial\|W^{[l]}\|_F^2}{\partial W^{[l]}} = 2W^{[l]}\right)$

$W^{[l]} := W^{[l]} - \alpha \cdot dW^{[l]}$

"Weight decay": $W^{[l]} := W^{[l]} - \alpha \cdot \left[(\text{from backprop}) + \frac{\lambda}{m}W^{[l]}\right]$

$= W^{[l]} - \frac{\alpha\lambda}{m}W^{[l]} - \alpha(\text{from backprop})$

$= \left(1 - \frac{\alpha\lambda}{m}\right)W^{[l]} - \alpha(\text{from backprop})$

Regularization for Neural Network

$$J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|W^{[l]}\|_F^2$$

Frobenius norm: $\|W^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}\left(w_{ij}^{[l]}\right)^2$ $\quad W^{[l]}: (n^{[l]}, n^{[l-1]})$

Gradient descent: $dW^{[l]} = \boxed{(\text{from backprop}) + \frac{\lambda}{m}W^{[l]}}$ $\left(\because \frac{\partial\|W^{[l]}\|_F^2}{\partial W^{[l]}} = 2W^{[l]}\right)$

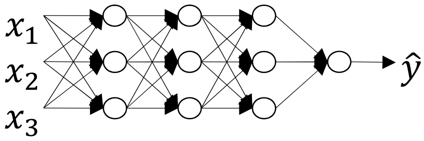
$W^{[l]} := W^{[l]} - \alpha \cdot dW^{[l]}$

"Weight decay": $W^{[l]} := W^{[l]} - \alpha \cdot \left[(\text{from backprop}) + \frac{\lambda}{m}W^{[l]}\right]$

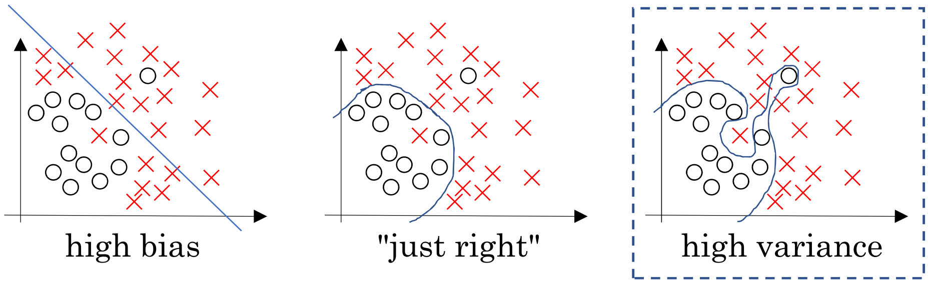
$= W^{[l]} - \frac{\alpha\lambda}{m}W^{[l]} - \alpha(\text{from backprop})$

$= \underbrace{\boxed{\left(1 - \frac{\alpha\lambda}{m}\right)}}_{<1}W^{[l]} - \alpha(\text{from backprop})$

How does regularization prevent overfitting?

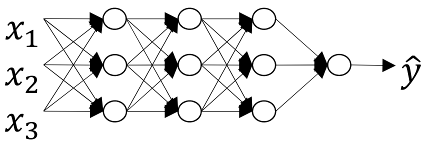


$$J(\ldots) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

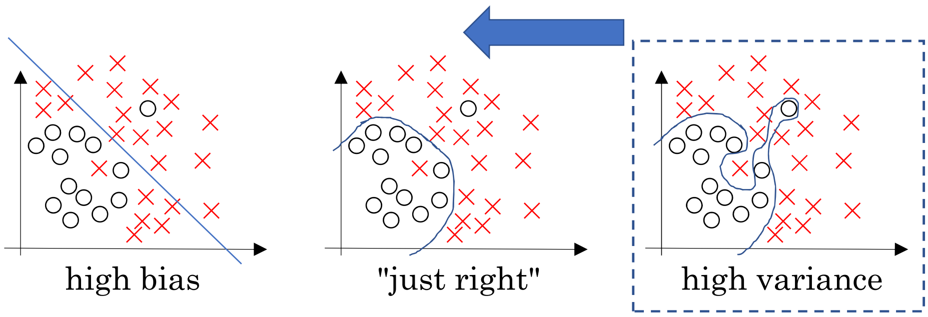


Let's assume overfitting

How does regularization prevent overfitting?

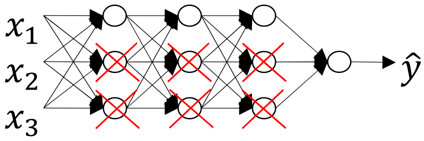


$$J(\ldots) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|W^{[l]}\|_F^2$$



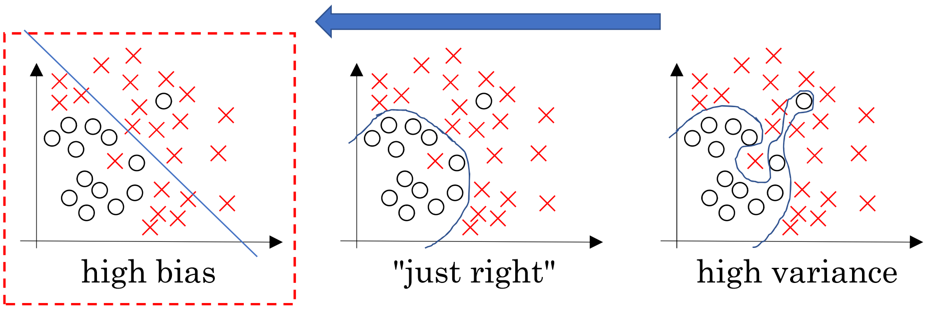
Let's assume overfitting

How does regularization prevent overfitting?

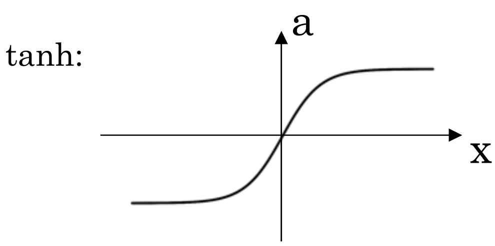


$$J(\dots) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|W^{[l]}\|_F^2$$

very large $\lambda \rightarrow W^{[l]} \approx spase\ matrix$
(very simple model!)

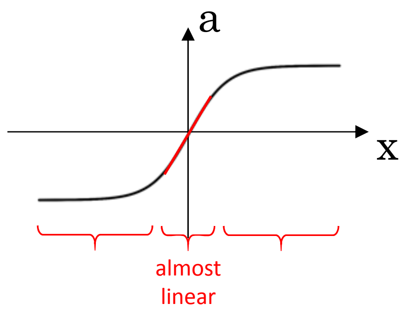


How does regularization prevent overfitting?



How does regularization prevent overfitting?

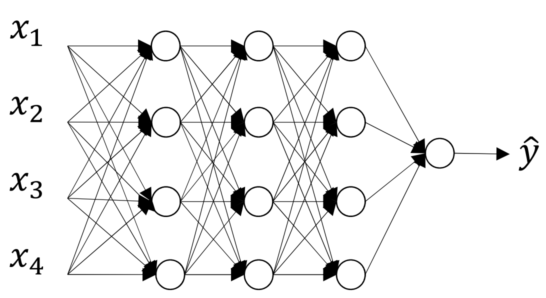
tanh:



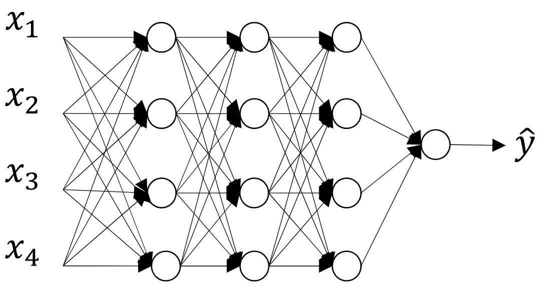
$\lambda \uparrow \Rightarrow |W^{[l]}{}_{ij}| \downarrow \Rightarrow |z^{[l]}| \downarrow \quad (\because z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]})$

$\Rightarrow$ every layers ≈ get close to linear ≈ get simple

Dropout Regularization

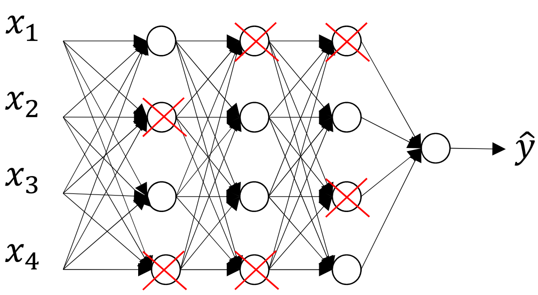


Dropout Regularization



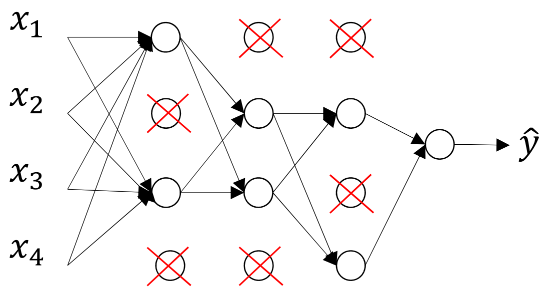
keeping prob = 0.5

Dropout Regularization



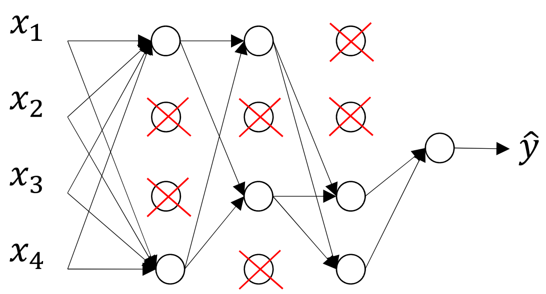
keeping prob = 0.5

Dropout Regularization



keeping prob = 0.5

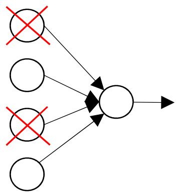
Dropout Regularization



keeping prob = 0.5

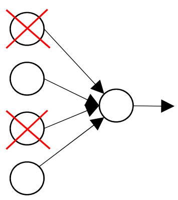
Why does dropout work?

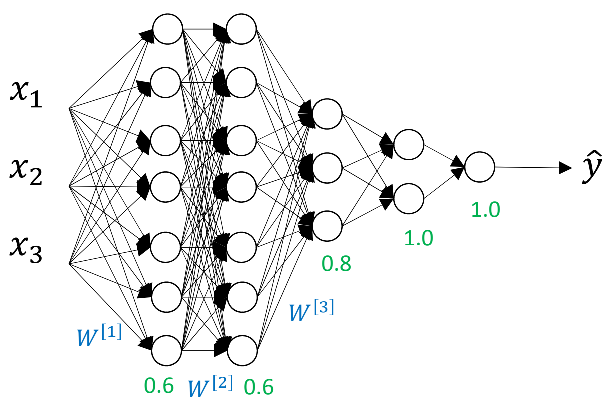
Intuition: Can't rely on any one feature, so have to spread out weights.



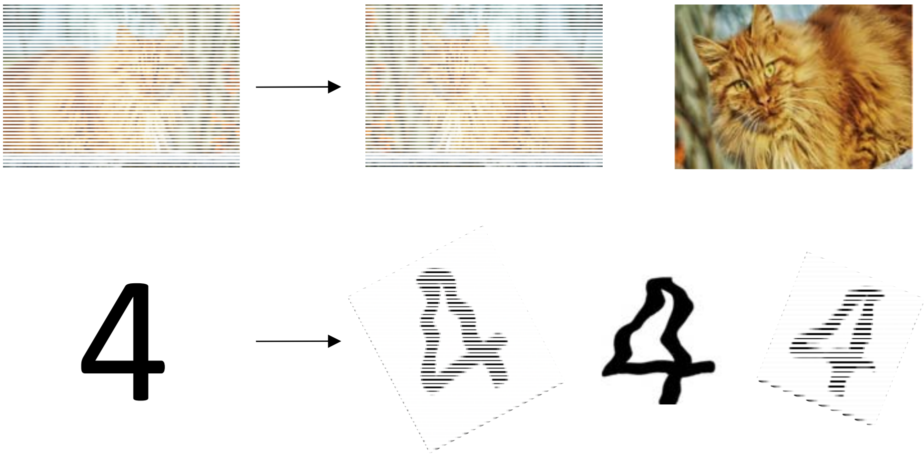
Why does dropout work?

Intuition: Can't rely on any one feature, so have to spread out weights.





Data augmentation



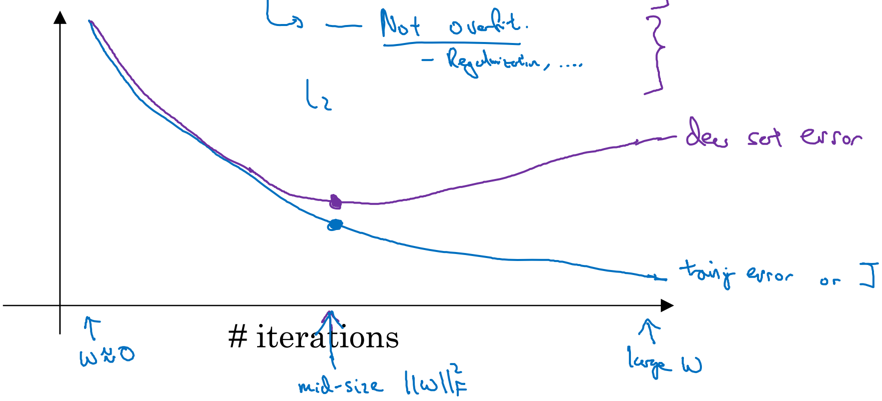
Early stopping

Orthogonalization.

- — Optimize cost function J
  - Gradient, ...
- — Not overfit.
  - Regularization, ...

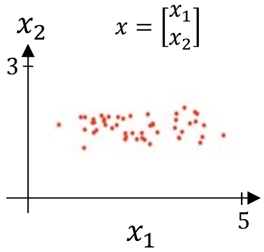
L_2

$J(w,b)$



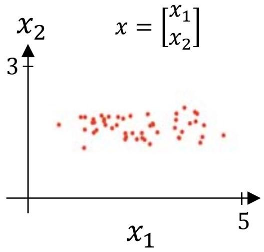
Normalizing training sets

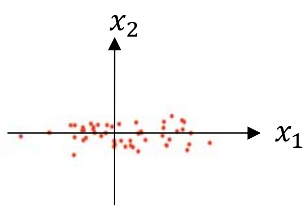
$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



Normalizing training sets

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$



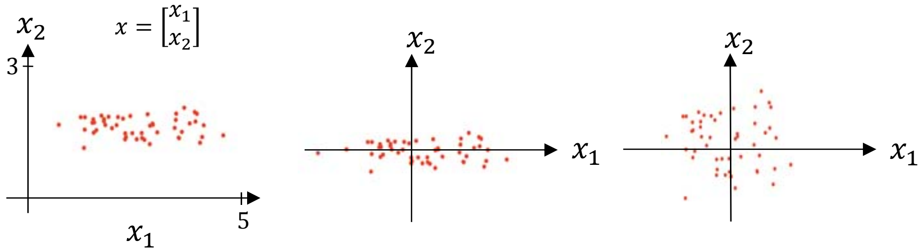


1. subtract mean:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$x^{(i)} := x^{(i)} - \mu$$

Normalizing training sets



1. subtract mean:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

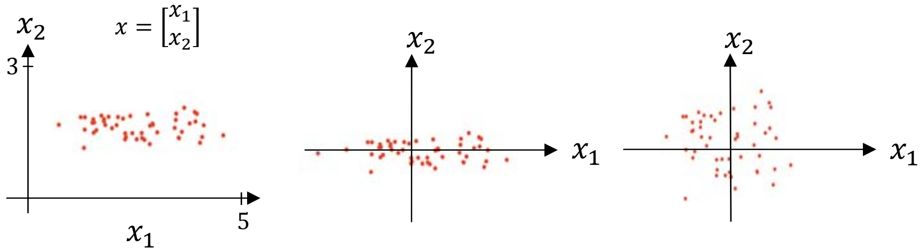
$$x^{(i)} := x^{(i)} - \mu$$

2. normalize variance:

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} \left\{x_j^{(i)}\right\}^2$$

$$x_j^{(i)} := x_j^{(i)} / \sigma_j$$

Normalizing training sets



1. subtract mean:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$x^{(i)} := x^{(i)} - \mu$$

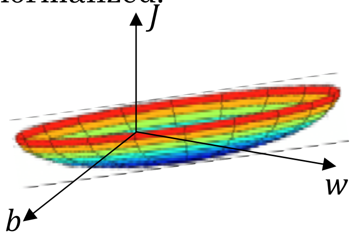
2. normalize variance:

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} \left\{x_j^{(i)}\right\}^2$$

$$x_j^{(i)} := x_j^{(i)} / \sigma_j$$

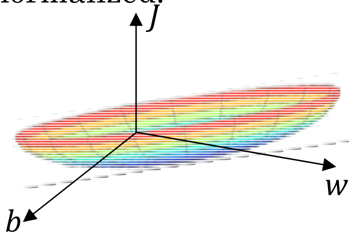
Why normalize inputs?

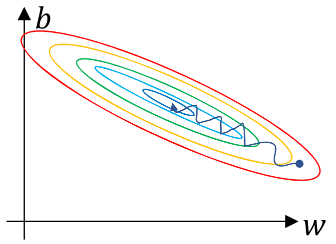
Unnormalized:



Why normalize inputs?

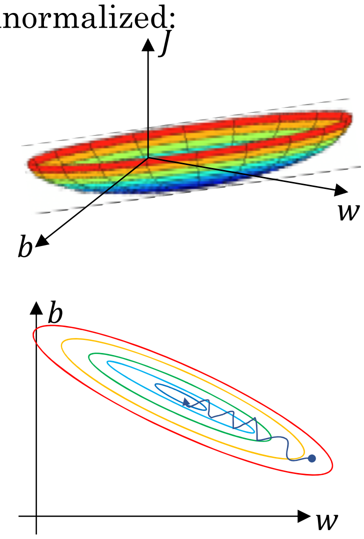
Unnormalized:



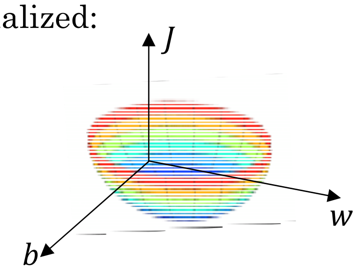


Why normalize inputs?

Unnormalized:

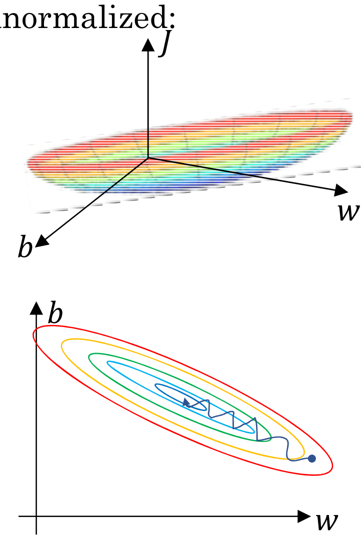


Normalized:

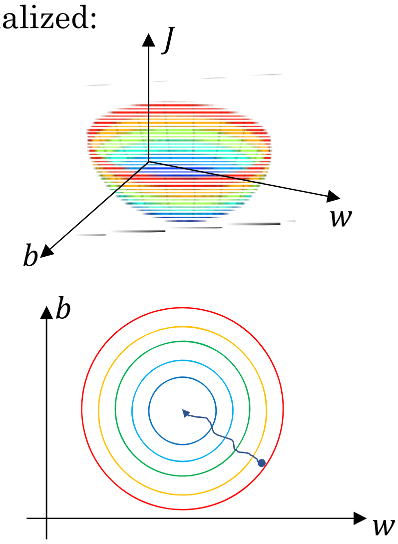


Why normalize inputs?

Unnormalized:

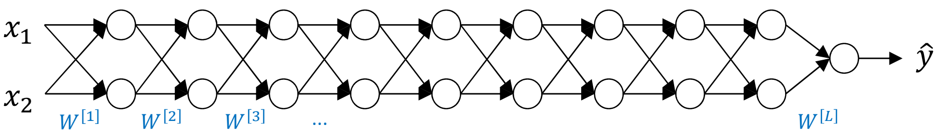


Normalized:



Vanishing/exploding gradients

Vanishing/exploding gradients

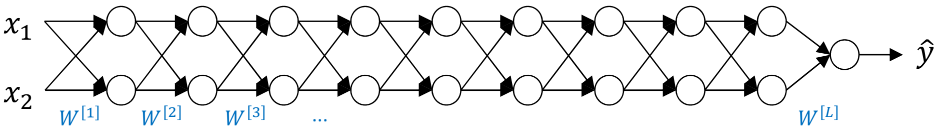


$g(z) = z \qquad b^{[l]} = 0$

$\hat{y} = W^{[L]} W^{[L-1]} W^{[L-2]} \ldots W^{[3]} W^{[2]} W^{[1]} x$

$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \Rightarrow \hat{y} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^L x = \begin{bmatrix} 1.5^L & 0 \\ 0 & 1.5^L \end{bmatrix} x \Rightarrow$ activations increase exponentially!! "Explode"

Vanishing/exploding gradients



$g(z) = z \qquad b^{[l]} = 0$

$\hat{y} = W^{[L]}W^{[L-1]}W^{[L-2]} \ldots W^{[3]}W^{[2]}W^{[1]}x$

$$W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \Rightarrow \hat{y} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^L x = \begin{bmatrix} 1.5^L & 0 \\ 0 & 1.5^L \end{bmatrix} x \Rightarrow$$

activations increase exponentially!!

$$W^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \Rightarrow \hat{y} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^L x = \begin{bmatrix} 0.5^L & 0 \\ 0 & 0.5^L \end{bmatrix} x \Rightarrow$$

activations decrease exponentially!!
"Vanish"

Vanishing/exploding gradients

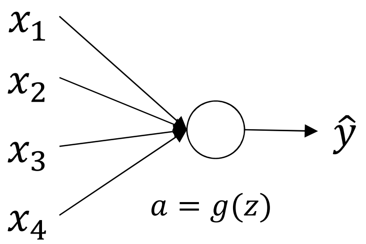
$*$ Remember :

$$\left.\begin{aligned} dz^{[l]} &= da^{[l]} * g^{[l]\prime}(z^{[l]}) \\ dW^{[l]} &= dz^{[l]} \cdot a^{[l-1]T} \\ db^{[l]} &= dz^{[l]} \\ da^{[l-1]} &= W^{[l]T} \cdot dz^{[l]} \end{aligned}\right\} \text{recursive definition !}$$

$$\therefore dW^{[l]} = W^{[l+1]T} \cdot W^{[l+2]T} \cdot \ldots \cdot W^{[L]T} \cdot g^{[l]\prime}(z^{[l]}) * g^{[l+1]\prime}(z^{[l+1]}) * \ldots * g^{[L]\prime}(z^{[L]}) \ldots$$

$$db^{[l]} = \underbrace{\underbrace{W^{[l+1]T} \cdot W^{[l+2]T} \cdot \ldots \cdot W^{[L]T}}_{\text{Weight Matrix Multiplication}} \cdot \underbrace{g^{[l]\prime}(z^{[l]}) * g^{[l+1]\prime}(z^{[l+1]}) * \ldots * g^{[L]\prime}(z^{[L]})}_{\text{Activation Function Derivative Multiplication}}}_{\text{can vanish or explode gradient ...}} \ldots$$

Weight Variance initialization for deep network

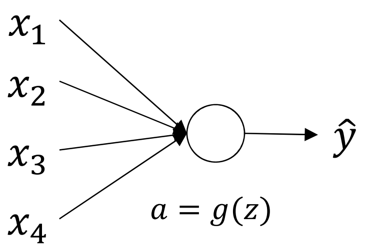


$z = w_1x_1 + w_1x_1 + \cdots + w_nx_n + b$

if large $n \rightarrow$ smaller variance of w_i

$\therefore \mathrm{Var}(w_i) = \frac{1}{n}$ when b exists for general case

Weight Variance initialization for deep network

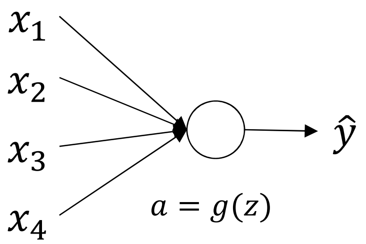


$z = w_1x_1 + w_1x_1 + \cdots + w_nx_n + b$

if large $n \rightarrow$ smaller variance of w_i

$\therefore \mathrm{Var}(w_i) = \frac{2}{n}$ for ReLU activation, which has been claimed by some experiments

Weight Variance initialization for deep network



$z = w_1x_1 + w_1x_1 + \cdots + w_nx_n + b$

if large $n \rightarrow$ smaller variance of w_i

$\therefore \mathrm{Var}(w_i) = \frac{2}{n}$ for ReLU activation

Other variants for tanh activation

$$\mathrm{Var}(w_i) = \frac{1}{n^{[l-1]}}$$

: Xavier initialization

$$\mathrm{Var}(w_i) = \frac{2}{n^{[l-1]} + n^{[l]}}$$

: proposed by Bengio