Chapter 7: Regularization for Deep Learning

PART 1

Deep Learning Textbook Study Group, SF

Safak Ozkan, shafaksan@gmail.com

April 15, 2017

Chapter 7: Regularization for Deep Learning

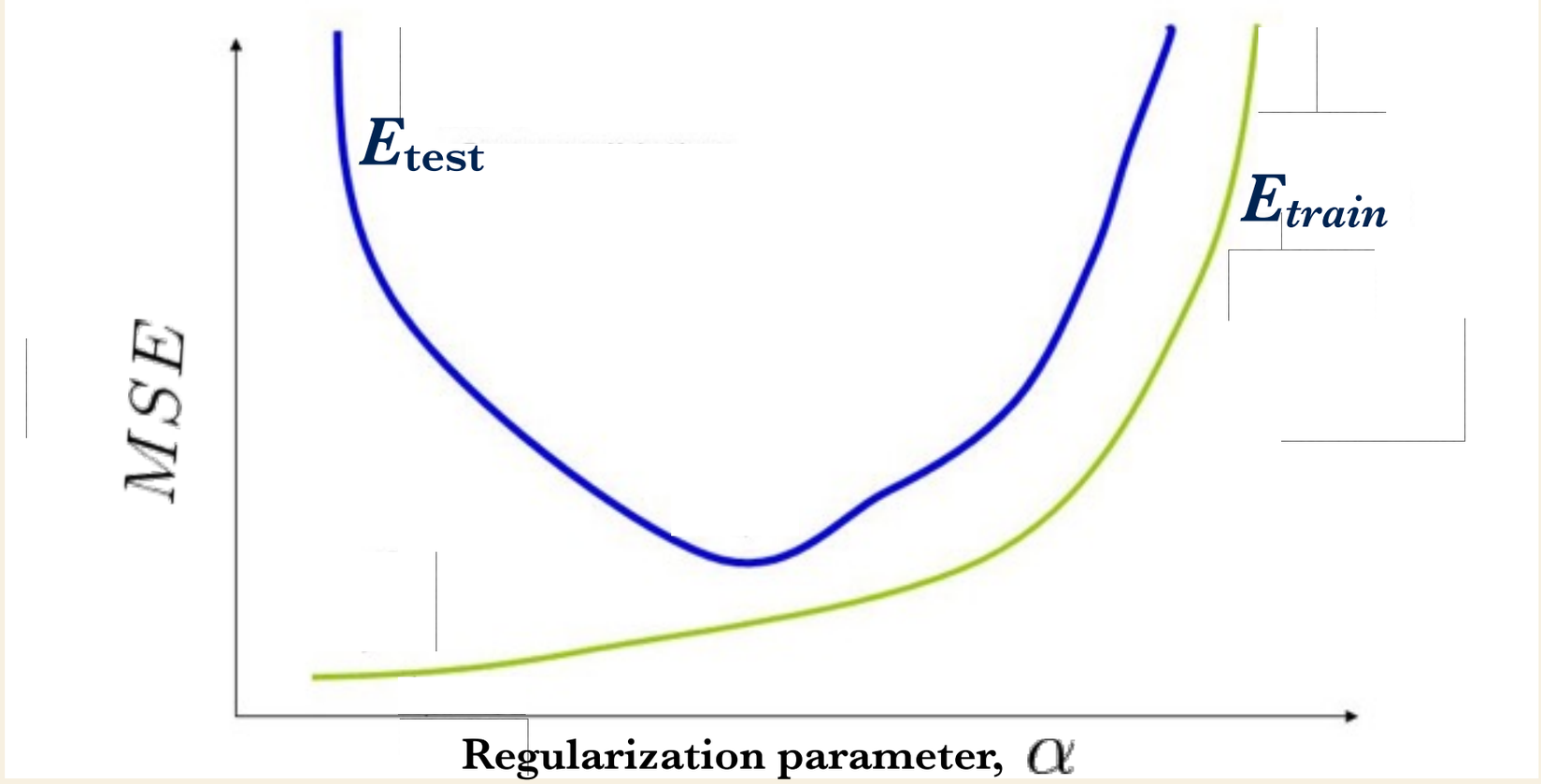
- L^2 Parameter Regularization
- L^1 Parameter Regularization
- Norm Penalties and Constrained Optimization
- Regularization and Under-Constrained Problems
- Dataset Augmentation
- Noise Robustness
- Injecting Noise at Output Targets
- Early Stopping
- Semi Supervised Learning
- Multi-Task Learning
- Parameter Tying and Parameter Sharing
- Bagging and Other Ensemble Methods
- Dropout
- Adversarial Training
- Tangent Distance, Manifold Tangent Classifier

Definition

- **"Regularization** is any modification we make to a learning algorithm that is intended to reduce its **test error** but **NOT** its **training error**."

E_{train} : **Training Error**

E_{test} : **Test Error (or Generalization Error)**



L^2 Regularization

(a.k.a. Weight decay, Tikhonov regularization, Ridge regression)

- **Regularization increases bias and reduces variance.**

$$J_{reg}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{1}{2}\alpha \|\boldsymbol{w}\|^2$$

Regularization parameter

Regularized cost function

Unregularized Cost function

Regularization term (excludes the bias term)

- **Gradient Descent update rule:**

$$\boldsymbol{w} = \boldsymbol{w} - \epsilon\alpha\boldsymbol{w} - \epsilon\nabla J(\boldsymbol{w})$$

Additional term

L^2 Regularization

- **Lagrangian Constrained Optimization**

$$J_{reg}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{1}{2}\alpha \|\boldsymbol{w}\|^2$$

Lagrangian multiplier

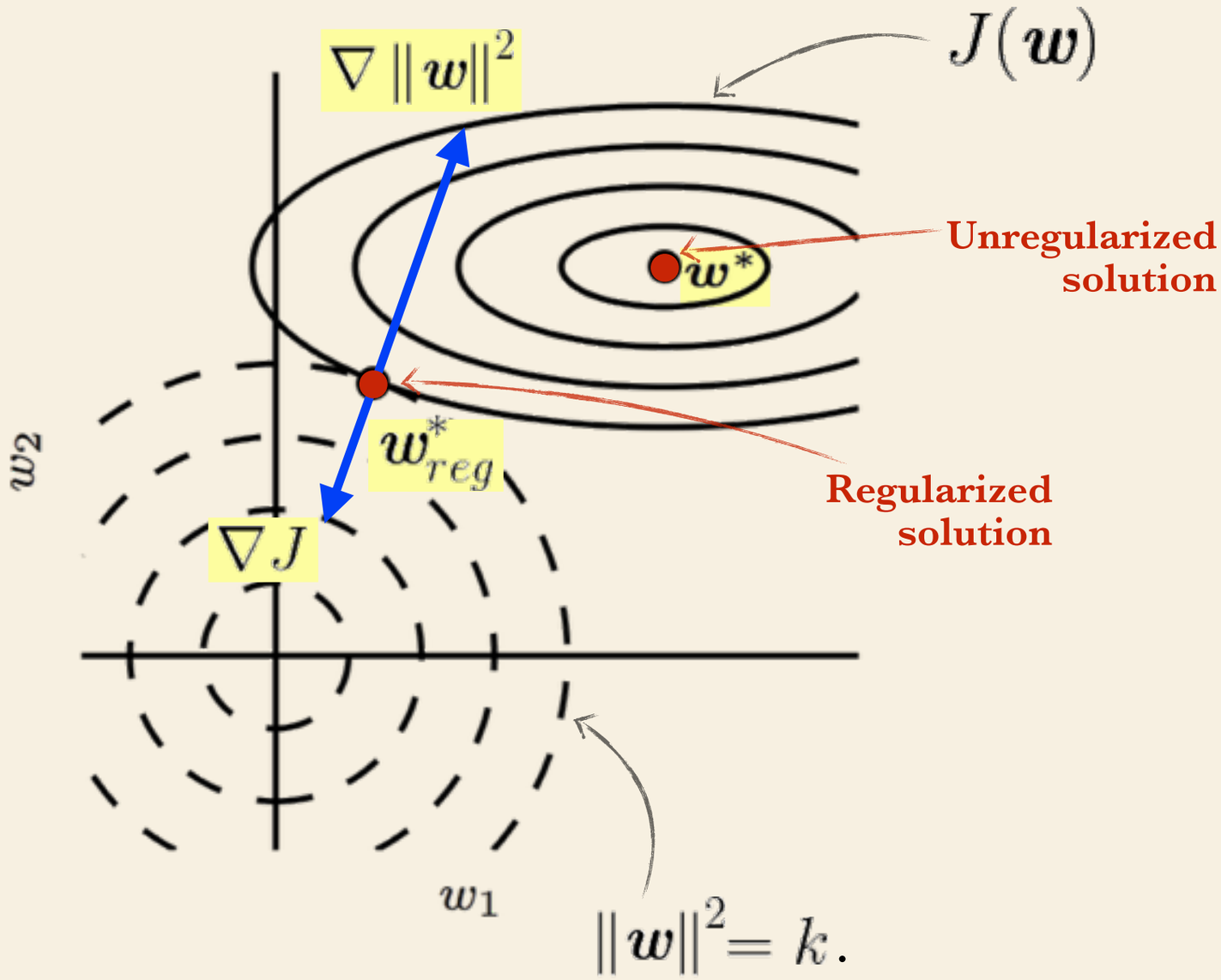
is equivalent to optimizing $J(\boldsymbol{w})$

subject to constraint $\|\boldsymbol{w}\|^2 = k$.

$$J_{reg}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{1}{2}\alpha(\|\boldsymbol{w}\|^2 - k)$$

L^2 Regularization

- **Lagrangian Constrained Optimization**



$$J(\boldsymbol{w}) + \alpha\left(\|\boldsymbol{w}\|^2 - k\right)$$

- We typically don't set k explicitly, We set α .

$$\alpha = \frac{\|\nabla J\|}{\left\|\nabla \|\boldsymbol{w}\|^2\right\|}$$

- Large $\alpha \Rightarrow$ small constraint region
- Large $\alpha \Rightarrow \boldsymbol{w}^* \rightarrow \boldsymbol{0}$

L^2 Regularization - *eigenvector analysis*

- 2nd degree Taylor Approximation of $J(\boldsymbol{w})$ around $\boldsymbol{w}^*$:

unregularized cost function →
$$J(\boldsymbol{w}) \approx J(\boldsymbol{w}^*) + \tfrac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*)$$

regularized cost function →
$$J_{reg}(\boldsymbol{w}) = J(\boldsymbol{w}) + \tfrac{1}{2}\alpha \|\boldsymbol{w}\|^2$$

At $\boldsymbol{w} = \boldsymbol{w}^*_{reg}$, $\quad \nabla J_{reg}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) + \alpha \boldsymbol{w} = 0$

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

- Analysis through e-vector decomposition

$$\boldsymbol{H} = \boldsymbol{V} \Lambda \boldsymbol{V}^T$$

$$\boldsymbol{w}^*_{reg} = \boldsymbol{V}(\Lambda + \alpha \boldsymbol{I})^{-1} \Lambda \boldsymbol{V}^T \boldsymbol{w}^*$$

Stretching in i^{th} eigen direction $\dfrac{\lambda_i}{\lambda_i + \alpha}$

N.B: **small eigen-directions** will be affected **more** than **larger eigen-directions**.

L^2 Regularization - *covariance analysis*

- **Normal Equations for Linear Regression**

$$\boldsymbol{X} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}_{m \times n}, \quad \boldsymbol{y} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}_{m \times 1}$$

$$\boldsymbol{w}^*_{reg} = (\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

covariance of input features

L^2 regularization causes the learning algorithm to **perceive** the input $\boldsymbol{X}$ with increased variance.

covariance of input features with the target values.

Example: $\boldsymbol{X}^T\boldsymbol{y} = \begin{bmatrix} small \\ big \\ big \end{bmatrix}$

Then, w_1 would shrink more than other components.

L^1 Regularization

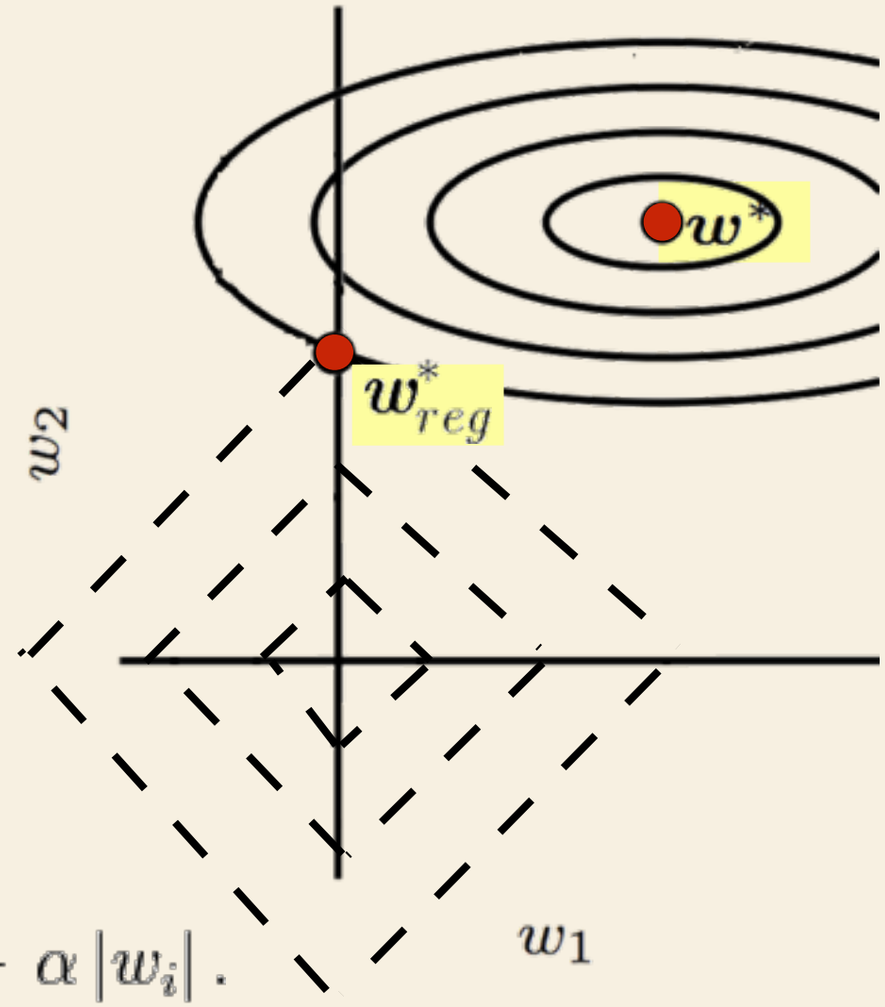
(a.k.a. LASSO)

Regularization Term

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

$$J_{reg}(\boldsymbol{w}) = J(\boldsymbol{w}) + \alpha \|\boldsymbol{w}\|_1$$

$$\nabla J_{reg}(\boldsymbol{w}) = \nabla J(\boldsymbol{w}) + \alpha \, sign(\boldsymbol{w})$$



- **2nd degree Taylor Approximation of $J(\boldsymbol{w})$ around $\boldsymbol{w}^*$:**

$$J_{reg}(\boldsymbol{w}) \approx J_{reg}(\boldsymbol{w}^*) + \sum_i \tfrac{1}{2} H_{i,i}(w_i - w_i^*)^2 + \alpha |w_i| \, .$$

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

$$w_i^* \leq \frac{\alpha}{H_{i,i}} \longrightarrow w_i = 0$$

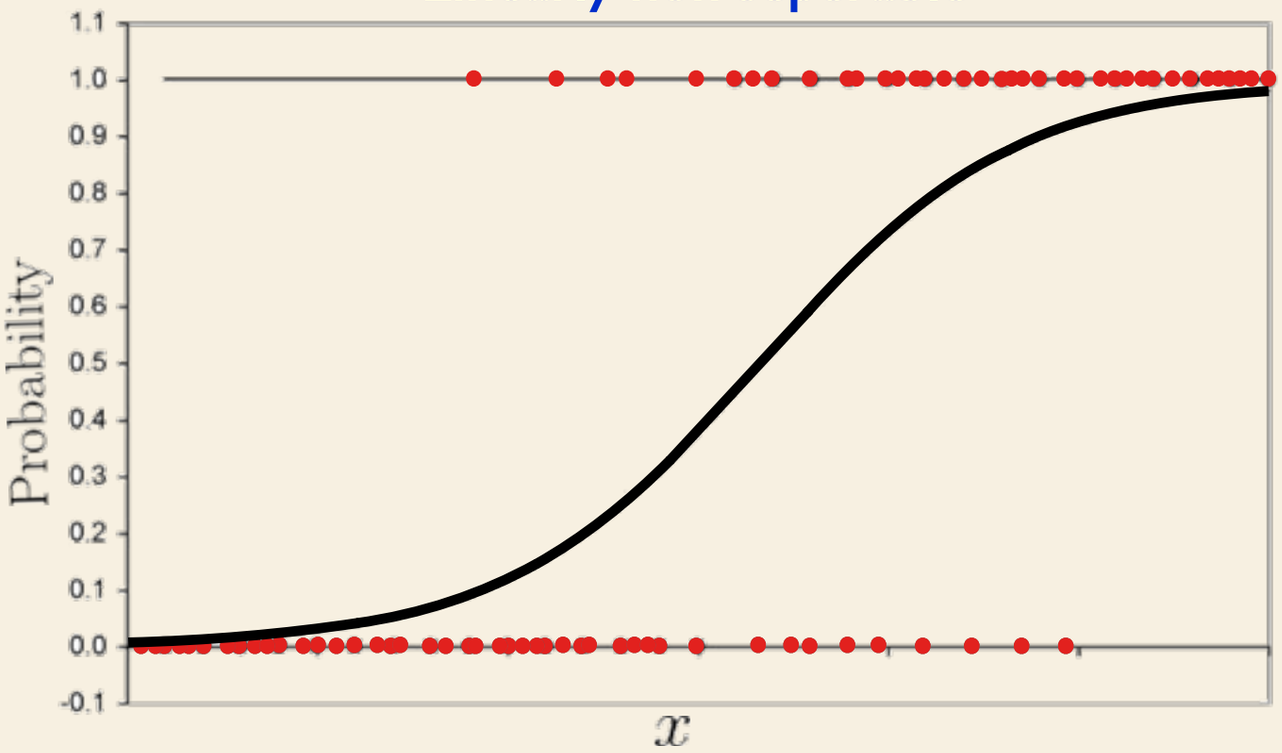
(Induces Sparsity)

$$w_i^* > \frac{\alpha}{H_{i,i}} \longrightarrow w_i = w_i - \frac{\alpha}{H_{i,i}}$$

Under-Constrained Problems

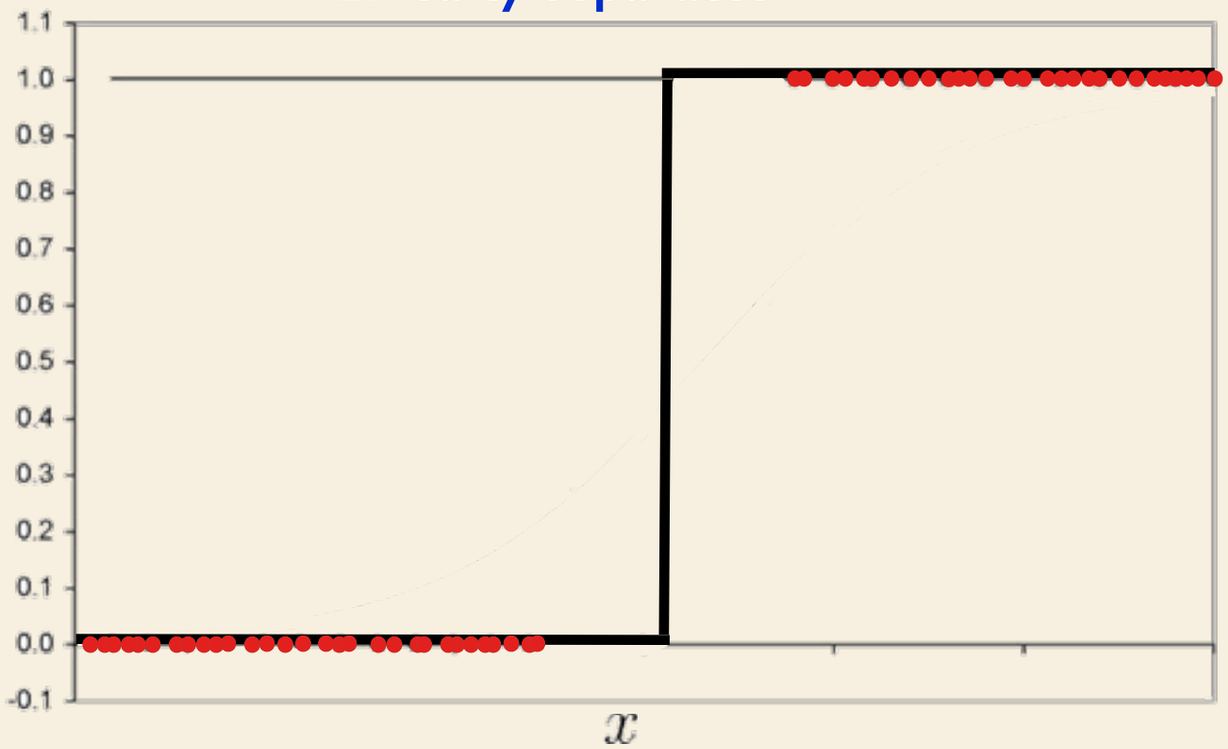
- E.g. Logistic Regression

Linearly non seperable



- Well behaved problem.

Linearly separable



- Under-determined problem. ($\boldsymbol{w}$ will continue to increase in a GD Algorithm)

$$\boldsymbol{J}(\boldsymbol{w}) = -\boldsymbol{y}\, log(\boldsymbol{w}^T\boldsymbol{x}) - (1-\boldsymbol{y})\, log(1-\boldsymbol{w}^T\boldsymbol{x}) + \frac{1}{2}\alpha \left\|\boldsymbol{w}\right\|^2$$

Data Augmentation

- Best way to **improve generalization** of a model is to train it on **more data**.
- Data Augmentation works particularly well for **Object Recognition** tasks.
- Injecting **noise** to input works well for **Speech Recognition**.

Original Input Image

Affine Distortion

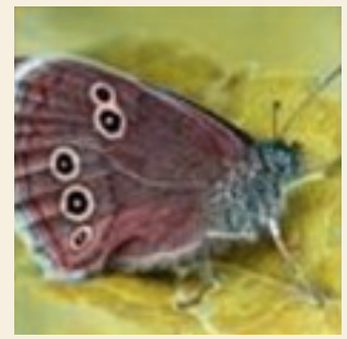
Noise

Elastic Deformation

Horizontal Flip

Random Translation

Hue Shift



Noise Robustness

- Addition of **noise with a small variance** is equivalent to imposing **norm penalty** on weights.
- **Noise on weights:** A stochastic implementation of Bayesian Inference (uncertainty on weights are represented by a probability distribution)

$$J = \mathbb{E}\left[(\hat{y}(\boldsymbol{x}) - y)^2\right]$$

$$\hat{y}_\epsilon(\boldsymbol{x}) = \hat{y}(\boldsymbol{x}) + \epsilon$$

$$\epsilon \sim \mathcal{N}(0, \eta \boldsymbol{I}).$$

For each input data, apply noise on weights

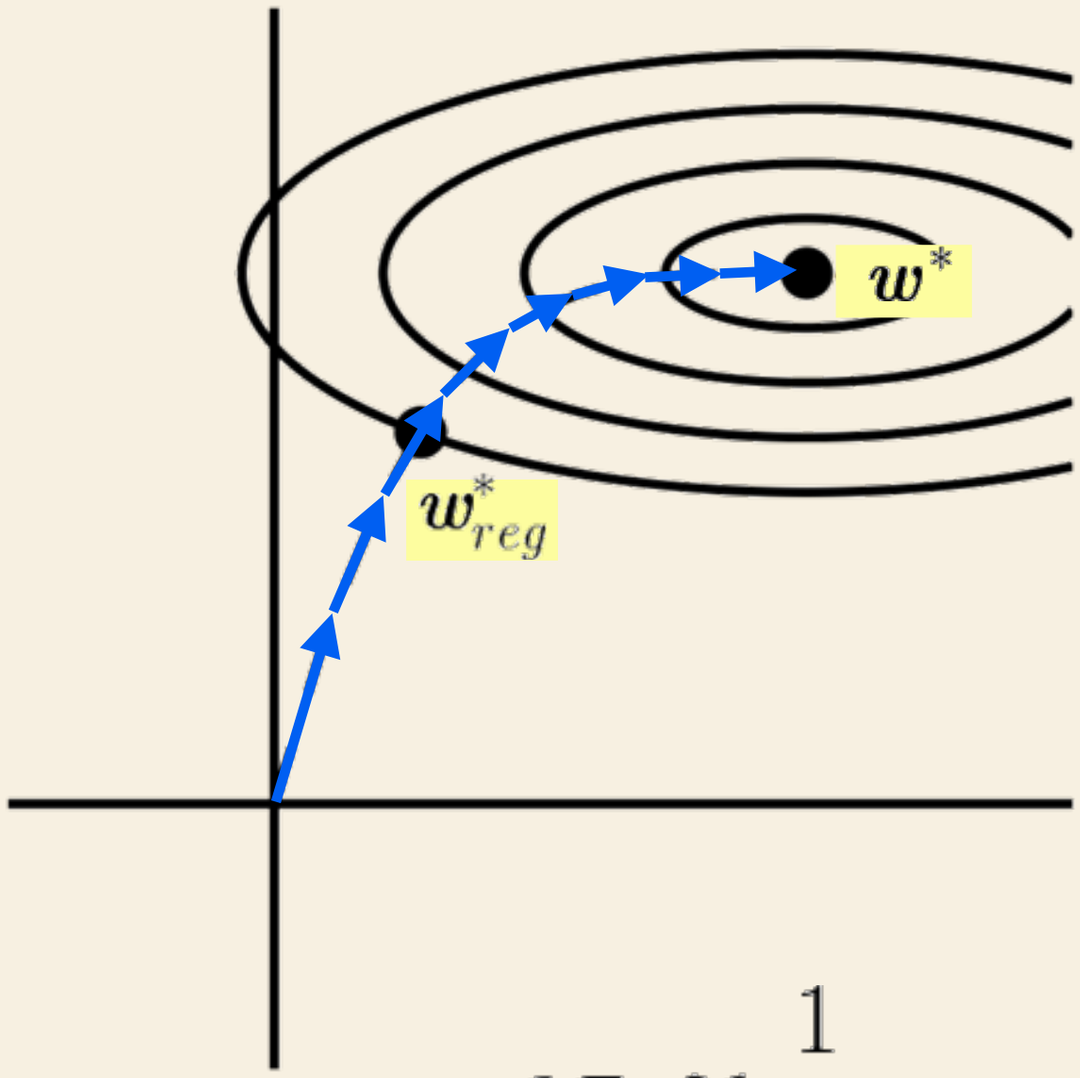
$$J_{reg} = \mathbb{E}\left[(\hat{y}_\epsilon(\boldsymbol{x}) - y)^2\right]$$

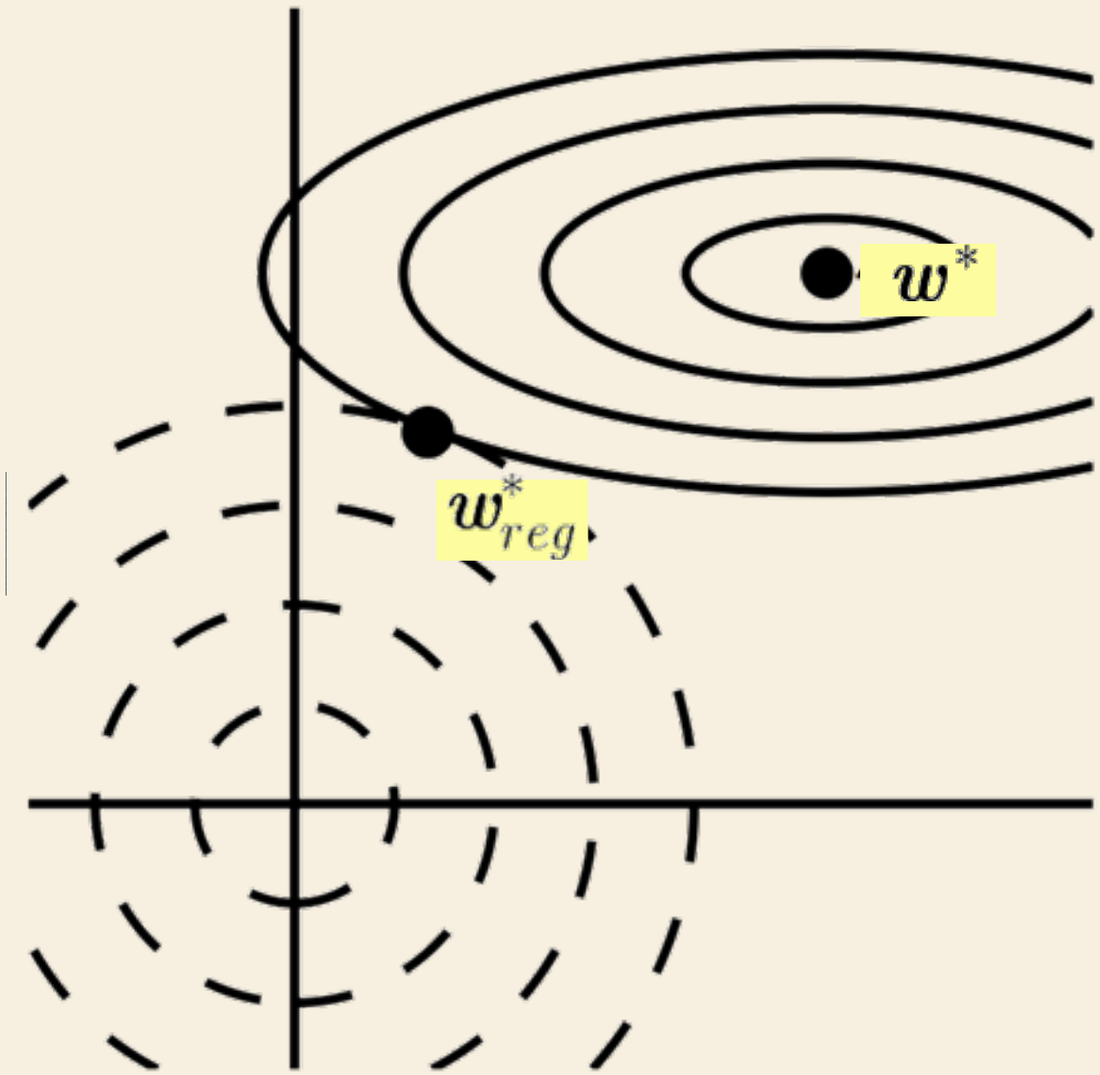
modified cost function

$$J_{reg} = J + \eta\, \mathbb{E}\left[\nabla \left\|\hat{y}_\epsilon(\boldsymbol{x})\right\|^2\right]$$

regularization term

Early Stopping





$$\epsilon\,\tau \;\propto\; \frac{1}{\alpha}$$

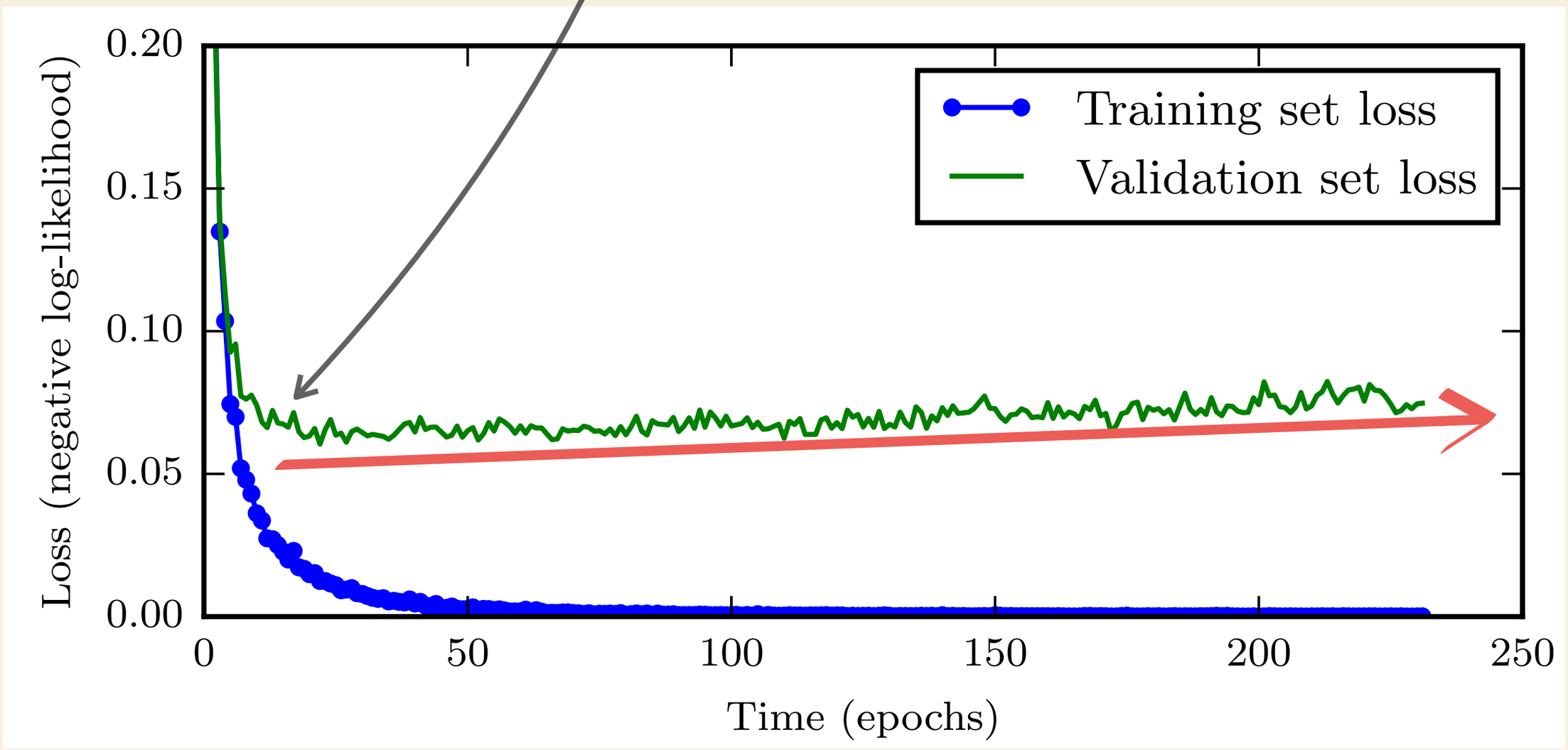
learning rate

number of steps

regularization parameter

Early Stopping

Early stopping: **Terminate while validation set performance is better**



Chapter 7: Regularization for Deep Learning

PART 2

Deep Learning Textbook Study Group, SF

Safak Ozkan, shafaksan@gmail.com

April 22, 2017

Parameter Sharing

- **Parameter sharing** scheme is used in **ConvNets** to control the number of parameters. It also reflects the natural character of some problems (translational invariance).

  WITHOUT parameter sharing, for instance, for a training dataset with centered faces—the **ConvNet model** would look for the eyes, the nose and mouth always at the same regions of the image. Instead we use a *locally-connected layer (a densely connected layer* would still be an overkill)

Parameter Tying

- Training a Neural Network which has similarities to another pretrained Neural Network.

$$\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$$

$$\hat{y}^{(B)} = f(\boldsymbol{w}^{(B)}, \boldsymbol{x})$$

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

Known weights

To be trained

Sparse Representation

- Invoking a penalty on **activations** instead of on **weights**.

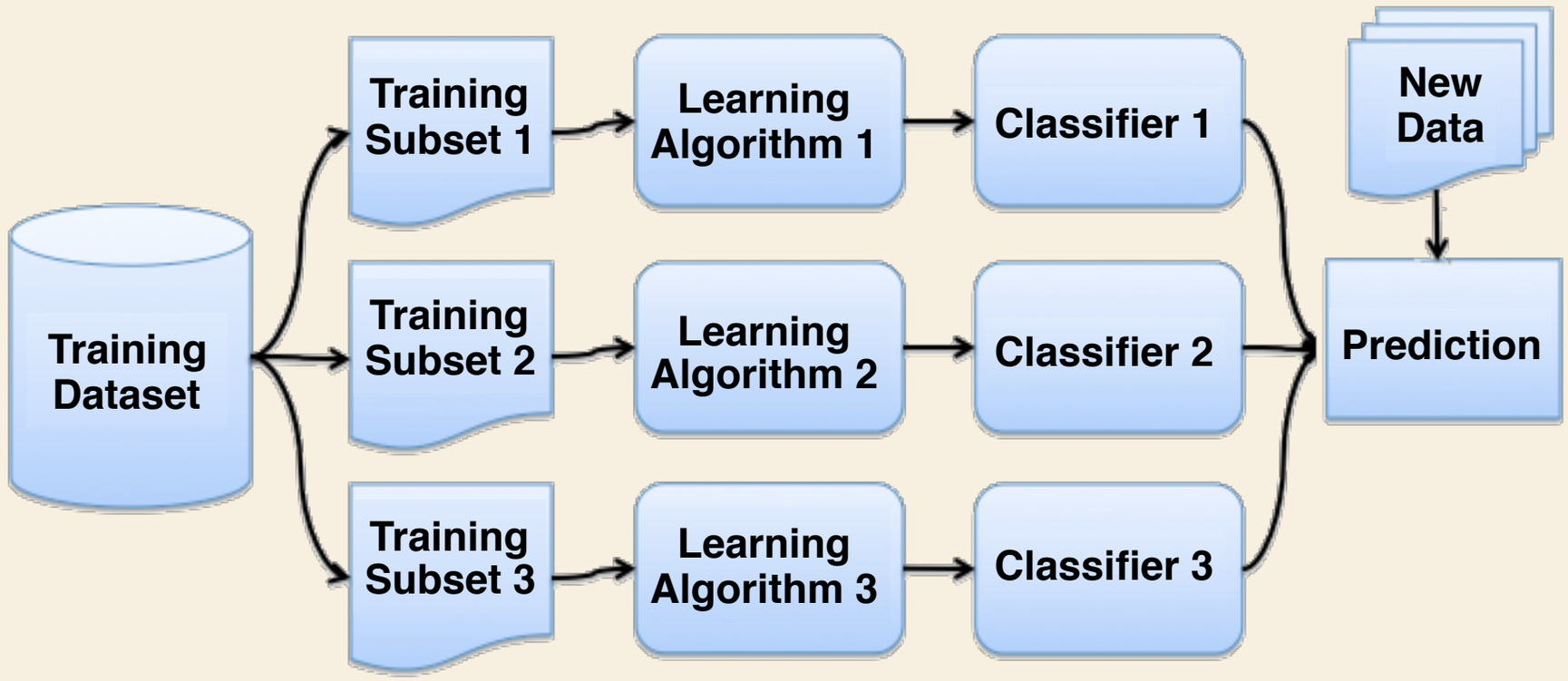
$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{B} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{h} \in \mathbb{R}^n$

$$J_{reg}(\boldsymbol{w}) = J(\boldsymbol{w}) \; + \; \alpha\Omega(\boldsymbol{h})$$
$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1$$

Bootstrap Aggregation (Bagging)

- **Bagging:** Constructs k different data sets via **boostrapping**.
  Train each i^{th} model on i^{th} dataset.



Bootstrap Aggregation (Bagging)

- **Ensemble Methods:** Train separate models (on same dataset). Then average the output of all model predictions.
- Assume k separate models with different architectures trained on the **same data set**.
- Each i^{th} individual model will have an error of ϵ_i .

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

Ensemble output

v, **variance of each model error**

covariance of errors (Assumed zero)

Conclusion:
- The error decreases **linearly** with ensemble size k
- The ensemble will perform **better than its best model**.

Bootstrap Aggregation (Bagging)

Aside: Assume original data set of size n.

Boostrapping with resampling a new data set of same size.

- On average $\frac{1}{3}$ of the original data set will be left out of each subset.

$$p(not\ picking\ i^{th} data\ point) = \left(1 - \frac{1}{n}\right)^n$$

- The limit of following sequence …

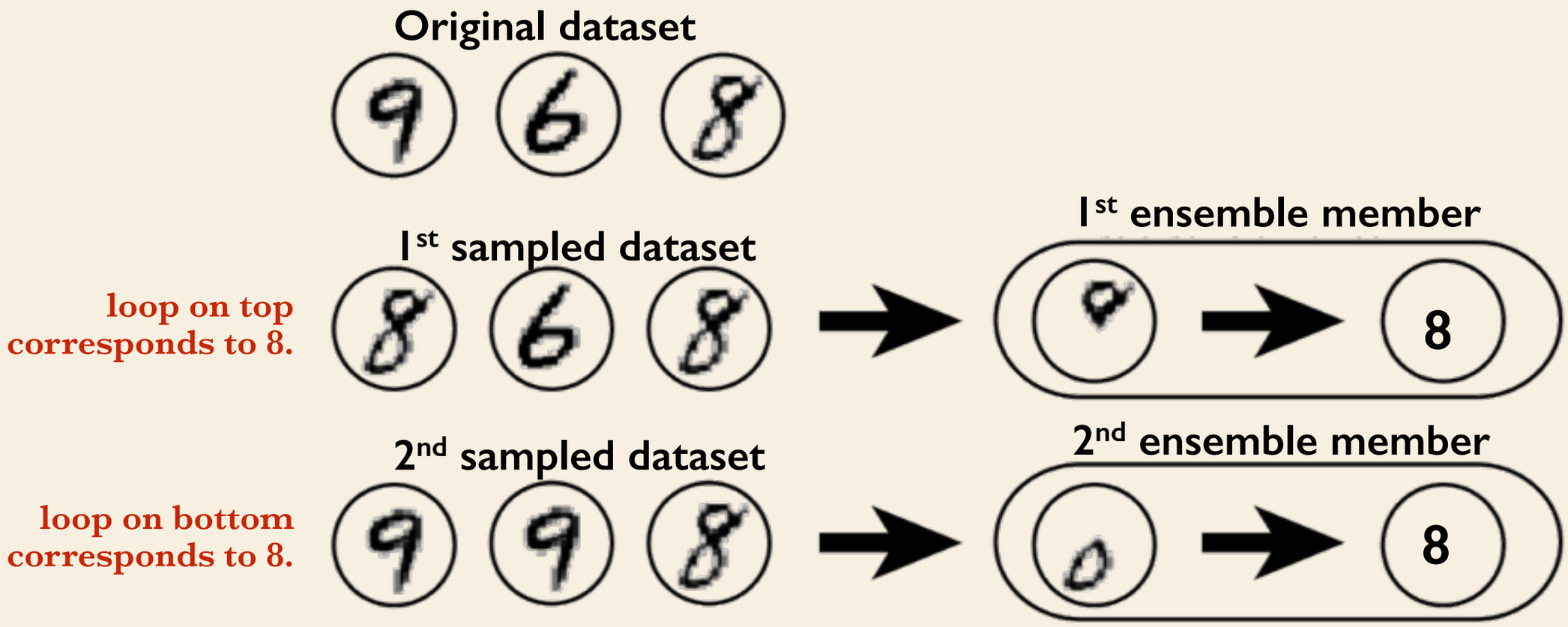
$$e^x = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

Hence,

$$p(not\ picking\ i^{th} data\ point) = e^{-1} \approx 0.36$$

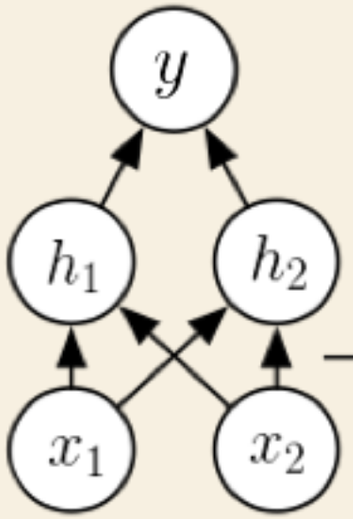
Bootstrap Aggregation (Bagging)

- **A conceptual depiction:** Same model trained on 3 separate bootstrapped datasets.

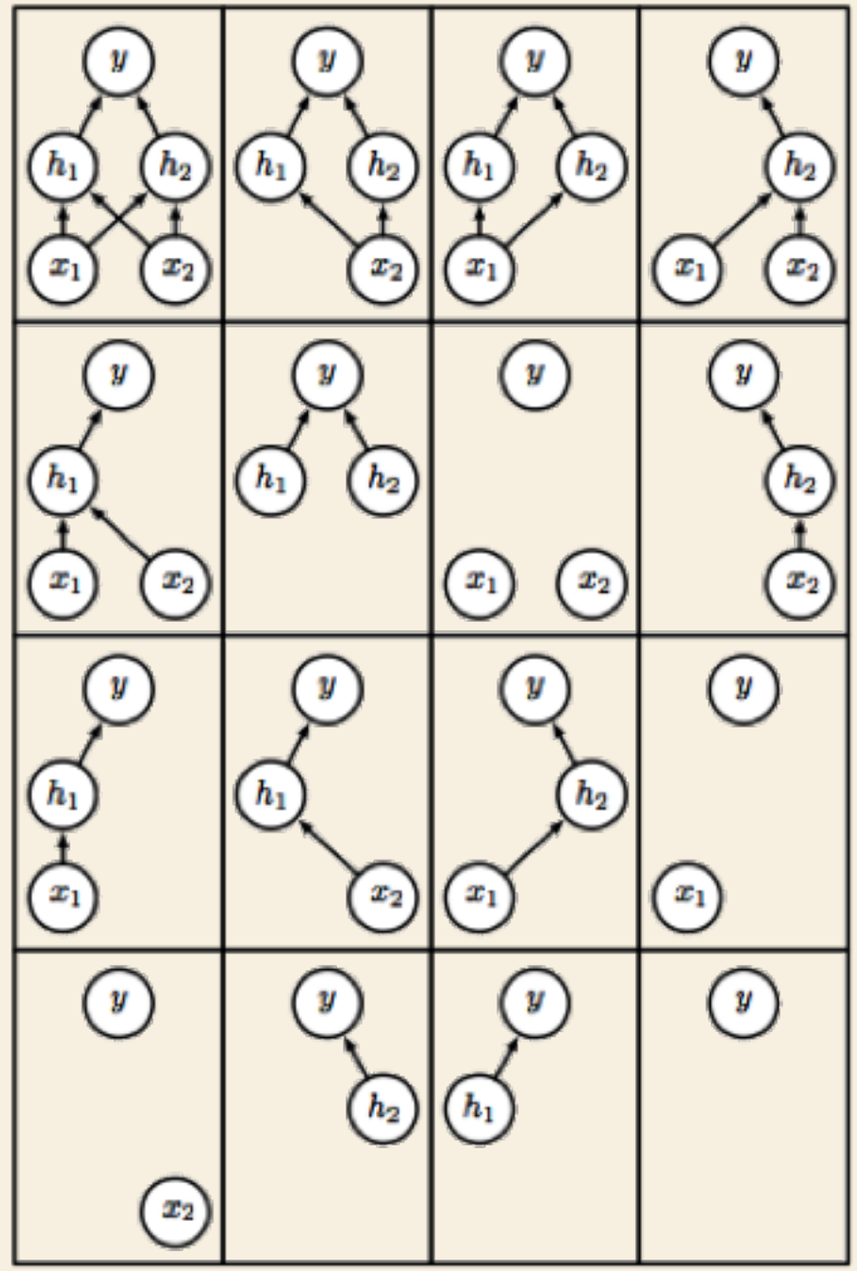


Dropout

- Derived from concept of **Bagging**
- Apply a **binary mask** μ to each **input layer** x, and each **hidden layer** h.
- Train the NN model on k **minibatches** of the dataset at a time (SGD), where each i^{th} **dataset** is used to train each i^{th} **submodel**.



Base model



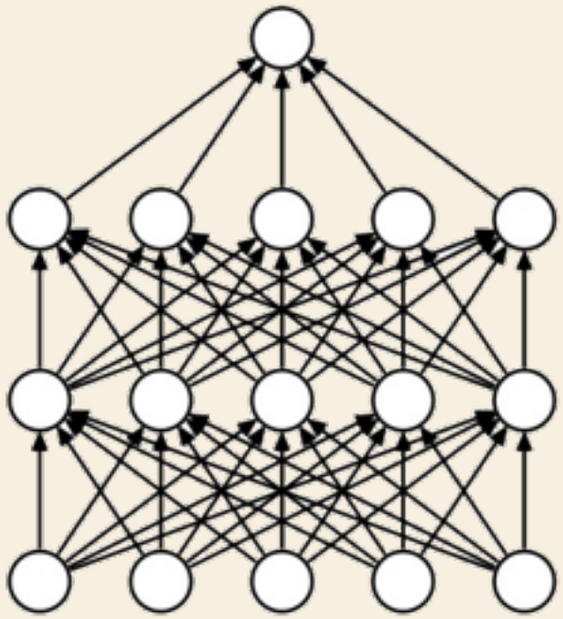
Ensemble of submodels

Differences from Bagging:

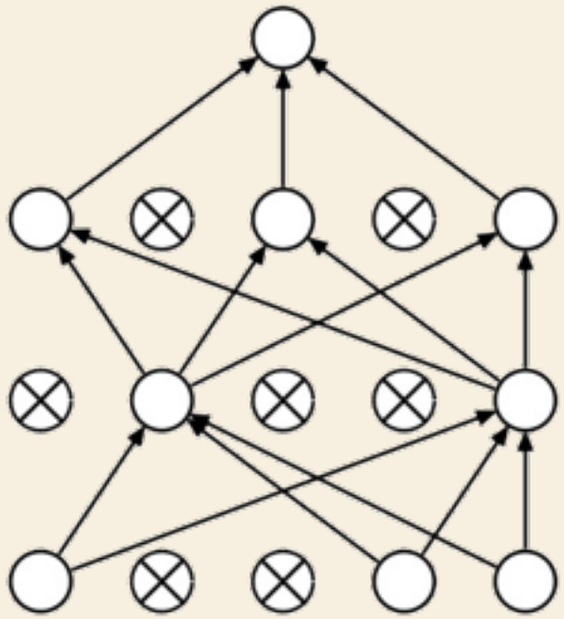
- All submodels **share parameters**.
- Each submodel is trained for a **single step**.

Dropout

- Apply a **binary mask** μ to each unit **input layer** x , and each **hidden layer** h .



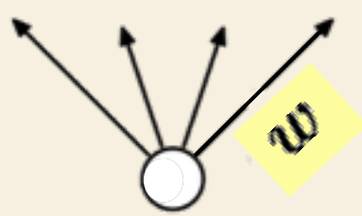
Standard NN



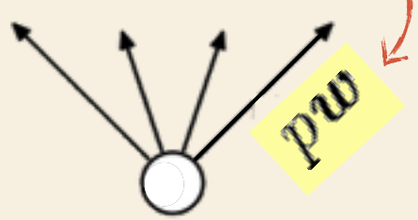
After applying dropout

$$\mu_2 = \{1, 0, 1, 0, 1\}$$
$$\mu_1 = \{0, 1, 0, 0, 1\}$$
$$\mu_{input} = \{1, 0, 0, 1, 1\}$$

Weight Scaling

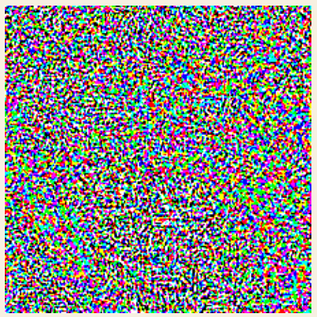


At training time



At test time

Adversarial Training

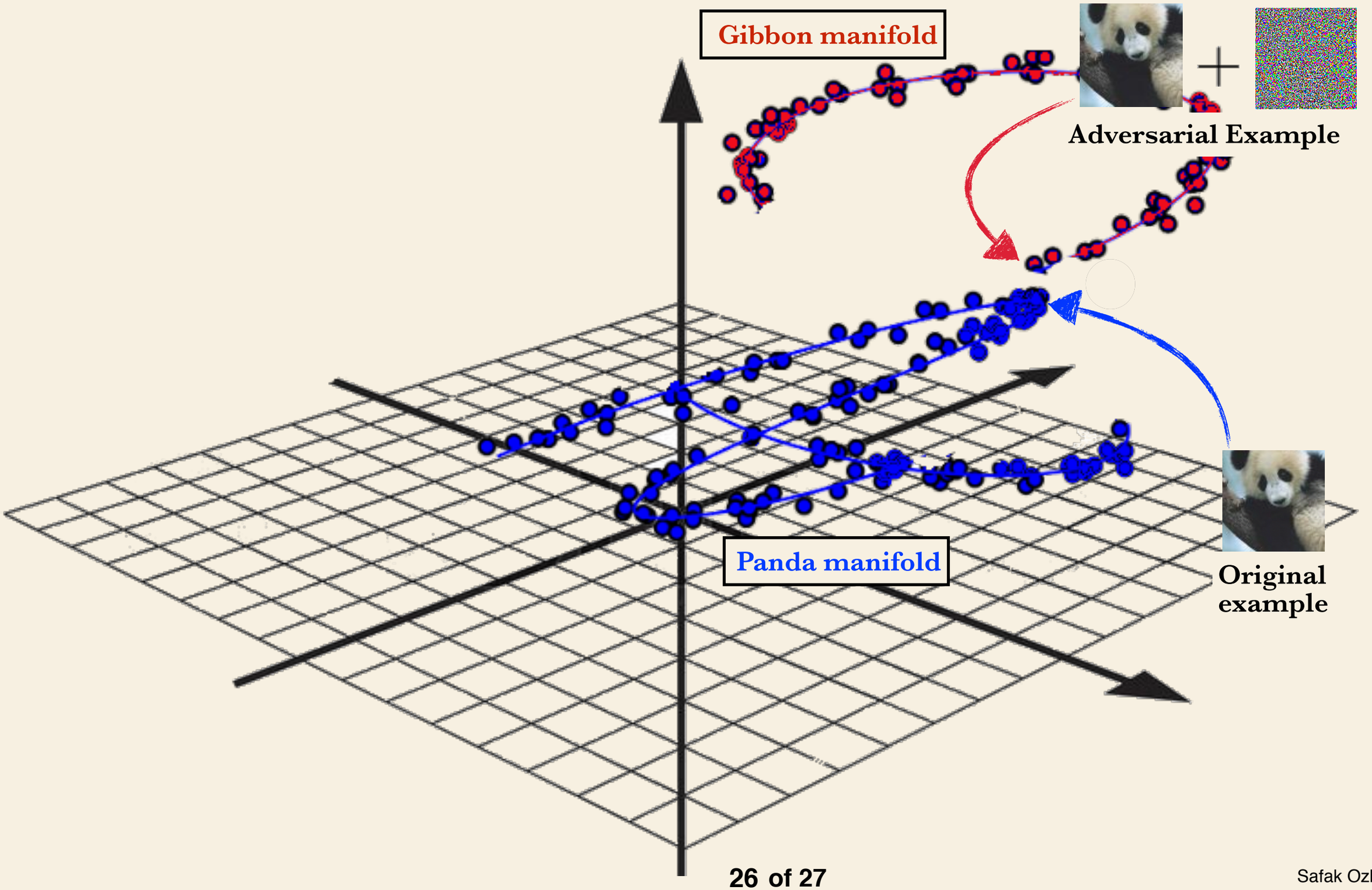

$+ .007 \times$ $=$

Adversarial Example

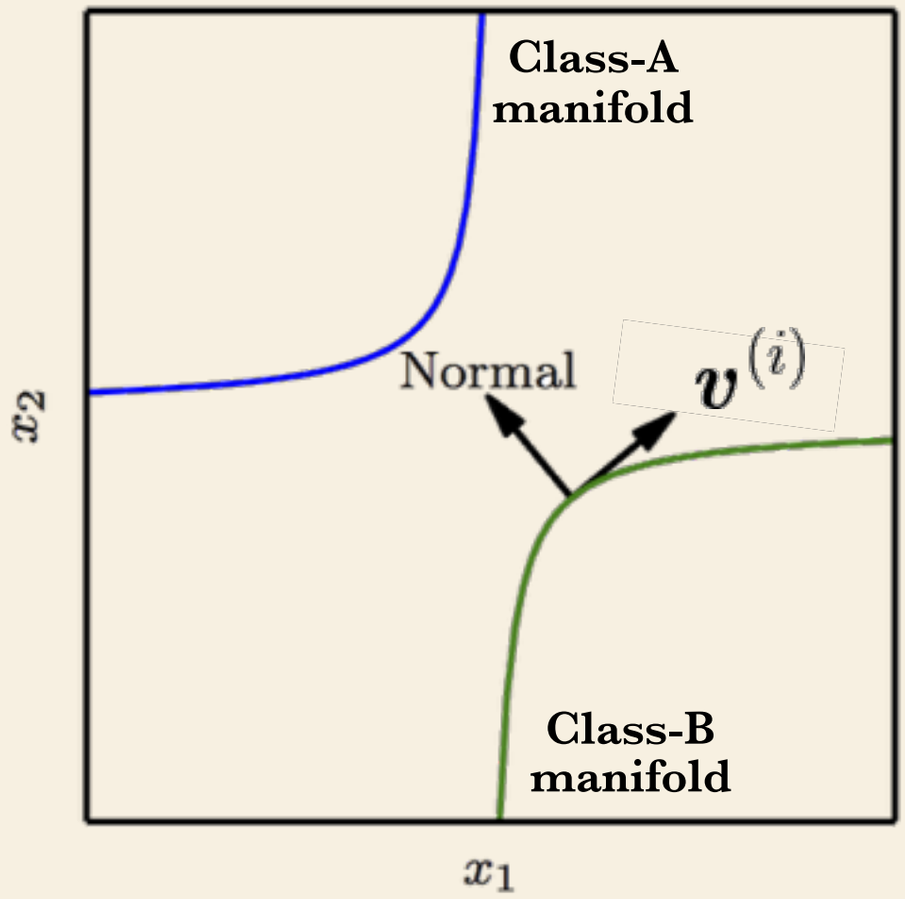
$\boldsymbol{x}$	$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda"	"nematode"	"gibbon"
w/ 57.7%	w/ 8.2%	w/ 99.3 %
confidence	confidence	confidence

- Neural Networks are prone to **curse of dimensionality**.
- "Excessive linearity" causes the error at each pixel to add up linearly.
- **Adversarial Training** tries make the NN output a constant value at training example x.
- Training on **"adversarially perturbed"** examples from the training set.

Adversarial Training



Manifold Tangent Classifier



$$\Omega(f) = \sum_i \left((\nabla f(\boldsymbol{x}))^T \boldsymbol{v}^{(i)} \right)^2$$

Each output of the NN model

tangent direction

- **Tangent propagation** is related to **data augmentation.**
- User specifies "tangent directions", $\boldsymbol{v}^{(i)}$.
- It learns to resist perturbation in the directions **normal** to class manifold.