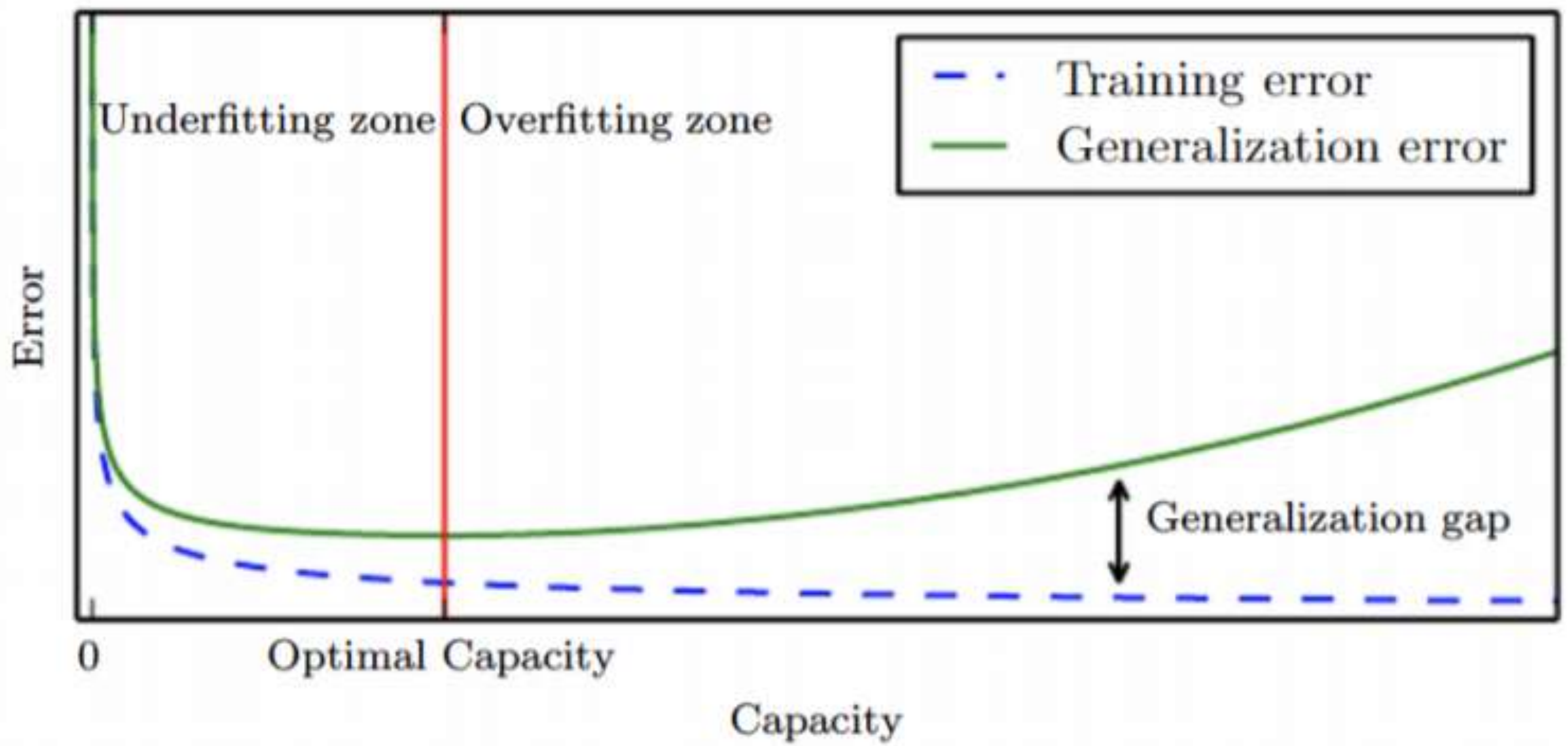
Regularization

Deep Learning

Why Regularization?



Definition: Regularization

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

Model Types and Regularization

- Three types of model families
  1. Excludes the true data generating process
     - Implies underfitting and inducing high bias
  2. Matches the true data generating process
  3. Overfits
     - Includes true data generating process but also many other processes
- Goal of regularization is to take model from third regime to second

Norm Penalty

- When our training algorithm minimizes the regularized objective function

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

  - it will decrease both the original objective J on the training data and some measure of the size of the parameters θ
- Different choices of the parameter norm Ω can result in different solutions preferred
  - We discuss effects of various norms

L^2 parameter Regularization

- Simplest and most common kind
- Called *Weight decay*
- Drives weights closer to the origin
  - by adding a regularization term to the objectve function $\boxed{\Omega(\theta) = \frac{1}{2} \| w \|_2^2}$
- In other communities also known as *ridge regression* or *Tikhonov regularization*

Gradient of Regularized Objective

- Objective function (with no bias parameter)

$$\boxed{\tilde{J}(w;X,y) = \frac{\alpha}{2} w^T w + J(w;X,y)}$$

- Corresponding parameter gradient

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

- To perform single gradient step, perform update:

$$\boxed{\boldsymbol{w} \leftarrow \boldsymbol{w} - \varepsilon\left(\alpha \boldsymbol{w} + \nabla_w J\left(\boldsymbol{w}; X, \boldsymbol{y}\right)\right)}$$

- Written another way, the update is

$$\boxed{\boldsymbol{w} \leftarrow (1-\varepsilon\alpha)\boldsymbol{w} - \varepsilon \nabla_w J\left(\boldsymbol{w}; X, \boldsymbol{y}\right)}$$

  - We have modified learning rule to shrink $\boldsymbol{w}$ by constant factor $1\text{-}\varepsilon\alpha$ at each step

Effect of L^2 regularization on optimal w

Objective function: $\boxed{\tilde{J}(w;X,y) = \frac{\alpha}{2} w^T w + J(w;X,y)}$

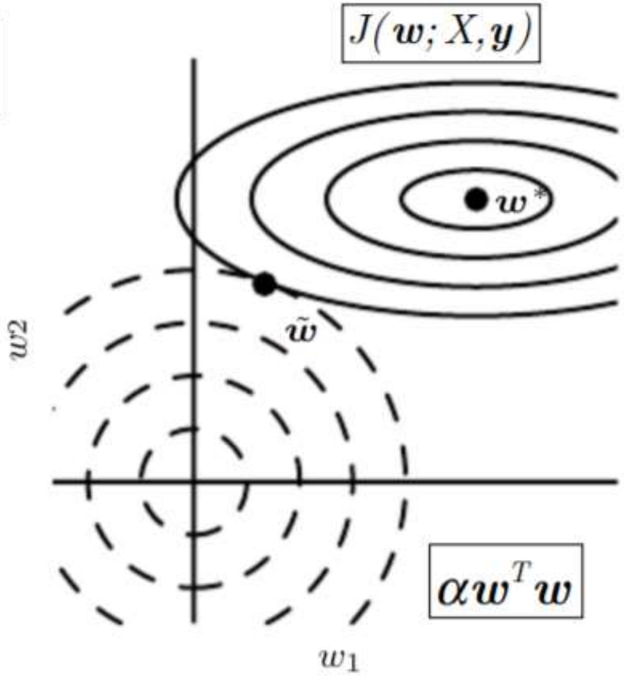
Solid ellipses:

- contours of equal value of unregularized objective $\boxed{J(\boldsymbol{w};X,\boldsymbol{y})}$

Dotted circles:

- contours of equal value of L^2 regularizer $\boxed{\alpha \boldsymbol{w}^T \boldsymbol{w}}$

At point $\tilde{\boldsymbol{w}}$ competing objectives reach equilibrium

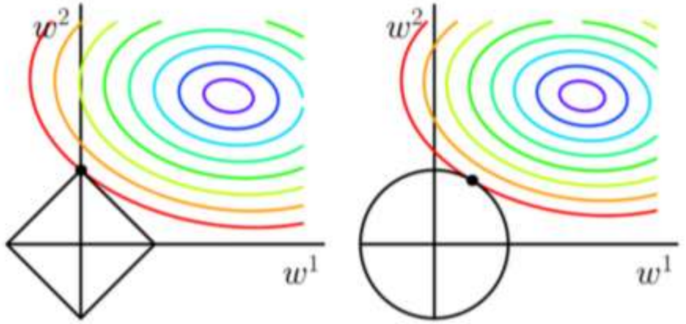


L^1 Regularization

- L^2 weight decay is common weight decay
- Other ways to penalize model parameter size
- L^1 regularization is defined as

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

  – which sums the absolute values of parameters

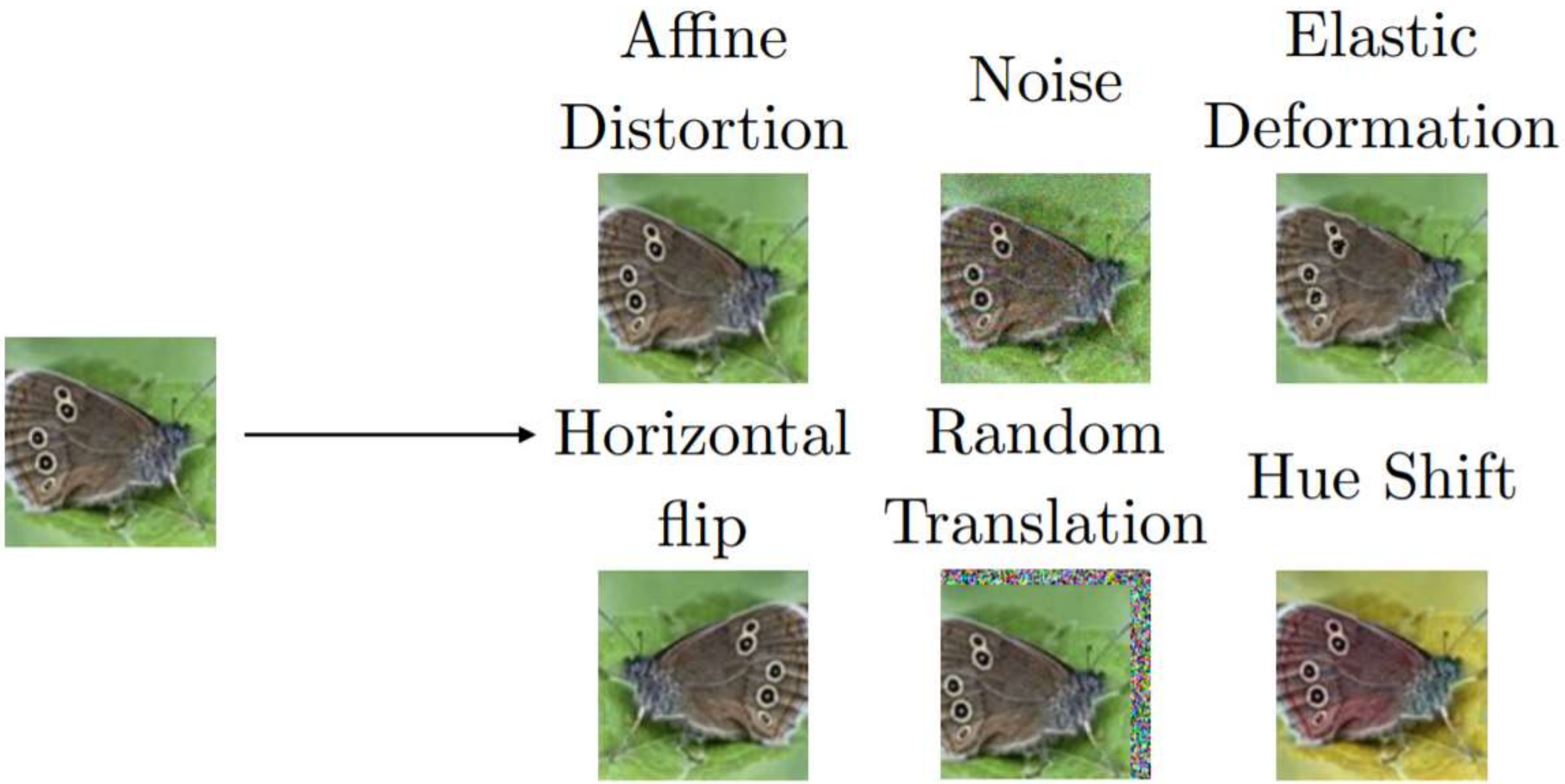


(a) ℓ_1-ball meets quadratic function. ℓ_1-ball has corners. It's very likely that the meet-point is at one of the corners.

(b) ℓ_2-ball meets quadratic function. ℓ_2-ball has no corner. It is very unlikely that the meet-point is on any of axes.

Image Source: https://zhuanlan.zhihu.com/p/28023308

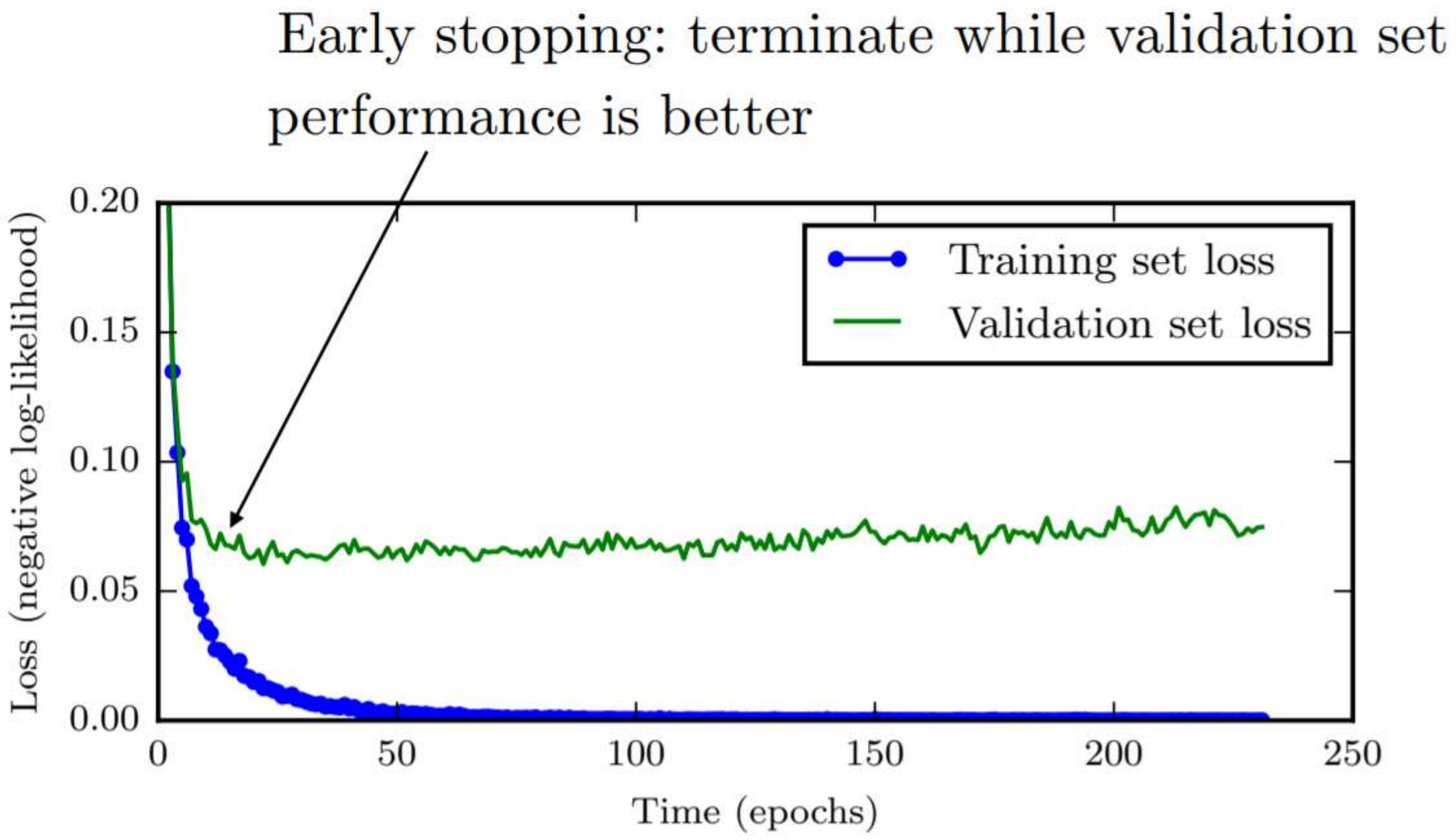
Data Augmentation



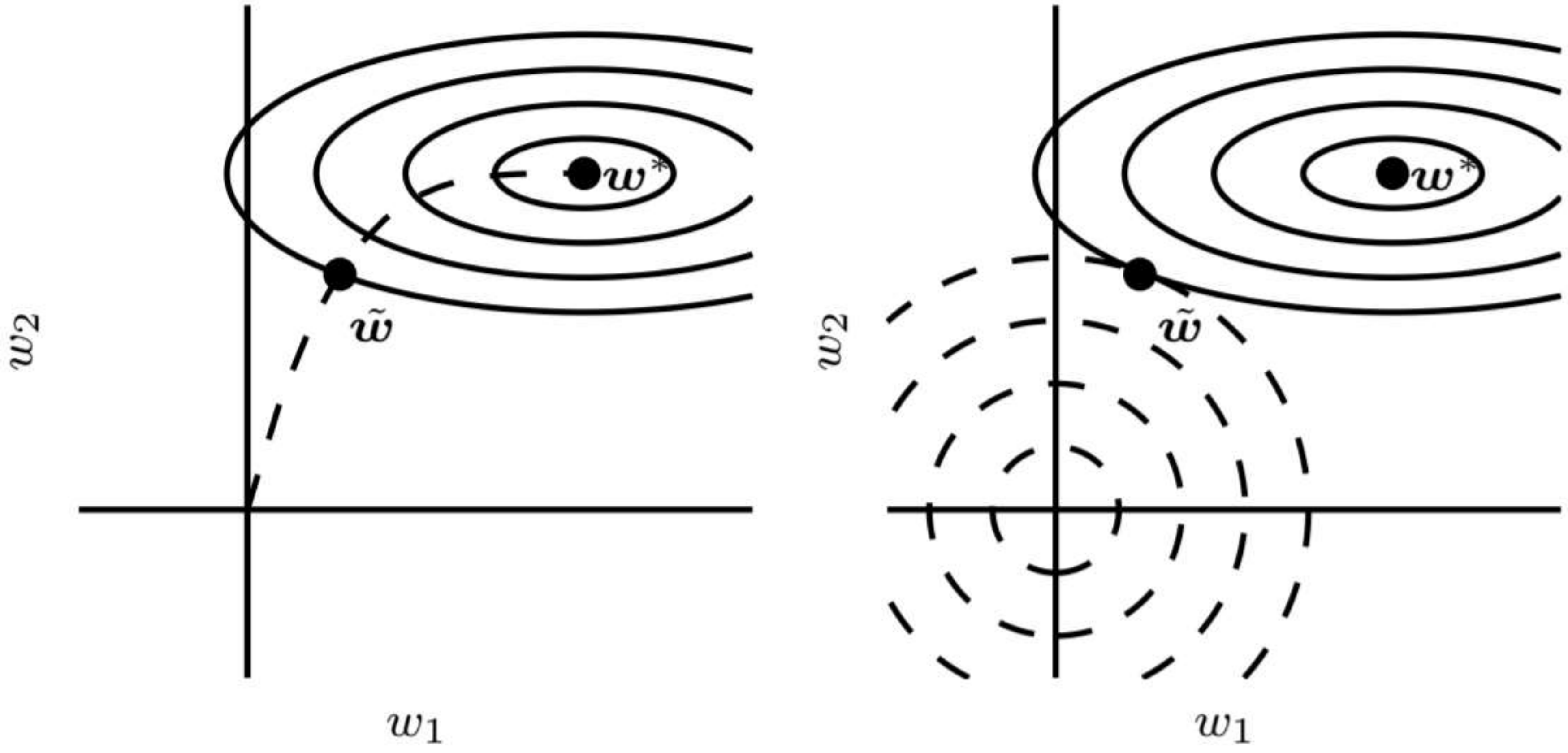
Caution in Data Augmentation

- Not apply transformation that would change the class
- OCR example: ‘b’ vs ‘d’ and ‘6’ vs ‘9’
  - Horizontal flips and 180 degree rotations are not appropriate ways
- Some transformations are not easy to perform
  - Out of plane rotation cannot be implemented as a simple geometric operation on pixels

Early Stopping



Early Stopping and Weight Decay



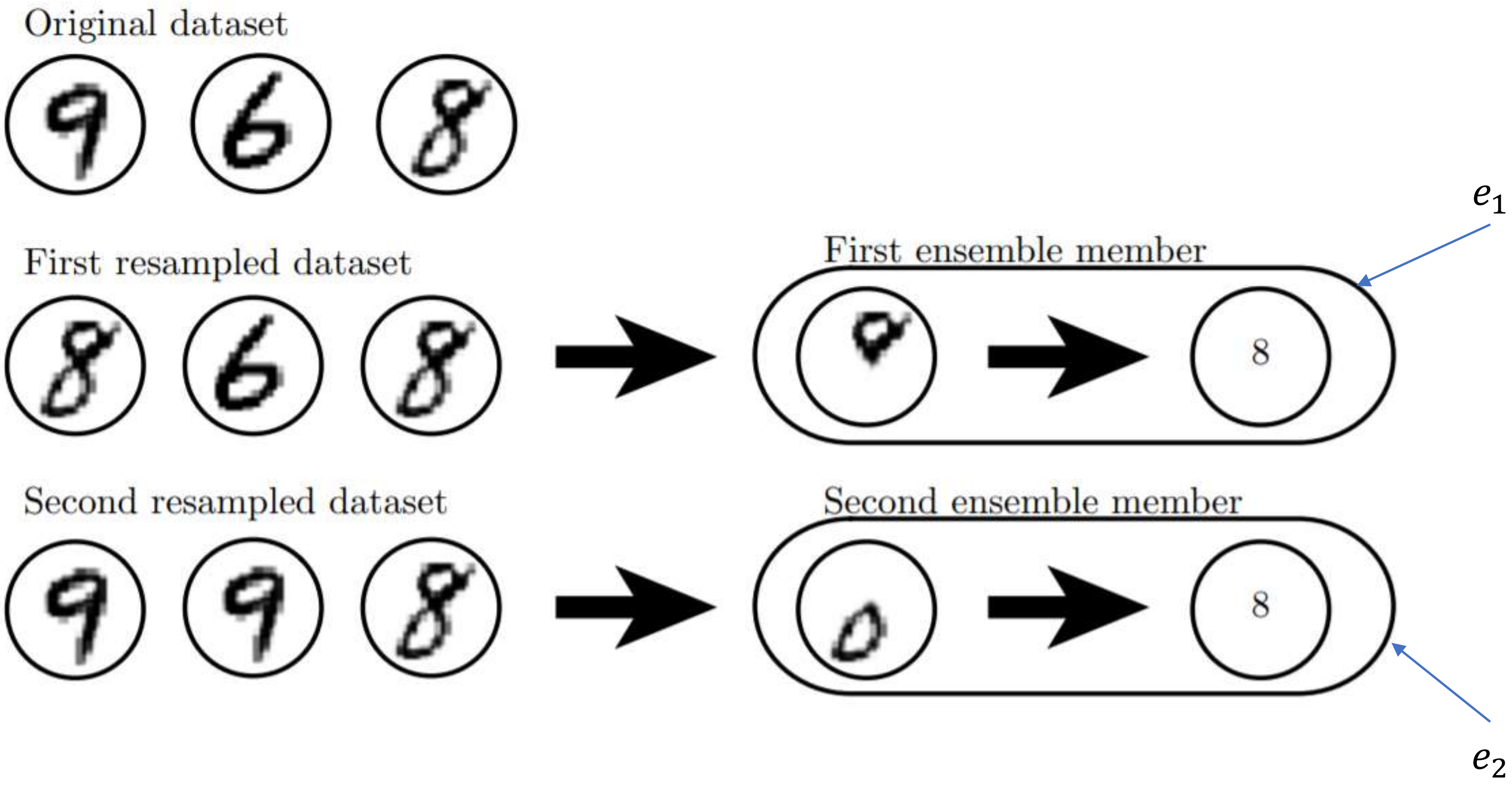
Bagging/Model Ensemble

- Train k models.
- Suppose that the errors have variance v and covariance c.

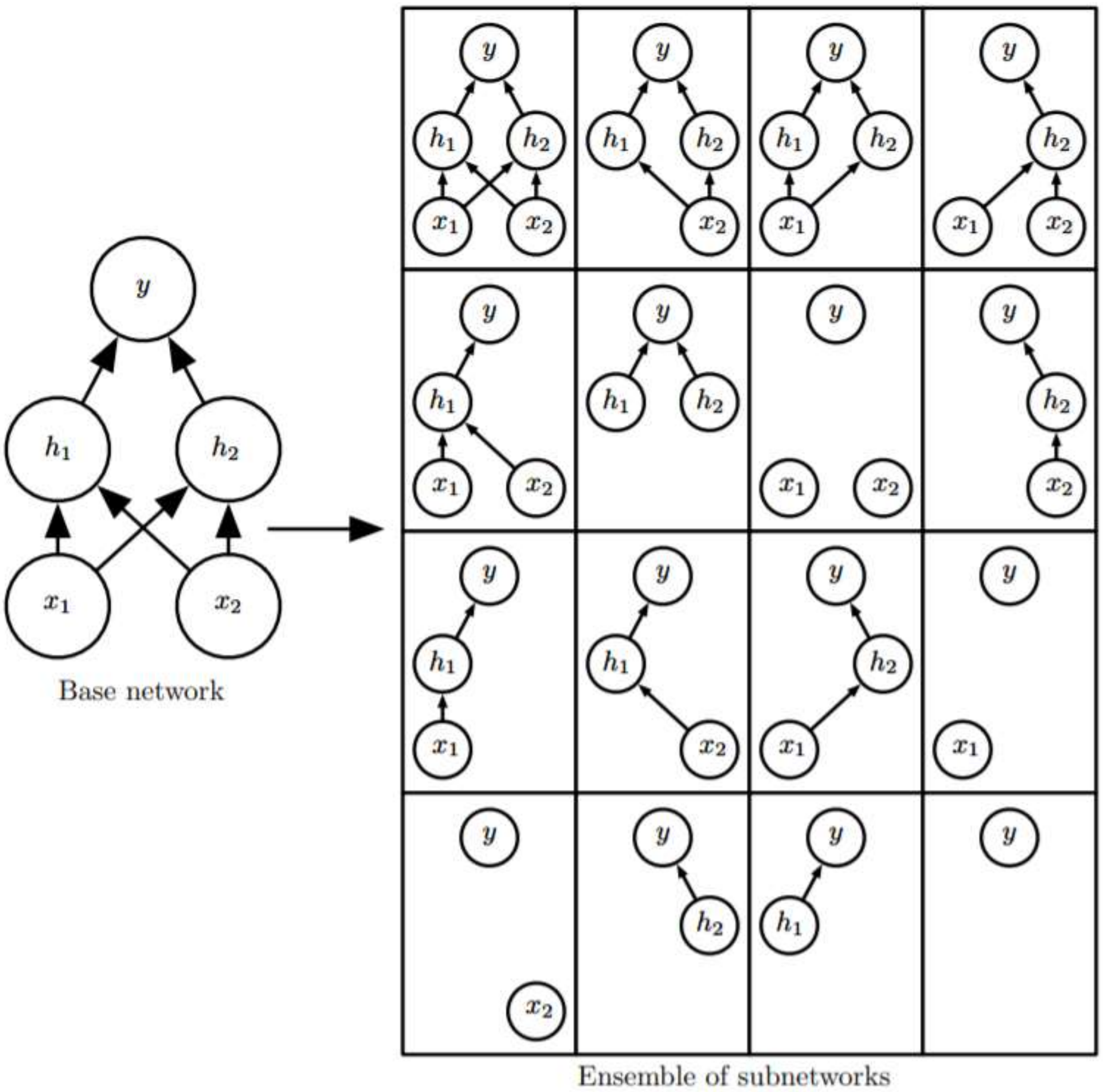
$$\bar{e} = \frac{1}{k}\sum_i e_i$$

$$\begin{aligned}
\mathbb{E}\left[\bar{e}\right] &= \mathbb{E}\left[\left(\frac{1}{k}\sum_i e_i\right)^2\right] \\
&= \frac{1}{k^2}\mathbb{E}\left[\left(\sum_i e_i\right)^2\right] \\
&= \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(e_i^2 + \sum_{i\neq j} e_i e_j\right)\right] \\
&= \frac{1}{k^2}\left(k\mathbb{E}\left[e_i^2\right] + \left(k^2 - k\right)\mathbb{E}\left[e_i e_j\right]\right) \\
&= \frac{1}{k}\mathbb{E}\left[e_i^2\right] + \frac{k-1}{k}\mathbb{E}\left[e_i e_j\right] \qquad = \frac{1}{k}v + \frac{k-1}{k}c
\end{aligned}$$

Bagging/Model Ensemble



Dropout



Dropout: Implementation

The expected value of the neuron with dropout is:

$$E[ra] = \sum_r p(r)ra = pa + (1-p)0 = pa$$

where a is o/p of neuron, $r \in \{0,1\}$, dropout probability $p(r): p$

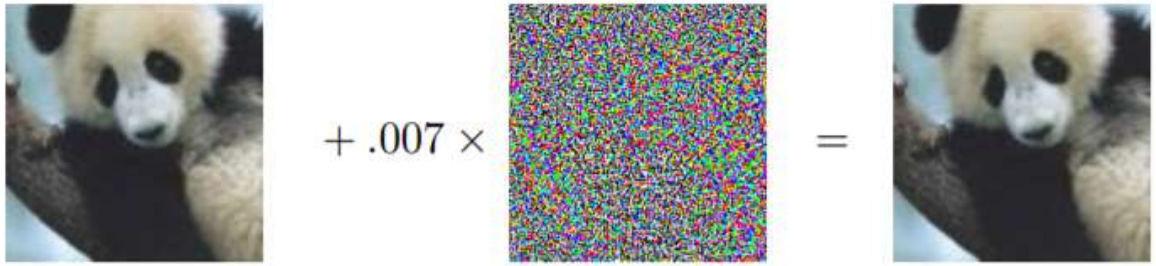
Dropout

```
def train_step(X):
    hidden_layer_1 = np.maximum(0, np.dot(W1, X) + b1)
    dropout_mask_1 = np.random.binomial(1, keep_prob, hidden_layer_1.shape)
    hidden_layer_1 *= dropout_mask_1
    hidden_layer_2 = np.maximum(0, np.dot(W2, hidden_layer_1) + b2)
    dropout_mask_2 = np.random.binomial(1, keep_prob, hidden_layer_2.shape)
    hidden_layer_2 *= dropout_mask_2
    out = np.dot(W3, hidden_layer_2) + b3

def predict(X):
    # ensembled forward pass
    hidden_layer_1 = np.maximum(0, np.dot(W1, X) + b1) * keep_prob
    hidden_layer_2 = np.maximum(0, np.dot(W2, hidden_layer_1) + b2) * keep_prob
    out = np.dot(W3, hidden_layer_2) + b3
```

Inverted Dropout

```
def train_step(X):
    hidden_layer_1 = np.maximum(0, np.dot(W1, X) + b1)
    dropout_mask_1 = np.random.binomial(1, keep_prob, hidden_layer_1.shape) / keep_prob
    hidden_layer_1 *= dropout_mask_1
    hidden_layer_2 = np.maximum(0, np.dot(W2, hidden_layer_1) + b2)
    dropout_mask_2 = np.random.binomial(1, keep_prob, hidden_layer_2.shape) / keep_prob
    hidden_layer_2 *= dropout_mask_2
    out = np.dot(W3, hidden_layer_2) + b3

def predict(X):
    # ensembled forward pass
    hidden_layer_1 = np.maximum(0, np.dot(W1, X) + b1)
    hidden_layer_2 = np.maximum(0, np.dot(W2, hidden_layer_1) + b2)
    out = np.dot(W3, hidden_layer_2) + b3
```

Adversarial Examples



Training on adversarial examples is mostly intended to improve security, but can sometimes provide generic regularization.

http://karpathy.github.io/2015/03/30/breaking-convnets/

Adversarial Examples Intution

$$P(y=1 \mid x; w, b) = \sigma(w^T x + b), \text{ where } \sigma(z) = 1/(1+e^{-z})$$

Suppose

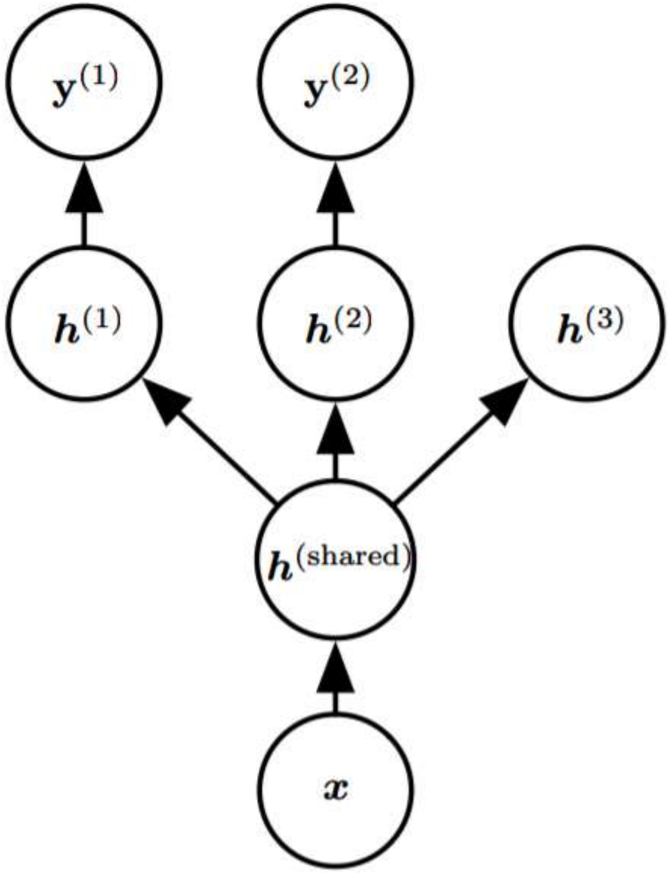
```
x = [2, -1, 3, -2, 2, 2, 1, -4, 5, 1] // input
w = [-1, -1, 1, -1, 1, -1, 1, 1, -1, 1] // weight vector
```

Hence, probability of class 1 is `1/(1+e^(-(-3))) = 0.0474`

We're now going to try to fool the classifier.

```
// xad = x + 0.5w gives:
xad = [1.5, -1.5, 3.5, -2.5, 2.5, 1.5, 1.5, -3.5, 4.5, 1.5]
```

Multitask Learning



Sparse representation:

Direct versus Representational Sparsity

$$\underset{\boldsymbol{y} \in \mathbb{R}^m}{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}} = \underset{\boldsymbol{A} \in \mathbb{R}^{m \times n}}{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}} \tag{7.46}$$

$$\underset{\boldsymbol{y} \in \mathbb{R}^m}{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}} = \underset{\boldsymbol{B} \in \mathbb{R}^{m \times n}}{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}} \underset{\boldsymbol{h} \in \mathbb{R}^n}{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}} \tag{7.47}$$

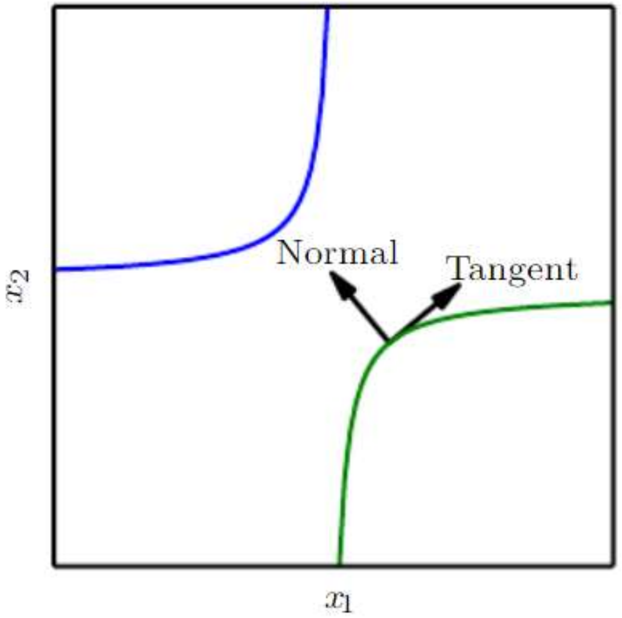
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

$$\Omega(\boldsymbol{h}) = ||\boldsymbol{h}||_1 = \sum_i |h_i|$$

Injecting noise

- Injecting noise into the input of a neural network can be seen as data augmentation
- Neural networks are not robust to noise
- To improve robustness, train them with random noise applied to their inputs
  - Part of some unsupervised learning, such as denoising autoencoder
- Noise can also be applied to hidden units
- Dropout, a powerful regularization strategy, can be viewed as constructing new inputs by multiplying by noise
- Noise applied to weights
- Injecting noise at the output targets

Tangent prop algorithm



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

END