Regularization for Deep Learning

Lecture slides for Chapter 7 of *Deep Learning*
www.deeplearningbook.org
Ian Goodfellow
2016-09-27
Adapted by m.n. for CMPS 392

Definition

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

- Developing more effective regularization strategies has been one of the major research efforts in the field.

- Deep learning take:
  - the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately!

Regularization strategies

- Constraints: adding restrictions on the parameter values.
- Soft constraints: Adding extra terms in the objective function:
  - Encode Prior knowledge.
  - Generic preference for a simpler model
- Ensemble methods:
  - Combine multiple hypotheses to explain the training data

Parameter norm penalties

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

- θ*:* all learnable parameters (weights and biases)
- *w:* parameters affected by a norm penalty
  - we take weights and exclude biases
- $\alpha \in [0, \infty)$
  - $\alpha = 0$, no regularization
- Ω: norm function
  - L1
  - L2

L^2 Parameter regularization
aka. ridge regression

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

- $\nabla_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{w} = \boldsymbol{w}$
- Update step: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\, \alpha\, \boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J$
  - $\boldsymbol{w} \leftarrow \boldsymbol{w}(1 - \epsilon\alpha) - \epsilon \nabla_{\boldsymbol{w}} J$
- Let $\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, J$
- Let $\tilde{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \tilde{J}$
- Approximating J in the neighborhood of $\boldsymbol{w}^*$:
  - $\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + (\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*)$

Weights are shrunk by a multiplicative factor

No first order term since w* is the minimum ($\nabla J(\boldsymbol{w}^*) = \boldsymbol{0}$)

How $\tilde{\boldsymbol{w}}$ (regularized solution) compares to unregularized solution $\boldsymbol{w}^*$?

- What is the gradient of $\hat{J}(\boldsymbol{w})$ at $\tilde{\boldsymbol{w}}$*?*
- $\nabla \hat{J}(\tilde{\boldsymbol{w}}) = \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*)$
- $\nabla \tilde{J}(\tilde{\boldsymbol{w}}) = \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) + \alpha\tilde{\boldsymbol{w}} = \boldsymbol{0}$
  - $(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$
  - $\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*$
- $\boldsymbol{H}$ is real and symmetric
  - $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}}$
- $\tilde{\boldsymbol{w}} = \left(\boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}} + \alpha\boldsymbol{I}\right)^{-1}\left(\boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}}\right)\mathbf{w}^* = \left(\boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}} + \boldsymbol{Q}\alpha\boldsymbol{I}\boldsymbol{Q}^{\boldsymbol{T}}\right)^{-1}\left(\boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}}\right)\mathbf{w}^*$
- $\tilde{\boldsymbol{w}} = \left(\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^{\boldsymbol{T}}\right)^{-1}\left(\boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}}\right)\mathbf{w}^* = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{\Lambda}\mathbf{Q}^{\mathbf{T}}\mathbf{w}^*$

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

Interpretation

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

- $\mathbf{w}^{*}$ projections against the eigen vectors of $\boldsymbol{H}$ are scaled
  - Component i is multiplied by $\frac{\lambda_i}{\lambda_i + \alpha}$
  - $\lambda_i \gg \alpha \Rightarrow$ the effect of regularization is small
  - $\lambda_i \ll \alpha \Rightarrow$ the corresponding component is shrunk by a factor of α

Weight Decay

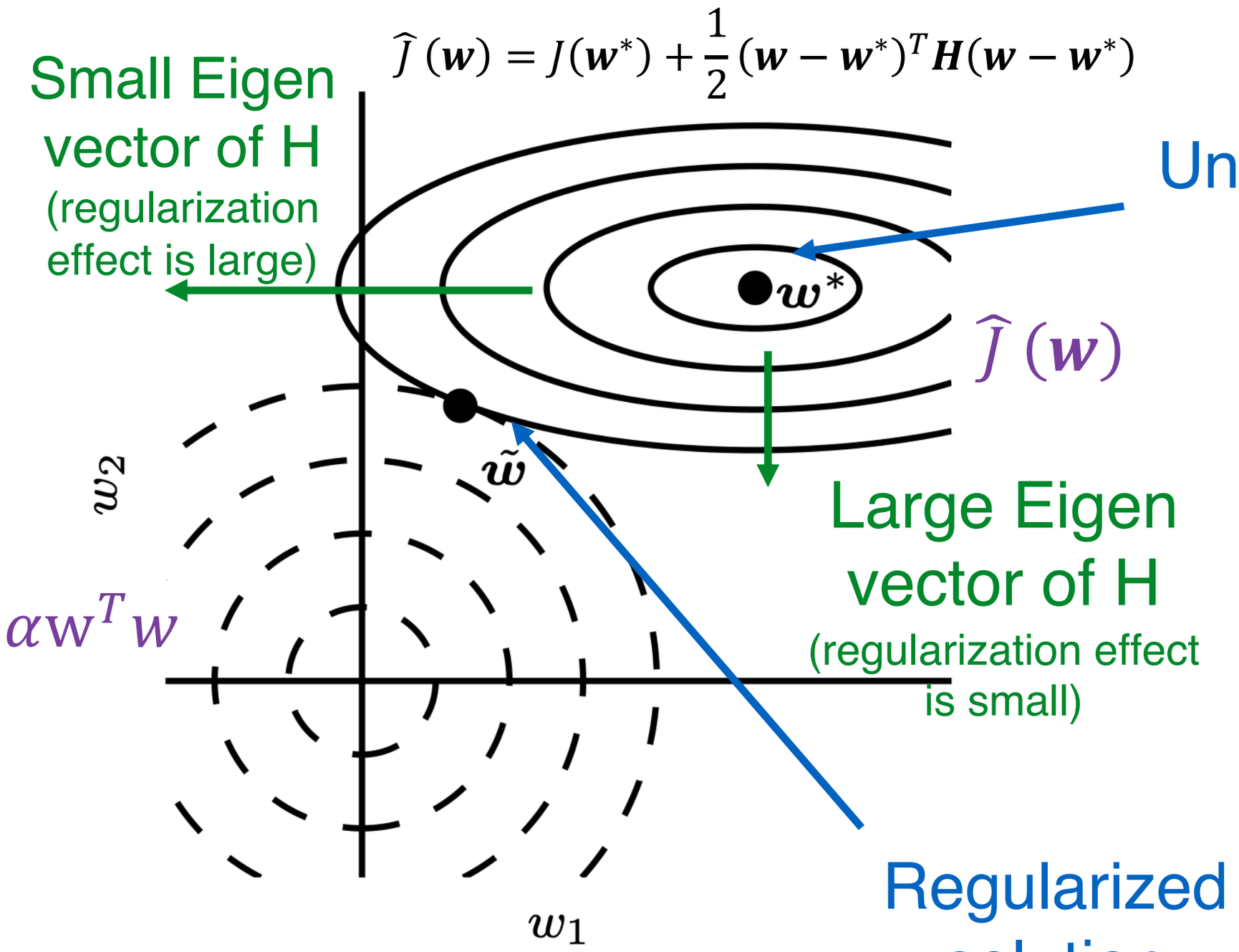


Figure 7.1

Special case: Linear Regression

- Cost function: $(\boldsymbol{Xw} - \boldsymbol{y})^T(\boldsymbol{Xw} - \boldsymbol{y}) + \frac{1}{2}\alpha \boldsymbol{w^T w}$
- *Normal equations*
  - $\boldsymbol{X^T X w} - \boldsymbol{X^T y} + \alpha \boldsymbol{w} = \boldsymbol{0} \Rightarrow (\boldsymbol{X^T X} + \alpha \boldsymbol{I})\boldsymbol{w} = \boldsymbol{X^T y}$
  - $\boldsymbol{w} = \left(\boldsymbol{X^T X} + \alpha \boldsymbol{I}\right)^{-1} \boldsymbol{X^T y}$ ← Covariance feature-output
  - Proportional to the covariance matrix
- Basically, we are adding α to the diag.
  - The diag. elements correspond to the variance of each feature
- We perceive the data as having higher variance
  - A feature having low covariance with output got shrunk even more due to this added variance

L^1 regularization

- $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$
- $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$
  - $\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\, \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$
- $\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$
  - $\nabla \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$
- Assume that $\boldsymbol{H} = \text{diag}([H_{1,1}, \ldots, H_{n,n}])$, $H_{i,i} > 0$
  - Linear regression after PCA
- $\tilde{J}(\boldsymbol{w}) \approx J(\boldsymbol{w}^*) + \sum_i \left[\frac{1}{2} H_{i,i}(w_i - w_i^*)^2 + \alpha|w_i|\right]$
- Solution: $w_i = sign(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$

Interpretation

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

- If $w_i^* > 0$:
  - $w_i^* > \frac{\alpha}{H_{i,i}} \Rightarrow w_i$ is shifted towards 0 by $\frac{\alpha}{H_{i,i}}$
  - $w_i^* \leq \frac{\alpha}{H_{i,i}} \Rightarrow w_i = 0$
- If $w_i^* < 0$:
  - $-w_i^* > \frac{\alpha}{H_{i,i}} \Rightarrow w_i = w_i + \frac{\alpha}{H_{i,i}}$
    - w_i is shifted towards 0 by $\frac{\alpha}{H_{i,i}}$
  - $-w_i^* \leq \frac{\alpha}{H_{i,i}} \Rightarrow w_i = 0$

L^1 regularization sparsity

- The sparsity property induced by L1 regularization can be used as a feature selection mechanism
  - LASSO regression (least absolute shrinkage and selection operator)
- Equivalent to MAP Bayesian estimation with Laplace prior
  - the prior is an isotropic Laplace distribution over $w \in \mathbb{R}^n$:
    - $\text{Laplace}\left(w_i; 0, \frac{1}{\alpha}\right) = \frac{1}{2\alpha}\exp(-\alpha|w_i|)$
    - $\log \text{Laplace}\left(w_i; 0, \frac{1}{\alpha}\right) = -\log 2\alpha - \alpha|w_i|$

$$\log posterior \propto \log likelihood + \log prior$$

$$\max \log posterior \Leftrightarrow \min\left(\text{negative log likelihood} - \log \text{prior}\right)$$

Norm Penalties

- MAP: Maximum A-Posteriori
- L1:
  - Encourages sparsity,
  - equivalent to MAP Bayesian estimation with Laplace prior
- Squared L2:
  - Encourages small weights,
  - equivalent to MAP Bayesian estimation with Gaussian prior

Explicit constraints

- We want to constrain $\Omega(\theta)$ to be less than some constant k
  - construct a generalized Lagrange function

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k).$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \alpha).$$

- We can fix α but lose k

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha^* \Omega(\boldsymbol{\theta}).$$

- The regularized training problem $\tilde{J}$ is equivalent to the explicit constraints problem for an unknown k!

Projection

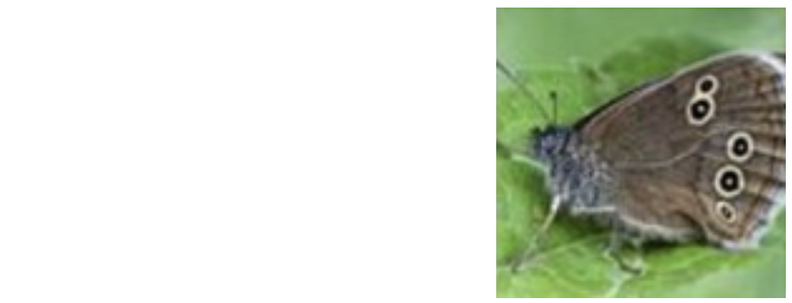
- Sometimes we may wish to use explicit constraints rather than penalties.
  - we can modify algorithms such as stochastic gradient descent to take a step downhill on $J(\theta)$ and then project θ back to the nearest point that satisfies $\Omega(\theta) < k$.
- How to project?
  - Project into unit L2 ball: $\mathbf{x} = \frac{\mathbf{y}}{\max\{1, \|\mathbf{y}\|_2\}}$.
  - Project into unit L1 ball:
    - No closed-form solution
    - Numerical solution

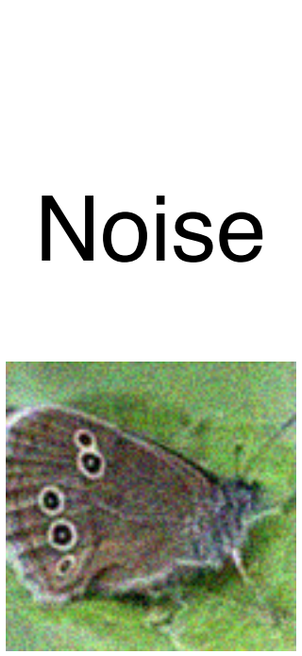
Dataset Augmentation

- Best way to regularize is to train with more data
  - create fake data and add it to the training set.
  - We can generate new $(\boldsymbol{x}, y)$ pairs easily just by transforming the $\boldsymbol{x}$ inputs in our training set.
    - particularly effective for *object recognition*
      - translating the training images a few pixels in each direction
      - rotating the image or scaling
- Some inappropriate transformations:
  - horizontal flips: ‘b’ and ‘d’,
  - 180◦ rotations: ‘6’ and ‘9’,

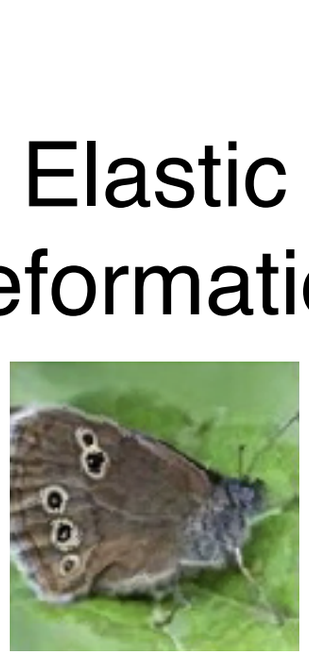
Dataset Augmentation

Horizontal flip





Random Translation



Hue Shift

(Goodfellow 2016)

Noise Robustness

- Noise with infinitesimal variance can be added:
  - At the input
  - At the hidden layers
  - At the weights: $\epsilon_{\boldsymbol{W}} \sim \mathcal{N}(\epsilon; \mathbf{0}, \eta \boldsymbol{I})$

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x}, y, \epsilon_{\boldsymbol{W}})} \left[(\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - y)^2 \right]$$
$$= \mathbb{E}_{p(\boldsymbol{x}, y, \epsilon_{\boldsymbol{W}})} \left[\hat{y}^2_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - 2y\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) + y^2 \right]$$

Injecting noise at the weights

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon_W})}\left[(\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) - y)^2\right]$$

$$= \mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon_W})}\left[\hat{y}^2_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) - 2y\hat{y}_{\boldsymbol{\epsilon W}}(\boldsymbol{x}) + y^2\right]$$

- For η small:
  - $\hat{y}_{\epsilon_{\boldsymbol{W}}} = \hat{y}(\boldsymbol{W} + \boldsymbol{\epsilon}) = \hat{y}(\boldsymbol{W}) + \boldsymbol{\epsilon}^T \nabla_{\boldsymbol{w}} \hat{y}(\boldsymbol{W})$
  - $\mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon_W})}\left[\hat{y}^2_{\epsilon_{\boldsymbol{W}}}\right] = \mathbb{E}_{p(\boldsymbol{x},y)}[\hat{y}^2] +$ $\mathbb{E}_{p(\boldsymbol{\epsilon_W})}[\epsilon^2]\mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_{\boldsymbol{w}}\hat{y}\|^2] + 0$
  - $\mathbb{E}_{p(\boldsymbol{x},y,\boldsymbol{\epsilon_W})}\left[\hat{y}^2_{\epsilon_{\boldsymbol{W}}}\right] = \mathbb{E}_{p(\boldsymbol{x},y)}[\hat{y}^2] + \eta\mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_w \hat{y}\|^2]$
  - $\tilde{J}_{\boldsymbol{W}} = J + \eta\ \mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_w \hat{y}(\boldsymbol{x})\|^2]$
  - Equivalent to adding a regularization term
- Pushes the model into regions where the model is relatively insensitive to small variations in the weights

Special case: linear regression

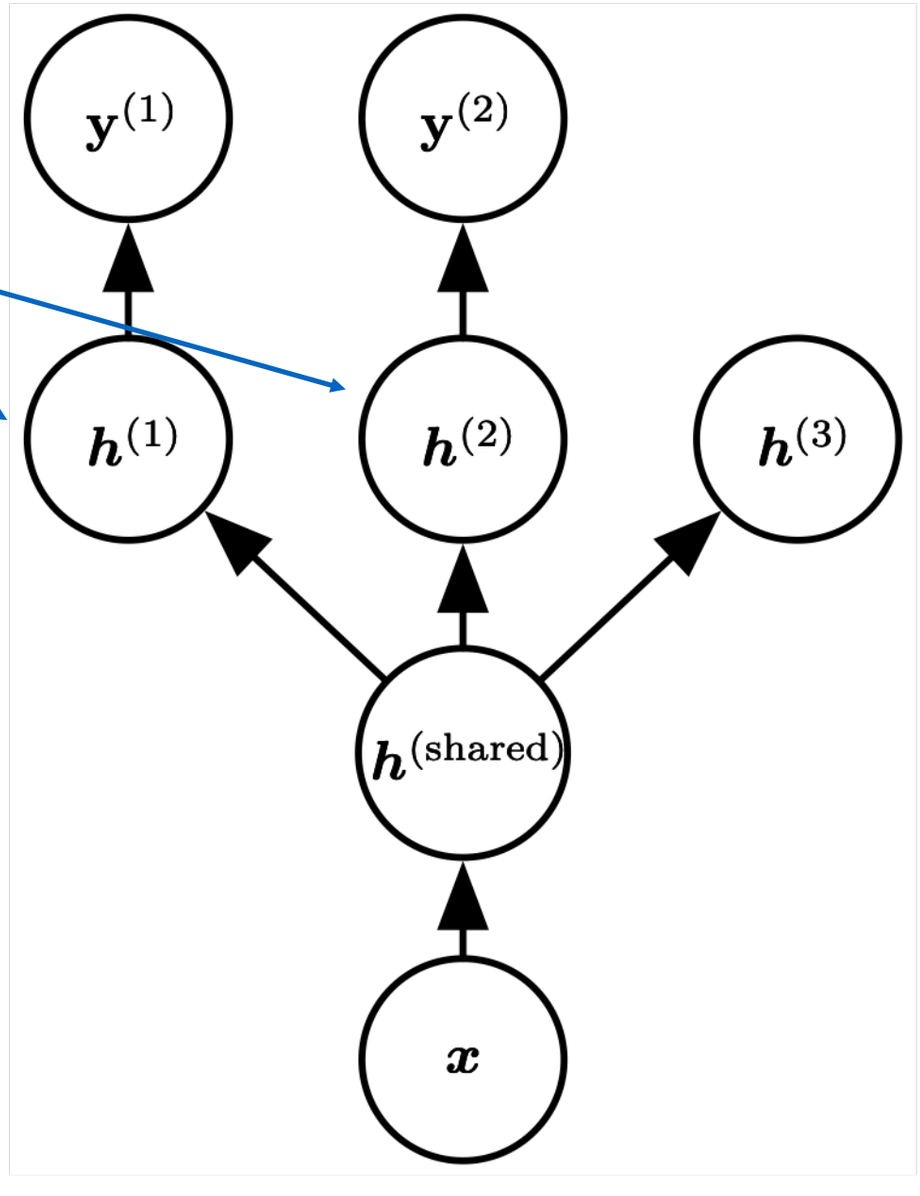
- $\tilde{J}_{\boldsymbol{W}} = J + \eta\ \mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_w \hat{y}(\boldsymbol{x})\|^2]$
- $\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$
- $\mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_{\boldsymbol{w}} \hat{y}(x)\|^2] = \mathbb{E}_{p(\boldsymbol{x})}[\|\boldsymbol{x}\|^2]$
- which is not a function of parameters and therefore does not contribute to the cost function w.r.t $\boldsymbol{w}$:
  - No regularization effect!

Injecting noise at the output targets

- Most datasets have some amount of mistakes in the y labels.
- It can be harmful to maximize $\log p(y \mid x)$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels.
  - For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1 - \epsilon$,
  - and otherwise any of the other possible labels might be correct.
- This assumption is easy to incorporate into the cost function analytically,
  - rather than by explicitly drawing noise samples.
  - For example, **label smoothing** regularizes a model based on a softmax with k output values
    - by replacing the hard 0 by $\frac{\epsilon}{k-1}$
    - and 1 by $1 - \epsilon$
- Label smoothing has the advantage of preventing the pursuit of hard probabilities without discouraging correct classification.

Multi-Task Learning

Task specific parameters



Unsupervised Learning context

Among the factors that explain the variations observed in the data associated with the different tasks, some are shared across two or more tasks.

Figure 7.2

Learning Curves

Early stopping: terminate while validation set performance is better

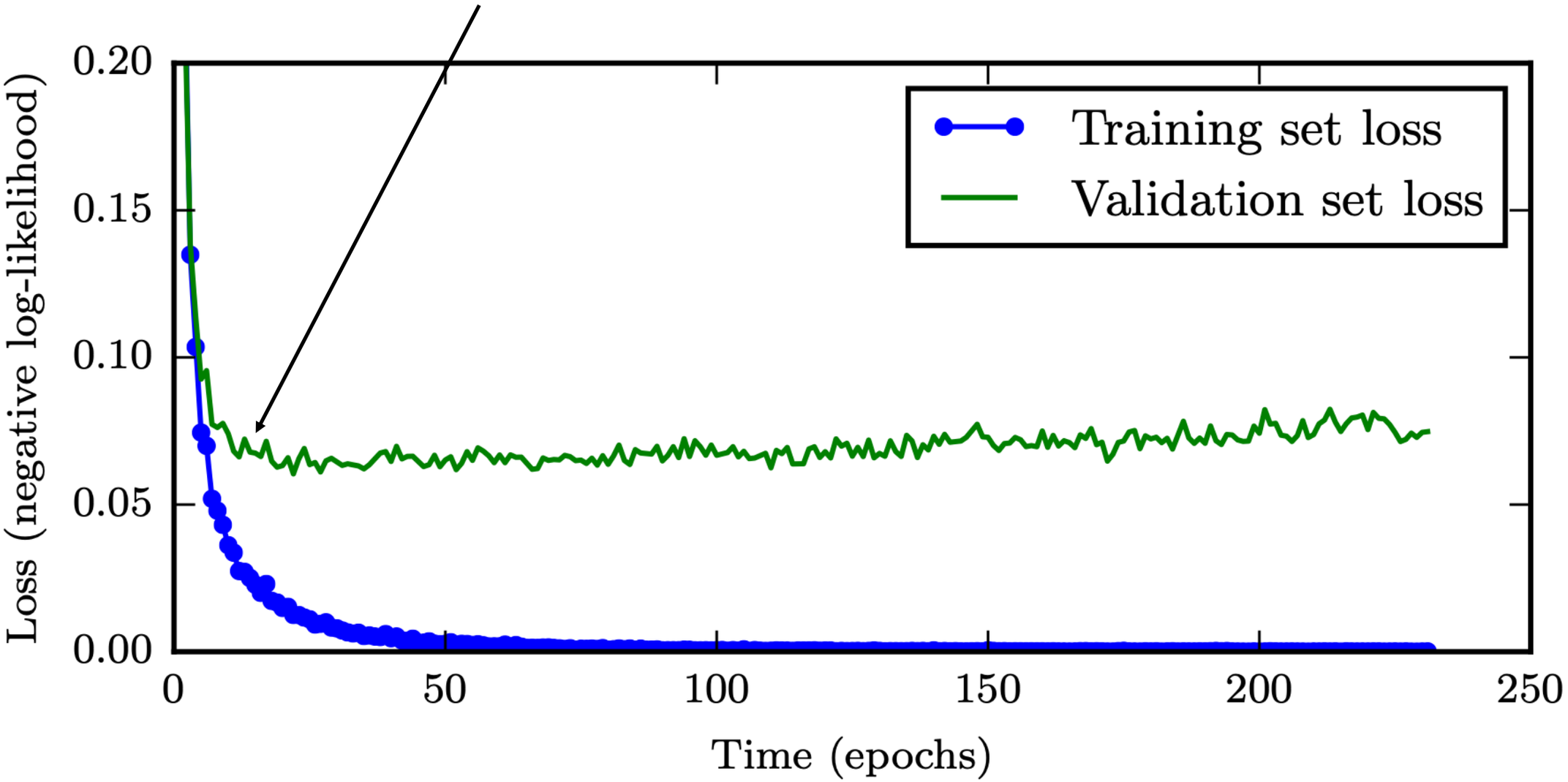


Figure 7.3

Early stopping

- probably the most commonly used form of regularization in deep learning.
  - the number of training steps (or training time) is just another hyperparameter.
- The cost is running the validation set evaluation periodically during training
  - Reduce the validation set
  - Evaluate the validation loss less frequently
- Periodically save the trained model

Early stopping algorithm

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Re-use the validation set

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Less well-behaved →

Algorithm 7.3 Meta-algorithm using early stopping to determine at what objective value we start to overfit, then continue training until that value is reached.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This updates $\boldsymbol{\theta}$.
$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$
while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ **do**
  Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for n steps.
end while

Early stopping as a regularizer

- ϵ (learning rate) and τ (number of training steps) limits the the volume of parameters reachable from $\boldsymbol{\theta_0}$ (initial parameters)
- Early stopping is equivalent to L2 regularization in the case of:
  - a simple linear model
  - with a quadratic error function
  - and simple gradient descent
- $\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$
  - $\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$
- $\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}\left(\boldsymbol{w}^{(\tau-1)}\right) = \boldsymbol{w}^{(\tau-1)} - \epsilon\, \boldsymbol{H}\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right)$
  - $\boldsymbol{w}^{(\tau)} = (\boldsymbol{I} - \epsilon \boldsymbol{H})\boldsymbol{w}^{(\tau-1)} + \epsilon \boldsymbol{H} \boldsymbol{w}^*$
  - $\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})\boldsymbol{w}^{(\tau-1)} + (\epsilon \boldsymbol{H} - \boldsymbol{I})\boldsymbol{w}^*$
  - $\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \mathbf{w}^*)$

The number of steps τ corresponds to some value of the weight decay coefficient α

- $\boldsymbol{w^{(\tau)} - w^* = (I - \epsilon H)(w^{(\tau-1)} - \mathbf{w}^*)}$
  - $\boldsymbol{H = Q\Lambda Q^T}$
  - $\boldsymbol{w^{(\tau)} - w^* = Q(I - \epsilon\Lambda)Q^T\left(w^{(\tau-1)} - \mathbf{w}^*\right)}$
  - $\boldsymbol{Q^T\left(w^{(\tau)} - w^*\right) = (I - \epsilon\Lambda)Q^T\left(w^{(\tau-1)} - \mathbf{w}^*\right)}$
    - if ϵ small $\Rightarrow |1 - \epsilon\lambda_i| < 1$, every step brings closer to $\boldsymbol{w^*}$
- Assume we start with $\boldsymbol{w^{(0)} = 0}$:
  - $\boldsymbol{Q^T w^{(1)} = \left(I - (I - \epsilon\Lambda)\right)Q^T w^*}$
  - $\boldsymbol{Q^T w^{(2)} = \left(I - (I - \epsilon\Lambda)^2\right)Q^T w^*}$
  - $\boldsymbol{Q^T w^{(\tau)} = (I - (I - \epsilon\Lambda)^\tau)Q^T w^*}$
- L2 regularization:
  - $\boldsymbol{Q^T\tilde{w} = (\Lambda + \alpha I)^{-1}\Lambda \mathbf{Q^T w}^*}$
  - $\boldsymbol{Q^T\tilde{w} = I - (\Lambda + \alpha I)^{-1}\alpha\, \mathbf{Q^T w}^*}$

$$\frac{\lambda_i}{\lambda_i + \alpha} = 1 - \frac{\alpha}{\lambda_i + \alpha}$$

- Compare:
  - $\boldsymbol{I - (\Lambda + \alpha I)^{-1}\alpha}$ and $\boldsymbol{I - (I - \epsilon\Lambda)^\tau}$

$$1 - (1 - \epsilon\lambda_i)^\tau$$

  - $\boldsymbol{(\Lambda + \alpha I)^{-1}\alpha = (I - \epsilon\Lambda)^\tau}$

Early stopping advantage

- $\frac{\alpha}{\lambda_i+\alpha} = (1-\epsilon\lambda_i)^\tau \Rightarrow \tau \log(1-\epsilon\lambda_i) = \log\left(\frac{1}{1+\frac{\lambda_i}{\alpha}}\right)$
- Assume $\log(1+x) \approx x$ for small enough x
  - Assume $\frac{\lambda_i}{\alpha} \ll 1$ and $\epsilon\lambda_i \ll 1$
- $-\tau\epsilon\lambda_i \approx -\frac{\lambda_i}{\alpha_i} \Rightarrow \alpha \approx \frac{1}{\tau\epsilon}$
  - the number of training iterations τ plays a role inversely proportional to the L2 regularization parameter,
  - and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient.
- Early stopping advantage over weight decay:
  - early stopping automatically determines the correct amount of regularization
  - while weight decay requires many training experiments with different values of its hyperparameter.

Early Stopping and Weight Decay

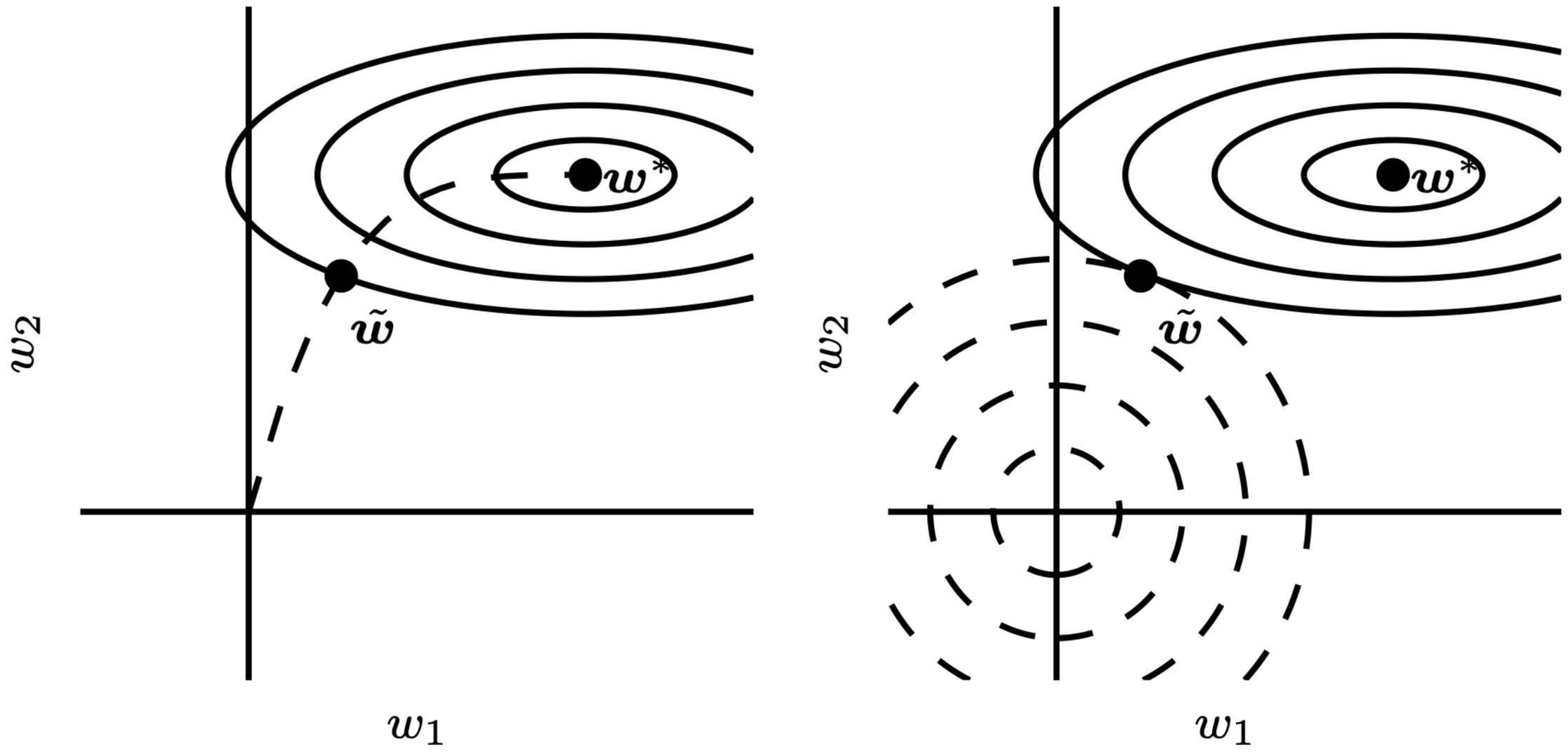


Figure 7.4

Parameter tying

- Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$
- The two models map the input to two different, but related outputs:
  - $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$
  - $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$
  - $\forall i, \boldsymbol{w}^{(A)}$ should be close to $\boldsymbol{w}^{(B)}$
- Regularization

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

Parameter sharing (e.g. CNN)

- Force sets of parameters to be equal.
- Advantage:
  - only a subset of the parameters (the unique set) needs to be stored in memory.
- Natural images have many statistical properties that are invariant to translation.
  - a photo of a cat remains a photo of a cat if it is translated one pixel to the right
  - Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters

Sparse Representations

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{A} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{x} \in \mathbb{R}^n$

Sparse parameters

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

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

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

Sparse representations

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

Bagging

- Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models
  - train several different models separately
  - the models vote on the output for test examples
- Bagging is an example of model averaging.
  - The general term is Ensemble methods.
- The reason that model averaging works is that different models will usually not make all the same errors on the test set.

Bagging example

- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a zero-mean multivariate normal distribution
  - with variances $E[\epsilon_i^2] = v$
  - and covariances $E[\epsilon_i \epsilon_j] = c$
- Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$
- The expected squared error of the ensemble predictor is:

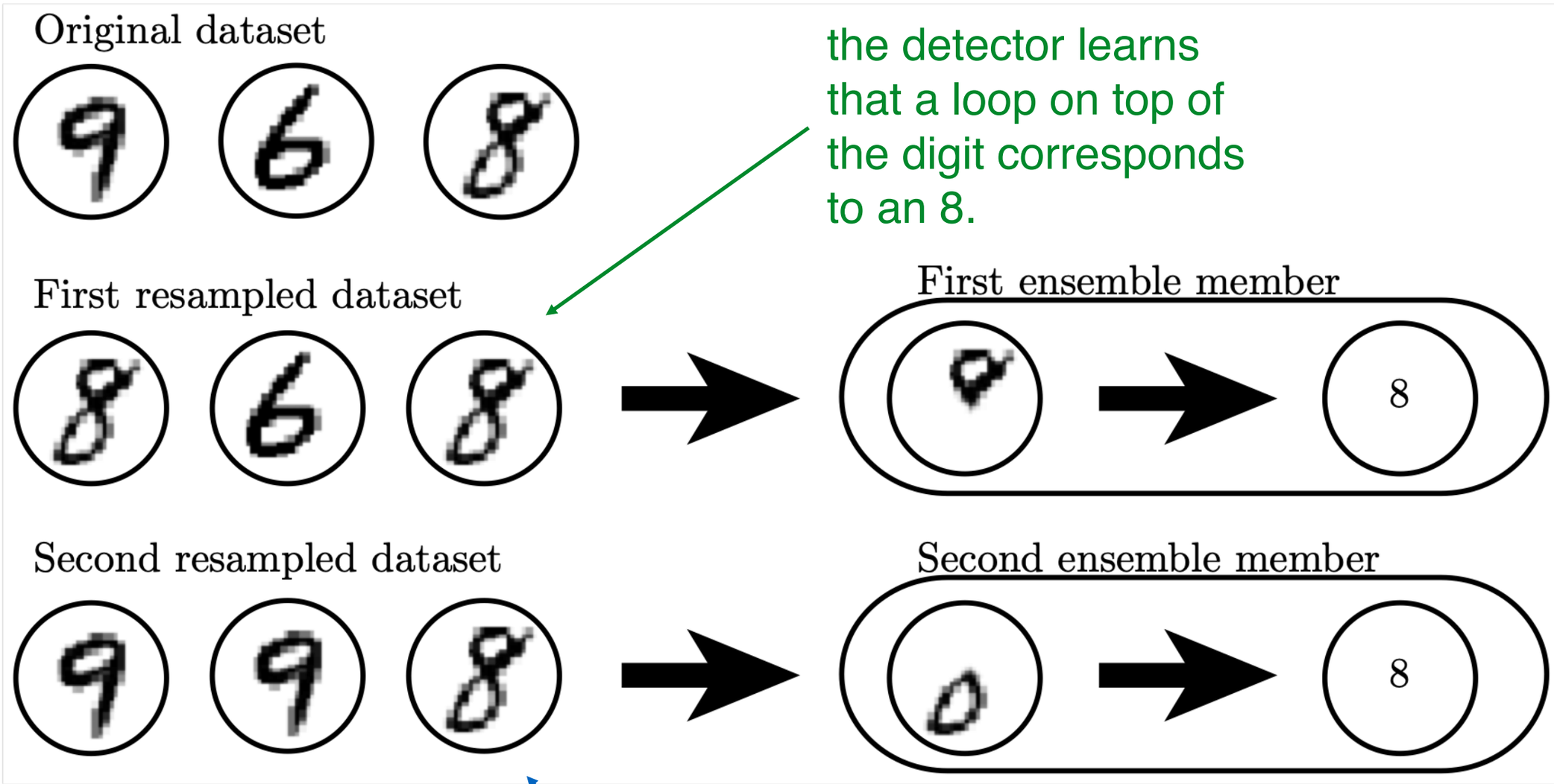
$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c.$$

  - $c = v \Rightarrow$ no gain, the expected error remains v
  - $c = 0 \Rightarrow$ max gain, the expected error is v/k

Ensemble methods vs. bagging

- Different ensemble methods construct the ensemble of models in different ways.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times
- Bagging involves constructing k different datasets.
  - Each dataset has the same number of examples as the original dataset,
  - but each dataset is constructed by sampling with replacement from the original dataset.
    - with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples
    - on average around 2/3 of the examples from the original dataset are found in the resulting training set, if it has the same size as the original

Bagging


Original dataset
the detector learns
that a loop on top of
the digit corresponds
to an 8.
First resampled dataset
First ensemble member
8
Second resampled dataset
Second ensemble member
8
the detector learns
that a loop on the bottom
of the digit corresponds to
an 8.

Why 2/3 ?

- *N: number of items*
- *K: number of unique items*
- *A: number of drawn items*

$$P(k) = \frac{\frac{N!}{(N-k)!}}{N^A} \begin{Bmatrix} A \\ k \end{Bmatrix}$$

All permutations of k among N items

All ways to distribute A items among k subsets such as no subset is left empty

All possible ways to draw A items

- $\begin{Bmatrix} A \\ k \end{Bmatrix}$ is a Stirling number of the second kind

Expected number of duplicates

- The indicator d_i corresponds to original item i, taking the value of one if i is present and zero if not
- $P(d_i = 0) = 1 - \left(\frac{1}{N}\right)^A$
- $E[d_i] = 1 - \left(1 - \frac{1}{N}\right)^A$
- $E[\sum d_i] = \sum E[d_i] = NE[d_i] = N\left(1 - \left(1 - \frac{1}{N}\right)^A\right)$
- $A = N \Rightarrow E[k] = N\left(1 - \left(1 - \frac{1}{N}\right)^N\right) \rightarrow N(1 - e^{-1})$
- $E[k] \approx 0.632\ N$

More about bagging

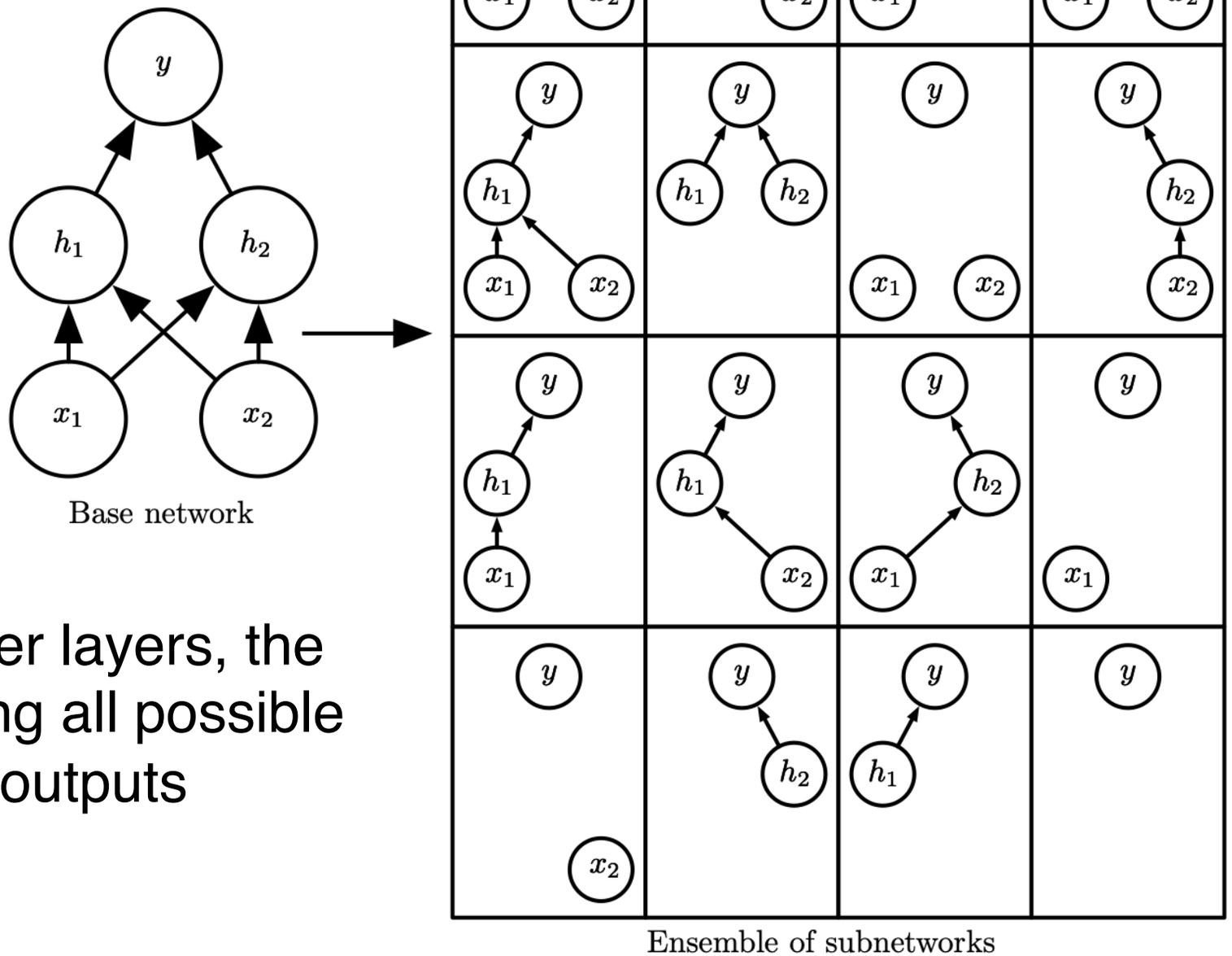
- Neural networks reach a wide enough variety of solution points that they can often benefit from model averaging
- Model averaging is an extremely powerful and reliable method for reducing generalization error.
  - Its use is usually discouraged when benchmarking algorithms for scientific papers
- Machine learning contests are usually won by methods using model averaging over dozens of models.

Dropout

- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.
  - removing non-output units from an underlying base network
    - by multiplying its output value by zero
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
  - The probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins.
    - Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5

Dropout

Figure 7.6



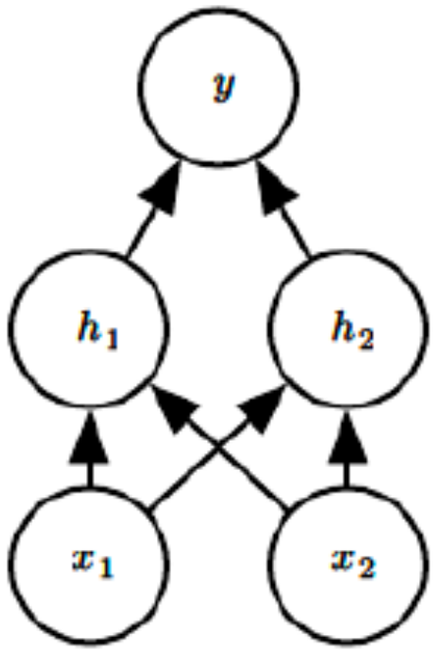
In networks with wider layers, the probability of dropping all possible paths from inputs to outputs becomes smaller.

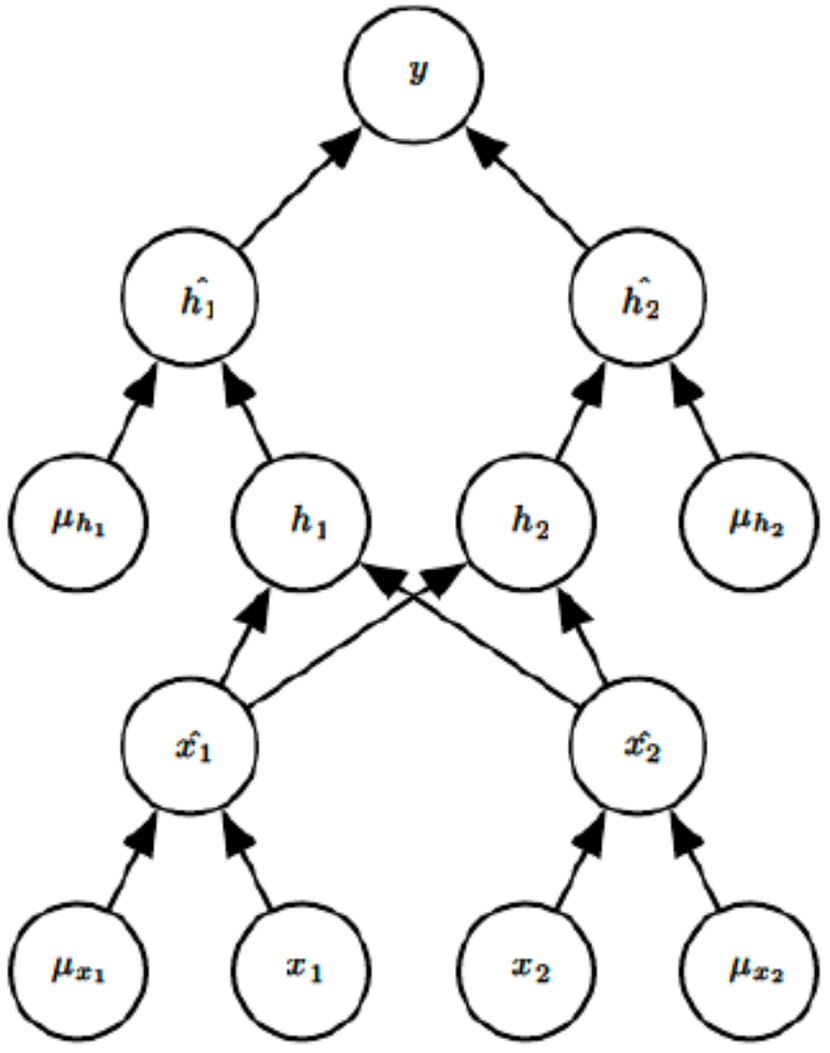
(Goodfellow 2016)

Dropout vs. bagging

- More formally, suppose that a mask vector $\boldsymbol{\mu}$ specifies which units to include, and $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$.
  - Then dropout training consists in minimizing $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$.
  - The expectation contains exponentially many terms (2^d)
- Dropout training is not quite the same as bagging training.
  - In the case of bagging, the models are all independent.
  - In the case of dropout, the models share parameters
  - In bagging, each model is trained to convergence on its respective training set
  - In dropout, a tiny fraction of the possible sub-networks are each trained for a single step
  - In both, the training set encountered by each sub-network is a subset of the original training set sampled with replacement

Computational graph of dropout





- The entries of $\boldsymbol{\mu}$ are binary and are sampled independently from each other,
  - And is not a function of the current value of the model parameters or the input example

Inference

- To make a prediction, a bagged ensemble must accumulate votes from all of its members.
  - We refer to this process as inference
- In bagging, the prediction of the ensemble is $\frac{1}{\mathrm{k}}\sum_{i=1}^{k} p^{(i)}(y|\boldsymbol{x})$
- In dropout, the arithmetic mean is $\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y|\boldsymbol{x}, \boldsymbol{\mu})$
- The geometric mean is

$$\tilde{p}_{\text{ensemble}}(y|\boldsymbol{x}) = \sqrt[2^n]{\prod_{\boldsymbol{\mu}} p(y|\boldsymbol{x}, \boldsymbol{\mu})}$$

- To guarantee that the result is a probability distribution,
  - we impose that none of the sub-models assigns probability 0 to any event,
  - and we renormalize the resulting distribution.

Weight scaling inference rule

- Evaluate with the trained model with all units,
  - But with the weights going out of unit i multiplied by the probability of including unit i (e.g. ½)
  - This corresponds to predict the geometric mean of the ensemble!
- Consider a softmax regression classifier with n input variables represented by the vector v:

$$P(y = y_i|\boldsymbol{v}) = \text{softmax}\left(\boldsymbol{W^T v} + \boldsymbol{b}\right)_i$$

- To index into the family of submodels:

$$P(y = y_i|\boldsymbol{v}) = \text{softmax}\left(\boldsymbol{W^T}(\boldsymbol{v} \odot \boldsymbol{d}) + \boldsymbol{b}\right)_i$$

$$\tilde{p}_{\text{ensemble}}\,(y = y_i|\boldsymbol{x}) = \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} \text{softmax}(\boldsymbol{W^T}(\boldsymbol{v} \odot \boldsymbol{d}) + \boldsymbol{b})_i}$$

$$\tilde{p}_{\text{ensemble}}\,(y = y_i|\boldsymbol{x}) \propto \exp\left(\frac{1}{2^n}\sum_{\boldsymbol{d}\in\{0,1\}^n}\left(\boldsymbol{W^T}(\boldsymbol{v} \odot \boldsymbol{d}) + \boldsymbol{b}\right)_i\right)$$

$$= \exp\left(\frac{1}{2^n}\left(2^{n-1}\,\boldsymbol{W^T v} + 2^n \boldsymbol{b}\right)_i\right) = \exp\left(\frac{1}{2}\boldsymbol{W^T v} + \boldsymbol{b}\right)_i$$

Another perspective of dropout

- (1) Droput is bagging with parameter sharing
- (2) Information erasing: Each hidden unit must be able to perform well regardless of which other hidden units are in the model
  - Dropout thus regularizes each hidden unit to be not merely a good feature but a feature that is good in many contexts.
  - For example, if the model learns a hidden unit h_i that detects a face by finding the nose,
  - then dropping h_i corresponds to erasing the information that there is a nose in the image.
  - The model must learn another h_i,
    - either that redundantly encodes the presence of a nose,
    - or that detects the face by another feature, such as the mouth

Adversarial examples

- Search for an input x' near a data point x such that the model output is very different at x'
- In many cases, x' can be so similar to x that a human observer cannot tell the difference between the original example and the adversarial example,
  - but the network can make highly different predictions.
- Adversarial training
  - training on adversarially perturbed examples from the training set
- Adversarial examples are interesting in the context of regularization
  - because one can reduce the error rate on the original i.i.d. test set via adversarial training

Adversarial Examples

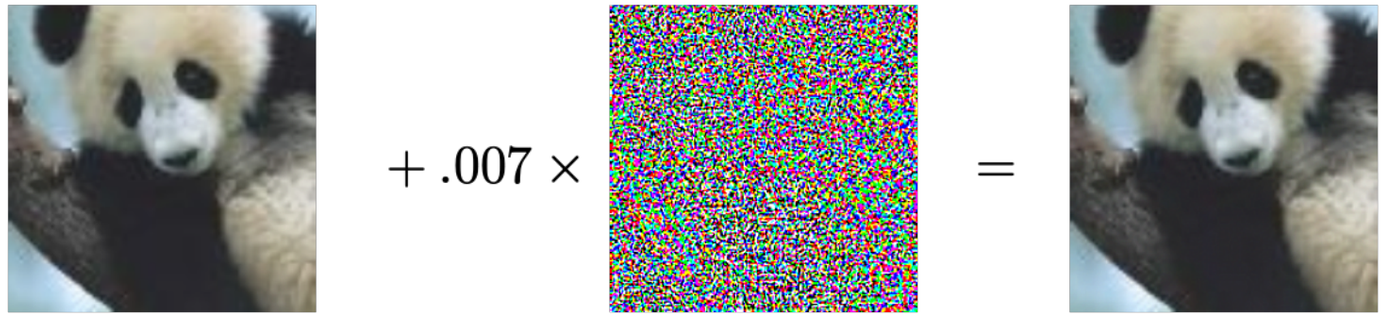


Figure 7.8

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

Aversarial training

- The value of a linear function can change very rapidly if it has numerous inputs.
  - If we change each input by ϵ, then a linear function with weights w can change by as much as $\epsilon \|\boldsymbol{w}\|_1$, which can be a very large amount if $\boldsymbol{w}$ is high-dimensional.
- Adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data.
- This can be seen as a way of explicitly introducing a local constancy prior into supervised neural nets.
  - The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}$'.
  - The assumption motivating this approach is that different classes usually lie on disconnected manifolds, and a small perturbation should not be able to jump from one class manifold to another class manifold.

Conclusion

- This chapter has described most of the general strategies used to regularize neural networks.
- Regularization is a central theme of machine learning

Our next topic is: optimization