

# Chapter 7

## Regularization for Deep Learning

## Regularization

---

- In machine learning, we optimize a cost function defined w.r.t. the training set

$$J(\boldsymbol{\theta}) = E_{\boldsymbol{x}, y \sim \hat{p}_{\text{data}}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

where

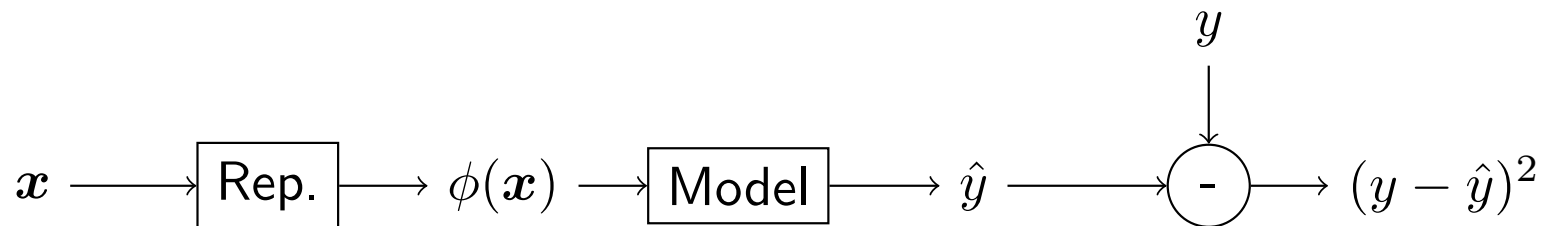
- $L$  is the per-example loss function
  - $f(\boldsymbol{x}; \boldsymbol{\theta})$  is the model prediction,
  - $y$  is the target output
  - $\hat{p}_{\text{data}}$  is the empirical distribution
- We however hope that doing so will minimize the expected loss over the true data-generating distribution  $p_{\text{data}}(\boldsymbol{x}, y)$

$$J^*(\boldsymbol{\theta}) = E_{\boldsymbol{x}, y \sim p_{\text{data}}} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$$

- If we knew the true distribution  $p_{\text{data}}(\mathbf{x}, y)$ , minimizing  $J^*(\boldsymbol{\theta})$  would become a pure *optimization problem*; but, when we do not know the true distribution but only the empirical distribution  $\hat{p}_{\text{data}}(\mathbf{x}, y)$  over the training data, we have a *machine learning problem*
- One central problem in machine learning is how to **make an algorithm work well** not just only on training data, but also **on new inputs**
- Strategies used to reduce test error, possibly at the expense of **increased training error**, are known collectively as **regularization**

## Revisiting Capacity, Underfitting and Overfitting

- To characterize analytically the relationship between a model's capacity and the phenomena of underfitting and overfitting when it is trained using the maximum likelihood
- Example: Linear Regression



- To predict  $y$  from  $x$ , we construct a model of the form

$$\hat{y}(x) = f(x; w) = w^T \phi(x)$$

and make a point estimate of the parameters  $w$  by minimizing

$$E_{x, y \sim \hat{p}_{\text{data}}} [(y - \hat{y}(x))^2]$$

- This is equivalent to maximizing the expected likelihood function  $E_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} p(y|\mathbf{x})$  by assuming the following data model

$$p(y|\mathbf{x}) = \mathcal{N}(y; \hat{y}(\mathbf{x}), \sigma^2)$$

- The optimal prediction which achieves the minimum expected (squared) generalization error

$$g^*(\mathbf{x}) = \arg \min_{g(\cdot)} E_{\mathbf{x}, y \sim p_{\text{data}}} [(y - g(\mathbf{x}))^2]$$

is given by the conditional mean

$$g^*(\mathbf{x}) = E_{y \sim p_{\text{data}}(y|\mathbf{x})}[y]$$

- The expected generalization error of the model  $\hat{y}(\mathbf{x})$  is then seen as the sum of Bayes error and the expected error between the optimal and the model predictions

$$\begin{aligned}
& E_{\mathbf{x}, y \sim p_{\text{data}}} [(y - \hat{y}(\mathbf{x}))^2] \\
&= E_{\mathbf{x}, y \sim p_{\text{data}}} [(y - g^*(\mathbf{x}) + g^*(\mathbf{x}) - \hat{y}(\mathbf{x}))^2] \\
&= \underbrace{E_{\mathbf{x}, y \sim p_{\text{data}}} [(y - g^*(\mathbf{x}))^2]}_{\text{Bayes error}} + E_{\mathbf{x} \sim p_{\text{data}}} [(g^*(\mathbf{x}) - \hat{y}(\mathbf{x}))^2]
\end{aligned}$$

where the cross-term

$$\begin{aligned}
& E_{\mathbf{x}, y \sim p_{\text{data}}} [2(y - g^*(\mathbf{x}))(g^*(\mathbf{x}) - \hat{y}(\mathbf{x}))] \\
&= E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} E_{y \sim p_{\text{data}}(y|\mathbf{x})} [2(y - g^*(\mathbf{x}))(g^*(\mathbf{x}) - \hat{y}(\mathbf{x}))] \\
&= E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [2E_{y \sim p_{\text{data}}(y|\mathbf{x})} [(y - g^*(\mathbf{x}))] (g^*(\mathbf{x}) - \hat{y}(\mathbf{x}))] \\
&= 0
\end{aligned}$$

- The Bayes error, which arises from the intrinsic noise on data, represents the minimum achievable value of the expected generalization error, and is independent of  $\hat{y}(\mathbf{x})$  and training data

- On the other hand, the expected error between the optimal and the model predictions has to do with training data because  $\hat{y}(\mathbf{x})$  is obtained by making a point estimate of  $\mathbf{w}$  based on a particular training data set  $\mathcal{D} = \{\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})}\}$

$$E_{\mathbf{x} \sim p_{\text{data}}}[(g^*(\mathbf{x}) - \hat{y}(\mathbf{x}))^2]$$

- Assume we are concerned with how the model performs over an ensemble of training data sets, and denote the model  $\hat{y}(\mathbf{x})$  trained with a particular data set  $\mathcal{D}$  as  $\hat{y}(\mathbf{x}; \mathcal{D})$
- For a given  $\mathbf{x}$ , we then evaluate the expected error between the optimal and model predictions w.r.t. the distribution of training data to be

$$\begin{aligned} & E_{\mathcal{D}}[(g^*(\mathbf{x}) - \hat{y}(\mathbf{x}; \mathcal{D}))^2] \\ &= E_{\mathcal{D}}[(g^*(\mathbf{x}) - E_{\mathcal{D}}[\hat{y}(\mathbf{x}; \mathcal{D})] + E_{\mathcal{D}}[\hat{y}(\mathbf{x}; \mathcal{D})] - \hat{y}(\mathbf{x}; \mathcal{D}))^2] \\ &= \underbrace{(g^*(\mathbf{x}) - E_{\mathcal{D}}(\hat{y}(\mathbf{x}; \mathcal{D})))^2}_{(\text{bias})^2} + \underbrace{E_{\mathcal{D}}[(E_{\mathcal{D}}[\hat{y}(\mathbf{x}; \mathcal{D})] - \hat{y}(\mathbf{x}; \mathcal{D}))^2]}_{\text{variance}} \end{aligned}$$

where the cross-term is again computed to be zero

- The  $(\text{bias})^2$  represents the extent to which the average model prediction over all training data sets differs from the optimal prediction
- The variance measures the extent to which the model  $y(\mathbf{x}; \mathcal{D})$  is sensitive to the particular choice of training set
- Both terms can be further averaged over different  $\mathbf{x}$ 's to obtain the expected generalization error of the model  $\hat{y}(\mathbf{x})$

$$\text{Bayes error} + E_{\mathbf{x} \sim p_{\text{data}}}[(\text{bias})^2] + E_{\mathbf{x} \sim p_{\text{data}}}[\text{variance}]$$

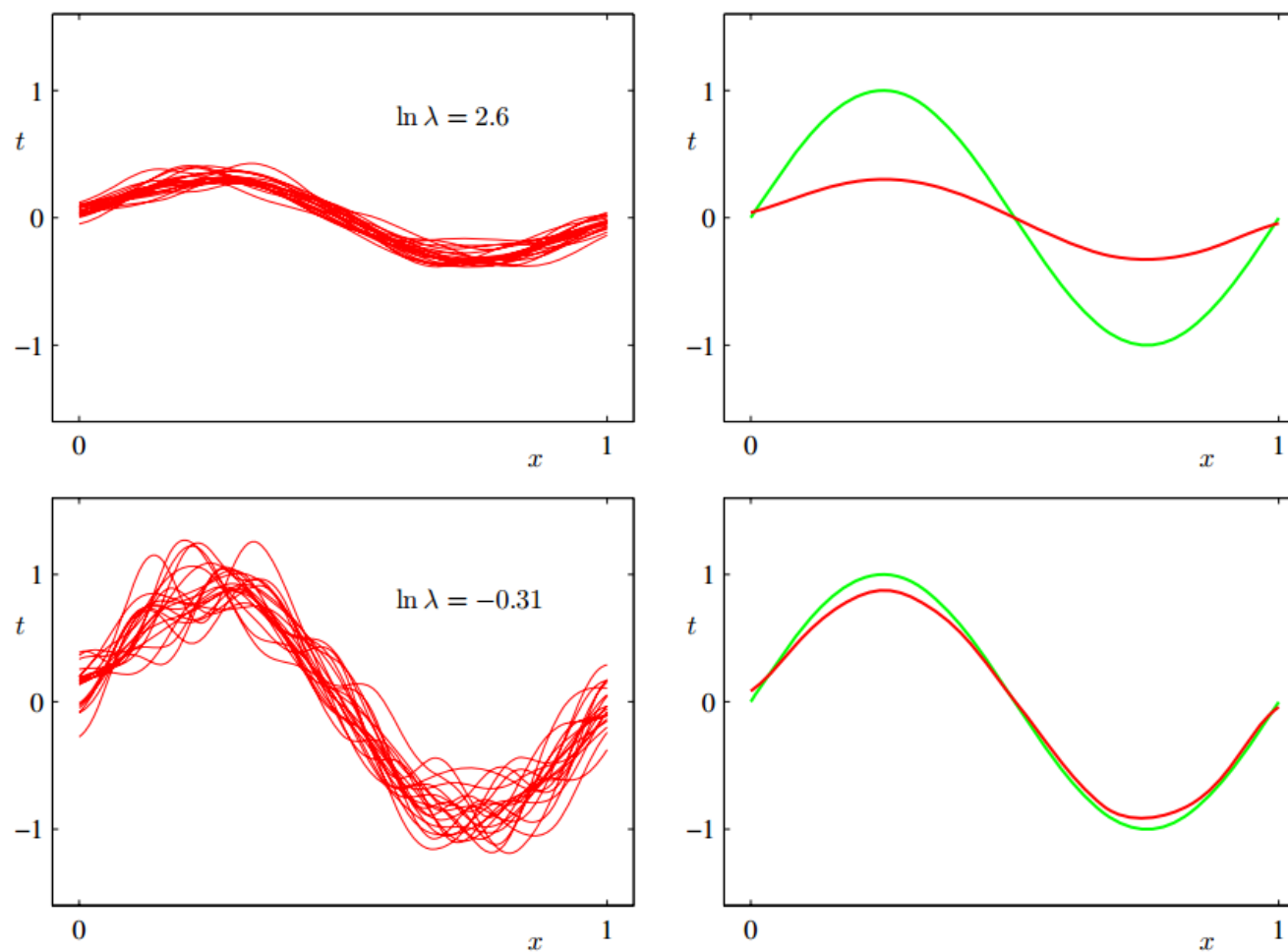


## Fitting Sinusoidal Functions

---

- Setting
  - Data:  $y = \sin(2\pi x) + \epsilon$ ,  $p(\epsilon) = \mathcal{N}(\epsilon; 0, \sigma^2)$
  - Model:  $\hat{y} = \mathbf{w}^T \boldsymbol{\phi}(x)$ ,  $\boldsymbol{\phi}(x)$  is Gaussian basis
  - 100 training data sets, each having 25 data points  $(x, y)$
- Training

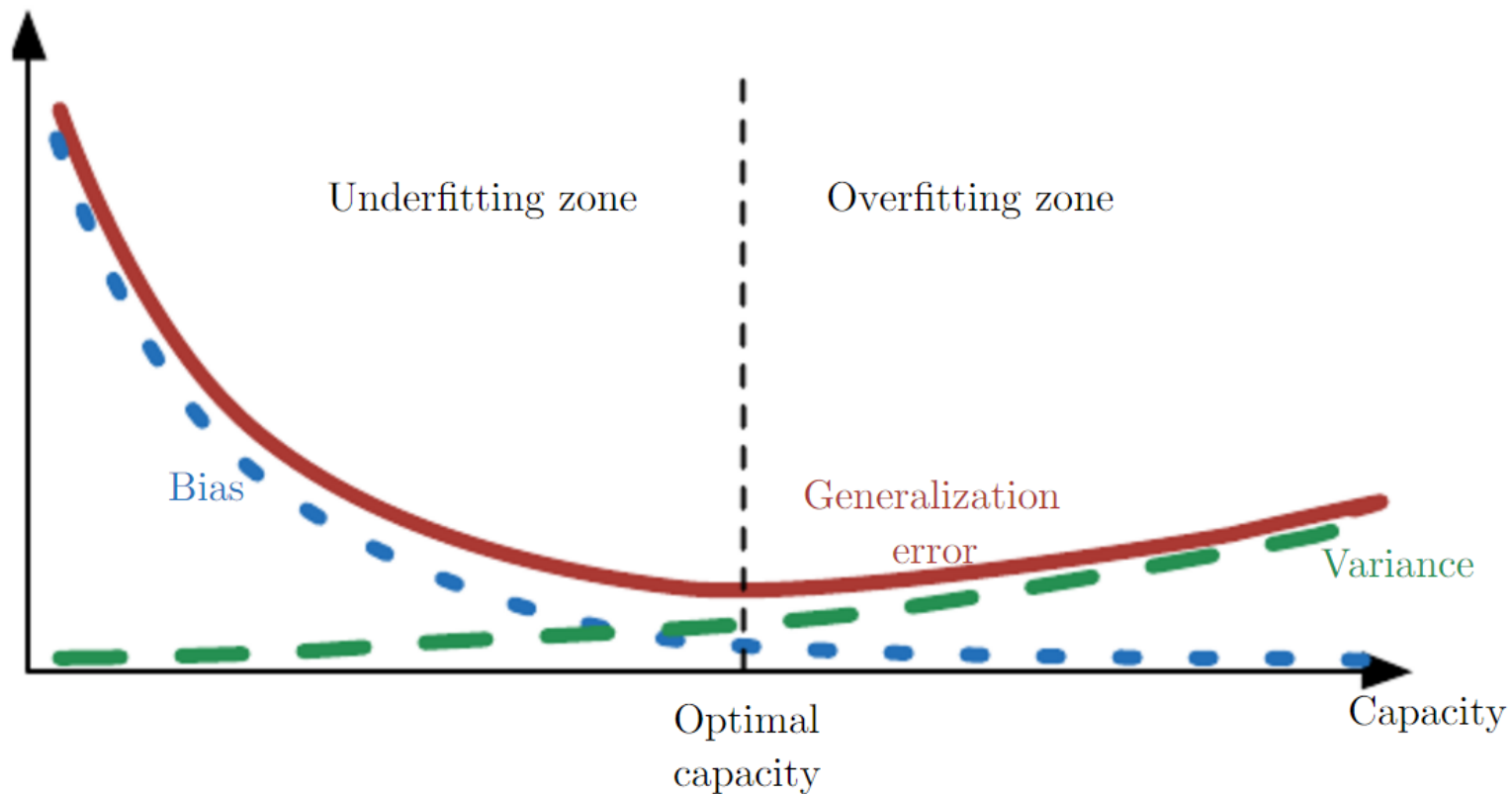
$$\min E_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} [(y - \hat{y}(\mathbf{x}))^2] + \lambda \mathbf{w}^T \mathbf{w}$$



Left:  $\hat{y}(x; \mathcal{D})$  with different training sets

Right:  $g^*(x) = \sin(2\pi x)$  (Green);  $E_{\mathcal{D}}[y(x; \mathcal{D})]$  (Red)

## Trading off Bias and Variance



In general, models of high capacity have low bias and high variance, whereas models of low capacity have high bias and low variance

## Parameter Norm Penalties

---

- Limiting the model capacity by adding a norm penalty  $\Omega(\theta)$

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

where  $X, y$  are training data and  $\alpha \in [0, \infty)$  weights the relative contribution of the norm penalty to the objective function

- Generally, for neural networks, only the weights  $w$  of the affine transformation  $w^T x + b$  are penalized, with the bias  $b$  often left unregularized
- This is because regularizing the bias can introduce a significant amount of underfitting, e.g., in the linear regression problem
- Hereafter we denote as  $w$  weights that should be regularized and as  $\theta$  all the parameters  $\{w, b\}$

## $L^2$ Regularization

- The  $L^2$  parameter regularization drives the weights closer to the origin by adding a  $L^2$ -norm penalty  $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$  (i.e. weight decay)

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

- The gradient of  $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$  w.r.t.  $\boldsymbol{w}$  is

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

- To gain insight into the behavior of weight decay, we make a quadratic approximation to  $J$  around  $\boldsymbol{w}^* = \arg \min_{\boldsymbol{w}} J(\boldsymbol{w})$

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

where  $\boldsymbol{H}$  is the Hessian matrix of  $J$  evaluated at  $\boldsymbol{w}^*$

- We then solve for the minimum of  $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$  by substituting  $\hat{J}$  for  $J$

and setting to zero its gradient w.r.t.  $w$

$$\nabla_w \tilde{J}(w; X, y) \approx H(w - w^*) + \alpha w = 0$$

- The regularized solution  $\tilde{w}$  is given by

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

- Going further, we know that  $H$  must have the factorization

$$H = Q\Lambda Q^T$$

because the Hessian matrix is real and symmetric, and is positive semi-definitive when evaluated at  $w^*$

- We then have

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

$$= Q \underbrace{(\Lambda + \alpha I)^{-1} \Lambda}_{\text{re-scaling}} Q^T w^*$$

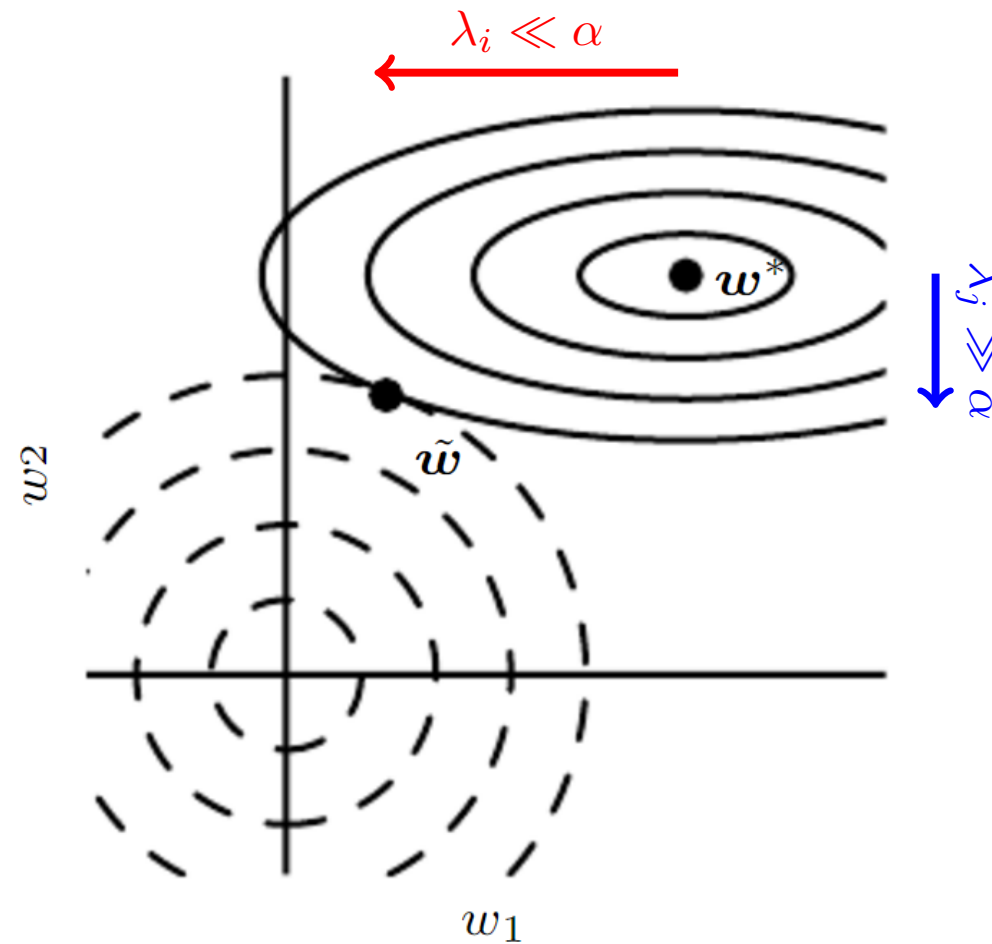
- From the above, the component of  $w$  that is aligned with the  $i$ -th eigenvector is re-scaled by  $\frac{\lambda_i}{\lambda_i + \alpha}$
- Recall that

$$\begin{aligned} J(w; X, y) &\approx J(w^*) + \frac{1}{2}(w - w^*)^T H(w - w^*) \\ &= J(w^*) + \frac{1}{2}(w - w^*)^T Q \Lambda Q^T (w - w^*) \end{aligned}$$

where along directions corresponding to large  $\lambda_i$ , a further deviation from  $w^*$  contributes significantly to increasing the objective function

- The effect of  $L^2$  regularization is to decay away components of  $w^*$  along unimportant directions with  $\lambda_i \ll \alpha$

## Effect of $L^2$ Regularization





## $L^1$ Regularization

- Another popular parameter norm regularization is to add a  $L^1$ -norm penalty  $\Omega(\boldsymbol{\theta}) = \|\mathbf{w}\|_1 = \sum |w_i|$

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \alpha \|\mathbf{w}\|_1$$

- As with  $L^2$  regularization, we hope to analyze the effect of  $L^1$  regularization by making a quadratic approximation to  $J$  at  $\mathbf{w}^*$

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) \approx J(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) + \alpha \|\mathbf{w}\|_1$$

- It is however noticed that the full general Hessian does not admit a clean algebraic solution to the following problem

$$\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) \approx \mathbf{H}(\mathbf{w} - \mathbf{w}^*) + \alpha \text{sign}(\mathbf{w}) = \mathbf{0}$$

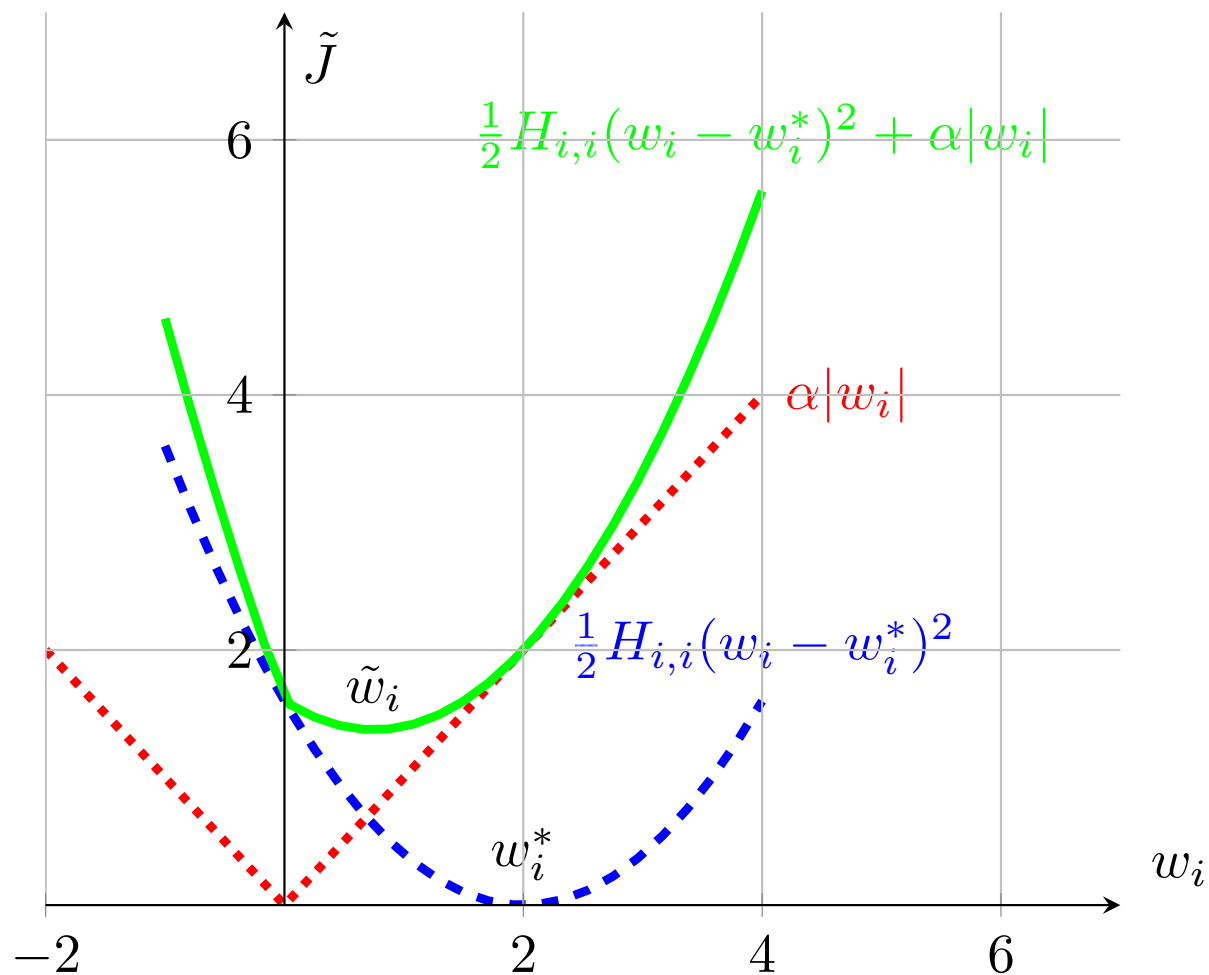
- We then make a further assumption that  $\mathbf{H}$  is diagonal

$$\mathbf{H} = \begin{bmatrix} H_{1,1} & 0 & \cdots & 0 \\ 0 & H_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & H_{n,n} \end{bmatrix}$$

to arrive at

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) \approx J(\mathbf{w}^*) + \sum_i \left[ \frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right]$$

- Without loss of generality, let us assume  $w_i^* > 0$ ; it can then be seen that the optimal  $w_i$  which minimizes  $\tilde{J}$  lies in  $[0, w_i^*]$



- Setting to zero the partial derivative of  $\tilde{J}$  w.r.t.  $w_i$  yeilds

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

- The regularized solution is then given by

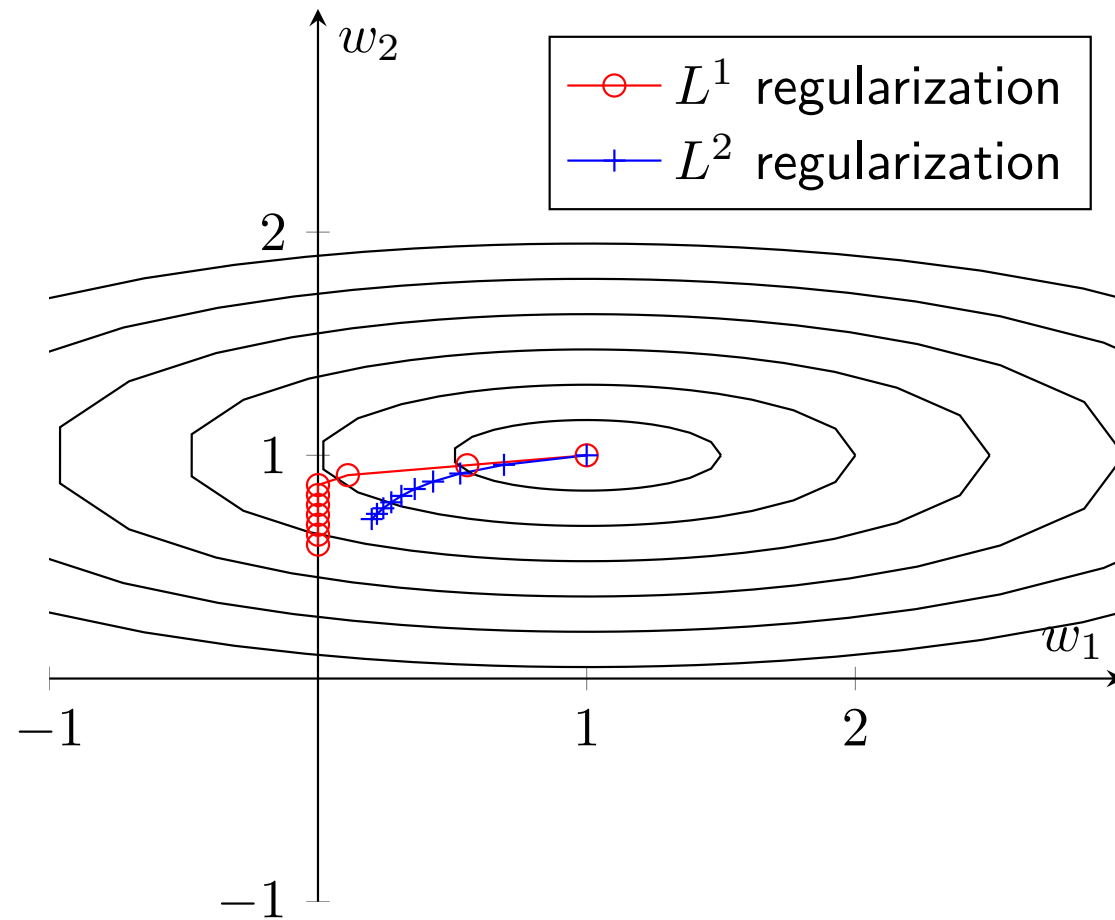
$$\tilde{w}_i = \begin{cases} w_i^* - \frac{\alpha}{H_{i,i}} & \text{if } w_i^* \geq \frac{\alpha}{H_{i,i}} \\ 0 & \text{otherwise} \end{cases}$$

- It is seen that  $L^1$  regularization results in a solution that is more sparse (i.e. having more zero weights); a similar result occurs when  $w_i^* < 0$
- In contrast,  $L^2$  regularization in the present case does NOT cause the parameters to become sparse

$$\tilde{w}_i = \frac{H_{i,i}}{H_{i,i} + \alpha} w_i^*,$$

- **Least absolute shrinkage and selection operator (LASSO):** a

feature selection mechanism based on  $L^1$  penalty + linear model + least-squares cost



## Norm Penalties as Constrained Optimization

---

- Regularized training problem of minimizing  $\tilde{J}$

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

can be thought of as constrained optimization with a weight constraint

$$\arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) \text{ s.t } \Omega(\boldsymbol{\theta}) \leq k$$

- To solve the problem, we construct the Lagrangian function

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

- The solution is then give by

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \max_{\alpha, \alpha > 0} \mathcal{L}(\boldsymbol{\theta}, \alpha; \mathbf{X}, \mathbf{y})$$

- When  $\alpha$  is fixed at its optimal value  $\alpha^*$ ,  $\theta$  is found by minimizing

$$J(\theta; \mathbf{X}, \mathbf{y}) + \alpha^* \Omega(\theta),$$

which has exactly the same form as  $\tilde{J}$

- The optimal solution  $\theta^*$  must satisfy (see the next two slides)

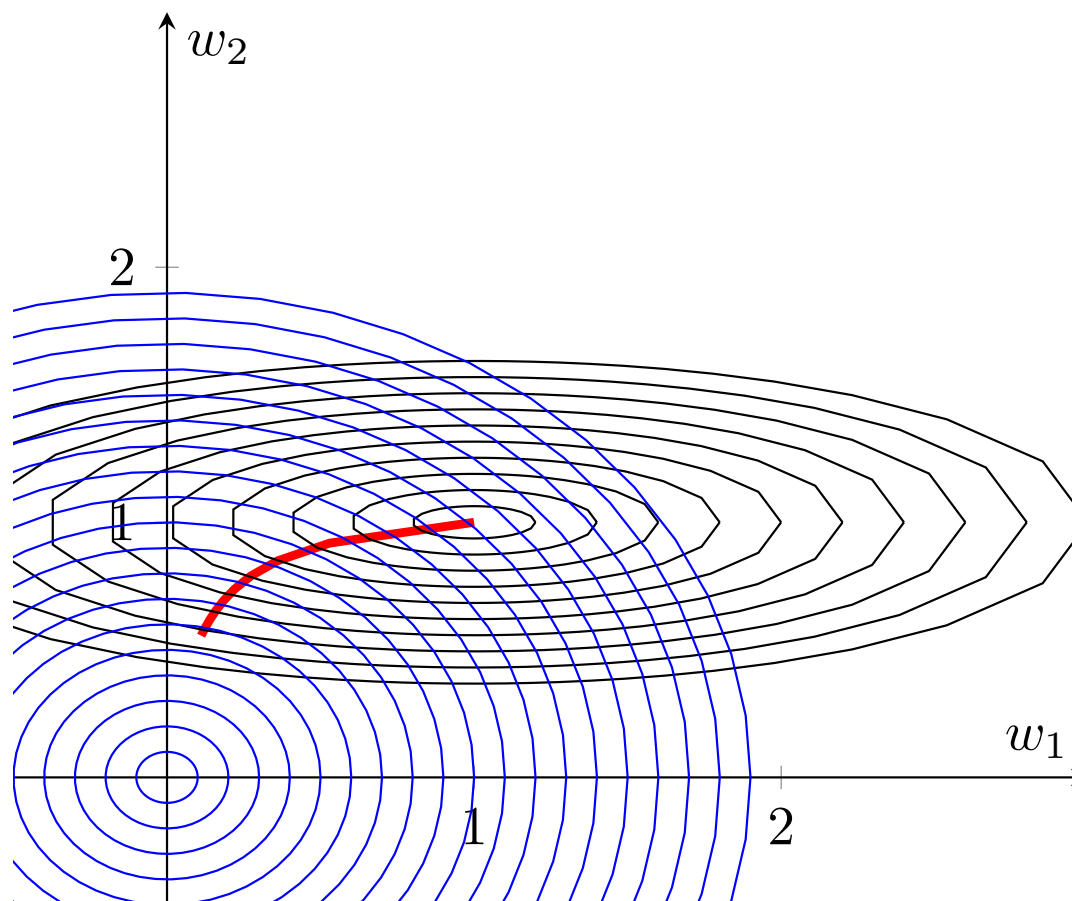
$$\nabla_{\theta} J(\theta^*; \mathbf{X}, \mathbf{y}) = -\alpha^* \nabla_{\theta} \Omega(\theta^*)$$

- Note that the value of  $\alpha^*$  does not directly tell us the value of  $k$
- Some use explicit constraints rather than penalties

1. Take a step downhill on  $J(\theta)$
2. Project  $\theta$  back to the set  $\{\theta : \Omega(\theta) < k\}$

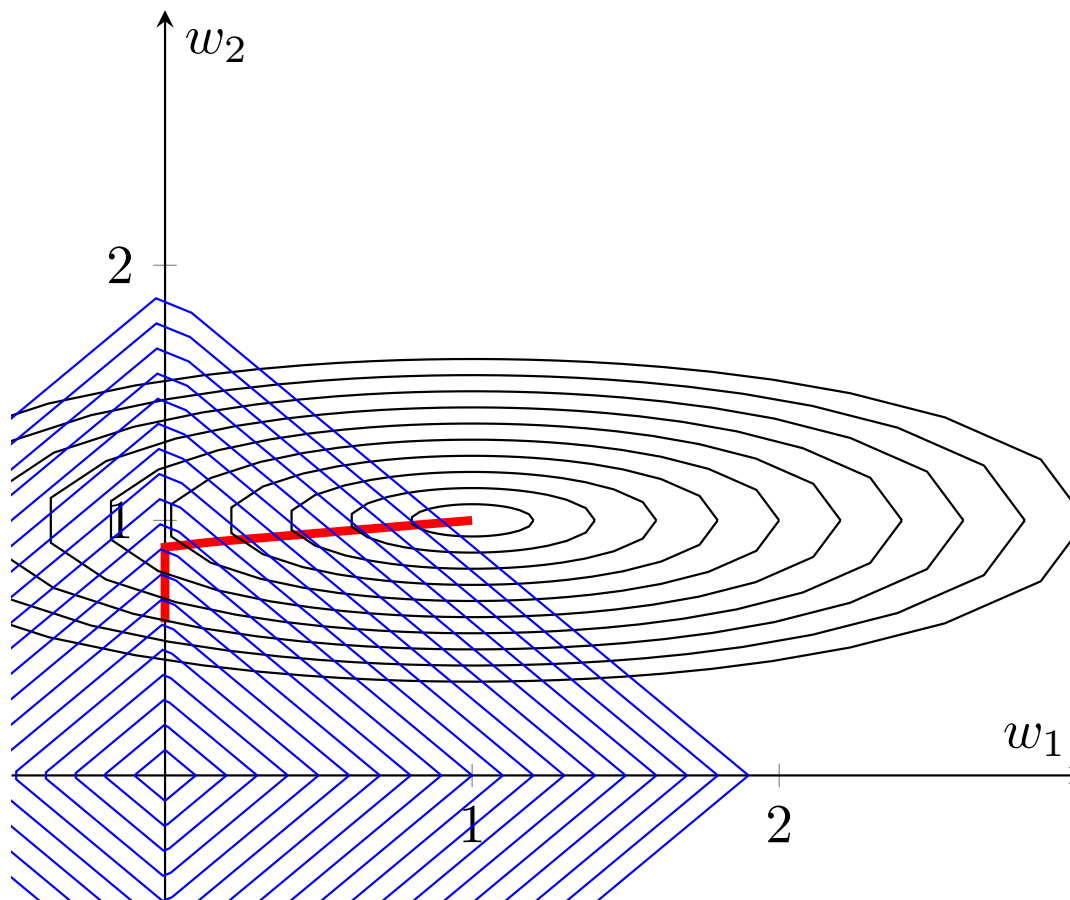
(Repeat steps 1 and 2 until some stopping criterion is satisfied)

## Trajectory of $L^2$ Regularization



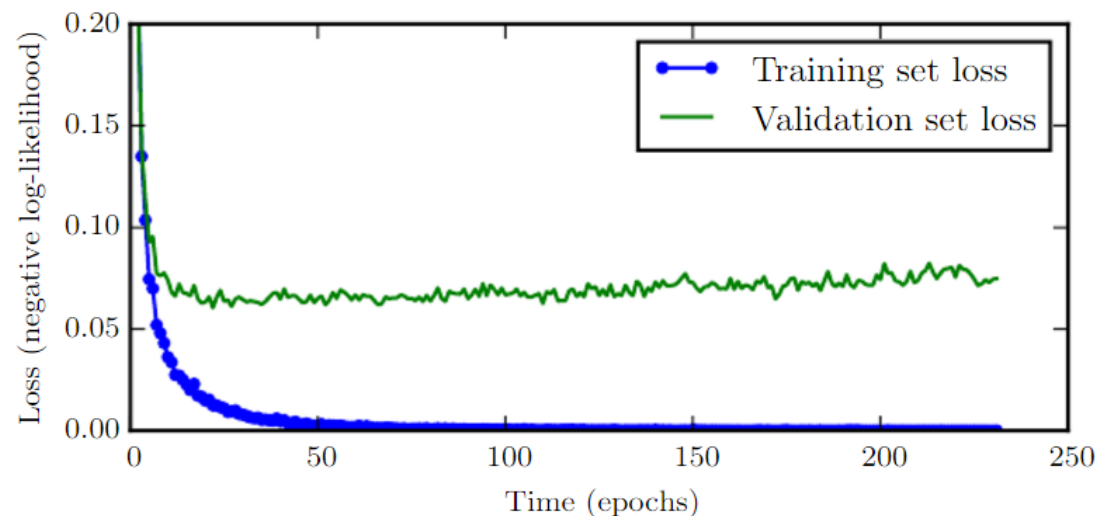


## Trajectory of $L^1$ Regularization



## Early Stopping

- One effective way of determining when the training process should stop
  1. Run training for  $n$  steps
  2. Check validation error
  3. Store model parameters if validation error reduces
  4. Repeat 1-3 until validation error does not improve after a few trials
  5. Return model parameters



## Early Stopping as Regularization

- In a sense, early stopping restricts the training to a small volume of parameter space around the initial  $\mathbf{w}_0$
- The reachable volume is determined by the product  $\tau\epsilon$  of the training iterations  $\tau$  and learning rate  $\epsilon$
- The product  $\tau\epsilon$  plays a similar role to  $\alpha^{-1}$  in  $L^2$  regularization

$$\arg \min_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w},$$

which is equivalent to

$$\min_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y}) \text{ s.t. } \mathbf{w}^T \mathbf{w} \leq k_{\alpha}$$

(Check Section 7.8 for an approximate analysis)

- Early stopping however has the advantage of automatically determining the correct amount of regularization (i.e. the value of  $\tau\epsilon \approx \alpha^{-1}$ )

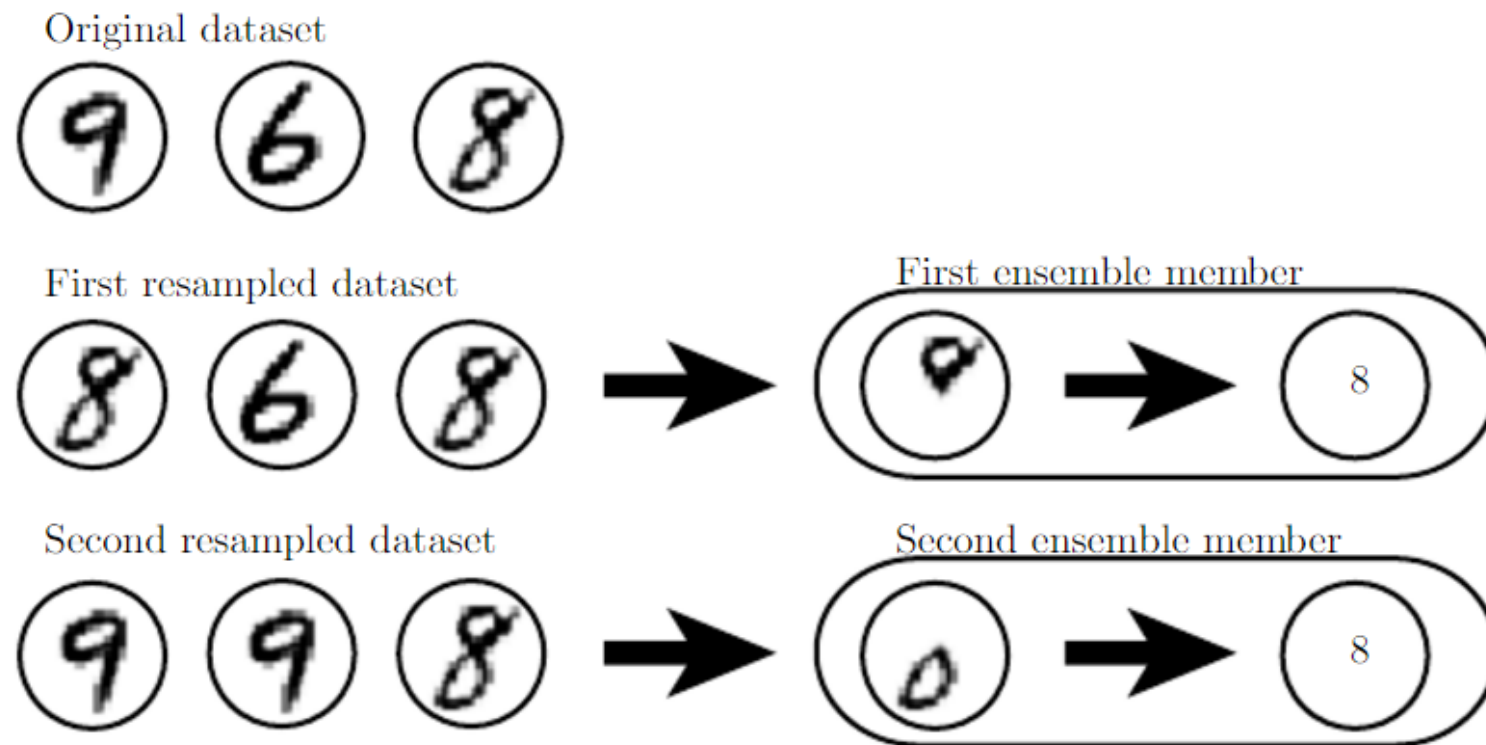
## Bootstrap Aggregating (Bagging)

- **Idea:** To train several models separately and have them vote on the output for test examples (a.k.a. model averaging or ensemble methods)
- Suppose that each model makes a random error  $\epsilon_i$  on each example with mean zero,  $E[\epsilon_i^2] = v$ , and  $E[\epsilon_i \epsilon_j] = c$
- Then, the error made by the average prediction of all  $k$  models is

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

- When errors are highly correlated, i.e.  $E[\epsilon_i \epsilon_j] = c = v$ , the mean squared error reduces to  $v$ ; the model averaging does not help
- When they are uncorrelated, the mean squared error is reduced by a factor of  $1/k$

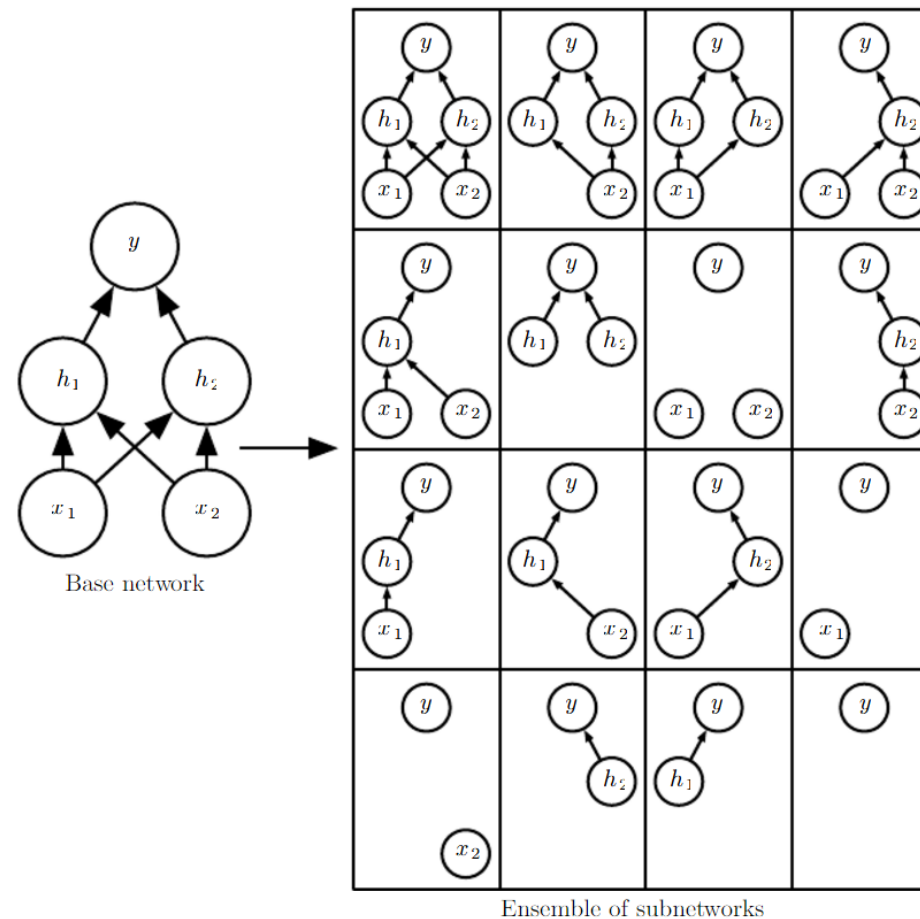
- In other words, **on average**, the ensemble will perform at least as well as any of its members, and significantly better if the members make independent errors



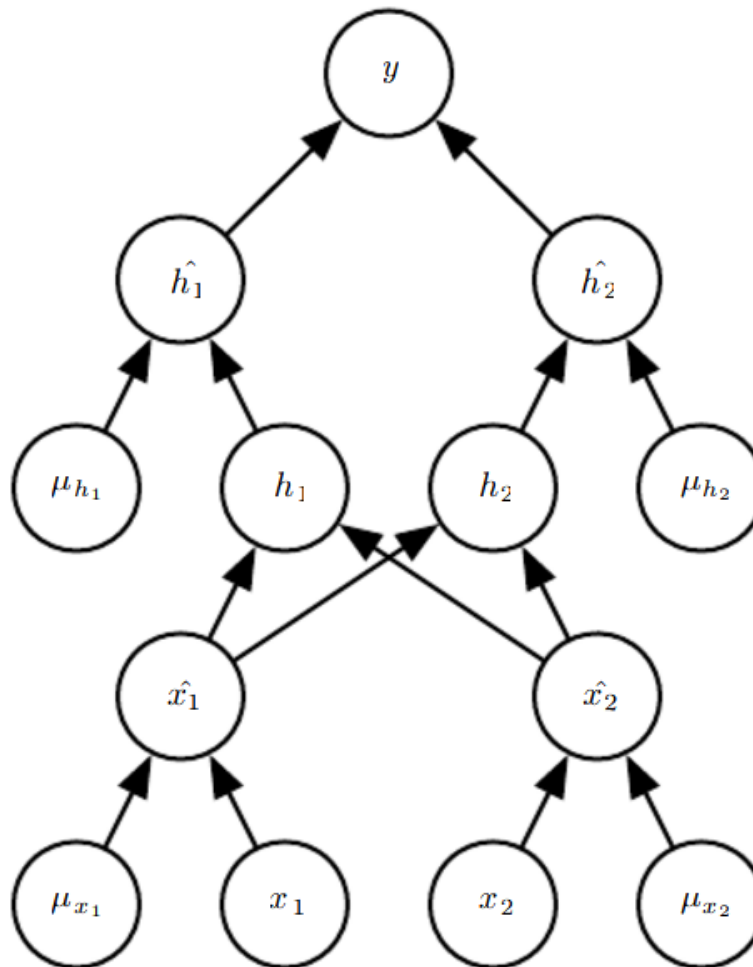
Cartoon depiction of bagging

# Dropout

- **Idea:** To train a bagged ensemble of exponentially many neural networks that consist of all subnetworks of a base network



- **Subnetwork construction:** To remove nonoutput units through multiplication of their output values by zero, with a mask vector  $\mu$  indicating which units to keep



- Typically, an input unit is included with probability 0.8 and a hidden unit with probability 0.5
- **Dropout training:** In each mini-batch step, we randomly sample a binary mask  $\mu$ ; run forward- and back-propagation; and update parameters as usual
- This amounts to minimizing

$$E_{\mu}J(\theta, \mu),$$

with each subnetwork inheriting a different subset of parameters from the parent neural network (i.e. subnetworks share parameters)

- **Ensemble inference:** To accumulate votes from all the subnetworks

$$p(y|x) = \sum_{\mu} p(\mathbf{u})p(y|x, \mu) = E_{\mu}p(y|x, \mu),$$

where  $p(\mathbf{u})$  is the distribution used to sample  $\mu$  at training time

- The summation over  $\mu$  involves an exponential number of terms, and



is thus practically intractable

- One workaround is to approximate the inference with sampling

$$p(y|\mathbf{x}) = E_{\boldsymbol{\mu}} p(y|\mathbf{x}, \boldsymbol{\mu}) \approx \frac{1}{N} \sum_{i=1}^N p(y|\mathbf{x}, \boldsymbol{\mu}^{(i)})$$

- Another empirical approach, termed the **weight scaling inference rule**, allows us to approximate  $p(y|\mathbf{x})$  in one model: the model with all units, but with the weights going out of unit  $i$  multiplied by the probability of including unit  $i$
- The motivation is to capture the right expected value of output from that unit, or to make sure that the expected total input to a unit at test time is roughly the same as that at training time
- The weight scaling inference rule is exact in some settings, e.g. deep networks that have hidden layers **without** non-linearities

- As an example, it is shown empirically that, in the present context,

$$p(y|\mathbf{x}) = E_{\boldsymbol{\mu}} p(y|\mathbf{x}, \boldsymbol{\mu}) \approx c \times \sqrt[2^d]{\prod_i p(y|\mathbf{x}, \mathbf{u}^{(i)})}$$

where  $c$  is for normalization and  $2^d$  is the number of all possible  $\mathbf{u}$ 's

- For a softmax regression classifier with input  $\mathbf{x}$  and dropout, we have

$$p(y|\mathbf{x}; \mathbf{u}) = \text{softmax}(\mathbf{W}^T(\mathbf{u} \odot \mathbf{x}) + \mathbf{b})_y,$$

with each element  $u_i$  having an equal probability of being 0 or 1

- By an application of the geometric mean approximation, we obtain an ensemble softmax classifier with

$$p(y|\mathbf{x}) \propto \exp\left(\underbrace{\frac{1}{2} \mathbf{W}_{y,:}^T}_{\text{}} \mathbf{x} + b_y\right)$$

(Check Section 7.12 for details)

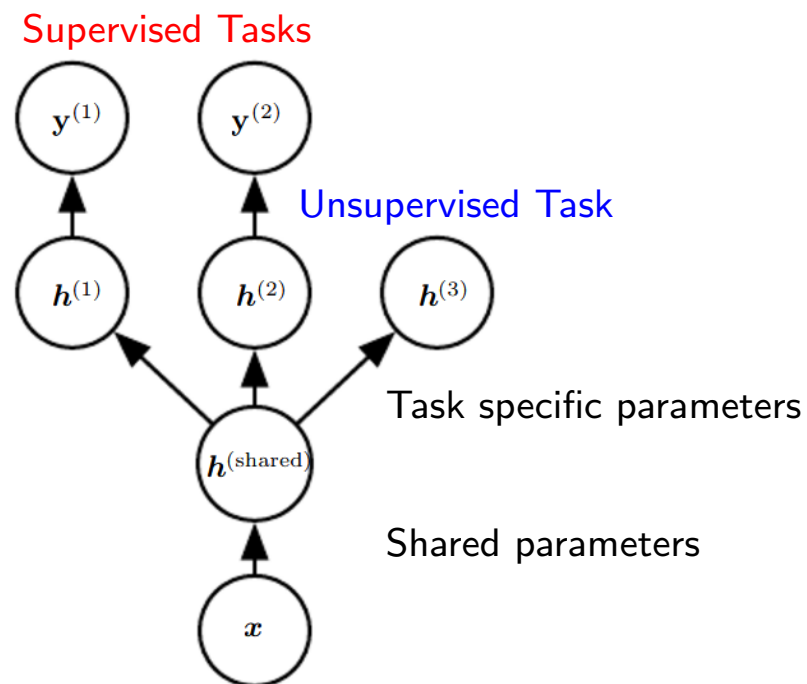
## Pros and Cons

---

- Values of dropout go beyond bagging, e.g. removal of hidden units is similar to adaptive destruction of high-level contents
- Shared hidden units learn features useful in many contexts/subnetworks
- Applicable to many types of models
- More effective than other standard regularizers
- Computationally cheap  $\mathcal{O}(n)$  for training and storage
- Increased model size needed for more capable subnetworks
- Less effective with few labeled training examples
- etc.

## Multitask Learning

- **Idea:** To improve generalization by pooling examples for several tasks

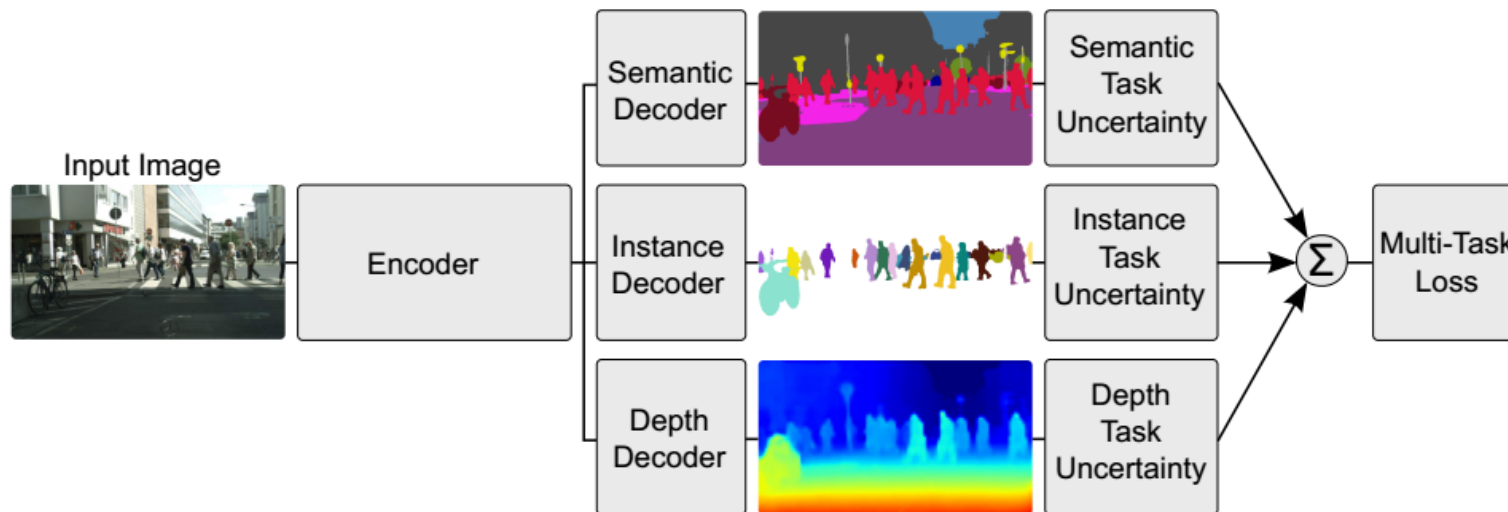


- Assumption: there exists a common pool of factors that explain the data variations, while each task is linked to a subset of these factors
- The cost function may involve both supervised and unsupervised parts

- The dominant approach is to perform a weighted linear sum of losses

$$L_{total} = \sum_i w_i L_i$$

- The model performance is however sensitive to weight selection  $w_i$



Source: A. Kendall, Y. Gal and R. Cipolla, "Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics," CVPR 2018.

## Multitask Likelihoods

- **Idea:** Learning the relative weights  $w_i$  of the losses  $L_i$
- Suppose  $\mathbf{f}^{\mathbf{W}}(\mathbf{x})$  is the shared representation of input  $\mathbf{x}$  with weights  $\mathbf{W}$  and the model output is composed of a continuous vector  $\mathbf{y}_1$  for a regression task and a discrete vector  $\mathbf{y}_2$  for a classification task
- For regression, we define the likelihood as a Gaussian

$$p(\mathbf{y}_1 | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_1) = \mathcal{N}(\mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_1^2)$$

- For classification, we define the likelihood as a softmax that squashes a **scaled** version of the shared representation

$$p(\mathbf{y}_2 | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_2) = \text{Softmax}\left(\underbrace{\frac{1}{\sigma_2^2}} \mathbf{f}^{\mathbf{W}}(\mathbf{x})\right)$$

- The joint loss is given by the negative joint conditional log-likelihood

$$\begin{aligned}\mathcal{L}(\mathbf{W}, \sigma_1, \sigma_2) &= -\log p(\mathbf{y}_1, \mathbf{y}_2 = c | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_1, \sigma_2) \\ &= -\log p(\mathbf{y}_1 | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_1) p(\mathbf{y}_2 = c | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_2) \\ &= -\log p(\mathbf{y}_1 | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_1) - \log p(\mathbf{y}_2 = c | \mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma_2) \\ &\approx \frac{1}{2\sigma_1^2} \mathcal{L}_1(\mathbf{W}) + \frac{1}{\sigma_2^2} \mathcal{L}_2(\mathbf{W}) + \log \sigma_1^2 + \log \sigma_2^2\end{aligned}$$

where

$$\mathcal{L}_1(\mathbf{W}) = \|\mathbf{y}_1 - \mathbf{f}^{\mathbf{W}}(\mathbf{x})\|^2$$

$$\mathcal{L}_2(\mathbf{W}) = -\log \text{Softmax}(\mathbf{y}_2, \mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$

## Data Augmentation

---

- **Idea:** To add fake data to make the model generalize better
- Effective for some specific tasks, e.g. image recognition
  - Translation, rotation, scaling, etc. of training images
  - Noise injection in inputs, hidden units, outputs, and weights
- Not as readily applicable to many other tasks, e.g. density estimation



## Adversarial Training

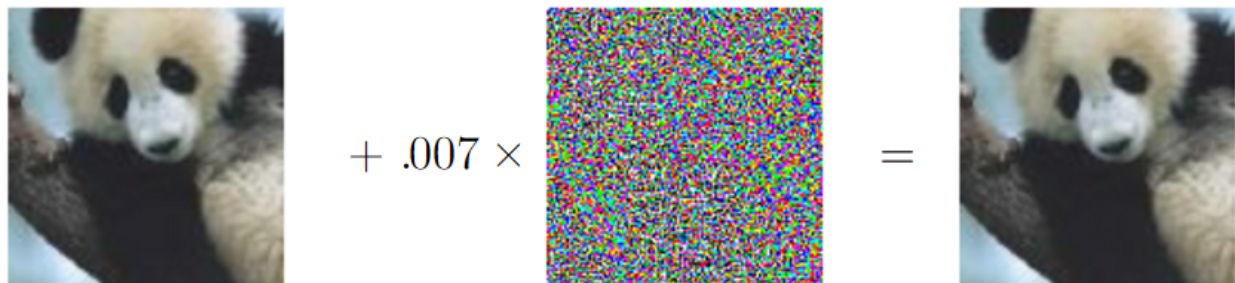
- **Idea:** To encourage the model  $\hat{y}(x)$  to be locally constant in the vicinity of training data  $x$  by including adversarial examples for training

$$x' \rightarrow x, \hat{y}(x') \rightarrow \hat{y}(x)$$

- This can be easily violated with simple linear models  $\hat{y}(x) = w^T x$

$$|\hat{y}(x + \epsilon) - \hat{y}(x)| \leq |w|^T |\epsilon| \approx \|w\|_1 c, \text{ with } |\epsilon_i| = c$$

- Adversarial examples



$$x + .007 \times \text{sign}(\nabla_x J(\theta, x, y)) = x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$$

## Review

---

- Regularization as variance and bias trade-off
- Parameter norm penalties:  $L^2$  and  $L^1$  regularization
- Early stopping
- Bagging
- Dropout
- Multitask learning
- Data augmentation
- Adversarial training
- etc.