Chapter 7

Regularization for Deep Learning

Regularization

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

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

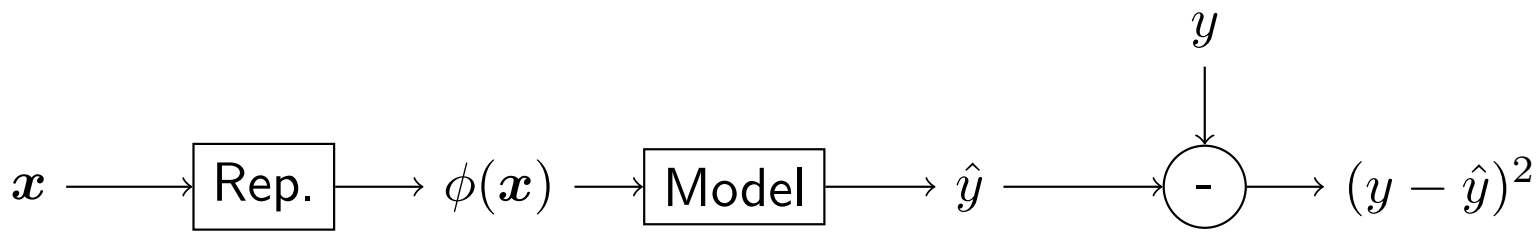
  where
  - L is the per-example loss function
  - $f(\boldsymbol{x};\boldsymbol{\theta})$ is the model prediction,
  - y is the target output
  - $\hat{p}_{\text{data}}$ is the empirical distribution
- We however hope that doing so will minimize the expected loss over the true data-generating distribution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

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

- If we knew the true distribution $p_{\text{data}}(\boldsymbol{x}, y)$, minimizing $J^*(\boldsymbol{\theta})$ would become a pure *optimization problem*; but, when we do not know the true distribution but only the empirical distribution $\hat{p}_{\text{data}}(\boldsymbol{x}, y)$ 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 $\boldsymbol{x}$, we construct a model of the form

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

and make a point estimate of the parameters $\boldsymbol{w}$ by minimizing

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

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

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

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

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

  is given by the conditional mean

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

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

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

where the cross-term

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

- 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 $\hat{y}(\boldsymbol{x})$ and training data

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

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

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

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

where the cross-term is again computed to be zero

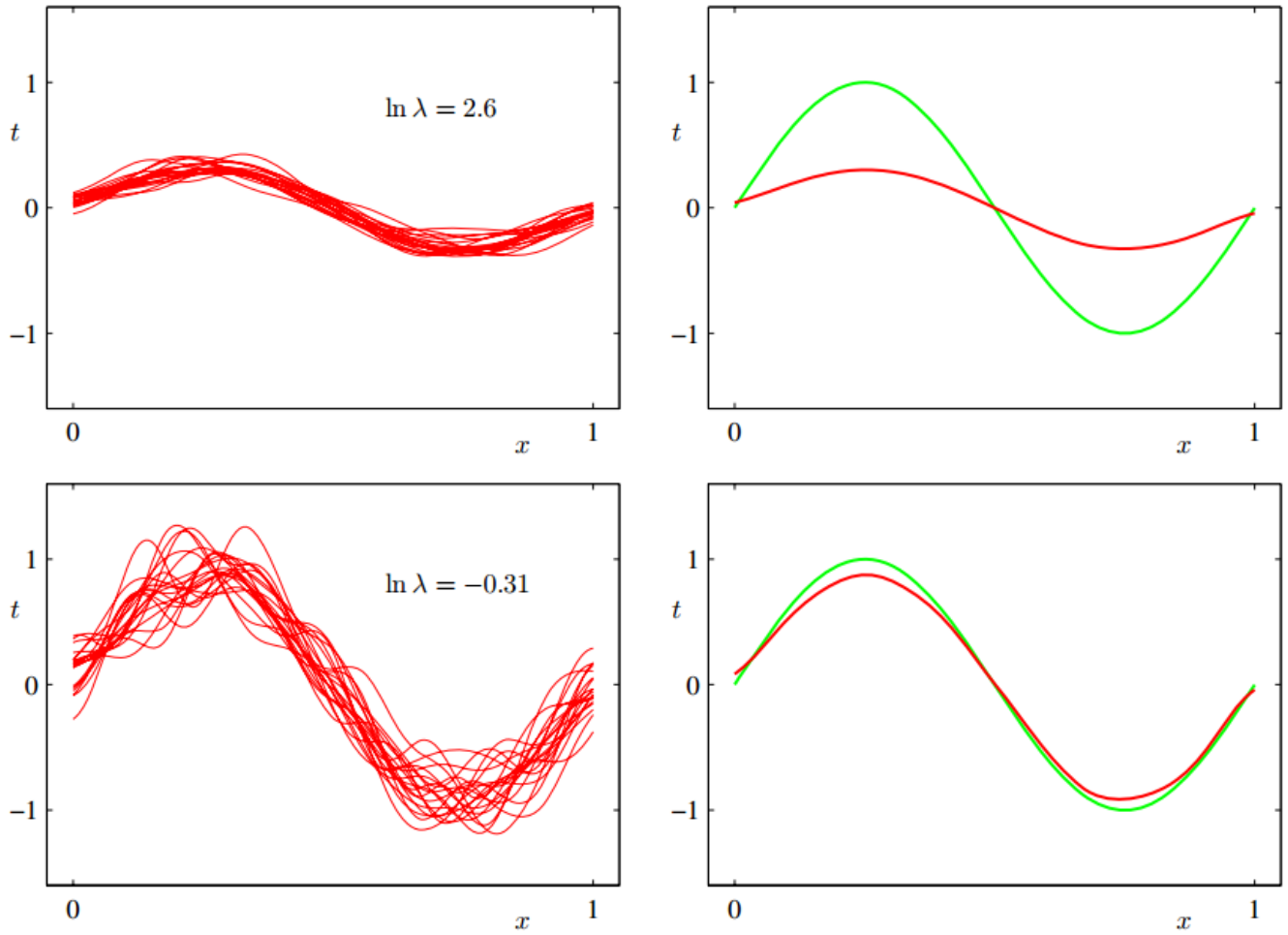
- The $(\text{bias})^2$ represents the extent to which the average model prediction over all training data sets differs from the optimal prediction
- The variance measures the extent to which the model $y(\boldsymbol{x}; \mathcal{D})$ is sensitive to the particular choice of training set
- Both terms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

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

Fitting Sinusoidal Functions

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

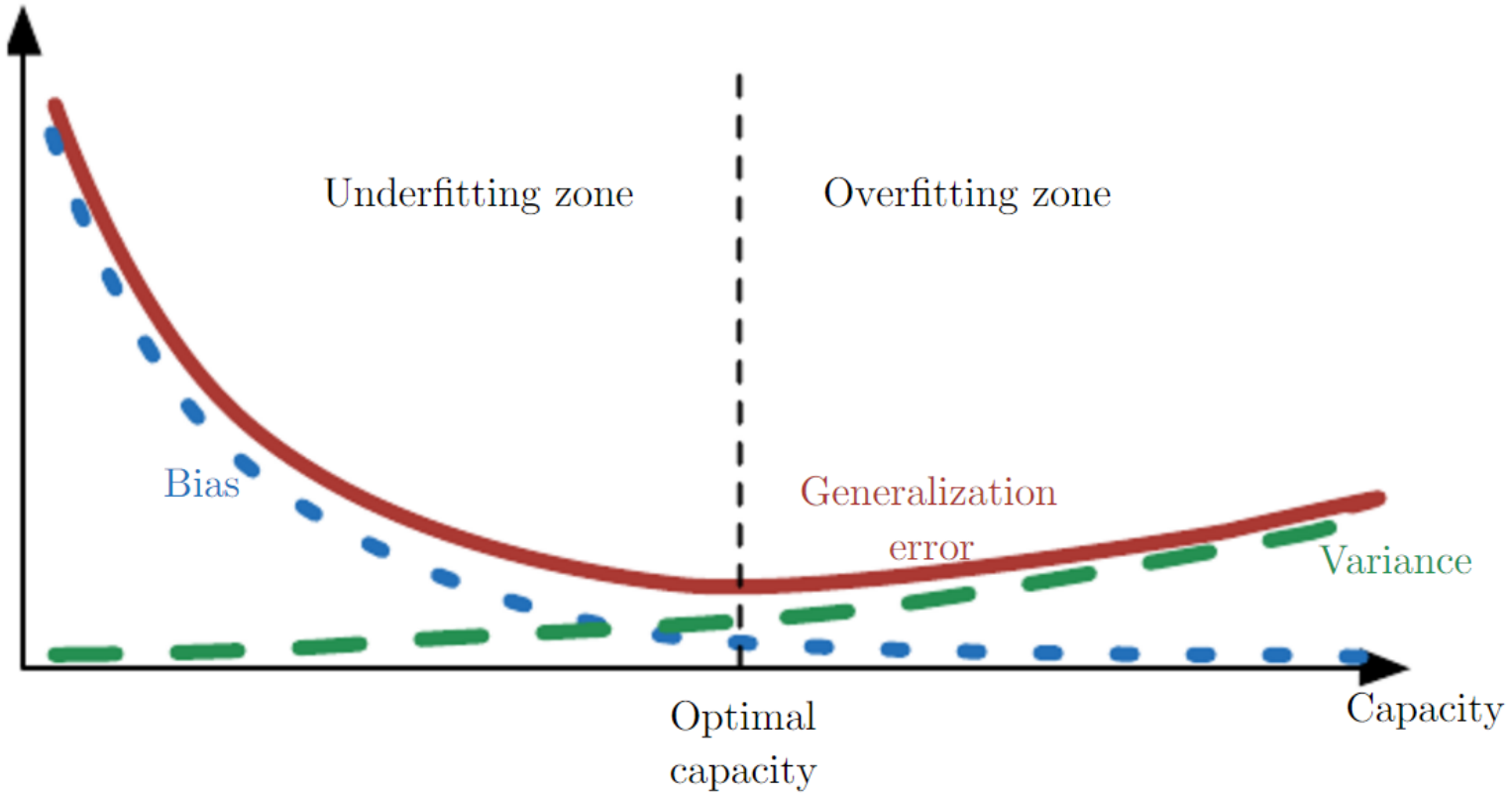
$$\min E_{\boldsymbol{x}, y \sim \hat{p}_{\text{data}}}[(y - \hat{y}(\boldsymbol{x}))^2] + \lambda \boldsymbol{w}^T \boldsymbol{w}$$



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 $\Omega(\boldsymbol{\theta})$

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

  where $\boldsymbol{X}, \boldsymbol{y}$ 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 $\boldsymbol{w}$ of the affine transformation $\boldsymbol{w}^T\boldsymbol{x} + b$ 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 $\boldsymbol{w}$ weights that should be regularized and as $\boldsymbol{\theta}$ all the parameters $\{\boldsymbol{w}, b\}$

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. $\boldsymbol{w}$

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

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

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

- Going further, we know that $\boldsymbol{H}$ must have the factorization

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

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

- We then have

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

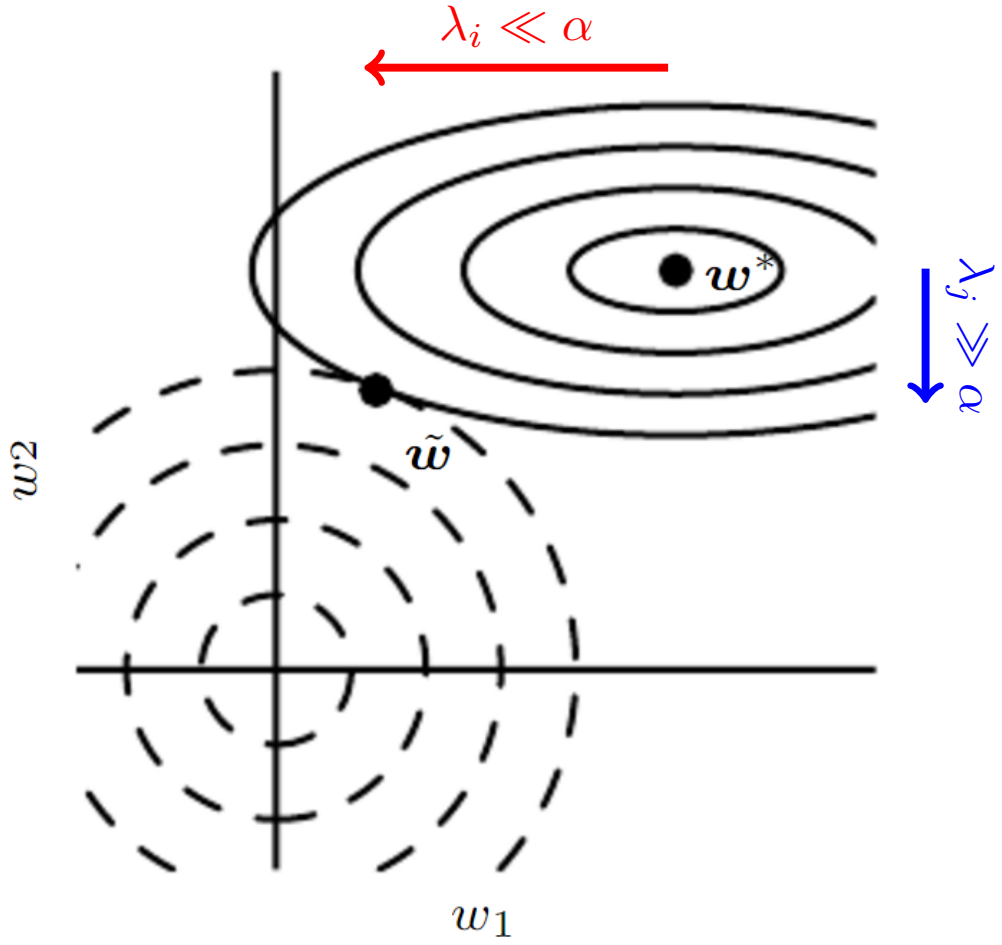
$$= \boldsymbol{Q}\underbrace{(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}}\boldsymbol{Q}^T\boldsymbol{w}^*$$

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

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

  where along directions corresponding to large λ_i, a further deviation from $\boldsymbol{w}^*$ contributes significantly to increasing the objective function
- The effect of L^2 regularization is to decay away components of $\boldsymbol{w}^*$ 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 $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum |w_i|$

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

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

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

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

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

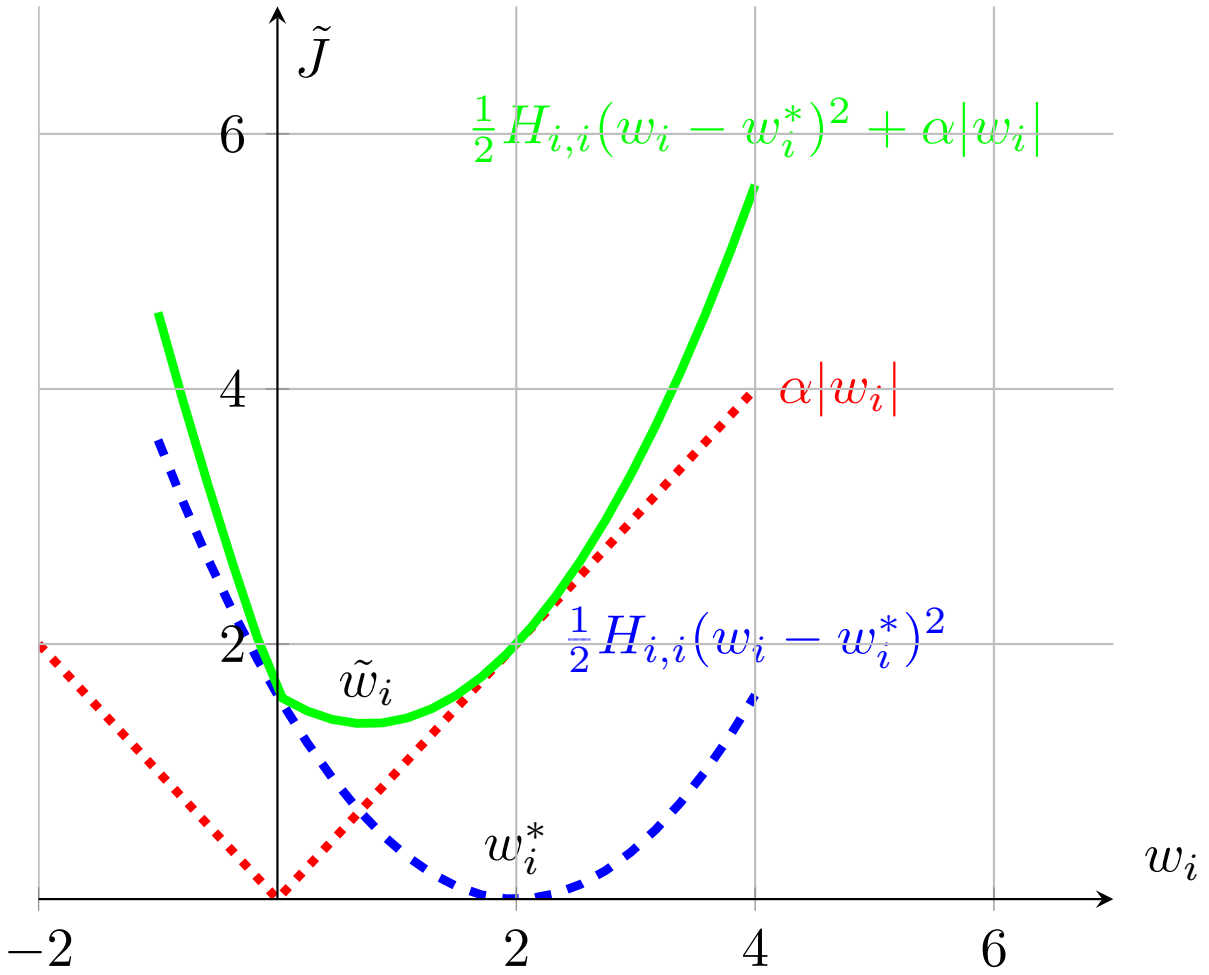
- We then make a further assumption that $\boldsymbol{H}$ is diagonal

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

  to arrive at

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

- 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 $[0, w_i^*]$


$\tilde{J}$
$\frac{1}{2}H_{i,i}(w_i - w_i^*)^2 + \alpha|w_i|$
$\alpha|w_i|$
$\frac{1}{2}H_{i,i}(w_i - w_i^*)^2$
$\tilde{w}_i$
w_i^*
w_i
6
4
2
−2
2
4
6

- 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

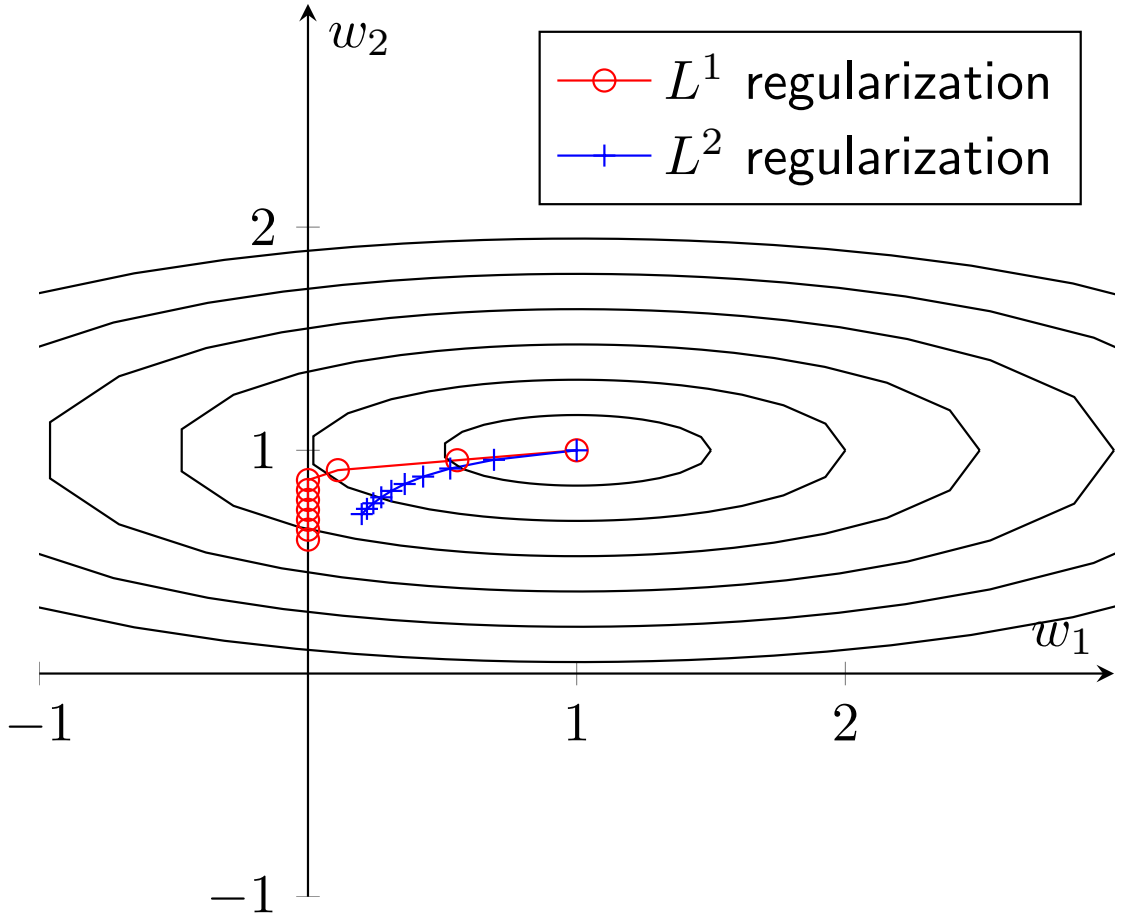
$$\tilde{w}_i = \begin{cases} w_i^* - \frac{\alpha}{H_{i,i}} & \text{if } w_i^* \geq \frac{\alpha}{H_{i,i}} \\ 0 & \text{otherwise} \end{cases}$$

- 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 $\tilde{J}$

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

  can be thought of as constrained optimization with a weight constraint

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

- To solve the problem, we construct the Lagragian function

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

- The solution is then give by

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

- When α is fixed at its optimal value α^*, $\boldsymbol{\theta}$ is found by minimizing

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

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

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

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

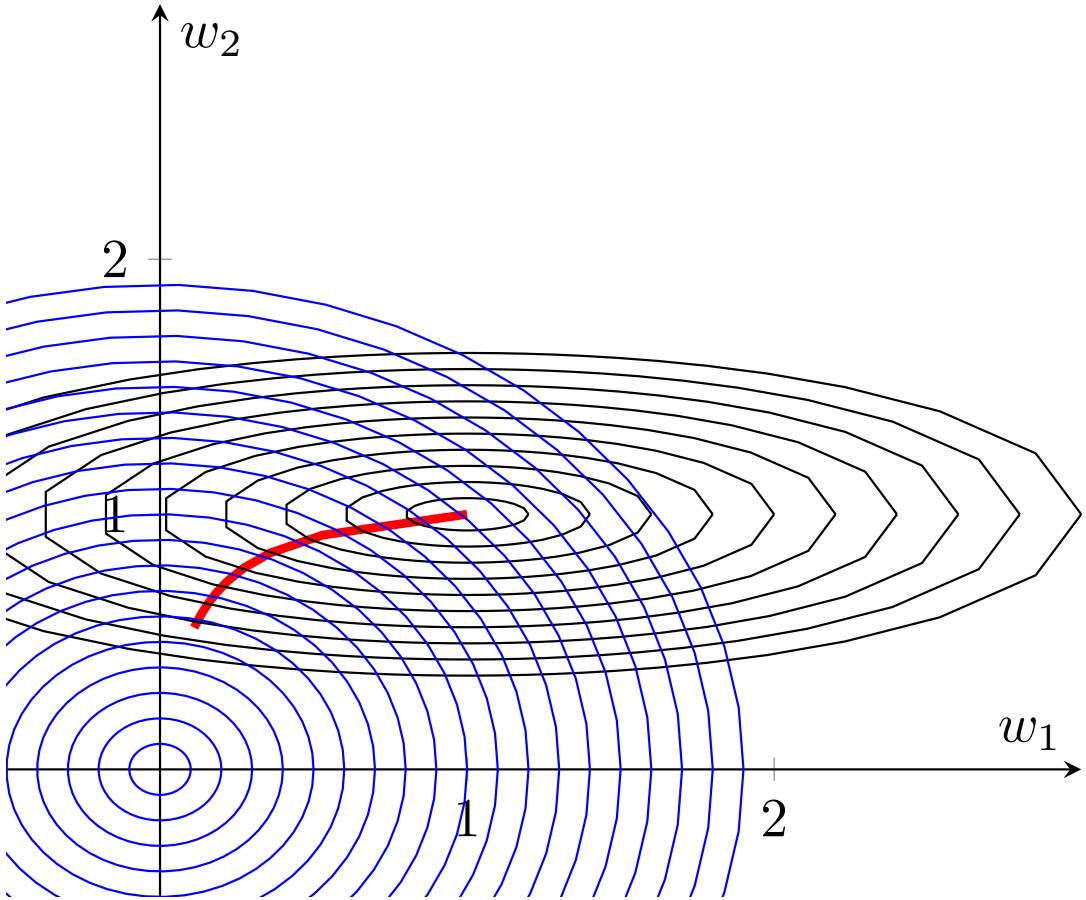
- Note that the value of α^* does not directly tell us the value of k

- Some use explicit constraints rather than penalties

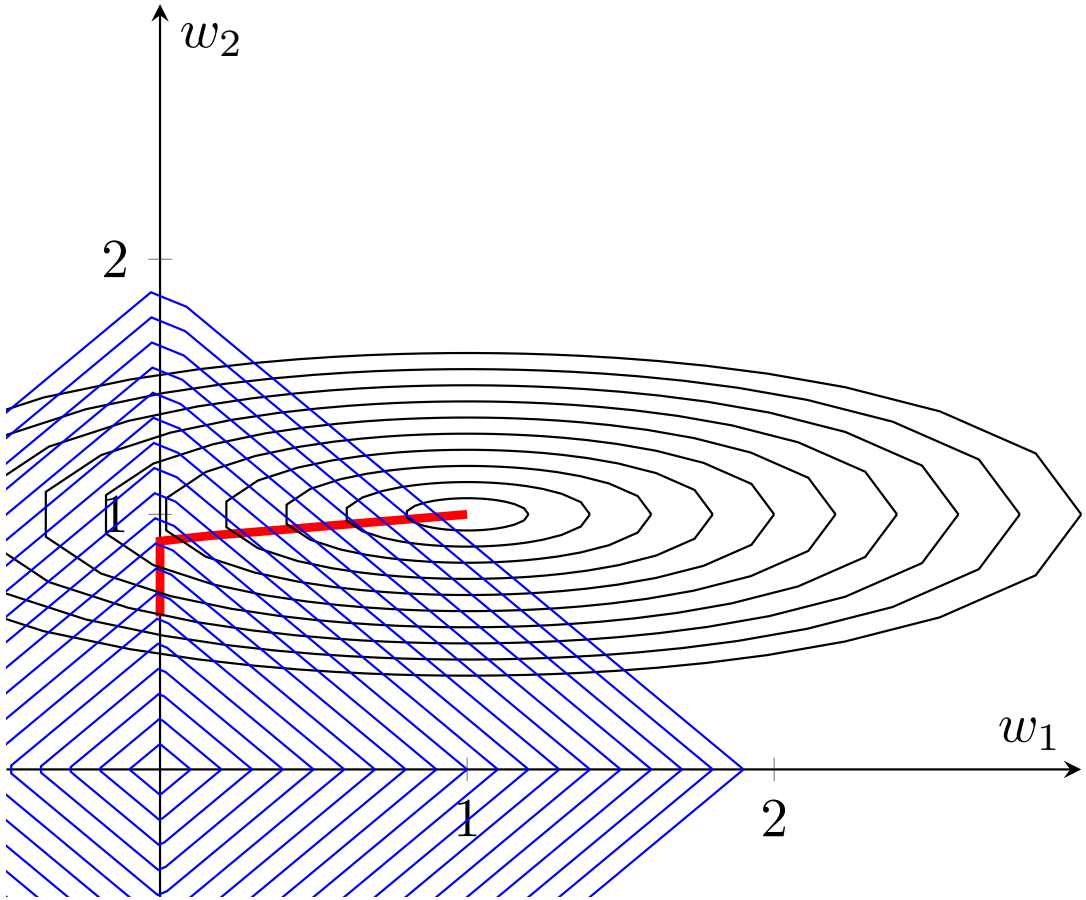
  1. Take a step downhill on $J(\boldsymbol{\theta})$
  2. Project $\boldsymbol{\theta}$ back to the set $\{\boldsymbol{\theta} : \Omega(\boldsymbol{\theta}) < k\}$

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

Trajectory of L^2 Regularization

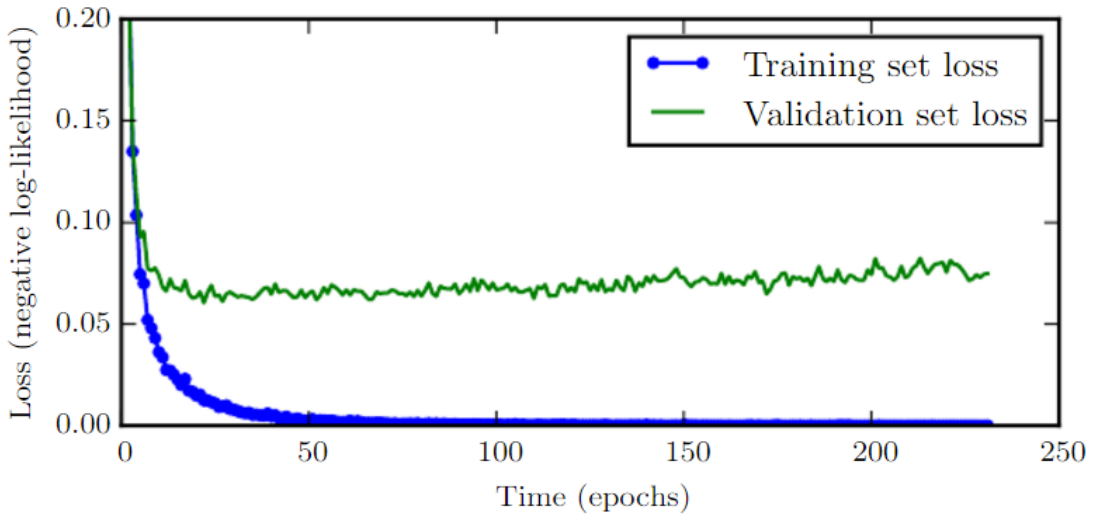


Trajectory of L^1 Regularization



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 $\boldsymbol{w}_0$
- The reachable volume is determined by the product $\tau\epsilon$ of the training iterations τ and learning rate ϵ
- The product $\tau\epsilon$ plays a similar role to α^{-1} in L^2 regularization

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

which is equivalent to

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

(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 $\tau\epsilon \approx \alpha^{-1}$)

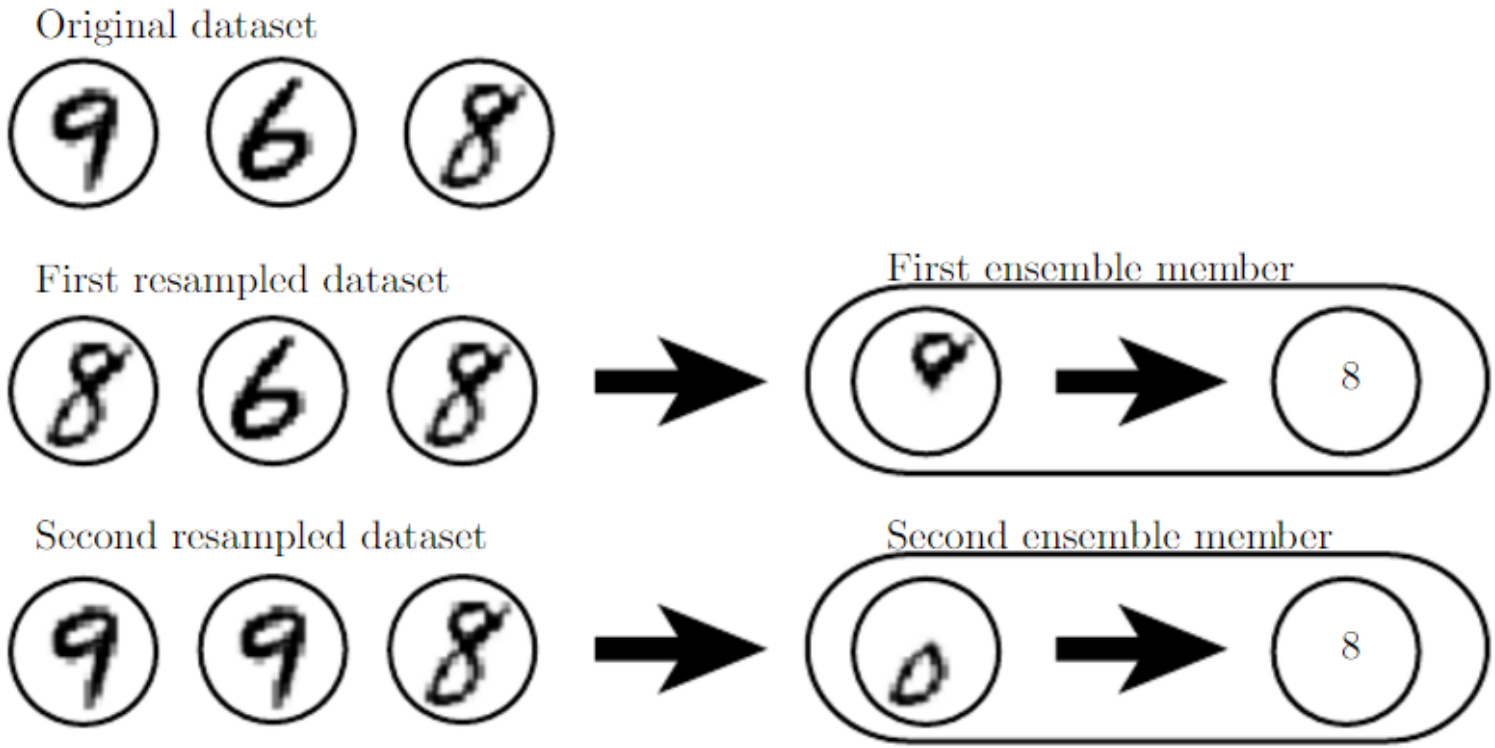
Bootstrap Aggregating (Bagging)

- **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 ϵ_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 $1/k$

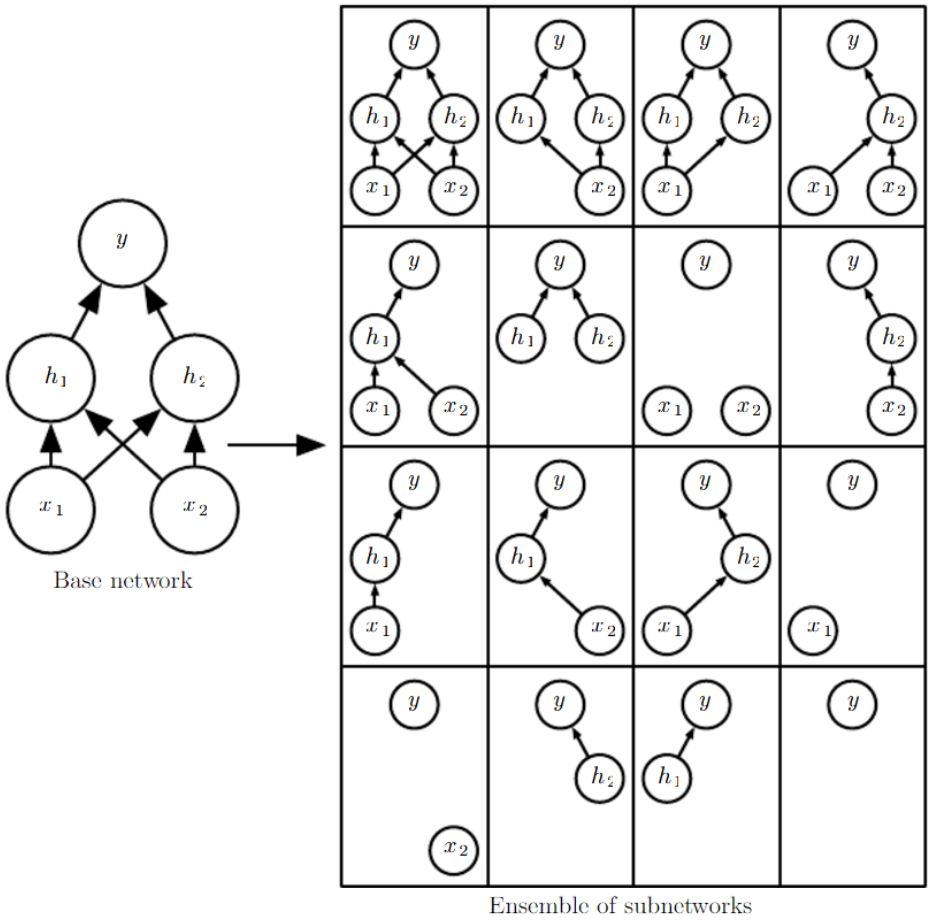
- In other words, **on average**, the ensemble will perform at least as well as any of its members, and significantly better if the members make independent errors



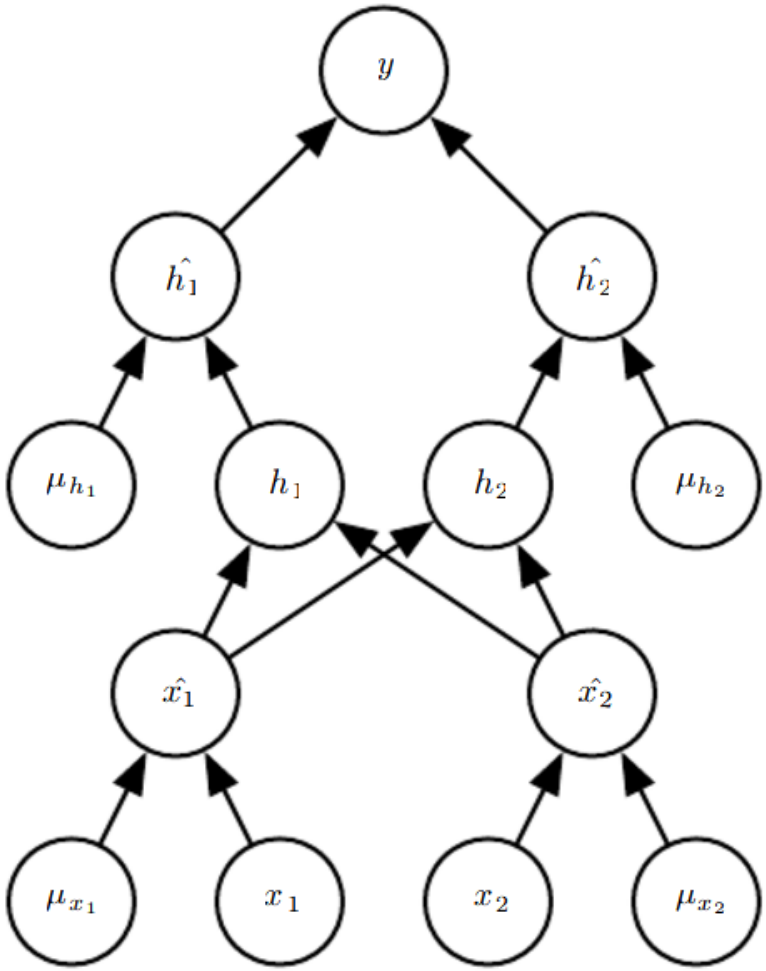
Cartoon depiction of bagging

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 $\boldsymbol{\mu}$ 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 $\boldsymbol{\mu}$; run forward- and back-propagation; and update parameters as usual
- This amounts to minimizing

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

  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

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

  where $p(\boldsymbol{u})$ is the distribution used to sample $\boldsymbol{\mu}$ at training time
- The summation over $\boldsymbol{\mu}$ involves an exponential number of terms, and

is thus practically intractable

- One workaround is to approximate the inference with sampling

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

- Another empirical approach, termed the **weight scaling inference rule**, allows us to approximate $p(y|\boldsymbol{x})$ 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,

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

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

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

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

  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

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

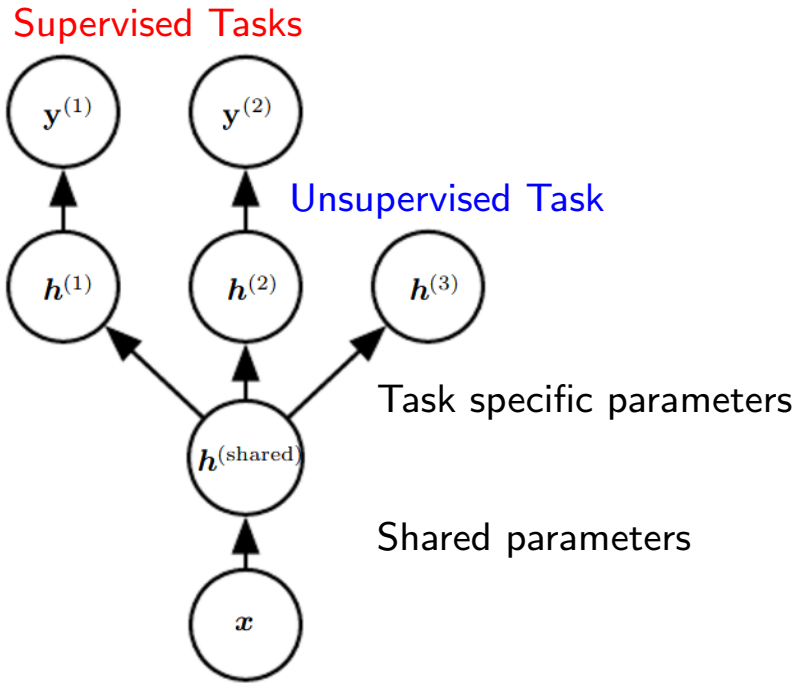
  (Check Section 7.12 for details)

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

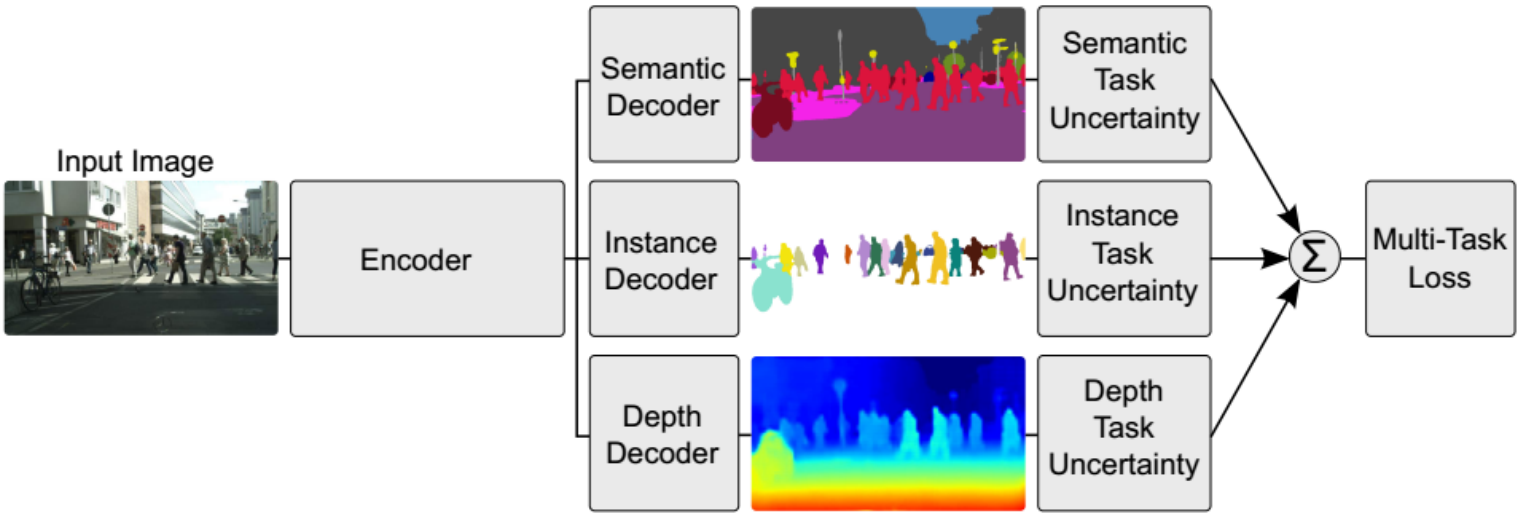


- Assumption: there exists a common pool of factors that explain the data variations, while each task is linked to a subset of these factors
- The cost function may involve both supervised and unsupervised parts

- 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 $\boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})$ is the shared representation of input $\boldsymbol{x}$ with weights $\boldsymbol{W}$ and the model output is composed of a continuous vector $\boldsymbol{y}_1$ for a regression task and a discrete vector $\boldsymbol{y}_2$ for a classification task
- For regression, we define the likelihood as a Gaussian

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

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

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

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

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

where

$$\begin{aligned}
\mathcal{L}_1(\boldsymbol{W}) &= \|\boldsymbol{y}_1 - \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x})\|^2 \\
\mathcal{L}_2(\boldsymbol{W}) &= -\log \mathsf{Softmax}(\boldsymbol{y}_2, \boldsymbol{f}^{\boldsymbol{W}}(\boldsymbol{x}))
\end{aligned}$$

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

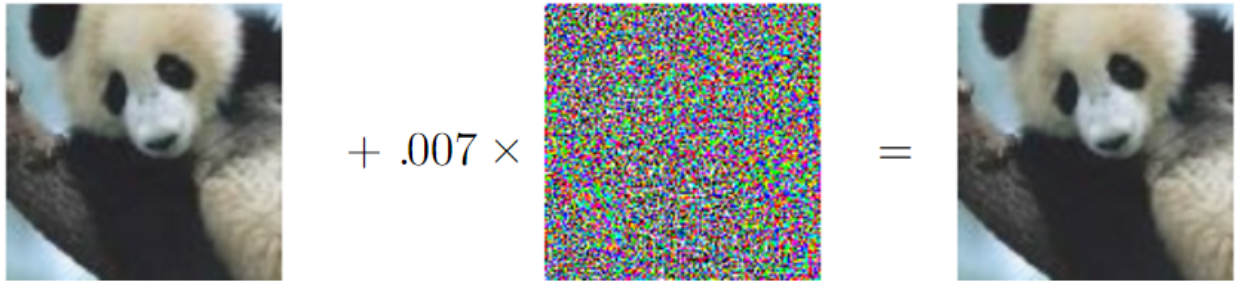
- **Idea:** To encourage the model $\hat{y}(\boldsymbol{x})$ to be locally constant in the vicinity of training data $\boldsymbol{x}$ by including adversarial examples for training

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

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

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

- Adversarial examples



$\boldsymbol{x}$ $\quad$ $\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ $\quad$ $\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

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.