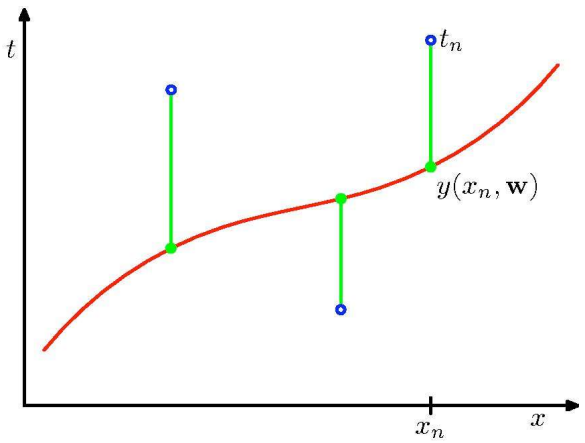
Underfitting, Overfitting and Training Tricks

Vahid Tarokh

ECE 685D, Fall 2025

Example: Polynomial Curve Fitting

• As for the least squares example: we can minimize the sum of the squares of the errors between the predictions $y(x_n, \mathbf{w})$ for each data point x_n and the corresponding target values t_n.



Loss function: sum-of-squared error function:

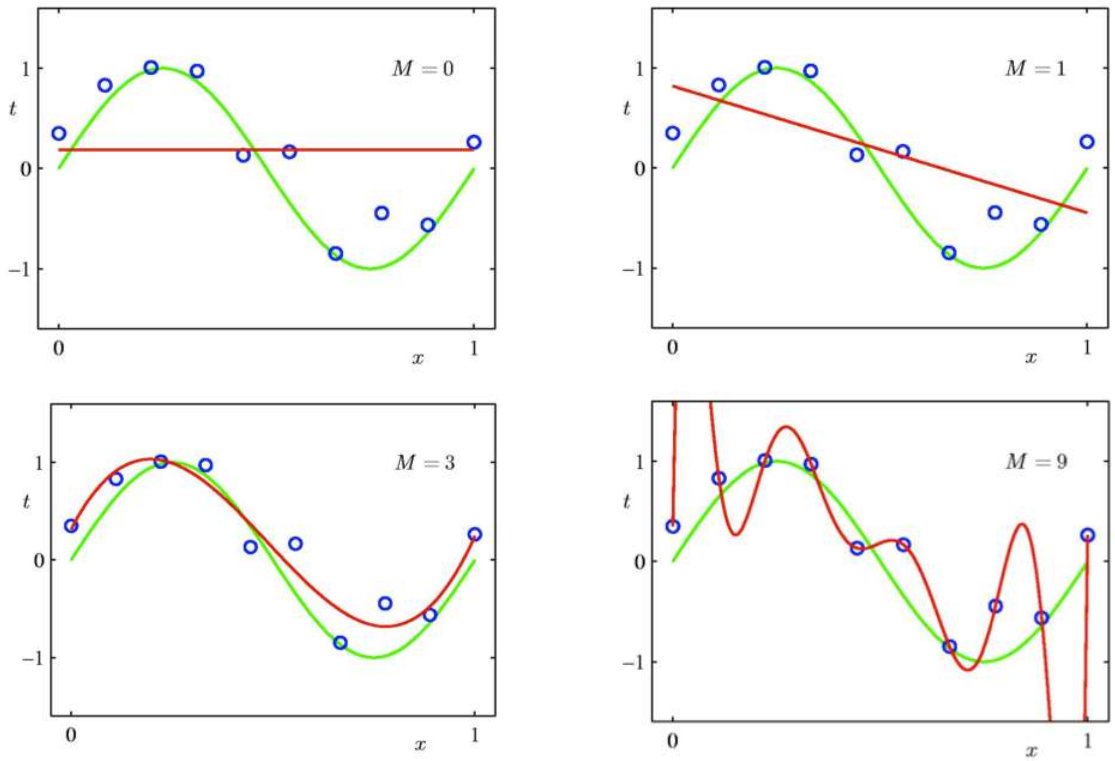
$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2.$$

• Similar to the linear least squares: Minimizing sum-of-squared error function has a unique solution $\mathbf{w}^*$.

• The model is characterized by M+1 parameters $\mathbf{w}^*$.

• How do we choose M? ! **Model Selection**.

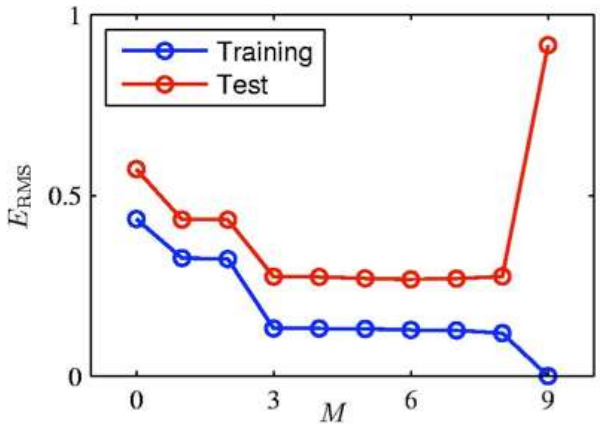
Some Fits to the Data

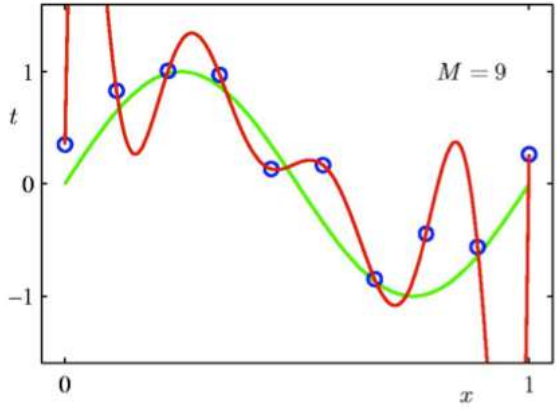


For M=9, we have fitted the training data perfectly.

Overfitting

• Consider a separate **test set** containing 100 new data points generated using the same procedure that was used to generate the training data.



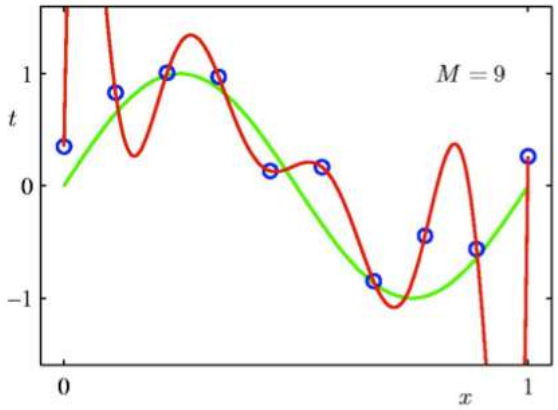


• For M=9, the training error is zero ! The polynomial contains 10 degrees of freedom corresponding to 10 parameters **w**, and so can be fitted exactly to the 10 data points.

• However, the test error has become very large. Why?

Overfitting

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

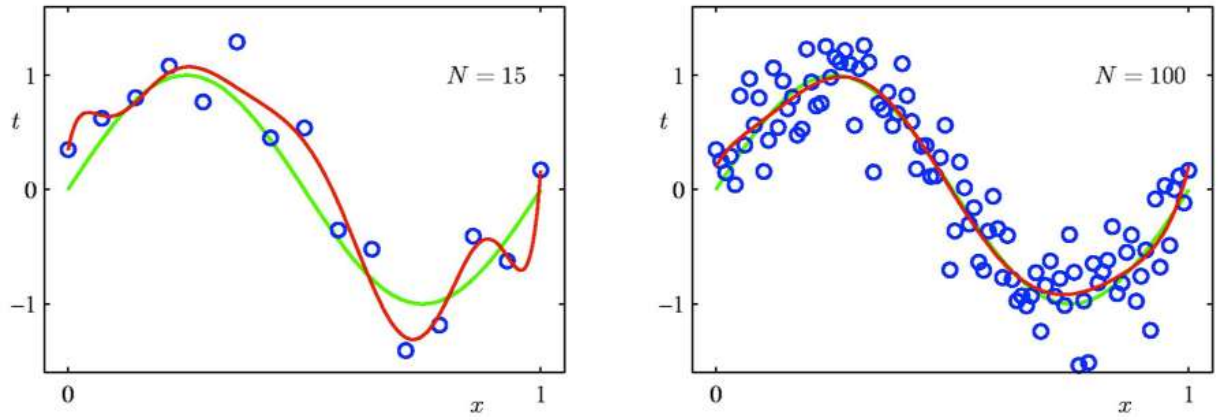


- As M increases, the magnitude of coefficients gets larger.
- For M=9, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

  More flexible polynomials with larger M tune to the random noise on the target values.

Varying the Size of the Data

9th order polynomial



• For a given model complexity, the overfitting problem becomes less severe as the size of the dataset increases.

• However, the number of parameters is not necessarily the most appropriate measure of the model complexity.

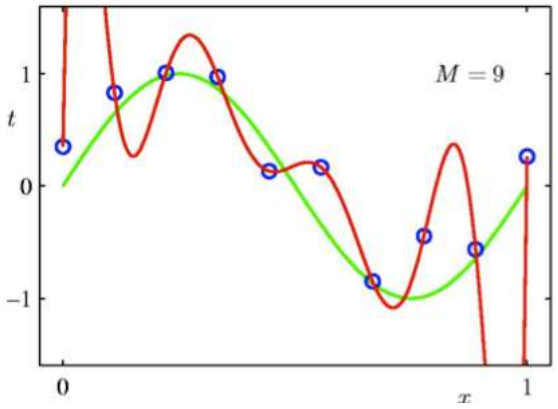
Generalization

• The goal is achieve good **generalization** by making accurate predictions for new test data that is not known during learning.

• Choosing the values of parameters that minimize the loss function on the training data may not be the best option.

• We would like to model the true regularities in the data and ignore the noise in the data:

- It is hard to know which regularities are real and which are accidental due to the particular training examples we happen to pick.



• Intuition: We expect the model to generalize if it explains the data well given the complexity of the model.

• If the model has as many degrees of freedom as the data, it can fit the data perfectly. But this is not very informative.

• Some theory on how to control model complexity to optimize generalization.

A Simple Way to Penalize Complexity

One technique for controlling over-fitting phenomenon is **regularization**, which amounts to adding a penalty term to the error function.

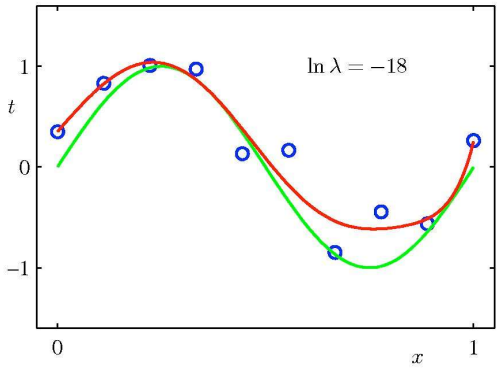
penalized error function

target value

regularization parameter

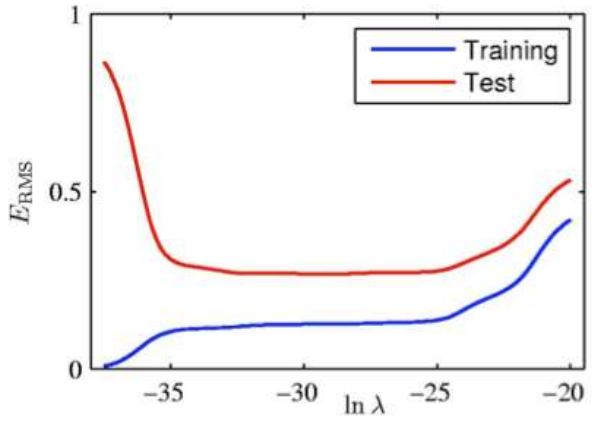
$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where $||\mathbf{w}|| = \mathbf{w}^T\mathbf{w} = w_1^2 + w_2^2 + ... + w_M^2$ and ¸ is called the regularization term. Note that we do not penalize the bias term w_0.



- The idea is to "shrink" estimated parameters towards zero (or towards the mean of some other weights).
- Shrinking to zero: penalize coefficients based on their size.
- For a penalty function which is the sum of the squares of the parameters, this is known as "**weight decay**", or "**ridge regression**".

Regularization



	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Graph of the root-mean-squared training and test errors vs. ln¸ for the M=9 polynomial.

How to choose λ?

Cross Validation

If the data is plentiful, we can divide the dataset into three subsets:

- Training Data: used to fitting/learning the parameters of the model.
- Validation Data: not used for learning but for selecting the model, or choosing the amount of regularization that works best.
- Test Data: used to get performance of the final model.

For many applications, the supply of data for training and testing is limited.
To build good models, we may want to use as much training data as possible.
If the validation set is small, we get noisy estimate of the predictive performance.

S fold cross-validation

• The data is partitioned into S groups.
• Then S-1 of the groups are used for training the model, which is evaluated on the remaining group.
• Repeat procedure for all S possible choices of the held-out group.
• Performance from the S runs are averaged.

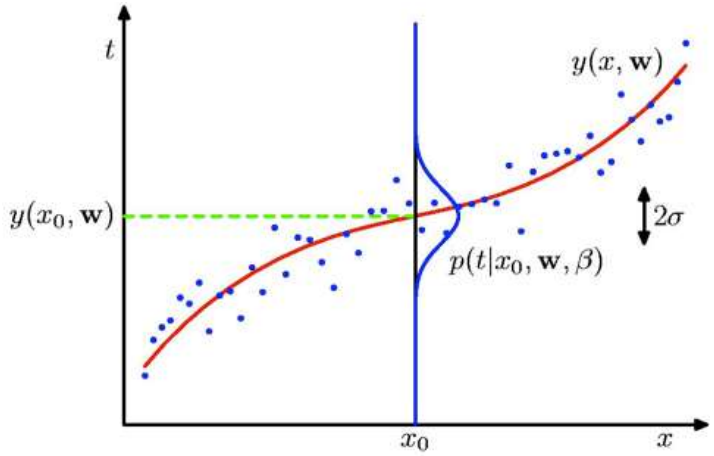
Probabilistic Perspective Of Polynomial Regression

• So far we saw that polynomial curve fitting can be expressed in terms of error minimization. We now view it from probabilistic perspective.

• Suppose that our model arose from a statistical model:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

where ϵ is a random error having Gaussian distribution with zero mean, and is independent of $\mathbf{x}$.



Thus we have:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}),$$

where β is a precision parameter, corresponding to the inverse variance.

We will use probability distribution and probability density interchangeably. It should be obvious from the context.

Maximum Likelihood

If the data are assumed to be independently and identically distributed (*i.i.d assumption*), the likelihood function takes form:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}).$$

It is often convenient to maximize the log of the likelihood function:

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \underbrace{-\frac{\beta}{2} \sum_{n=1}^{N} (y(\mathbf{x}_n, \mathbf{w}) - t_n)^2}_{\beta E(\mathbf{w})} + \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi).$$

• Maximizing log-likelihood with respect to **w** (under the assumption of a Gaussian noise) is equivalent to minimizing the *sum-of-squared error* function.

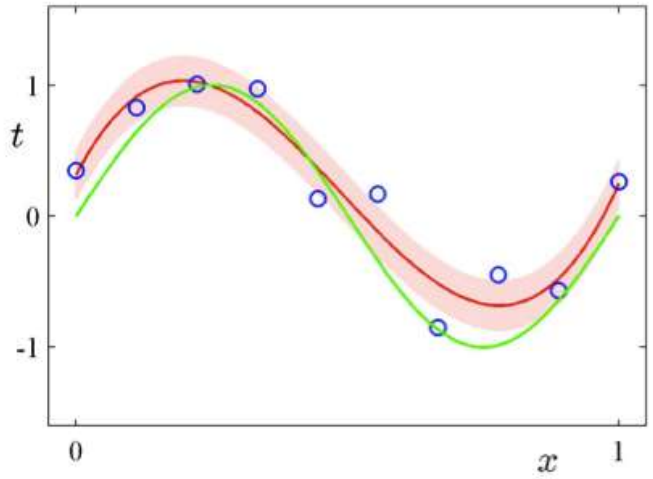
• Determine $\mathbf{w}_{ML}$ by maximizing log-likelihood. Then maximizing w.r.t. β :

$$\frac{1}{\beta_{ML}} = \frac{1}{N} \sum_{n} (y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n)^2.$$

Predictive Distribution

Once we determined the parameters **w** and β, we can make prediction for new values of **x**:

$$p(t|\mathbf{x}, \mathbf{w}_{ML}, \beta_{ML}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{ML}), \beta_{ML}^{-1}).$$



Maximum Likelihood

• As before, assume observations arise from a deterministic function with an additive Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon,$$

which we can write as:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

• Given observed inputs $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, and corresponding target values $\mathbf{t} = [t_1, t_2, ..., t_N]^T$, , under i.i.d assumption, we can write down the likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{i=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta),$$

where $\boldsymbol{\phi}(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), ..., \phi_{M-1}(\mathbf{x}))^T.$

Maximum Likelihood

Taking the logarithm, we obtain:

$$\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \sum_{i=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n), \beta)$$

$$= -\frac{\beta}{2}\underbrace{\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)\right)^2}_{\text{sum-of-squares error function}} + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi).$$

Differentiating and setting to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^{N} \left\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

Maximum Likelihood

Differentiating and setting to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

Solving for **w**, we get:

$$\mathbf{w}_{\mathrm{ML}} = \underbrace{\left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \right)^{-1}}_{\text{Depends on Data}} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t}$$

The Moore-Penrose pseudo-inverse of $\boldsymbol{\Phi}^{\dagger}$

where $\boldsymbol{\Phi}$ is known as the design matrix:

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

Sequential Learning

• The training data examples are presented one at a time, and the model parameters are updated after each such presentation (online learning):

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \bigtriangledown E_n$$

weights after seeing training case t+1

learning rate

vector of derivatives of the squared error w.r.t. the weights on the training case presented at time t.

• For the case of sum-of-squares error function, we obtain:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left(t_n - \mathbf{w}^{(t)^T} \boldsymbol{\phi}(\mathbf{x}_n) \right) \boldsymbol{\phi}(\mathbf{x}_n).$$

• Stochastic gradient descent: The training examples are picked at random (dominant technique when learning with very large datasets).

• Care must be taken when choosing learning rate to ensure convergence.

Regularized Least Squares

• Let us consider the following error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

λ is called the regularization coefficient.

• Using sum-of-squares error function with a quadratic penalization term, we obtain:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

which is minimized by setting:

Depends on Data

$$\mathbf{w} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$

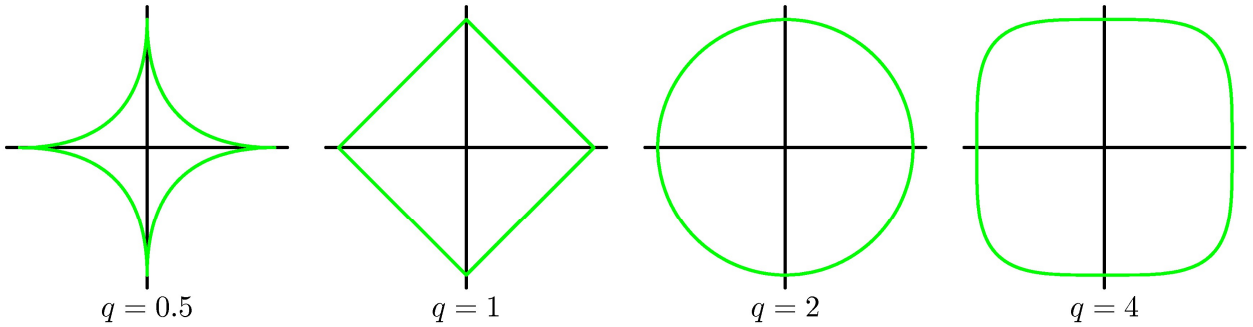
Ridge regression

The solution adds a positive constant to the diagonal of $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$. This makes the problem nonsingular, even if $\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}$ is not of full rank (e.g. when the number of training examples is less than the number of basis functions).

Other Regularizers

Using a more general regularizer, we get:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



Lasso Quadratic

The LASSO

• Penalize the absolute value of the weights:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \left[\frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{M-1} |w_j| \right].$$

• For sufficiently large λ, some of the coefficients will be driven to exactly zero, leading to a sparse model.

• The above formulation is equivalent to:

$$\mathbf{w}^{lasso} = \underset{\mathbf{w}}{\operatorname{argmin}} \underbrace{\frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) \right)^2}_{\text{unregularized sum-of-squares error}}, \text{ subject to } \sum_{j=1}^{M-1} |w_j| \leq \tau.$$

• The two approaches are related using Lagrange multiplies.

• The LASSO solution is a quadratic programming problem: can be solved efficiently.

Review of Inference

Assume that the training examples are drawn independently from the set of all possible examples, or from the same underlying distribution $p(\mathbf{x}, t)$.

We also assume that the training examples are identically distributed (i.i.d assumption).

Assume that the test samples are drawn in exactly the same way -- i.i.d from the same distribution as the training data.

These assumptions make it unlikely that some strong regularity in the training data will be absent in the test data.

Statistical Decision Theory

• We now develop a small amount of theory that provides a framework for developing many of the models we consider.

• Suppose we have a real-valued input vector **x** and a corresponding target (output) value t with joint probability $p(\mathbf{x}, t)$. distribution:

• Our goal is predict target t given a new value for **x**:

- for regression: t is a real-valued continuous target.
- for classification: t is a categorical variable representing class labels.

The joint probability distribution $p(\mathbf{x}, t)$ provides a complete summary of uncertainties associated with these random variables.

Determining $p(\mathbf{x}, t)$ from training data is known as the inference problem.

Example: Classification

Medical diagnosis: Based on the X-ray image, we would like determine whether the patient has cancer or not.

The input vector **x** is the set of pixel intensities, and the output variable t will represent the presence of cancer, class C_1, or absence of cancer, class C_2.

x -- set of pixel intensities

C_1: Cancer present

C_2: Cancer absent

Choose t to be binary: t=0 correspond to class C_1, and t=1 corresponds to C_2.

Inference Problem: Determine the joint distribution $p(\mathbf{x}, \mathcal{C}_k)$, or equivalently $p(\mathbf{x}, t)$. However, at the end, we must make a decision of whether to give treatment to the patient or not.

Example: Classification

Informally: Given a new X-ray image, our goal is to decide which of the two classes that image should be assigned to.

• We could compute conditional probabilities of the two classes, given the input image:

posterior probability of C_k given observed data.

probability of observed data given C_k

prior probability for class C_k

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x},\mathcal{C}_k)}{\sum_{k=1}^{K} p(\mathbf{x},\mathcal{C}_k)} = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

Bayes' Rule

• If our goal to minimize the probability of assigning **x** to the wrong class, then we should choose the class having the highest posterior probability.

Expected Loss

• Loss Function: overall measure of loss incurred by taking any of the available decisions.

Suppose that for **x**, the true class is C_k, but we assign **x** to class j ! incur loss of L_{kj} (k,j element of a loss matrix).

Consider medical diagnosis example: example of a loss matrix:

		Decision	
		cancer	normal
Truth	cancer	0	1000
	normal	1	0

Expected Loss: $$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\mathbf{x}, \mathcal{C}_k)\, \mathrm{d}\mathbf{x}$$

Goal is to choose decision regions $\mathcal{R}_j$ as to minimize expected loss.

Regression

Let **x** ∈ R^d denote a real-valued input vector, and t ∈ R denote a real-valued random target (output) variable with joint the $p(\mathbf{x}, t)$. distribution

• The decision step consists of finding an estimate y(**x**) of t for each input **x**.

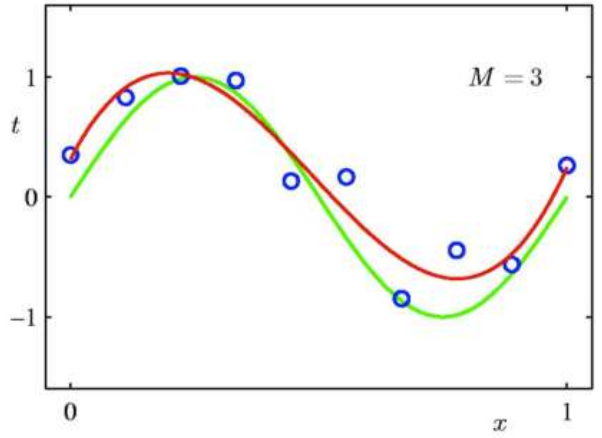
• To quantify what it means to do well or poorly on a task, we need to define a loss (error) function: $L(t, y(\mathbf{x}))$.

• The average, or expected, loss is given by:

$$\mathbb{E}[L] = \int \int L(t, y(\mathbf{x}))p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t.$$

• If we use squared loss, we obtain:

$$\mathbb{E}[L] = \int \int \big(t - y(\mathbf{x})\big)^2 p(\mathbf{x}, t)\mathrm{d}\mathbf{x}\mathrm{d}t.$$



Squared Loss Function

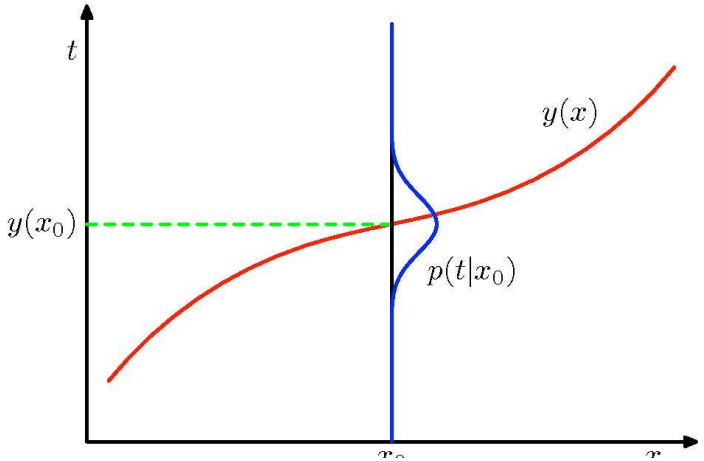
• If we use squared loss, we obtain:

$$\mathbb{E}[L] = \int \int \big(t - y(\mathbf{x})\big)^2 p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t.$$

• Our goal is to choose y(x) so as to minimize the expected squared loss.

• The optimal solution (if we assume a completely flexible function) is the conditional average:

$$y(\mathbf{x}) = \int t p(t|\mathbf{x}) \mathrm{d}t = \mathbb{E}[t|\mathbf{x}].$$



The regression function y(**x**) that minimizes the expected squared loss is given by the mean of the conditional distribution $p(t|\mathbf{x})$.

Squared Loss Function

- If we use squared loss, we obtain:

$$\begin{aligned}(y(\mathbf{x}) - t)^2 &= (y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}] + \mathbb{E}[t|\mathbf{x}] - t)^2 \\ &= \big(y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\big)^2 + 2\big(y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\big)\big(\mathbb{E}[t|\mathbf{x}] - t\big) + \big(\mathbb{E}[t|\mathbf{x}] - t\big)^2.\end{aligned}$$

- Plugging into expected loss:

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}]\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int \mathrm{var}\,[t|\mathbf{x}]\, p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

expected loss is minimized when $y(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]$.

intrinsic variability of the target values.

Because it is independent noise, it represents an irreducible minimum value of expected loss.

Other Loss Function

• Simple generalization of the squared loss, called the *Minkowski* loss:

$$\mathbb{E}[L] = \int \int \big(t - y(\mathbf{x})\big)^q p(\mathbf{x}, t) \mathrm{d}\mathbf{x} \mathrm{d}t.$$

• The minimum of $\mathbb{E}[L]$ is given by:

- the conditional mean for q=2,
- the conditional median when q=1

Bias-Variance Decomposition

• Introducing a regularization term can help us control overfitting. But how can we determine a suitable value of the regularization coefficient?

• Let us examine the expected squared loss function.
Remember:

$$\mathbb{E}[L] = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x} + \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$

intrinsic variability of the target values: The minimum achievable value of expected loss

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x})\,\mathrm{d}t.$$

• If we model $h(\mathbf{x})$ using a parametric function $y(\mathbf{x}, \mathbf{w})$, then from a Bayesian perspective, the uncertainly in our model is expressed through the posterior distribution over parameters **w**.

• We first look at the frequentist perspective.

Bias-Variance Decomposition

• From a frequentist perspective: we make a point estimate of $\mathbf{w}^*$ based on the dataset D.

• We next interpret the uncertainly of this estimate through the following thought experiment:

- Suppose we had a large number of datasets, each of size N, where each dataset is drawn independently from $p(\mathbf{x}, t)$.
- For each dataset D, we can obtain a prediction function $y(\mathbf{x}; \mathcal{D})$.
- Different datasets will give different prediction functions.
- The performance of a particular learning algorithm is then assessed by taking the average over the ensemble of these datasets.

• Let us consider the expression:

$$\{y(\mathbf{x}; \mathcal{D}) - h(\mathbf{x})\}^2.$$

• Note that this quantity depends on a particular dataset D.

Bias-Variance Decomposition

• Consider:

$$\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2.$$

• Adding and subtracting the term $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]$, we obtain

$$\begin{aligned}
&\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2 \\
&= \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 \\
&= \{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2 + \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 \\
&\quad +2\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}.
\end{aligned}$$

• Taking the expectation over $\mathcal{D}$, the last term vanishes, so we get:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - h(\mathbf{x})\}^2\right] \\
&= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right]}_{\text{variance}}.
\end{aligned}$$

Bias-Variance Trade-off

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

Average predictions over all datasets differ from the optimal regression function.

Solutions for individual datasets vary around their averages -- how sensitive is the function to the particular choice of the dataset.

Intrinsic variability of the target values.

$$(\text{bias})^2 = \int \{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - h(\mathbf{x})\}^2 p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

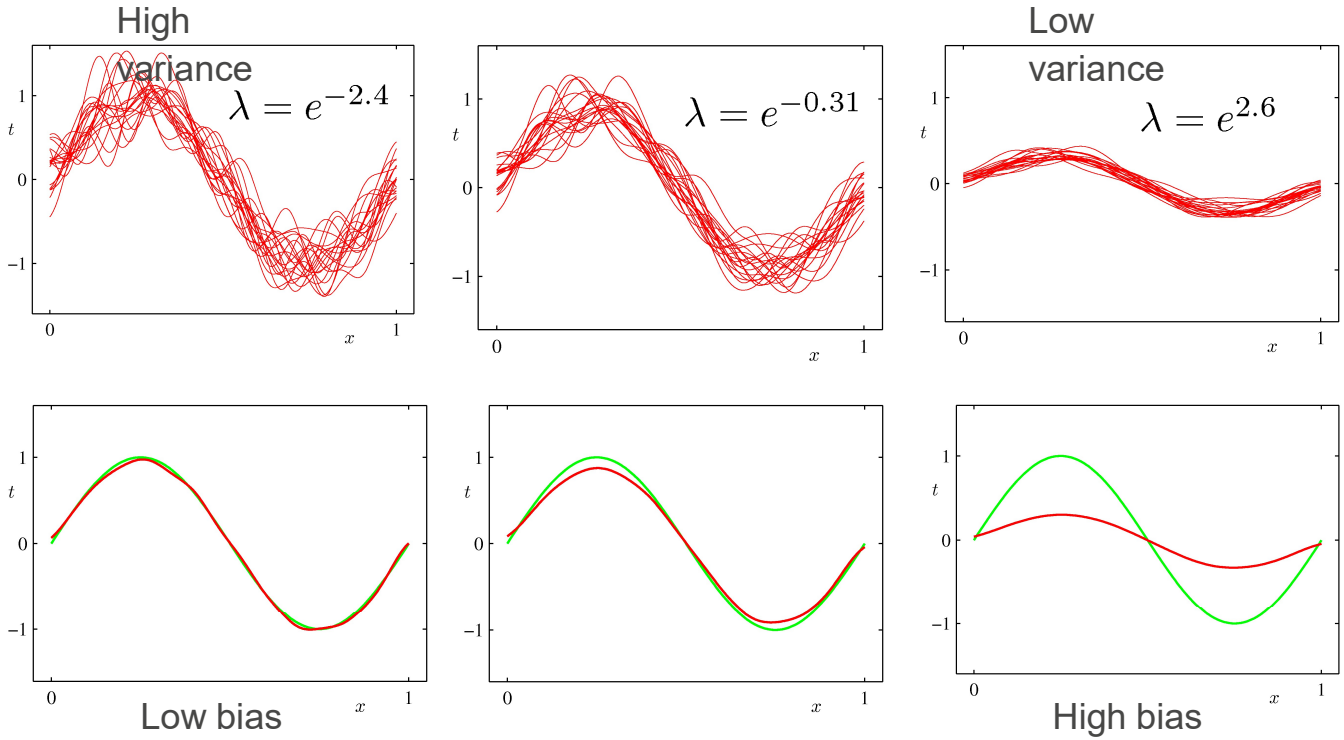
$$\text{variance} = \int \mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})]\}^2\right] p(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

$$\text{noise} = \iint \{h(\mathbf{x}) - t\}^2 p(\mathbf{x},t)\,\mathrm{d}\mathbf{x}\,\mathrm{d}t$$

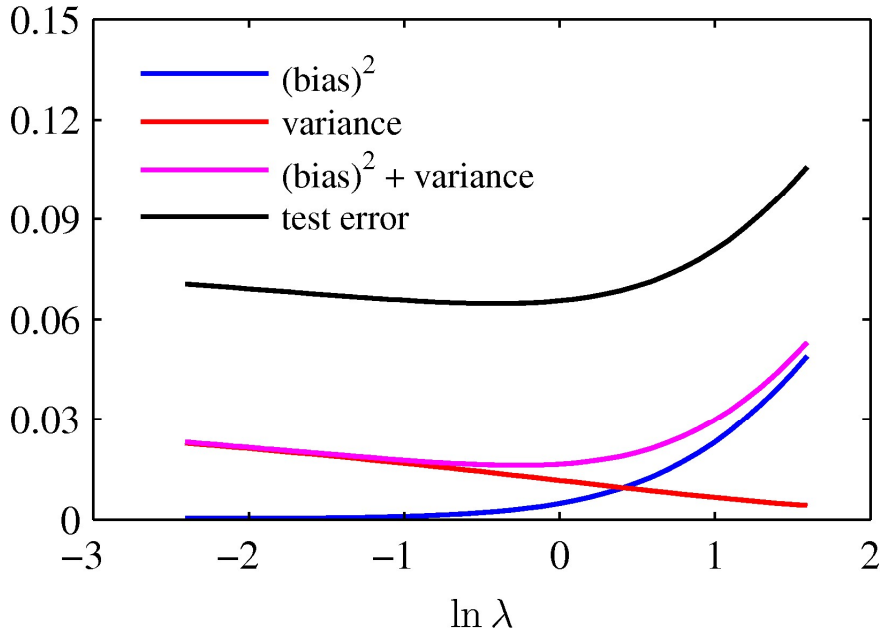
- Trade-off between bias and variance: With very flexible models (high complexity) we have low bias and high variance; With relatively rigid models (low complexity) we have high bias and low variance.
- The model with the optimal predictive capabilities has to balance between bias and variance.

Bias-Variance Trade-off

• Consider the sinusoidal dataset. We generate 100 datasets, each containing N=25 points, drawn independently from $h(x) = \sin 2\pi x$.



Bias-Variance Trade-off



From these plots note that over-regularized model (large λ) has high bias, and under-regularized model (low λ) has high variance.

Beating the Bias-Variance Trade-off

• We can reduce the variance by averaging over many models trained on different datasets:

- In practice, we only have a single observed dataset. If we had many independent training sets, we would be better off combining them into one large training dataset.

• Given a standard training set D of size N, we could generate new training sets, of size N, by sampling examples from D uniformly and with replacement.

- This is called bagging and it works quite well in practice (ad hoc).

• Given enough computation, we could also resort to the Bayesian framework:

- Combine the predictions of many models using the posterior probability of each parameter vector as the combination weight.

Applications to Deep Networks

Overfitting Issues in Deep Networks

Deep nets may have many hidden layers

With limited training data, over-fitting may happen

Methods to reduce overfitting (Regularization)

- Cross-validation set (discussed before)
- Weight regularization, e.g., L_1 and L_2 regularization (discussed before)
- Dropout

In order to understand dropout, we need to first understand bagging.

Bagging (bootstrap aggregating) is a method of averaging over several models to improve generalization. The idea is to train several different models separately, then have all the models vote on the output for test examples.

This is an example of a general strategy in machine learning called model averaging.

- Bagging: average the predictions of all possible settings of the parameters

Bagging

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 co-variances $E(\epsilon_i \epsilon_j) = c$.

Then the error made by the average prediction of all the ensemble models has variance $\frac{v}{k} + c\frac{k-1}{k}$

If $c = 0$, the bagging reduces square error by a factor of k. If $c = v$, then no gains is achieved.

Typically, 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.

Model i is then trained on dataset i. The differences between which examples are included in each dataset result in differences between the trained models.

Bagging

In bagging:

- The classification probability of the ensemble of neural networks is given by the arithmetic mean of all the corresponding distributions
  - Model i produces the prediction probability as $p^{(i)}\ (y|\boldsymbol{x})$
  - Prediction of ensemble of k models is the arithmetic mean $\frac{1}{k}\sum_{i=1}^{k} p^{(i)}\ (y|\boldsymbol{x})$
  - This gives equal weight to ensemble elements predictions
  - It is possible to give unequal weights to ensemble elements giving $\sum_{i=1}^{k} w_i\ p^{(i)}\ (y|\boldsymbol{x})$ with $w_i \geq 0$ and $\sum_{i}^{k} w_i = 1$.
  - Use geometric mean rather than arithmetic mean of the ensemble member's predicted distributions is *intuitively equivalent* to ensemble log-likelihood optimization.

Bagging

Neural networks reach a wide enough variety of solution points that they can often benefit from model averaging even if all the models are trained on the same dataset.

Differences in random initialization, in random selection of mini-batches, or in outcomes of nondeterministic implementations of neural networks are often enough to cause different members of the ensemble to make partially independent errors.

This means that bagging can be very useful in neural networks.

Dropout

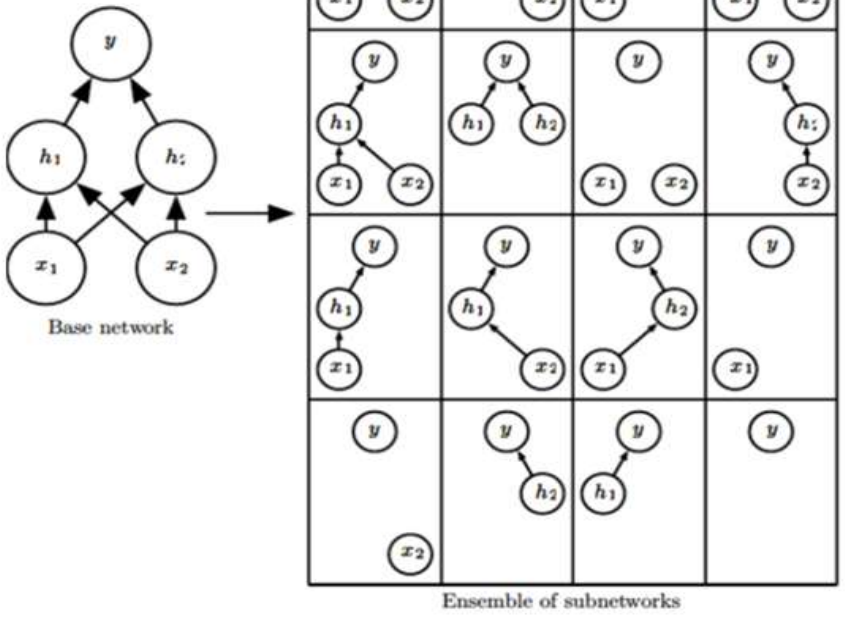
Specifically, dropout trains the ensemble consisting of all subnetworks that can be formed by removing non-output units from an underlying base network.

Parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory.

Dropout aims to approximate the above, but with an exponentially large number of neural networks.

Note that in most modern neural networks, based on a series of affine transformations and nonlinearities, we can effectively remove a unit from a network by multiplying its output value by zero.

Subnetworks Example



Dropout

In bagging, each model is trained to converge on its respective training set.

In dropout, typically most models are not explicitly trained at all.

- A small fraction of the possible subnetworks trained for a single step.

Model parameters for subnetworks are shared

The parameter sharing causes the remaining subnetworks to perform well too.

Dropout

Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.

Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all the input and hidden units in the network.

- The mask for each unit is sampled independently from all the others.
- The probability of sampling a mask value of one (causing a unit to b included) is a hyper-parameter fixed before training begins.
- It is not a function of the current value of the model parameters or the input example.
- Typically, an input unit is included with probability 0.8, and a hidden unit is included with probability 0.5.

We then run forward propagation, back-propagation, and the learning update as usual.

Mathematical Model of Dropout

Denote μ as a mask vector

- which units to include and which ones to be removed

Denote $J(\theta, \mu)$ as the cost of the model prediction

Training with dropout consists of minimizing $\mathbb{E}_{\mu}(J(\theta, \mu))$

- Expected value contains exponential # of terms

One can get an unbiased estimate of its gradient by sampling values of μ

Mask for dropout training

Denote μ as a mask vector

- each subnetwork is this defined by mask vector μ
  - μ determines which units to include and to remove

Hence, each subnetwork outputs a probability distribution $p(y \mid x, \mu)$.

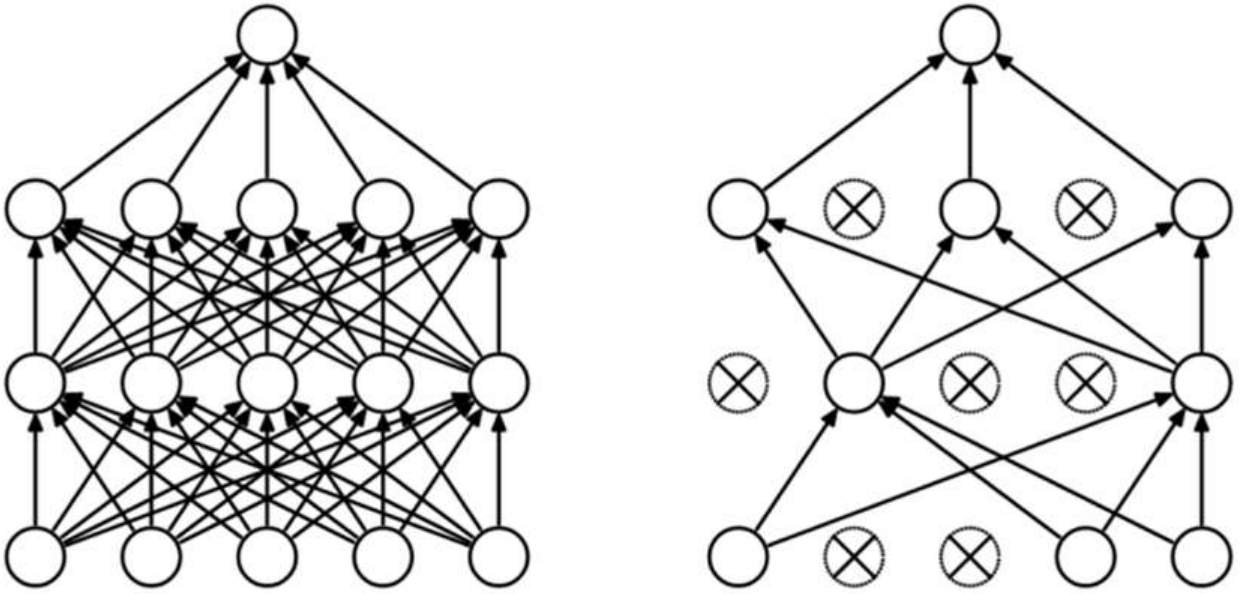
If subnetworks are chosen according to probability/weight $p(\mu)$, then the arithmetic mean **over all masks** is given $\sum_{i=1}^{k} p^{(i)}\,(y|\boldsymbol{x})\,p(\mu)$

$p(\mu)$ is the distribution used to sample μ at training time

Geometric mean is preferred (average $log\,[\,p^{(i)}\,(y|\boldsymbol{x})]$).

This sum may include an exponential number of terms.

Visualization of Drop-out



(a) Standard Neural Net

(b) After applying dropout.

Dropout

Approximate the sum using sampling

1 - By averaging together the output from many masks

- 10-20 masks are sufficient for good performance

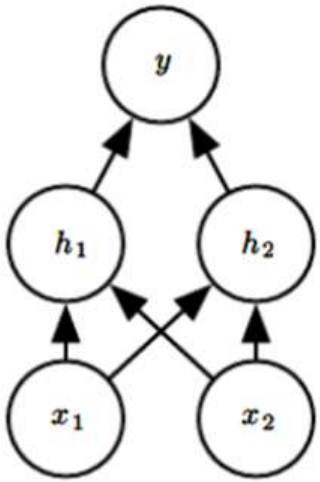
2 - Use geometric mean rather than arithmetic mean of the ensemble member's predicted distributions

$$p_{ensemble}(y|x) = \sqrt[2^d]{\prod_{\mu} p(y|\boldsymbol{x}, \mu)}$$

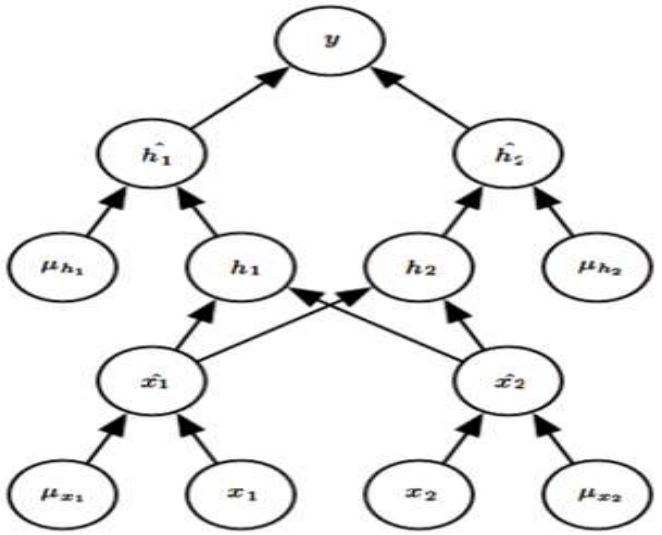
3- We have to make sure that none of the models assign probability zero to any event

Forward Propagation with Dropout

Base network



Forward propagation with dropout



μ is a random binary vector (mask) with one entry for each input

The probability of each entry being 1 is a hyper-parameter, usually 0.5 for the hidden layer, and 0.8 for the input.

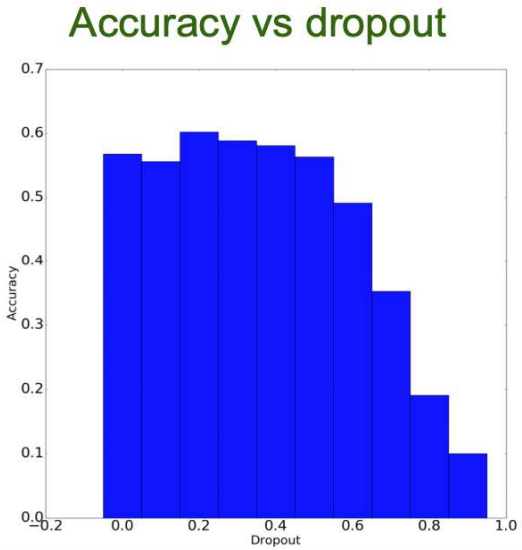
Each unit is multiplied by the corresponding mask μ

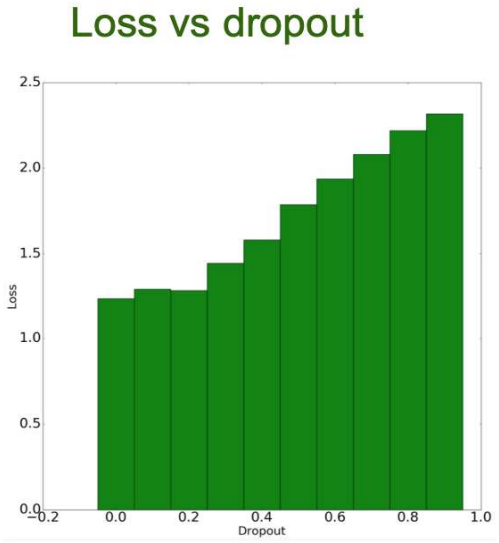
Equivalent to randomly selecting one of the subnetworks of previous slide

What is the effect of probability of drop

CIFAR-10 test dataset

- Three convolution layers of size 64, 128 and 256
- Followed by two densely connected layers of size 512
- output layer dense layer of size 10





Classification accuracy with or without Dropout