ECBM E6040 Neural Networks and Deep Learning

Lecture #5: Machine Learning Basics (cont'd)

Aurel A. Lazar

Columbia University
Department of Electrical Engineering

February 16, 2016

Outline of Part I

Outline of Part II

Part I

Review of Previous Lecture

Topics Covered

- Machine Learning Algorithms
- Capacity, Overfitting and Underfitting
- Estimation, Bias and Variance
- Maximum Likelihood Estimation

Learning Objectives

- Learning as the means of attaining the ability to perform a task.
- Unsupervised learning: learning the probability distribution of observed examples of a dataset.
- Supervised learning: learning the conditional probability distribution of labeled examples of a dataset.
- Generalization in machine learning as the ability to perform well on previously unobserved inputs and not just on data the model was trained on.
- Review of basic concepts in statistics that have applications in machine learning.

Part II

Today's Lecture

Basics of Bayesian Statistics

There are two main modeling approaches/perpectives in statistics:

(i) Frequentist statistics. In the frequentist perspective on statistics the true parameter value $\boldsymbol{\theta}$ is fixed but unknown, while the point estimate $\hat{\boldsymbol{\theta}}$ is a random variable on account of it being a function of the data. The data is assumed to be random.

(ii) Bayesian statistics. The Bayesian perspective on statistics uses probability to reflect degrees of certainty of states of knowledge. The data is directly observed and so it is not assumed to be random. On the other hand, the true parameter $\boldsymbol{\theta}$ is unknown or uncertain and thus it is represented as a random variable.

Bayes Rule

Knowledge about $\boldsymbol{\theta}$ is represented using the prior probability distribution (or prior), $p(\boldsymbol{\theta})$.
Let $\{x^1, ..., x^m\}$ be a set of data samples.
Our belief about $\boldsymbol{\theta}$ is given by the posterior distribution via Bayes rule:

$$p(\boldsymbol{\theta}|x^1, ..., x^m) = \frac{p(x^1, ..., x^m|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(x^1, ..., x^m)},$$

where $p(x^1, ..., x^m|\boldsymbol{\theta})$ is the likelihood of observing the data samples $\{x^1, ..., x^m\}$ given $\boldsymbol{\theta}$.

Comparison of MLE and Bayesian Estimates

Unlike the maximum likelihood point estimate of $\boldsymbol{\theta}$, the Bayesian makes decision with respect to a full distribution over $\boldsymbol{\theta}$. For example, after observing m examples, the predicted distribution over the next data sample, x^{m+1}, is given by

$$\begin{aligned} p(x^{m+1}|x^1,...,x^m) &= \int_{\mathbb{D}} p(x^{m+1},\boldsymbol{\theta}|x^1,...,x^m)d\boldsymbol{\theta} \\ &= \int_{\mathbb{D}} p(x^{m+1}|x^1,...,x^m,\boldsymbol{\theta})p(\boldsymbol{\theta}|x^1,...,x^m)d\boldsymbol{\theta}. \end{aligned}$$

Here each value of $\boldsymbol{\theta}$ with positive probability density contributes to the prediction of the next example, with the contribution weighted by the posterior density itself. After having observed $\{x^1,...,x^m\}$, if we are still quite uncertain about the value of $\boldsymbol{\theta}$, then this uncertainty is incorporated directly into any predictions we might make.

Comparison of MLE and Bayesian Estimates

The frequentist statistics addresses the uncertainty in a given point estimator of $\boldsymbol{\theta}$ by evaluating its variance. The variance of the estimator is an assessment of how the estimate might change with alternative samplings of the observed (or training) data.

The Bayesian answer to the question of how to deal with the uncertainty in the estimator is to simply integrate over it, which tends to protect well against overfitting.

The Bayesian prior shifts the probability mass density towards regions of the parameter space that are preferred a priori. In practice, the prior often expresses a preference for models that are simpler or more smooth. One important effect of the prior is to actually reduce the uncertainty (or entropy) in the posterior density over $\boldsymbol{\theta}$.

A Bayesian Estimation Approach to Learning
Example: Linear Regression

We learn a linear mapping from an input vector $\mathbf{x} \in \mathbb{R}^n$ to predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y} = \mathbf{w}^T \mathbf{x},$$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of parameters called weights.
Given m training samples $(\mathbf{X}^{train}, \mathbf{y}^{train})$, the prediction of y over the entire training set is given by

$$\hat{\mathbf{y}}^{train} = \mathbf{X}^{train}\mathbf{w}.$$

Expressed as Gaussian conditional distribution on $\mathbf{y}^{train}$, we have

$$p(\mathbf{y}^{train}|\mathbf{X}^{train}, \mathbf{w}) = \mathcal{N}(\mathbf{y}^{train}; \mathbf{X}^{train}\mathbf{w}, \mathbf{I})$$

$$\propto \exp\left(-\frac{1}{2}(\mathbf{y}^{train} - \mathbf{X}^{train}\mathbf{w})^T(\mathbf{y}^{train} - \mathbf{X}^{train}\mathbf{w})\right).$$

The assumption here is that the variance of y is 1.

Example: Linear Regression (cont'd)

For real-valued parameters it is common to use a Gaussian as a prior distribution for the model parameter vector $\mathbf{w}$:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \boldsymbol{\mu}_0, \boldsymbol{\Lambda}_0) \propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^T \boldsymbol{\Lambda}_0^{-1} (\mathbf{w} - \boldsymbol{\mu}_0)\right),$$

where $\boldsymbol{\mu}_0$ and $\boldsymbol{\Lambda}_0$ are the prior distribution mean vector and the covariance matrix, respectively. We typically assume a diagonal covariance matrix $\boldsymbol{\Lambda}_0 = \text{diag}(\lambda_0)$.
In what follows, we refer to $(\mathbf{X}^{train}, \mathbf{y}^{train})$ as simply $(\mathbf{X}, \mathbf{y})$. The posterior distribution of the model parameters amounts to

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{X}, \mathbf{y}, \mathbf{w})}{p(\mathbf{X}, \mathbf{y})} = \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{X}, \mathbf{w})}{p(\mathbf{X}, \mathbf{y})} \propto p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})$$

Example: Linear Regression (cont'd)

$$p(\mathbf{w}|\mathbf{X},\mathbf{y}) \propto$$

$$\propto \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w})\right)\exp\left(-\frac{1}{2}(\mathbf{w}-\boldsymbol{\mu}_0)^T\boldsymbol{\Lambda}_0^{-1}(\mathbf{w}-\boldsymbol{\mu}_0)\right)$$

$$\propto \exp\left(-\frac{1}{2}(-2\mathbf{y}^T\mathbf{X}\mathbf{w}+\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}+\mathbf{w}^T\boldsymbol{\Lambda}_0^{-1}\mathbf{w}-2\boldsymbol{\mu}_0^T\boldsymbol{\Lambda}_0^{-1}\mathbf{w})\right).$$

In the derivation above we dropped all terms that do not include the parameter vector $\mathbf{w}$. With the substitution

$$\boldsymbol{\Lambda}_m = (\mathbf{X}^T\mathbf{X}+\boldsymbol{\Lambda}_0^{-1})^{-1} \quad \text{and} \quad \boldsymbol{\mu}_m = \boldsymbol{\Lambda}_m(\mathbf{X}^T\mathbf{y}+\boldsymbol{\Lambda}_0^{-1}\boldsymbol{\mu}_0)$$

we obtain ($\boldsymbol{\Lambda}_m$ is symmetric)

$$p(\mathbf{w}|\mathbf{X},\mathbf{y}) \propto \exp\left(-\frac{1}{2}(\mathbf{w}-\boldsymbol{\mu}_m)^T\boldsymbol{\Lambda}_m^{-1}(\mathbf{w}-\boldsymbol{\mu}_m)+\frac{1}{2}\boldsymbol{\mu}_m^T\boldsymbol{\Lambda}_m^{-1}\boldsymbol{\mu}_m\right).$$

Example: Linear Regression (cont'd)

Finally,

$$p(\mathbf{w}|\mathbf{X}, \mathbf{y}) \propto \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_m)^T \boldsymbol{\Lambda}_m^{-1}(\mathbf{w} - \boldsymbol{\mu}_m)\right).$$

Note that the posterior distribution has the form of a Gaussian distribution with mean vector $\boldsymbol{\mu}_m$ and covariance matrix $\boldsymbol{\Lambda}_m$. This justifies our dropping all terms unrelated to $\mathbf{w}$, since the posterior distribution must be normalized and, as a Gaussian, we know what that normalization constant must be (where n is the dimension of the input):

$$p(\mathbf{w}|\mathbf{X}^{train}, \mathbf{y}^{train}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Lambda}_m|}} \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_m)^T \boldsymbol{\Lambda}_m^{-1}(\mathbf{w} - \boldsymbol{\mu}_m)\right).$$

Example: Linear Regression (cont'd)

Examining this posterior distribution allows us to gain some intuition for the effect of Bayesian inference. In most situations, we set $\boldsymbol{\mu}_0 = 0$. If we set $\boldsymbol{\Lambda}_m = \frac{1}{\alpha}\mathbf{I}$, then $\boldsymbol{\mu}_m$ gives the same estimate of $\mathbf{w}$ as does frequentist linear regression with a weight decay penalty of $\alpha \mathbf{w}^T \mathbf{w}$.

One difference is that the Bayesian estimate is undefined if $\alpha = 0$ - we are not allowed to begin the Bayesian learning process with an infinitely wide prior on $\mathbf{w}$. The more important difference is that the Bayesian estimate provides a covariance matrix, showing how likely all the different values of $\mathbf{w}$ are, rather than providing only the estimate $\boldsymbol{\mu}_m$.

Maximum A Posteriori Estimation (MAP) Basics

The MAP estimate chooses the point of maximal posterior probability (maximal probability density in case of continuous $\boldsymbol{\theta}$):

$$\boldsymbol{\theta}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \,|\, \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}).$$

On the right hand side, $\log p(\boldsymbol{x} \,|\, \boldsymbol{\theta})$, is the standard log likelihood term and $\log p(\boldsymbol{\theta})$ corresponds to the prior distribution.
The prior on $\boldsymbol{\theta}$ effects the MAP estimate by leveraging information other than that contained in the training data. This additional information helps to reduce the variance in the MAP point estimate (in comparison to the ML estimate). However, it does so at the price of increased bias.

Example: Regularized Linear Regression

Given a set of m training samples of input output pairs $(\mathbf{X}^{train}, \mathbf{y}^{train})$, and using the Bayesian approach to linear regression, the prediction of y over the entire training set is:

$$\hat{\mathbf{y}}^{train} = \mathbf{X}^{train}\mathbf{w},$$

where prediction is parametrized by the vector $\mathbf{w} \in \mathbb{R}^n$.
The MLE for the model parameters also minimizes the MSE and it is, therefore, given by

$$\hat{\mathbf{w}}_{ML} = (\mathbf{X}^{(train)T}\mathbf{X}^{train})^{-1}\mathbf{X}^{(train)T}\mathbf{y}^{train}.$$

Assuming that the mean is $\boldsymbol{\mu}_0 = 0$ and the covariance matrix of the prior is $\boldsymbol{\Lambda}_0 = \lambda_0\mathbf{I}$, the MAP estimate of the mean of the Gaussian posterior density becomes

$$\hat{\mathbf{w}}_{MAP} = \boldsymbol{\Lambda}_m\mathbf{X}^{(train)T}\mathbf{y}^{train}.$$

Example: Regularized Linear Regression (cont'd)

Since

$$\mathbf{\Lambda}_m = (\mathbf{X}^{(train)T}\mathbf{X}^{train} + \lambda_0^{-1}\mathbf{I})^{-1}.$$

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^{(train)T}\mathbf{X}^{train} + \lambda_0^{-1}\mathbf{I})^{-1}\mathbf{X}^{(train)T}\mathbf{y}^{train}.$$

Recall that

$$\hat{\mathbf{w}}_{ML} = (\mathbf{X}^{(train)T}\mathbf{X}^{train})^{-1}\mathbf{X}^{(train)T}\mathbf{y}^{train}.$$

Therefore, as the variance of the prior distribution tends to infinity, the MAP estimate tends to the ML estimate. As the variance of the prior tends to zero, the MAP estimate tends to zero (i.e., the value of $\boldsymbol{\mu}_0 = 0$).

Example: Regularized Linear Regression (cont'd)

Here we explore the model capacity tradeoff between the ML estimate and the MAP estimate by analyzing the bias and variance of these estimates.
The ML estimate is unbiased, i.e., $\mathbb{E}\hat{\mathbf{w}}_{ML} = \mathbf{w}$ and the variance is given by

$$\mathsf{Var}(\hat{\mathbf{w}}_{ML}) = (\mathbf{X}^{(train)T}\mathbf{X}^{train})^{-1}.$$

Work it out!

Example: Regularized Linear Regression (cont'd)

We now calculate the average of the MAP estimate:

$$\begin{aligned}\mathbb{E}\hat{\mathbf{w}}_{MAP} &= \mathbb{E}\boldsymbol{\Lambda}_m \mathbf{X}^{(train)T}\mathbf{y}^{train} = \mathbb{E}\boldsymbol{\Lambda}_m[\mathbf{X}^{(train)T}(\mathbf{X}^{train}\mathbf{w} + \boldsymbol{\epsilon})] \\ &= \boldsymbol{\Lambda}_m \mathbf{X}^{(train)T}\mathbf{X}^{train}\mathbf{w} + \boldsymbol{\Lambda}_m \mathbf{X}^{(train)T}\mathbb{E}\,\boldsymbol{\epsilon} \\ &= (\mathbf{X}^{(train)T}\mathbf{X}^{train} + \lambda_0^{-1}\mathbf{I})^{-1}\mathbf{X}^{(train)T}\mathbf{X}^{train}\mathbf{w}\end{aligned}$$

Note that the expected value of the ML estimate is the true parameter value $\mathbf{w}$ (i.e. the parameters that we assume generated the data); the expected value of the MAP estimate is a weighted average of w and the prior mean $\mathbf{w}$.
The bias amounts to

$$\text{Bias}(\hat{\mathbf{w}}_{MAP}) = \mathbb{E}\hat{\mathbf{w}}_{MAP} - \mathbf{w} = -(\lambda_0 \mathbf{X}^{(train)T}\mathbf{X}^{train} + \mathbf{I})^{-1}\mathbf{w}$$

As the variance of the prior $\lambda_0 \to \infty$, the bias tends to zero. As the variance of the prior $\lambda_0 \to 0$, the bias tends to $\mathbf{w}$.

Example: Regularized Linear Regression (cont'd)

In order to compute the variance, we use the identity $\mathrm{Var}(\hat{\theta}) = \mathbb{E}\,\hat{\theta}^2 - (\mathbb{E}\,\hat{\theta})^2$. Let us denote by $\mathbf{\Xi}^{train} = \mathbf{X}^{train}\mathbf{X}^{(train)T}$

$$
\begin{aligned}
&\mathbb{E}\hat{\mathbf{w}}_{MAP}\hat{\mathbf{w}}_{MAP}^T \\
&= \mathbb{E}\mathbf{\Lambda}_m\mathbf{X}^{(train)T}\hat{\mathbf{y}}^{train}\hat{\mathbf{y}}^{(train)T}\mathbf{X}^{train}\mathbf{\Lambda}_m \\
&= \mathbb{E}\mathbf{\Lambda}_m\mathbf{X}^{(train)T}(\mathbf{X}^{train}\mathbf{w}+\boldsymbol{\epsilon})(\mathbf{X}^{train}\mathbf{w}+\boldsymbol{\epsilon})^T\mathbf{X}^{train}\mathbf{\Lambda}_m \\
&= \mathbf{\Lambda}_m\mathbf{\Xi}^{(train)T}\mathbf{w}\mathbf{w}^T\mathbf{\Xi}^{(train)T}\mathbf{\Lambda}_m+\mathbf{\Lambda}_m\mathbf{X}^{(train)T}\mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]\mathbf{X}^{train}\mathbf{\Lambda}_m \\
&= \mathbf{\Lambda}_m\mathbf{\Xi}^{(train)T}\mathbf{w}\mathbf{w}^T\mathbf{\Xi}^{(train)T}\mathbf{\Lambda}_m+\mathbf{\Lambda}_m\mathbf{\Xi}^{(train)T}\mathbf{\Lambda}_m \\
&= \mathbb{E}\hat{\mathbf{w}}_{MAP}\,\mathbb{E}\hat{\mathbf{w}}_{MAP}^T + \mathbf{\Xi}^{(train)T}\mathbf{\Lambda}_m.
\end{aligned}
$$

Example: Regularized Linear Regression (cont'd)

With $\mathbb{E}\hat{\mathbf{w}}_{MAP}\,\hat{\mathbf{w}}_{MAP}^T$ thus computed, the variance of the MAP estimate of our linear regression model is given by:

$$
\begin{aligned}
\mathrm{Var}(\hat{\mathbf{w}}_{MAP}) &= \mathbb{E}\hat{\mathbf{w}}_{MAP}\,\hat{\mathbf{w}}_{MAP}^T - \mathbb{E}\hat{\mathbf{w}}_{MAP}\,\mathbb{E}\hat{\mathbf{w}}_{MAP}^T \\
&= \mathbb{E}\hat{\mathbf{w}}_{MAP}\,\mathbb{E}\hat{\mathbf{w}}_{MAP}^T + \mathbf{\Lambda}_m \mathbf{\Xi}^{(train)T}\mathbf{\Lambda}_m - \mathbb{E}\hat{\mathbf{w}}_{MAP}\,\hat{\mathbf{w}}_{MAP}^T \\
&= \mathbf{\Lambda}_m \mathbf{\Xi}^{(train)T}\mathbf{\Lambda}_m \\
&= (\mathbf{\Xi}^{(train)T} + \lambda_0^{-1}\mathbf{I})^{-1}\mathbf{\Xi}^{(train)T}(\mathbf{\Xi}^{(train)T} + \lambda_0^{-1}\mathbf{I})^{-1}.
\end{aligned}
$$

Recall that

$$
\mathrm{Var}(\hat{\mathbf{w}}_{ML}) = (\mathbf{X}^{(train)T}\mathbf{X}^{train})^{-1} = (\mathbf{\Xi}^{(train)T})^{-1}.
$$

Example: Regularized Linear Regression (cont'd)

To get some intuition, assume that $\mathbf{w}$ is one-dimensional (along with $\mathbf{x}$), it becomes a bit easier to see that, as long as λ_0 is bounded, then

$$\mathsf{Var}(\hat{w}_{ML}) = \frac{1}{\sum_{i=1}^m x_i^2} \mathsf{Var}(\hat{w}_{MAP}) = \frac{\lambda_0 \sum_{i=1}^m x_i^2}{(1 + \lambda_0 \sum_{i=1}^m x_i^2)^2}.$$

The role of the prior in the MAP estimate is to trade increased bias for a reduction in variance. The goal, of course, is to try to avoid overfitting. The incurred bias is a consequence of the reduction in model capacity caused by limiting the space of hypotheses to those with significant probability density under the prior.

Supervised learning algorithms learn to associate some input with some output, given a training set of examples of inputs $\mathbf{x}$ and outputs $\mathbf{y}$.

Most supervised learning algorithms are based on estimating a probability distribution $p(\mathbf{y}|\mathbf{x})$. We can do this simply by using maximum conditional likelihood estimation to find the best parameter vector $\boldsymbol{\theta}$ for a parametric family of distributions $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. The normal distribution over real-valued numbers that we used for linear regression was parameterized in terms of a mean. Linear regression corresponds to the family

$$p(y|\mathbf{x};\boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{\theta}^T\mathbf{x}, \mathbf{I}).$$

We can generalize linear regression to the classification scenario by defining a different family of probability distributions. If we have two classes, class 0 and class 1, then we need to only specify the probability of one of these classes.

Logistic Regression

A distribution over a binary variable is slightly more complicated, because its mean must always be between 0 and 1. One way to solve this problem is to use the logistic sigmoid function to squash the output of the linear function into the interval $(0, 1)$ and interpret that value as a probability:

$$p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T \mathbf{x}).$$

This approach is known as logistic regression. Solution is possible by minimizing the negative log-likelihood ratio via gradient descent.

Data Representation: A Central Theme in Deep Learning

Unsupervised learning algorithms are those that experience only“feature” but not a supervision signal. Informally, unsupervised learning refers to most attempts to extract information from a distribution that do not require human labor to annotate examples. The term is usually associated with density estimation, learning to draw samples from a distribution, learning to denoise data from some distribution, finding a manifold that the data lies near, or clustering the data into groups of related examples.

Learning a Representation of Data

A classic unsupervised learning task is to find the "best" representation of the data. By best we can mean different things, but generally speaking we are looking for a representation that preserves as much information about x as possible while obeying some penalty or constraint aimed at keeping the representation simpler or more accessible than x itself.

What is a simpler representation?

- Low-dimensional representations attempt to compress as much information about x as possible in a smaller representation.
- Sparse representations embed the dataset into a representation whose entries are mostly zeroes for most inputs.
- Independent representations attempt to disentangle the sources of variation underlying the data distribution such that the dimensions of the representation are statistically independent.

Principal Component Analysis (PCA)

A Representation with Uncorrelated Data Elements

Here we will demonstrate that the PCA decorrelates the data representation of an (n, m) design matrix $\mathbf{X}$. We will assume that the data has a mean of zero, $\mathbb{E}\mathbf{x} = 0$. The unbiased sample covariance matrix associated with $\mathbf{X}$ is given by:

$$\text{Var}(\mathbf{x}) = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}.$$

We consider the SVD of the matrix $\mathbf{X}$,

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T,$$

where $\mathbf{\Sigma}$ is an (n, m)-dimensional rectangular diagonal matrix with the singular values of $\mathbf{X}$ on the main diagonal, $\mathbf{U}$ is an (n, n) matrix whose columns are orthonormal (i.e. unit length and orthogonal) and $\mathbf{V}$ is an (m, m) matrix also composed of orthonormal column vectors.

Principal Component Analysis (cont'd)

$$(n-1)\mathrm{Var}(\mathbf{x}) = \mathbf{X}^T\mathbf{X} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

and, due to orthonormality $\mathbf{U}^T\mathbf{U} = \mathbf{I}$,

$$\mathrm{Var}(\mathbf{x}) = \frac{1}{n-1}\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T.$$

Similarly, for $\mathbf{z} = \mathbf{V}\mathbf{x}$, we have

$$\mathrm{Var}(\mathbf{z}) = \frac{1}{n-1}\mathbf{Z}^T\mathbf{Z} = \frac{1}{n-1}\mathbf{V}^T\mathbf{X}^T\mathbf{X}\mathbf{V} = \frac{1}{n-1}\mathbf{\Sigma}^2,$$

since $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. The above analysis shows that when we project the data $\mathbf{x}$ to $\mathbf{x}$, via the linear transformation $\mathbf{V}$, the resulting representation has a diagonal covariance matrix (as given by $\mathbf{\Sigma}^2$) which immediately implies that the individual elements of $\mathbf{z}$ are mutually uncorrelated.

Building Machine Learning Algorithms

All deep learning algorithms are built using a simple recipe: (i) combine a specification of a dataset, (i) a cost function, (iii) an optimization procedure and, (iv) a model.
For example, the linear regression algorithm combines a dataset $(\mathbf{X}, \mathbf{y})$, the cost function

$$J(\mathbf{w}, b) = -\mathbb{E}p_{model}(y|\mathbf{x}),$$

and the model specification

$$p_{model}(y|\mathbf{x}) = \mathcal{N}(y|\mathbf{x}^T\mathbf{x} + b, 1).$$

Typically we then observe that J simplifies to the mean squared error and we can choose to optimize this in closed form by solving the normal equations with the Moore-Penrose pseudo-inverse. By realizing that we can modify any of these components, we can obtain a very wide variety of algorithms.

Building Machine Learning Algorithms (cont'd)

The cost function typically includes at least one term that causes the learning process to perform statistical estimation. The most common cost function is the negative log-likelihood, so that minimizing the cost function causes maximum likelihood estimation.

This main term of the cost function often decomposes as a sum over training examples of some per-example loss function. For example, the negative conditional log-likelihood of the training data can be written as

$$J(\boldsymbol{\theta}) = \mathbb{E}\ L(\mathbf{x}, y, \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\mathbf{x}^i, y^i, \boldsymbol{\theta})$$

where y is the per-example loss $L(\mathbf{x}, y, \boldsymbol{\theta}) = \log p(y|\mathbf{x}; \boldsymbol{\theta}$. We can design many different cost functions just by taking the expectation across the training set of different per-example loss functions.

Building Machine Learning Algorithms (cont'd)

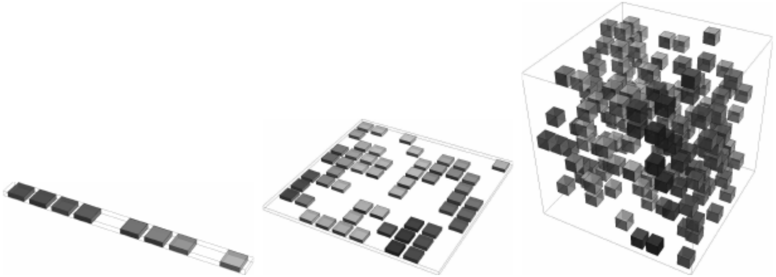
The cost function may also include additional terms, such as regularization terms. For example, we can add weight decay to the linear regression cost function to obtain

$$J(\mathbf{w}, b) = \lambda||\mathbf{x}||_2^2 - \mathbb{E}p_{model}(y|\mathbf{x}),$$

This still allows closed-form optimization.
If the model is non-linear, then most cost functions can no longer be optimized in close form. An iterative numerical optimization procedure, such as gradient descent, provides a solution.

The Curse of Dimensionality

In applications, the number of possible distinct configurations of the variables of interest often increases exponentially with the dimensionality. This is known as the curse of dimensionality.



The main challenge: the number of possible configurations of the variables of interest is much larger than the number of training examples.

Local Constancy and Smoothness Regularization

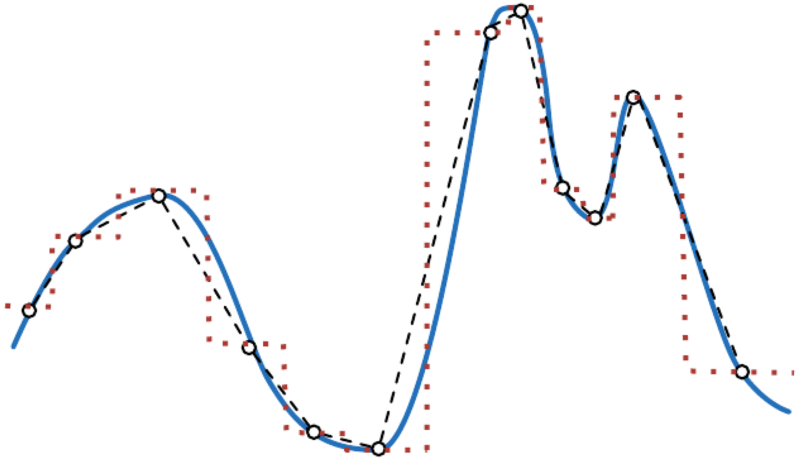
Because in high-dimensional spaces the number of configurations is huge, much larger than the number of examples, most configurations will have no training example associated with it.

To say something meaningful about these configurations, machine learning algorithms need to be guided by prior beliefs about the kind of function they should learn. These priors are incorporated as explicit beliefs in the form of probability distributions over parameters of the model.
Often, implicit "priors" are the smoothness prior or local constancy prior. The prior belief is that the learned function should be smooth or locally constant within a small region:

$$f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \epsilon)$$

for most $\mathbf{x}$ and small change ϵ.

Local Constancy and Smoothness Regularization (cont'd)



Interpolation between neighboring training examples.

Local Constancy and Smoothness Regularization (cont'd)

Non-parametric kernel density estimation methods and kernel regression methods construct a learned function f of the form

$$f(\mathbf{x}) = b + \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}^i)$$

for classification or regression. If the kernel function k always returns a discrete output such as 0 or 1, this includes the cases where f is piecewise constant. However, better results can be obtained if k is continuous. A common choice is the Gaussian kernel of the form

$$k(\mathbf{u} - \mathbf{v}) = \mathcal{N}(\mathbf{u} - \mathbf{v}; 0, \sigma^2 \mathbf{I}).$$

An important class of kernels is the family of local kernels where $k(\mathbf{u} - \mathbf{v})$ is large when $\mathbf{u} = \mathbf{v}$ and decreases as $\mathbf{u}$ and $\mathbf{v}$ grow farther apart from each other. A local kernel can be thought of as a similarity function that performs template matching, by measuring

Local Constancy and Smoothness Regularization (cont'd)

The use of a local kernel means that these methods are still essentially only interpolating between nearby training examples. One can think of the training examples as control knots of an interpolating spline.

Much of the modern motivation for deep learning is derived from studying the limitations of local template matching and how deep models are able to succeed in cases where local template matching fails.