Machine Learning

Adair Antonio da Silva Neto

January 31, 2025

Disclaimer

This text is a work in process. It is based on Stanford's "CS229 - Machine Learning" course and the book [HTF17]. For the Python implementations, check this GitHub page.

Contents

Chapter 1

Supervised Learning

1.1 Linear Regression and Gradient Descent

1.1.1 Linear Regression

Given a set of data x, our goal is to predict y, i.e., to find a **hypothesis** h such that $h(x) \approx y$.

In linear regression, we suppose that the relationship is linear:

$$h_\theta(x) = \theta_0 + \sum_{i=1}^{d} \theta_i x_i = \sum_{i=0}^{d} \theta_i x_i = \theta^T x$$

with $x_0 = 1$.

So our goal is to find the parameters θ_i that minimize the **cost function**

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

where n is the number of elements in the training set.

1.1.2 Batch / Stochastic Gradient Descent

Our first approach is the **batch gradient descent**:

1. Initialize θ_0 (equal to 0 or 1, for example);
2. For each j, update
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Notice that, for a single training example, we have

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{d} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y) x_j
\end{aligned}$$

So, our algorithm becomes to repeat, until convergence, the update

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

for all j.

A less costly alternative is the **stochastic gradient descent**, in which we repeat, for each $i = 1, \dots, n$,

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

for all j.

1.1.3 Normal Equations

To rewrite this procedure in matricial form, we let the **design matrix** X formed by the training inputs in each row, i.e. $(X_{ij}) = x_j^{(i)}$.

Since $h_\theta(x^{(i)}) = (x^{(i)})^T\theta$ and using that $z^Tz = \sum_i z_i^2$, it follows that

$$\frac{1}{2}(X\theta - y)^T(X\theta - y) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 = J(\theta)$$

Differentiating with respect to θ,

$$\begin{aligned}
DJ(\theta) &= D\left(\frac{1}{2}(X\theta - y)^T(X\theta - y)\right) \\
&= \frac{1}{2}D((X\theta)^TX\theta - (X\theta)^Ty - y^T(X\theta) + y^Ty) \\
&= \frac{1}{2}D(\theta^TX^TX\theta - \theta^TX^Ty - y^TX\theta) \\
&= \frac{1}{2}D(\theta^TX^TX\theta - y^TX\theta - y^TX\theta) \\
&= \frac{1}{2}D(\theta^TX^TX\theta - 2(X^Ty)^T\theta) \\
&= \frac{1}{2}(2X^TX\theta - 2X^Ty) \\
&= X^TX\theta - X^Ty
\end{aligned}$$

Setting this derivative to zero, we obtain the **normal equations**

$$X^TX\theta = X^Ty$$

Hence, the value of θ that minimizes $J(\theta)$ is

$$\theta = (X^TX)^{-1}X^Ty$$

1.2 Locally Weighted and Logistic Regression

1.2.1 Locally Weighted Regression

Parametric learning algorithm: fit a fixed set of parameters θ_i to data.

Non-parametric learning algorithm: the amount of data/parameters needed grows with the size of the training set.

In locally weighted regression, we fit θ to minimize

$$\sum_{i=1}^{n} w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$$

where $w^{(i)}$ is a **weighting function**, for example:

$$w^{(i)} = \exp\left(\frac{-(x^{(i)} - x)^2}{2\tau^2}\right)$$

and τ is called the **bandwidth** parameter.

Notice that, if $|x^{(i)} - x|$ is small, then $w^{(i)} \approx 1$. If $|x^{(i)} - x|$ is large, then $w^{(i)} \approx 0$. Here, x is the point where we want to predict.

1.2.2 Probabilistic Interpretation

Why do we use least squares?

Suppose that $y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$, where ε represents the unmodelled effects and random noise. We assume that the $\varepsilon^{(i)}$ are i.i.d and $\varepsilon^{(i)} \sim N(0, \sigma^2)$.

This means that

$$p(y^{(i)} \mid x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

i.e. $(y^{(i)} \mid x^{(i)}; \theta) \sim N(\theta^T x^{(i)}, \sigma^2)$. The ";" means "parameterized by".

The **likelihood** of θ is

$$L(\theta) = p(y \mid x; \theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

And the **log-likelihood** is simply

$$\begin{aligned} l(\theta) = \log(L(\theta)) &= \sum_{i=1}^{m}\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \log\exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)\right) \\ &= m\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \sum_{i=1}^{m}\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \end{aligned}$$

The **maximum likelihood estimation (MLE)** is (obviously!) to choose θ to maximize $L(\theta)$, which is the same as choosing θ to minimize

$$\frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2 = J(\theta)$$

This proves that minimizing the least square errors is finding the maximum likelihood estimate.

1.2.3 Logistic Regression

We consider a binary classification problem, i.e., $y \in \{0, 1\}$. Thus, $h_\theta(x) \in [0, 1]$.

In logistic regression, we choose

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is the **sigmoid** or **logistic** function.

Since

$$p(y = 1 \mid x; \theta) = h_\theta(x)$$

it follows that

$$p(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Hence,

$$p(y \mid x; \theta) = h_\theta(x)^y (1 - h_\theta(x))^{1-y}$$

In this case, the likelihood is

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) = \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

and the log likelihood,

$$l(\theta) = \sum_{i=1}^{m} [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

To choose θ to maximize $l(\theta)$, we use **batch gradient ascent**:

$$\theta_j := \theta_j + \alpha \frac{\partial}{\partial \theta_j} l(\theta)$$

Computing this partial derivative, we obtain

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

1.2.4 Newton's Method

Suppose we have a function f and want to find θ such that $f(\theta) = 0$. In our case, we have $f = l'$. It is simply to iterate

$$\theta^{(t+1)} := \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

When θ is a vector,

$$\theta^{(t+1)} := \theta^{(t)} - H^{-1} Dl(\theta^{(t)})$$

where H is the Hessian matrix.

The Newton's method has quadratic convergence.

1.3 Perceptron and Generalized Linear Model

1.3.1 Perceptron

The **perceptron** is similar to logistic regression, but instead of the sigmoid function, we use

$$g(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Again, we have $h_\theta(x) = g(\theta^T x)$ and the update rule is

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta^{(i)}(x)) x_j^{(i)}$$

Notice that $y^{(i)} - h_\theta^{(i)}(x)$ will be 0 if the algorithm got it right, or ± 1 if it got it wrong.

1.3.2 Exponential Family

The **exponential family** is a class of probability distributions with p.d.f.

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$$

where y is the data, η is the **natural parameter**, $T(y)$ is a sufficient statistic, $b(y)$ is the **base measure**, and $a(\eta)$ is the **log-partition function**.

Remark that $e^{-a(\eta)}$ is a normalization constant to make sure that $p(y;\eta)$ sums, over y, to 1.

Example (Bernoulli Distribution). Let φ be the probability of the event. Then,

$$\begin{aligned} p(y;\varphi) &= \varphi^y(1-\varphi)^{(1-y)} \\ &= \exp(\log(\varphi^y(1-\varphi)^{(1-y)})) \\ &= \exp\left[\log\left(\frac{\varphi}{1-\varphi}\right)y - \log(1-\varphi)\right] \end{aligned}$$

We have that $b(y) = 1$, $T(y) = y$, $\eta = \log\left(\frac{\varphi}{1-\varphi}\right)$ (i.e. $\varphi = \frac{1}{1+e^{-\eta}}$), and $a(\eta) = -\log(1-\varphi)$ (i.e. $-\log(1-\frac{1}{1+e^{-\eta}}) = \log(1+e^{\eta})$).

Example (Gaussian). Assume that $\sigma^2 = 1$. Then,

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}}\exp\left(\frac{-(y-\mu)^2}{2}\right) \\ &= \frac{1}{\sqrt{2\pi}}e^{\frac{-y^2}{2}}\exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

We have $b(y) = \frac{1}{\sqrt{2\pi}}e^{\frac{-y^2}{2}}$, $T(y) = y$, $\eta = \mu$, and $a(\eta) = \frac{\mu^2}{2} = \frac{\eta^2}{2}$.

Properties of Exponential Families.

1. The MLE with respect to η is concave, i.e., the negative log-likelihood is convex.
2. The expectation of y is
$$E[y;\eta] = \frac{\partial}{\partial\eta}a(\eta)$$
3. The variance is
$$\mathrm{Var}[y;\eta] = \frac{\partial^2}{\partial\eta^2}a(\eta)$$

In applications, we use the Gaussian distribution for real-valued data, Bernoulli for binary data, Poisson for counting, gamma or exponential distributions for positive real numbers, and beta and Dirichlet distributions.

1.3.3 Generalized Linear Models

In **generalized linear models**, we do the following assumptions (design choices):

1. $y \mid x;\theta$ is a member of exponential family of η.
2. $\eta = \theta^T x$, with $\theta, x \in \mathbf{R}^n$.
3. Test time: output will be $h_\theta(x) = E[y \mid x;\theta]$.

The idea is that given x, we have a linear model $\theta^T x$, which yields our parameter η. Then, based on our problem, we choose an exponential family and output the expectation.

During learning (training), we do gradient ascent on

$$\max_\theta \log p(y^{(i)}; \theta^T x^{(i)})$$

The learning update rule is the same for all distributions:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta^{(i)}(x))x_j^{(i)}$$

Terminology:

1. η is the **natural parameter**.

2. The expectation $E[y; \eta] = g(\eta)$ is the **canonical response function**.

3. The inverse $\eta = g^{-1}(\mu)$ is the **canonical link function**.

For the logistic regression, we use the Bernoulli distribution, whence

$$h_\theta(x) = E[y \mid x; \theta] = \varphi = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-\theta^T x}}$$

For the linear regression, we use the Gaussian distribution. Then,

$$h_\theta(x) = E[y \mid x; \theta] = \mu = \eta = \theta^T x$$

1.3.4 Softmax Regression (Multiclass Classification)

Consider a classification problem in which the response variable y can take $k \in \mathbf{N}$ possible values. For example, we may classify e-mails into three classes: spam, personal and work.

The labels y are of the form $\{0, 1\}^k$ (i.e. a vector with k entries, where all are 0 except for one which is equal to 1).

Each class k has its own set of parameters $\theta_{\text{class}} \in \mathbf{R}^n$.

Given x, we compute $\theta_{\text{class}}^T x$, exponentiate it and then normalize to obtain a probability distribution $\hat{p}$:

$$p(y = i \mid x; \theta) = \frac{e^{\theta_i^T x}}{\sum_{i=1}^k e^{\theta_j^T x}}$$

To obtain the real distribution p, we need to minimize the distance between the distributions, i.e., to minimize the cross-entropy between the two distributions.

The **cross-entropy** is

$$\text{CrossEnt}(p, \hat{p}) = -\sum_{y=1}^k p(y) \log \hat{p}(y) = -\log \hat{p}(y_0) = -\log \frac{e^{\theta_i^T x}}{\sum_{i=1}^k e^{\theta_j^T x}}$$

We treat this as the loss function and do the gradient descent.

1.4 Generative Learning Algorithms

A **discriminative learning algorithm** (e.g. logistic regression) learns $p(y \mid x)$, i.e., it learns

$$h_\theta = \begin{cases} 0 \\ 1 \end{cases}$$

directly.

A **generative learning algorithm** learns $p(x \mid y)$, i.e., it looks at the features x given the class y. It also learns $p(y)$, the **class prior**. That means that we look at each class at a time.

Using Bayes rule, given a new test example, we compute

$$p(y = 1 \mid x) = \frac{p(x \mid y = 1)p(y = 1)}{p(x)}$$

where $p(x) = p(x \mid y = 1)p(y = 1) + p(x \mid y = 0)p(y = 0)$.

1.4.1 Gaussian Discriminant Analysis (GDA)

Suppose $x \in \mathbf{R}^n$ (drop $x_0 = 1$ convention). The key assumption is that $p(x \mid y)$ is Gaussian. We model

$$\begin{aligned} y &\sim \text{Bernoulli}(\varphi) \\ x \mid y = 0 &\sim N(\mu_0, \Sigma) \\ x \mid y = 1 &\sim N(\mu_1, \Sigma) \end{aligned}$$

Our goal is to maximize the joint likelihood

$$L(\varphi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \varphi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{m} p(x^{(i)} \mid y^{(i)})p(y^{(i)})$$

Using MLE, we compute

$$\max_{\varphi,\mu_0,\mu_1,\Sigma} l(\varphi, \mu_0, \mu_1, \Sigma)$$

and obtain, for φ,

$$\varphi = \frac{\sum_{i=1}^m y^{(i)}}{m} = \frac{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=1\}}}{m}$$

For μ_0 and μ_1, we obtain the averages

$$\mu_0 = \frac{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=0\}} x^{(i)}}{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=0\}}}$$

and

$$\mu_1 = \frac{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=1\}} x^{(i)}}{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=1\}}}$$

For Σ, we have

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

Having fitted these parameters, to make a prediction, we choose

$$\arg\max_y p(y \mid x) = \arg\max_y \frac{p(x \mid y)p(y)}{p(x)} = \arg\max_y p(x \mid y)p(y)$$

1.4.2 Generative and Discriminative Comparison

In the discriminative model, we maximize the conditional likelihood

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

For fixed parameters, if we plot $p(y = 1 \mid x; \varphi, \mu_0, \mu_1, \Sigma)$ as a function of x, we have the sigmoid function.

When is the GDA superior and when is the discriminative model superior?

The GDA implies logistic regression, but the converse is not true. This means that the GDA has stronger assumptions and thus it does better (if the assumptions hold).

We can also prove that if

$$\begin{aligned} y &\sim \text{Bernoulli}(\varphi) \\ x \mid y = 0 &\sim \text{Poisson}(\lambda_0) \\ x \mid y = 1 &\sim \text{Poisson}(\lambda_1) \end{aligned}$$

then $p(y = 1 \mid x)$ is also logistic.

This means that logistic regression works fine for Gaussian and Poisson data. However, if we assume our data is Poisson and it is Gaussian, for example, the model will work poorly.

GDA is also more computationally efficient.

1.4.3 Naive Bayes

Consider the e-mail spam classification problem. How can we represent it as a feature vector?

Take all the words in the dictionary (or the top 10000 most used words) and define a vector x by putting $x_i = 1$ if the ith word of the dictionary appears in the e-mail and $x_i = 0$ otherwise. I.e., $x_i = \mathbf{1}_{\{\text{word } i \text{ appears in e-mail}\}}$. This is called the **multi-variate Bernoulli event model**.

We want to model $p(x \mid y)$ and $p(y)$. If we have 10000 words, there are 2^{10000} possible values of x.

Assume that the x_i's are conditionally independent given y. This means that

$$\begin{aligned} p(x_1, \ldots, x_{10000} \mid y) &= p(x_1 \mid y) p(x_2 \mid x_1, y) \cdots p(x_{10000} \mid x_1, x_2, \ldots, y) \\ &= p(x_1 \mid y) p(x_2 \mid y) \cdots p(x_{10000} \mid y) \end{aligned}$$

The parameters of this model are:

- If the word j is spam, what is the chance of it appearing?

$$\varphi_{j|y=1} = p(x_j = 1 \mid y = 1)$$

- If the word j is not spam, what is the chance of it appearing?

$$\varphi_{j|y=0} = p(x_j = 1 \mid y = 0)$$

- What is the chance that the next e-mail you receive is spam?

$$\varphi_y = p(y = 1)$$

The joint likelihood is

$$L(\varphi_y, \varphi_{j|y}) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \varphi_y, \varphi_{j|y})$$

The MLE yields

$$\varphi_y = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}}}{m}$$

and

$$\varphi_{j|y=1} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{x_j^{(i)}=1,\ y^{(i)}=1\}}}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=1\}}}$$

At prediction time, we compute

$$p(y = 1 \mid x) = \frac{p(x \mid y = 1)p(y = 1)}{p(x \mid y = 1)p(y = 1) + p(x \mid y = 0)p(y = 0)}$$

1.4.4 Laplace smoothing

Notice that if we have a new word, the probability will be zero divided by zero. To deal with this problem, we use Laplace smoothing.

In Laplace smoothing, we compute

$$p(x = j) = \frac{\sum_{j=1}^{m} \mathbf{1}_{\{x^{(i)}=j\}} + 1}{m + k}$$

which then yields, in the previous example,

$$\varphi_{j|y=0} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{x_j^{(i)}=1,\ y^{(i)}=0\}} + 1}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} + 2}$$

1.4.5 Event models

Instead of representing x as a vector of zeroes and ones, we can describe it as a vector in $\mathbf{R}^{n_i}$, where n_i is the length of the e-mail i, and each entry $x_j \in \{1, \ldots, 10000\}$ represents the j-th word of the e-mail. For example, if the first word of the e-mail corresponds to the number 1600, then $x_1 = 1600$. This is the **multinomial event model**.

In this model, we have (just as in Naive Bayes):

$$p(x, y) = p(x \mid y)p(y) = \prod_{j=1}^{n} p(x_j \mid y)p(y)$$

and now the parameters are - $\varphi_y = p(y = 1)$; - The choice of the j-th word being k if $y = 0$, i.e.,

$$\varphi_{k|y=0} = p(x_j = k \mid y = 0)$$

- The choice of the j-th word being k if $y = 1$.

The MLE is

$$\varphi_{k|y=0} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} \sum_{j=1}^{n_i} \mathbf{1}_{\{x_j^{(i)}=k\}}}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} n_i}$$

The denominator is the total number of words in all the non-spam emails, and the numerator counts, for non-spam emails, how many words in that email are the word k.

With Laplace smoothing,

$$\varphi_{k|y=0} = \frac{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} \sum_{j=1}^{n_i} \mathbf{1}_{\{x_j^{(i)}=k\}} + 1}{\sum_{i=1}^{m} \mathbf{1}_{\{y^{(i)}=0\}} n_i + 10000}$$

1.5 Support Vector Machines

Support vector machines generalize decision boundaries for classification.

Our labels will be $y = \pm 1$, h will output a value ± 1 and

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Dropping the $x_0 = 1$ convention, our hypothesis will be

$$h_{w,b}(x) = g(w^T x + b)$$

1.5.1 Optimal Margin Classifier (Separable Case)

Consider the logistic classifier $h_\theta(x) = g(\theta^T x)$, which predicts one if $\theta^T x > 0$ and zero otherwise. So, if $y^{(i)} = 1$, we want that $\theta^T x^{(i)} \gg 0$, and if $y^{(i)} = 0$, we want that $\theta^T x \ll 0$.

We define the **functional margin** of the hyperplane defined by (w, b) with respect to $(x^{(i)}, y^{(i)})$ as

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

We want $\hat{\gamma}^{(i)} \gg 0$. Notice that $\hat{\gamma}^{(i)} > 0$ means that $h(x^{(i)}) = y^{(i)}$.

We define the **functional margin** with respect to the entire training set as

$$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}$$

The functional margin measures how confident/accurate our classification is.

We can suppose, by rescaling, that $||w|| = 1$.

The **geometric margin** of the hyperplane defined by (w, b) with respect to $(x^{(i)}, y^{(i)})$ is

$$\gamma^{(i)} = \frac{y^{(i)}(w^T x^{(i)} + b)}{||w||}$$

Intuitively, the geometric margin is the Euclidean distance between the example $(x^{(i)}, y^{(i)})$ and the decision boundary.

Remark that the functional and geometric margin are related by

$$\gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{||w||}$$

The **geometric margin** with respect to the entire training set is defined as

$$\gamma = \min_i \gamma^{(i)}$$

The **optimal margin classifier** chooses the parameters w and b to maximize the geometric margin γ, that is, to

$$\begin{aligned} \max_{w,b} \quad & \gamma \\ \text{s.t.} \quad & \frac{y^{(i)}(w^T x^{(i)} + b)}{||w||} \geq \gamma, \quad i = 1, \dots, m \end{aligned}$$

We can reformulate this problem into

$$\begin{aligned} \min_{w,b} \quad & ||w||^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

Now, let us suppose that the parameter w can be written as a linear combination of the training examples, i.e.,

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

The **representer theorem** states that we can make this assumption without losing performance.

This is a reasonable assumption because using logistic regression with gradient descent (stochastic or batch), for example, no matter how many iterations, the parameter θ (or w) is always a linear combination of the training examples.

Another intuition for this is that w lies in the span of the training examples.

Our goal is to

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \dots, m \end{aligned}$$

Replacing our expression for w, we have

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}\right)^T \left(\sum_{j=1}^{m} \alpha_j y^{(j)} x^{(j)}\right) \\ &= \min_{w,b} \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)^T} x^{(j)} \\ &= \min_{w,b} \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle \end{aligned}$$

and

$$y^{(i)}\left(\left(\sum_{j=1}^{m} \alpha_j y^{(j)} x^{(j)}\right)^T x^{(i)} + b\right) = y^{(i)}\left(\sum_{j=1}^{m} \alpha_j y^{(j)} \langle x^{(j)}, x^{(i)} \rangle + b\right) \geq 1$$

The procedure here is 1. Solve for α_i and b; 2. Predict

$$h_{w,b}(x) = g(w^T x + b) = g\left(\sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b\right)$$

Using convex optimization, the problem above can be further simplified. This is called the **dual optimization problem**.

1.5.2 Kernels

The **kernel trick** is the following:

1. Write the algorithm in terms of inner products $\langle x^{(i)}, x^{(j)} \rangle = \langle x, z \rangle$.
2. Define a mapping from the set of features to higher-dimensional features, i.e., $x \mapsto \varphi(x)$.
3. Compute the **kernel function** $K(x, z) = \varphi(x)^T \varphi(z)$.
4. Replace $\langle x, z \rangle$ in the algorithm with $K(x, z)$.

The **support vector machine** is taking the optimal margin classifier and applying the kernel trick to it.

How to make kernels? If x and z are similar, then $K(x,z)$ should be large. Conversely, if x and z are dissimilar, then $K(x,z)$ is small. For example, we may define the **Gaussian kernel**

$$K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

This is a valid kernel if there exists a function φ such that $K(x,z) = \varphi(x)^T\varphi(z)$. Thus, it is necessary that $K(x,z)$ be a positive semi-definite function.

To prove that this is the case, let $\{x^{(1)},\dots,x^{(d)}\}$ be d points, and $K \in \mathbf{R}^{d\times d}$ be the **kernel matrix** defined by $K_{ij} = K(x^{(i)},x^{(j)})$.

Given any vector z,

$$\begin{aligned}
z^TKz &= \sum_i\sum_j z_iK_{ij}z_j \\
&= \sum_i\sum_j z_i\varphi(x^{(i)})^T\varphi(x^{(j)})z_j \\
&= \sum_i\sum_j z_i\sum_k \varphi_k(x^{(i)})\varphi_k(x^{(j)})z_j \\
&= \sum_k\sum_i\sum_j z_i\varphi_k(x^{(i)})\varphi_k(x^{(j)})z_j \\
&= \sum_k\left(\sum_i z_i\varphi_k(x^{(i)})\right)^2 \geq 0
\end{aligned}$$

This proves that the kernel matrix K is positive semi-definite. It turns out that this is also a sufficient condition.

Theorem (Mercer). K is a valid kernel function (i.e., there exists a function φ such that $K(x,z) = \varphi(x)^T\varphi(z)$) iff. for any d points, the corresponding kernel matrix K is positive semi-definite.

Another example of kernel is the **linear kernel** $K(x,z) = x^Tz$, with $\varphi(x) = x$.

1.5.3 Inseparable Case

If the data is not linearly separable, we reformulate our optimization as follows:

$$\begin{aligned}
\min_{w,b} \quad & \frac{1}{2}\|w\|^2 + c\sum_{i=1}^m \xi_i \\
\text{s.t.} \quad & y^{(i)}(w^Tx^{(i)}+b) \geq 1-\xi_i, \quad i=1,\dots,m
\end{aligned}$$

where $\xi_i \geq 0$. This is called the **L_1-norm soft margin SVM**.

The problem can also be simplified using the same calculation as the representer theorem.

Some examples of applications of SVMs are handwritten digit classification and protein sequence classifier.

1.6 Data Splits, Models and Cross-Validation

1.6.1 Bias and Variance

There is a tradeoff between bias and variance. Simple models have large bias, but small variance (underfitting). Too complex models have small bias, but large variance (overfitting).

1.6.2 Regularization

Regularization consists of adding a term, a **regularizer** $R(\theta)$, to the cost function, i.e.

$$J_\lambda(\theta) = J(\theta) + \lambda R(\theta)$$

If $\lambda = 0$, we return to the previous model. If we increase the value of λ, the term $\lambda R(\theta)$ penalizes the parameter θ being too big, making it harder to overfit the data. However, if λ is too large, we have underfitting.

An example of a regularizer, for linear regression, is $||\theta||^2/2$. Another one, for logistic regression, is $-||\theta||^2$.

Let us give an informal proof of why this procedure works. Given a training set S, we aim to find the most likely value of θ. By Bayes' rule,

$$p(\theta \mid s) = \frac{p(s \mid \theta)p(\theta)}{p(s)}$$

and we want to find

$$\arg\max_\theta p(\theta \mid s) = \arg\max_\theta p(s \mid \theta)p(\theta)$$

If we assume that $p(\theta)$ is Gaussian with $\theta \sim N(0, \tau^2 I)$ (this is our prior distribution for θ), we obtain the regularization technique above.

1.6.3 Train/Dev/Test splits

We divide the dataset S into three groups: S_{train}, S_{dev} and S_{test}.

Train each model (e.g. select the degree of the polynomial) on S_{train}, obtaining some hypothesis.

Then we measure the error on S_{dev} and pick the model with the lowest error. This helps not to overfit.

Finally, we evaluate the algorithm on the test set S_{test}.

1.6.4 Model Selection and Cross-Validation

For small datasets, one can use the k-**fold cross-validation**:

1. Divide the dataset into k subsets.
2. For $i = 1, \dots, k$, train on the $k - 1$ pieces, and test on the remaining one piece.
3. Compute the average test error.

Another strategy, for very small datasets, is the **leave-one-out cross validation**.

Which features should we select?

1. Start with a set of features $\mathcal{F} = \emptyset$.
2. Repeat:
   (a) Add each feature i to $\mathcal{F}$, and verify which single feature addition most improves the dev set performance.
   (b) Add that feature to $\mathcal{F}$.
   (c) Do this until adding a feature hurts the performance.

1.7 Approx/Estimation Error and ERM

We assume that our data is i.i.d., that is, the train and test set have the same distributions and the samples are independent.

Our process is as follows:

1. Given a set S of examples;
2. Sample from the data generating process D;
3. Feed this into a learning algorithm (estimator);
4. The output of the algorithm is the parameters of our hypothesis. The distribution of the parameter is called the sampling distribution.

Bias checks if the sampling distribution is centred around the true parameter (accuracy).

Variance measures how dispersed the sampling distribution is (precision).

Both bias and variance are properties of the first and second moments of the sampling distributions.

The algorithm is **consistent** if our parameter converges to the true parameter as m goes to infinity, i.e.,

$$\lim_{m\to\infty} \hat{\theta} = \theta^*$$

The estimator is an **unbiased estimator** if the sampling distribution is centred around the true parameter:

$$E[\hat{\theta}] = \theta^*$$

How can we 'fight' variance? One way is by taking $m \to \infty$. An alternative is using regularization.

Definition 1.7.1 (Risk/Generalization Error)**.** We define **risk** or **generalization error** as

$$\varepsilon(h) = E_{(x,y)\sim D}[\mathbf{1}_{h(x)\neq y}]$$

If g is the best possible hypothesis, then $\varepsilon(g)$ is the **Bayes error**, also called **irreducible error**.

The **approximation error** is $\varepsilon(h^*) - \varepsilon(g)$, where h^* is the best hypothesis in a class $\mathcal{H}$. It measures the price paid for limiting the hypothesis to a particular class.

The **estimation error** is $\varepsilon(\hat{h}) - \varepsilon(h^*)$.

Notice that $\varepsilon(\hat{\varepsilon})$ is the sum of estimation, approximation and irreducible errors.

The estimation error can be decomposed into estimation variance plus the estimation bias. Bias is the estimation bias plus the approximation error.

How do we 'fight' high bias? We can make the hypothesis class $\mathcal{H}$ bigger.

1.7.1 Empirical Risk Minimizer

Definition 1.7.2 (Empirical Risk)**.** The **empirical risk** is defined as

$$\hat{\varepsilon}_S(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{h(x^{(i)})\neq y^{(i)}}$$

The **empirical risk minimizer (ERM)** is a learning algorithm in which

$$\hat{h}_{ERM} = \arg\min_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}_{h(x^{(i)})\neq y^{(i)}}$$

1.7.2 Uniform Convergence

Consider the following questions. First, how does the empirical risk $\hat{\varepsilon}(h)$ and the generalization error $\varepsilon(h)$ compare? And how does the generalization error of our learned hypothesis $\varepsilon(\hat{h})$ compare with the best possible generalization error in the class $\varepsilon(h^*)$?

We will use two tools: the union bound ($P(A_1 \cup \cdots \cup A_k) \leq P(A_1) + \cdots + P(A_k)$) and the Hoeffding's inequality.

The **Hoeffding's inequality** states that, if $Z_1, \ldots, Z_m \sim \text{Bern}(\varphi)$, $\hat{\varphi} = \frac{1}{m}\sum_{i=1}^m Z_i$, and $\gamma > 0$ (margin), then

$$P[|\hat{\varphi} - \varphi| > \gamma] \leq 2\exp(-2\gamma^2 m)$$

Hence, for each hypothesis h_i,

$$P[|\hat{\varepsilon}(h_i) - \varepsilon(h_i)| > \gamma] \leq 2\exp(-2\gamma^2 m)$$

This means that, if we increase the size m, the empirical error will be more concentrated around the generalization error. Now our goal is to generalize this result for all h_i.

For a finite hypothesis class $\mathcal{H}$, with k elements, we use the union bound:

$$P[\exists h \in \mathcal{H} : |\hat{\varepsilon}(h) - \varepsilon(h)| > \gamma] \leq 2k\exp(-2\gamma^2 m)$$

which yields

$$P[\forall h \in \mathcal{H} : |\hat{\varepsilon}(h) - \varepsilon(h)| \leq \gamma] > 1 - 2k\exp(-2\gamma^2 m)$$

The value $\delta = 2k\exp(-2\gamma^2 m)$ is the **probability error** and γ is the **margin of error**. Fixing $\gamma, \delta > 0$, we have the **sample complexity** result:

$$m \geq \frac{1}{2\gamma^2}\log\frac{2k}{\delta}$$

Let us move to the second question now. We know that

$$\varepsilon(\hat{h}) \leq \hat{\varepsilon}(\hat{h}) + \gamma \leq \hat{\varepsilon}(h^*) + \gamma \leq \varepsilon(h^*) + 2\gamma$$

Thus, with probability $1 - \delta$,

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\sqrt{\frac{1}{2m}\log\frac{2k}{\delta}}$$

The case for infinite dimension uses VC dimension, which puts a size to $\mathcal{H}$, denoted by $VC(\mathcal{H})$. In this case,

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + O\left(\sqrt{\frac{VC(\mathcal{H})}{m}\log\frac{m}{VC(\mathcal{H})} + \frac{1}{m}\log\frac{1}{\delta}}\right)$$

1.8 Decision Trees and Ensemble Methods

We aim to split a given region R_p (the feature space) into a set of rectangles. To choose the splits, we define a function $L(R)$, which is the loss on each rectangle R.

Given C classes, define $\hat{p}_c$ to be the proportion of examples in R that are of class c. Then we have a **misclassification error**

$$L_{\text{misclass}} = 1 - \max_c \hat{p}_c$$

and

$$\max[L(R_p) - (L(R_1) + L(R_2))]$$

Misclassification loss is not great, since it is not very sensitive. We could, instead, use **cross-entropy loss**

$$L_{\text{cross}} = \sum_c \hat{p}_c \log \hat{p}_c$$

1.8.1 Regression Trees

For regression trees, we predict

$$\hat{y}_m = \frac{1}{|R_m|} \sum_{i \in R_m} y_i$$

and our loss is

$$L_{\text{squared}} = \frac{1}{|R_m|} \sum_{i \in R_m} (y_i - \hat{y}_m)^2$$

1.8.2 Regularization of Decision Trees

Decision trees are high-variance models. To regularize decision trees, we have the following heuristics:

1. Mininum leaf size;
2. Maximum depth;
3. Maximum number of nodes;
4. Minimum decrease in loss;
5. Prunning (use misclassification with validation set).

Runtime with n examples, f features and depth d: at test time we have $O(d)$, with $d < \log n$. At train time, each point is part of $O(d)$ nodes, the cost of a point at each node is $O(f)$ and the total cost is $O(nfd)$. The data matrix has size nf.

One problem with trees is that there is no additive structure.

Positive points: easy to explain, interpretable, works with categorical variables, and fast.

Negative points: high variance, bad at additive, low predictive accuracy.

1.8.3 Ensembling

Take X_i i.i.d. random variables. If, for a particular X_i we have $\text{Var}(X_i) = \sigma^2$, we can show that

$$\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n}\sum_i X_i\right) = \frac{\sigma^2}{n}$$

Dropping the independence assumption, with correlation ρ. Then

$$\text{Var}(\bar{X}) = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

Ways to ensemble:

1. Use different algorithms;
2. Use different training sets;
3. Bagging (Random Forests);
4. Boosting (Adaboost, xgboost).

1.8.4 Bagging: Bootstrap Aggregation

Suppose you have a true population P and a training set $S \sim P$. Assume that $P = S$ and take bootstrap samples $Z \sim S$.

Let $Z_1, \ldots, Z_M$ be bootstrap samples. Train model G_m on Z_m and aggregate

$$G(m) = \frac{1}{M} \sum_{m=1}^{M} G_m(x)$$

Bootstrapping drives the correlation ρ down. Increasing the number of bootstraps diminishes the variance, without overfitting.

However, it slightly increases the bias because of random subsampling.

Since decision trees have high variance and low bias, they are an ideal fit for bagging.

Random forests: consider only a fraction of the total features at each split.

1.8.5 Boosting

Here, we decrease the bias of the model. It also is more additive.

Gives weight to misclassifications. Addabost takes a weight α_m proportional to

$$\log\left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m}\right)$$

and then

$$G(x) = \sum_m \alpha_m G_m(x)$$

where each G_m is trained on a weighted training set.

Chapter 2

Deep Learning

Deep Learning is computationally expensive, so we need to parallelize the code.

2.1 Logistic Regression as Neural Networks

Classification goal: find cats in images. Returns 1 if there is a cat in the image and 0 otherwise. We suppose that there is at most one cat in the image.

If the image has size 64×64 and RGB color, then its size is $64 \times 64 \times 3$. The first step is to flatten this matrix into a vector. Then we can apply logistic regression $\hat{y} = \sigma(wx + b)$, where σ denotes the sigmoid function. The process is as follows:

1. Initialize w (weight) and b (bias);
2. Find the optimal w, b;
3. Use $\hat{y} = \sigma(wx + b)$ to predict.

To find the optimal weight and bias, we define a loss function

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

and use gradient descent to minimize the loss function.

In neural networks, we have

$$\text{neuron} = \text{linear} + \text{activation}$$

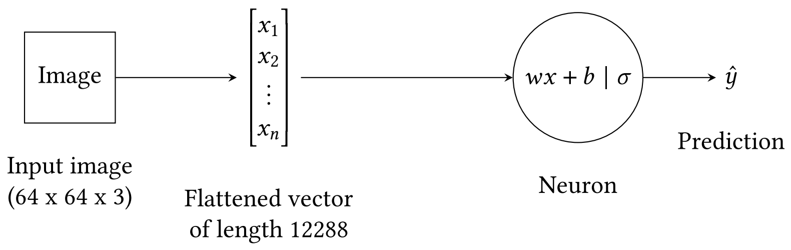

Figure 2.1: Logistic Regression for the cat example

In the logistic regression case, the linear part is $wx + b$ and the activation is the sigmoid function σ (see Figure 2.1).

Another important equation in neural networks is

$$\text{model} = \text{architecture} + \text{parameters}$$

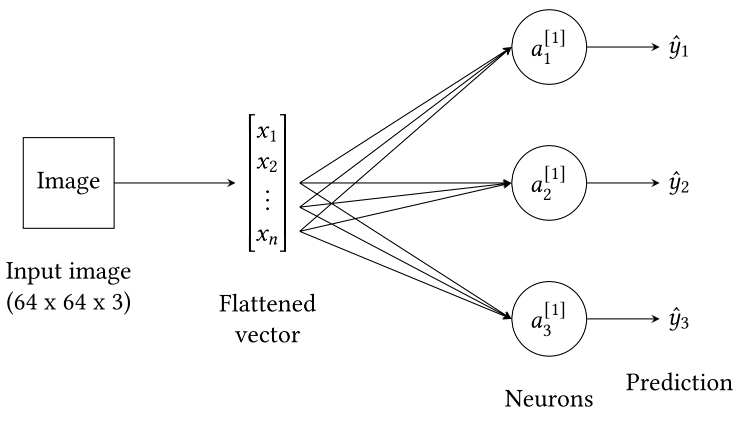

Figure 2.2: Logistic regression for the cat, lion and iguana example

Let us change our goal to find a cat, lion or iguana in the image. Here, instead of one neuron, we create three neurons $a_i^{[1]}$, with $i = 1, 2, 3$. The square brackets represent the layer (see Figure 2.2). In this case,

$$\hat{y}_i = a_i^{[1]} = \sigma(w_i^{[1]} x + b_i^{[1]})$$

Denote by $z_i^{[1]} = w_i^{[1]} x + b_i^{[1]}$ the linear part of the neuron.

How do we train the parameters? We need a new loss function

$$L = -\sum_{k=1}^{3} [y_k \log \hat{y}_k + (1 - y_k) \log(1 - \hat{y}_k)]$$

Now, let us add a constraint that there is a unique animal on an image. To do that, we use the softmax multiclass regression

$$z_i^{[1]} = \frac{e^{z_i^{[1]}}}{\sum_{k=1}^{3} e^{z_k^{[1]}}}$$

This works for the constraint because the sum of the outputs has to sum up to one.

The loss function will be the softmax cross-entropy loss

$$L = -\sum_{k=1}^{3} y_k \log \hat{y}_k$$

2.2 Neural Networks

Let us go back to our first goal: find if there is a cat in the image. We create three neurons in the first layer, as before, but add two neurons in the second layer and one neuron in the third and last layer.

The output layer has to have the same number of neurons as the number of classes of classification.

The number of parameters is $3n$ weights plus 3 biases in the input layer, $2 \times 3 + 2$ in the second layer and $2 \times 1 + 1$ in the output layer. The second layer is called the **hidden layer**. See Figure 2.3.

Consider now a house price prediction example. Assume that our inputs are the number of bedrooms, size of the house, zip code and wealth of the neighbourhood. We build a network with three neurons in the input layer and one in the output layer.

The zip code and wealth could predict the school's quality. The zip code could predict if the neighbourhood is walkable. The number of bedrooms and size could predict the family size. Using these pieces of information, we can predict the size.

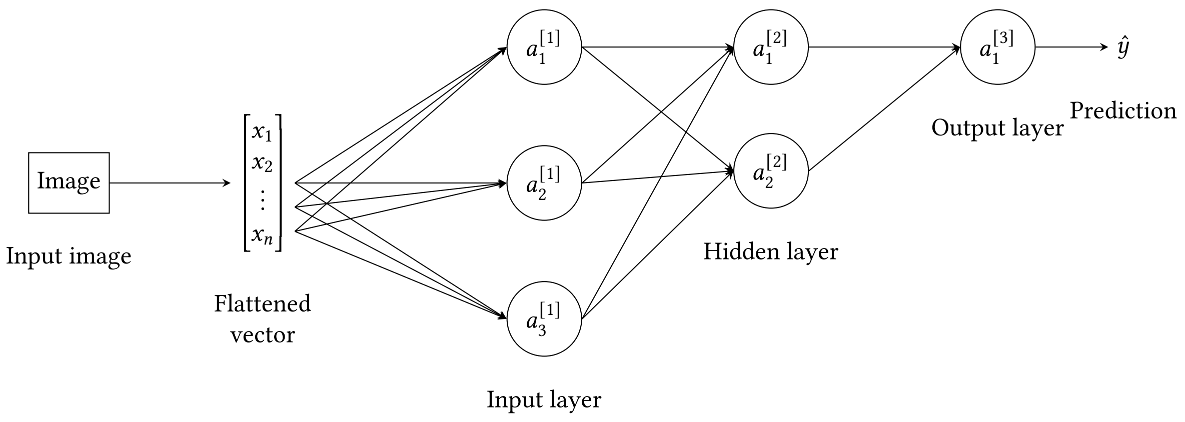

Figure 2.3: Neural Network

A **fully connected layer** means that all neurons, from one layer to another, are connected. Neural networks are also called **end-to-end learning** and **black box models**.

2.2.1 Propagation Equations

We have the following linear parts and activations:

$$\begin{aligned} z^{[1]} &= w^{[1]}x + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= w^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \\ z^{[3]} &= w^{[3]}a^{[2]} + b^{[3]} \\ \hat{y} = a^{[3]} &= \sigma(z^{[3]}) \end{aligned}$$

What happens for an input batch of m examples? Our X is not a single vector, but a matrix where $x^{(i)}$ is the i-th column of the matrix, representing the i-th example. Here, we have

$$Z^{[i]} = w^{[i]}x + b^{[i]}$$

Notice that $Z^{[i]}$ has dimension $(3, m)$, $w^{[i]}$ has dimension $(3, n)$, and b has dimension $(3, 1)$. To be able to sum properly, we use a technique called **broadcasting**: we define $\tilde{b}^{[i]}$ to be the vector $b^{[i]}$ repeated m times.

How can we optimize our parameters? The cost function will be

$$J(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} L^{(i)}$$

with

$$L^{(i)} = -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

To minimize, we compute, for all $l = 1, 2, 3$,

$$\begin{aligned} w^{[l]} &= w^{[l]} - \alpha \frac{\partial J}{\partial w^{[l]}} \\ b^{[l]} &= b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}} \end{aligned}$$

To make these computations easier, we use **backward propagation**:

$$\frac{\partial J}{\partial w^{[3]}} = \frac{\partial J}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}\frac{\partial z^{[3]}}{\partial w^{[3]}}$$

and

$$\frac{\partial J}{\partial w^{[2]}} = \frac{\partial J}{\partial z^{[3]}}\frac{\partial z^{[3]}}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial z^{[2]}}\frac{\partial z^{[2]}}{\partial w^{[2]}}$$

and

$$\frac{\partial J}{\partial w^{[1]}} = \frac{\partial J}{\partial z^{[2]}}\frac{\partial z^{[2]}}{\partial a^{[1]}}\frac{\partial a^{[1]}}{\partial z^{[1]}}\frac{\partial z^{[1]}}{\partial w^{[1]}}$$

Since

$$\frac{\partial L}{\partial w^{[3]}} = -(y^{(i)} - a^{[3]})a^{[2]^\top}$$

we have

$$\frac{\partial J}{\partial w^{[3]}} = -\frac{1}{m}\sum_{i=1}^{m} -(y^{(i)} - a^{[3]})a^{[2]^\top}$$

2.2.2 Improving Neural Networks

The first trick to improve neural networks is to use different activation functions. One alternative to the sigmoid function is the ReLU function

$$\mathrm{ReLU}(z) = \begin{cases} 0, & \text{if } z \leq 0 \\ z, & \text{if } z > 0 \end{cases}$$

and the hyperbolic tangent

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Sigmoid is good for classification, and works well in the linear regime, but does poorly in saturated regimes. This is because the network doesn't update the parameters properly (the gradient is nearly 0 for extreme values of z). The hyperbolic tangent has the problem. However, the ReLU function doesn't have this problem.

But why do we need activation functions? If we use the identity function, we get a linear regression.

The common practice is to use one activation in the layer and use one of these three activations.

The second trick is to use initialization methods, such as normalizing the input. This makes the gradient descent easier.

One problem in neural networks is vanishing or exploding gradients. One way to deal with this is to take small values of w_i if n is large; each w_i should be approximately $1/n$.

For a ReLU, using a 2 instead of the 1 above is better.

For tanh, **Xavier initialization** uses

$$w^{[l]} \approx \sqrt{\frac{1}{n^{l-1}}}$$

and **He initialization** uses

$$w^{[l]} \approx \sqrt{\frac{2}{n^l + n^{l-1}}}$$

To optimize our cost function, we can use **mini-batch stochastic gradient descent**, where we have a trade-off between stochasticity and vectorization. To do that, we divide the training examples (X, Y) into batches. The algorithm is as follows:

Algorithm 1: Mini-batch stochastic gradient descent

```
Data: learning rate $\alpha$, batch size $B$, number of iterations $n$.
Initialize $\theta$ randomly.
for $i = 1$ to $n$ do
    Sample $B$ batches $j_1, \dots, j_B$ without replacement and uniformly.
    Update $\theta$ by
        $$\theta := \theta - \frac{\alpha}{B} \sum_{k=1}^{B} \nabla_\theta J^{(j_k)}(\theta)$$
```

There is also the **momentum algorithm**:

$$\begin{cases} v = \beta v + (1 - \beta) \frac{\partial L}{\partial w} \\ w = w - \alpha v \end{cases}$$

2.3 Debugging ML Models and Error Analysis

Suppose we are building an anti-spam classifier and carefully chose a small set of a hundred words to use as features. We implement logistic regression with regularization (bayesian logistic regression) with gradient ascent, but the model gets 20% test error, which is unacceptably high.

$$\max_\theta \left(\sum_{i=1}^{m} \log p(y^{(i)} \mid x^{(i)}, \theta) - \lambda \|\theta\|^2 \right)$$

We could try different approaches: more training examples, smaller or larger sets of features, changing the features, running gradient ascent with more iterations, trying Newton's method, using a different value for λ, or using an SVM.

A better approach is to run diagnostics to determine the problem.

2.3.1 Diagnostics for debugging learning algorithms

One diagnostic that can help us is bias versus variance. In a high-variance model, the training error will be much lower than the test error. In a high bias, both errors will be high.

Going back to our previous approaches, do they fix high variance or high bias?

1. More training examples: high variance;
2. Smaller set of features: high variance;
3. Larger set of features: high bias;
4. Changing the features (e.g. use e-mail header): high bias.

Now consider another example: a logistic regression gets 2% error on span an on non-spam e-mail, which is unacceptably high on non-spam. An SVM works well for this problem, but we want to use logistic regression. Common questions:

1. Is the algorithm (gradient ascent) converging?
2. Are we optimizing the right function?

3. Are we using the correct value for λ?

Our goal is to optimize the weighted accuracy:

$$a(\theta) = \max_{\theta} \sum_i w^{(i)} \mathbf{1}_{\{h_\theta)x^{(i)}=y^{(i)}\}}$$

where the weights $w^{(i)}$ are different for spam and non-spam e-mail. Perhaps, optimizing the cost function $J(\theta)$ is not equivalent to optimising $a(\theta)$.

If $a(\theta_{\text{SVM}}) > a(\theta_{\text{BLR}})$ and $J(\theta_{\text{SVM}}) > J(\theta_{\text{BLR}})$, then θ_{BLR} fails to maximize J and the problem is with the convergence of the optimization algorithm.

If $a(\theta_{\text{SVM}}) > a(\theta_{\text{BLR}})$ but $J(\theta_{\text{SVM}}) \leq J(\theta_{\text{BLR}})$, then $J(\theta)$ is the wrong function to be maximizing, i.e., the problem is with the objective function of the maximization problem.

1. More training examples: high variance;
2. Smaller set of features: high variance;
3. Larger set of features: high bias;
4. Changing the features (e.g. use e-mail header): high bias;
5. Run gradient ascent with more iterations: fixes optimization algorithm;
6. Try Newton's method: fixes optimization algorithm;
7. Use a different value for λ: fixes optimization algorithm;
8. Using an SVM.

Now suppose we have a machine-learning algorithm to control a helicopter. We build a simulator of the helicopter, choose a cost function, say $J(\theta) = \|x - x_{\text{desired}}\|^2$, and run a reinforcement learning (RL) algorithm to fly the helicopter in the simulator, as to try to minimize the cost function

$$\theta_{RL} = \arg\min_{\theta} J(\theta)$$

However, the controller parameters θ_{RL} give a bad performance. What to do next? Improve simulator, modify cost function of modifying RL algorithm?

Suppose that the simulator is accurate, the RL algorithm correctly controls the helicopter to minimize $J(\theta)$ and that minimizing $J(\theta)$ corresponds to correct autonomous flight. Then the learned parameters θ_{RL} should fly well on the actual helicopter.

We can use the following diagnostics:

1. If θ_{RL} flies well in simulation, but not in real life, then the problem is in the simulator.
2. Let θ_{human} be the human control policy. If $J(\theta_{\text{human}}) < J(\theta_{RL})$, then the problem is in the reinforcement learning algorithm, i.e., it is failing to minimize the cost function.
3. If $J(\theta_{\text{human}}) \geq J(\theta_{RL})$, then the problem is in the cost function, i.e., optimizing it doesn't correspond to good autonomous flight.

2.3.2 Error analyses and ablative analysis

Many applications combine different learning algorithms into a pipeline. For example, a face recognition system. Given a camera image, it may first preprocess it (remove background), then detect the face, segment the face (eyes, nose, mouth), feed this information into a logistic regression, and then label.

How much error is in each step? We can feed a perfect response in each step to detect which step would be more improved.

Ablative analysis: remove components one at a time to see how the model performance is affected.

Chapter 3

Unsupervised Learning

In unsupervised learning, we start with unlabeled data. That means, without (x, y) examples, we are just given a set of examples $\{x^1, \ldots, x^{(m)}\}$ and our goal is to find interesting relations between them.

3.1 K-means Clustering

Suppose we want to find two clusters that separate our unlabeled data. The first step of k-means clustering is to pick two points (cluster centroids) that are our best guesses for the centres of the two clusters. Let us colour one of the clusters blue and the other red.

We repeatedly do the following:

1. Go through each of the training examples and color them either blue or red depending on which is the closer cluster centroid;

2. Compute the mean of all blue dots and move the blue centroid there. Do the same for the red dots.

So we have the following algorithm.

Algorithm 2: K-means clustering

```
Data: $\{x^1, \ldots, x^{(m)}\}$.
Initialize cluster centroid $\mu_1, \ldots, \mu_k \in \mathbf{R}^n$ randomly.
repeat
    Set $c^{(i)} := \arg\min_j \|x^{(i)} - \mu_j\|^2$;
    for $j = 1$ to $k$ do
        $\mu_j := \frac{\sum_{i=1}^m \mathbf{1}_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m \mathbf{1}_{\{c^{(i)}=j\}}}$
until convergence;
```

The best way to initialize the cluster centroid is to randomly pick k training examples.

The cost function is

$$J(c, \mu) = \sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

It can be proven that k-means makes this cost function converge.

How can we choose the number k? There are some criteria, but it can be picked by hand depending on the application.

3.2 Expectation-Maximization Algorithms

Suppose we want to create an anomaly detection algorithm. The idea is to model $\mathbf{P}(x)$ and, if $\mathbf{P}(x)$ is small, we flag an anomaly. This is a density estimation problem.

One possible model is the mixture of Gaussians. Suppose there is a latent (hidden/unobserved) random variable z and that $x^{(i)}, z^{(i)}$ have the joint distribution

$$\mathbf{P}(x^{(i)}, z^{(i)}) = \mathbf{P}(x^{(i)} \mid z^{(i)})\mathbf{P}(z^{(i)})$$

where $z^{(i)} \sim \text{Multinomial}(\varphi)$

$$x^{(i)} \mid z^{(i)} = j \sim N(\mu_j, \Sigma_j)$$

Notice the similarities here with Gaussian Discriminant Analysis. However, the variables $y^{(i)}$ were observed in GDA.

If we knew the $z^{(i)}$m we could use MLE on

$$l(\varphi, \mu, \Sigma) = \sum_{i=1}^{m} \log \mathbf{P}(x^{(i)}, z^{(i)}; \varphi, \mu, \Sigma)$$

and obtain the values for φ_j, μ_j, and Σ_j, just as in GDA.

What we use is the **Expectation-Maximization (EM) Algorithms** to guess the values of the $z^{(i)}$ and then use equations similar to those in the GDA.

Then *E-step* is to guess the values of the $z^{(i)}$. To do that, we set

$$w_j^{(i)} = \mathbf{P}(z^{(i)} = j \mid x^{(i)}; \varphi, \mu, \Sigma) = \frac{\mathbf{P}(x^{(i)} \mid z^{(i)} = j)\mathbf{P}(z^{(i)} = j)}{\sum_{l=1}^{k} \mathbf{P}(x^{(i)} \mid z^{(i)} = l)\mathbf{P}(z^{(i)} = l}$$

Since $\mathbf{P}(x^{(i)} \mid z^{(i)} = j) \sim N(\mu_j, \Sigma_j)$,

$$\mathbf{P}(x^{(i)} \mid z^{(i)} = j) = \frac{1}{(2\pi)^{n/2}|\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1}(x^{(i)} - \mu_j)\right)$$

And since $z^{(i)} \sim \text{Multinomial}(\varphi)$, we have $\mathbf{P}(z^{(i)} = j) = \varphi_j$. The terms in the denominator are similar.

The *M-step* is to update the parameters

$$\begin{aligned} \varphi_j &= \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)} \\ \mu_j &= \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}} \\ \Sigma_j &= \frac{\sum_{i=1}^{m} w_j^{(i)}(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}} \end{aligned} \tag{3.1}$$

3.2.1 EM Algorithm Derivation

To derive the EM algorithm more rigorously, assume we have a model for $\mathbf{P}(x, z; \theta)$, but we only observe x. Then

$$l(\theta) = \sum_{i=1}^{m} \log \mathbf{P}(x^{(i)}; \theta) = \sum_{i=1}^{m} \log \sum_{z^{(i)}} \mathbf{P}(x^{(i)} z^{(i)}; \theta)$$

and our goal is to find $\arg\max l(\theta)$.

We write

$$\sum_{i=1}^{m} \log \sum_{z^{(i)}} \mathbf{P}(x^{(i)}, z^{(i)}; \theta) = \sum_{i=1}^{m} \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

where $Q_i(z^{(i)})$ is a probability distribution.

By Jensen's inequality,

$$\begin{aligned}\sum_{i=1}^{m} \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} &= \sum_{i=1}^{m} \log \mathbf{E}_{z^{(i)} \sim Q_i} \left[\frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \\ &\geq \sum_{i=1}^{m} \mathbf{E}_{z^{(i)} \sim Q_i} \left[\log \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \\ &= \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}\end{aligned}$$

On a given iteration of EM (with parameters θ), we want

$$\log \mathbf{E}_{z^{(i)} \sim Q_i} \left[\frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] = \mathbf{E}_{z^{(i)} \sim Q_i} \left[\log \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

For this to hold, we need

$$\frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = \mathbf{E}_{z^{(i)} \sim Q_i} \left[\frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \quad \text{a.s.}$$

We can set $Q_i(z^{(i)}) \propto \mathbf{P}(x^{(i)}, z^{(i)}; \theta)$ by taking

$$Q_i(z^{(i)}) = \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{\sum_{z^{(i)}} \mathbf{P}(x^{(i)}, z^{(i)}; \theta)} = \mathbf{P}(z^{(i)} \mid x^{(i)}; \theta)$$

To summarize, on the *E-step*, we set $Q_i(z^{(i)}) = \mathbf{P}(z^{(i)} \mid x^{(i)}; \theta)$, and on the *M-step*, we take

$$\theta := \arg\max_{\theta} \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

i.e., we compute

$$\begin{aligned}&\max_{\varphi,\mu,\Sigma} \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \\ &\quad = \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \log \left(\frac{\frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2} (x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j) \right)}{w_j^{(i)}} \varphi_j \right)\end{aligned}$$

and obtain the parameters (3.1).

Let

$$J(\theta, Q) = \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{\mathbf{P}(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

We know that $l(\theta) \geq J(\theta, Q)$ for any θ and Q. At the *E-step*, we maximise J with respect to Q; at the *M-step*, we maximise J with respect to θ. This is called **coordinate ascent**.

3.3 Factor Analysis

A mixture of Gaussians works well for a training set with two features $n = 2$ and $m = 100$ examples, i.e., $m \gg n$. However, if $m \approx n$ or $m \ll n$, we need a different model.

We can model as a single Gaussian $x \sim N(\mu, \Sigma)$ and compute the MLE. If $m \ll n$, then Σ will be singular.

For example, suppose we have $n = 100$ psychological attributes and $m = 30$ persons, and we want to model how correlated are different psychological attributes. Then how do we build a model for $\mathbf{P}(x)$?

One option is to constrain Σ to be diagonal. Then the MLE for each entry in Σ will be

$$\sigma_j^2 = \frac{1}{m} \sum_i (x_j^{(i)} - \mu_j)^2$$

The problem is that this model assumes that all features are uncorrelated.

Another option is to constrain $\Sigma = \sigma^2 I$, so the Gaussians have circular contours. Here we have the same problem as before.

3.3.1 The Factor Analysis Model

Now consider the following model: $\mathbf{P}(x, z) = \mathbf{P}(x \mid z)\mathbf{P}(z)$ where z is hidden, $z \sim N(0, I)$ and $z \in \mathbf{R}^d$ with $d < n$. We assume that $x = \mu + \Lambda z + \varepsilon$ with $\varepsilon \sim N(0, \Psi)$, and Ψ is diagonal.

The parameters are $\mu \in \mathbf{R}^n$, $\Lambda \in \mathbf{R}^{n \times d}$ and $\Psi \in \mathbf{R}^{n \times n}$.

Notice that we can write $x = \mu + \Lambda z + \varepsilon$ as

$$x \mid z \sim N(\mu + \Lambda z, \Psi)$$

The conditional that Ψ is diagonal means that the noise in each of the d 'sensors' is independent of each other. This is the **factor analysis** model.

What kind of data can factor analysis model? We can take, for example, a 100 dimensional data and model the data as lying on a three-dimensional space with a little bit of noise.

3.3.2 EM Steps

To derive the EM model, we start by partition $x \in \mathbf{R}^{r+s}$ into $x_1 \in \mathbf{R}^r$ and $x_2 \in \mathbf{R}^s$ and partition μ and Σ similarly (Σ will break into four matrices, $\Sigma_{11} \in \mathbf{R}^{r \times r}$, $\Sigma_{12} \in \mathbf{R}^{r \times s}$ and so on).

Computing the marginals, we obtain $x_1 \sim N(\mu_1, \Sigma_{11})$. And computing the conditionals, $x_1 \mid x_2 \sim N(\mu_{1|2}, \Sigma_{1|2})$ where

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$
$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

Let us start by deriving $\mathbf{P}(x, z)$. We have that

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim N(\mu_{x,z}, \Sigma)$$

Now, $z \sim N(0, I)$ and $x = \mu + \Lambda z + \varepsilon$. Since $\mathbf{E}[z] = 0$ and $\mathbf{E}[x] = \mathbf{E}[\mu + \Lambda z + \varepsilon] = \mu$, it follows that

$$\mu_{xz} = \begin{bmatrix} 0 \\ \mu \end{bmatrix}$$

where $0 \in \mathbf{R}^{d \times 1}$ and $\mu \in \mathbf{R}^{r \times 1}$.

The covariance matrix will be

$$\Sigma = \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}$$

In fact,

$$\begin{aligned}
\Sigma_{22} &= \mathbf{E}[(x - \mathbf{E}[x])(x - \mathbf{E}[x])^T] \\
&= \mathbf{E}[(\Lambda z + \mu + \varepsilon - \mu)(\Lambda z + \mu + \varepsilon - \mu)^T] \\
&= \mathbf{E}[\Lambda z z^T \Lambda^T + \Lambda z \varepsilon^T | \varepsilon z^T \Lambda^T + \varepsilon\varepsilon^T] \\
&= \mathbf{E}[\Lambda z z^T \Lambda^T] + \mathbf{E}[\varepsilon\varepsilon^T] \\
&= \Lambda \mathbf{E}[z z^T] \Lambda^T + \Psi
\end{aligned}$$

and the other computations are similar.

For the *E-step*, we need to compute

$$Q_i(z^{(i)}) = \mathbf{P}(z^{(i)} \mid x^{(i)}; \theta)$$

Since $z^{(i)} \mid x^{(i)} \sim N(\mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}})$, with

$$\begin{aligned}
\mu_{z^{(i)}|x^{(i)}} &= 0 + \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}(x^{(i)} - \mu) \\
\Sigma_{z^{(i)}|x^{(i)}} &= I - \Lambda^T(\Lambda\Lambda^T + \Psi)^{-1}\Lambda
\end{aligned}$$

Notice that these equations come from (3.3.2).

For the *M-step*, we update

$$\begin{aligned}
\theta &:= \arg\max_\theta \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log \frac{\mathbf{P}(x^{(i)}, z^{(i)})}{Q_i(z^{(i)})} \, \mathrm{d}z^{(i)} \\
&= \arg\max_\theta \sum_i \mathbf{E}_{z^{(i)} \sim Q_i} \left[\log \frac{\mathbf{P}(x^{(i)}, z^{(i)})}{Q_i(z^{(i)})} \right]
\end{aligned}$$

3.4 Principal Component Analysis

Given unlabeled data $\{x^{(1)}, \dots, x^{(m)}\}$ with each $x^{(i)} \in \mathbf{R}^n$. We reduce the dimension from n to k, where $k \ll n$.

Pre-processing: normalize the data (standard score).

Notice that if $\|u\| = 1$, then the length of the projection of $x^{(i)}$ onto u is $u^T x^{(i)}$. In PCA, we choose u to maximize

$$\max_{u:\|u\|=1} \frac{1}{m} \sum_{i=1}^m (x^{(i)T} u)^2 = \max_{u:\|u\|=1} \frac{1}{m} \sum_{i=1}^m u^T x^{(i)} x^{(i)T} u = \max_{u:\|u\|=1} u^T \left(\frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \right) u$$

Let us denote $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T}$. Then, by Lagrangian multipliers, u is the principal eigenvector of Σ.

In general, to project data to k-dimensions, we set $u_1, \dots, u_k$ to be the top k eigenvectors of Σ (with corresponding eigenvalues $\lambda_1, \dots, \lambda_k$). The new represention for $x^{(i)}$ will be

$$y^{(i)} = (u_1^T x^{(i)}, \dots, u_k^T x^{(i)})$$

and $x^{(i)} \approx y_1^{(i)} u_1 + \cdots + y_k^{(i)} u_k$.

One of the applications of PCA is visualization (project from n dimensions to two or three dimensions). Another application is compression for machine learning efficiency. It can reduce overfitting but doesn't always work (use regularization instead). It also can be used for outlier detection and matching.

3.5 Independent Components Analysis

Suppose that there is a data $s \in \mathbf{R}^d$ generated by d independent sources. We observe $x = As$, where A is an unknown matrix called the **mixing matrix**. Repeated observation gives us a dataset. Our goal is to recover the sources $s^{(i)}$ that generated the data. To do that, we want to find $W = A^{-1}$, the **unmixing matrix**, so that $s = Wx$.

Since the Gaussian distribution is rotationally symmetric, ICA is not possible on Gaussian data. There are also ambiguities to the model.

There is a relationship between the distributions of x and s:

$$\mathbf{P}_x(x) = \mathbf{P}_s(Wx)\det(W)$$

Since the speakers are independent,

$$\mathbf{P}_s(s) = \prod_{j=1}^{n} \mathbf{P}_s(s_j)$$

where s_j denotes the j-th speaker.

Notice that

$$\mathbf{P}_x(x) = \mathbf{P}_x(Wx)\det(W) = \prod_{j=1}^{n} \mathbf{P}_s(W_j^T x)\det(W)$$

We use MLE to estimate W

$$l(W) = \sum_{i=1}^{m} \log\left[\left(\prod_{j=1}^{n} \mathbf{P}_s(W_j^T x)\right)\det(W)\right]$$

with stochastic gradient ascent.

Chapter 4

Reinforcement Learning

4.1 Markov Decision Process

Definition 4.1.1 (Markov Decision Process)**.** A **Markov Decision Process** is a tuple $(S, A, P_{sa}, \gamma, R)$, where

1. S is the set of states;
2. A is the set of actions;
3. P_{sa} are the state transition probabilities;
4. $\gamma \in [0, 1)$ is the **discount factor**;
5. $R : S \times A \longrightarrow \mathbf{R}$ is the **reward function**.

For example, we may consider a reward function $R(s, a)$ equal to 1 if the program does what we want, -1 if it doesn't and 0 or a small penalty for all other states. The discount factor γ is usually chosen to be slightly smaller than one. The **total payoff** is

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

The goal is to choose actions over time to maximize the expected total payoff.

Definition 4.1.2 (Policy)**.** A **policy** (or **controller**) is a function that maps from states to actions $\pi : S \longrightarrow A$. To **execute the policy** is to, at the state s, take the action $\pi(s)$.

Definition 4.1.3 (Value Function)**.** The **value function** for a policy π is $V^\pi : S \longrightarrow \mathbf{R}$ such that $V^\pi(s)$ is the expected total payoff for starting at state s and executing π, i.e.,

$$V^\pi(s) = \mathbf{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid \pi, s_0 = s]$$

The value function satisfies the Bellman equation:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s')$$

This yields a linear system of equations in terms of $V^\pi(s)$.

Definition 4.1.4 (Optimal Value Function)**.** The **optimal value function** is

$$V^*(s) = \max_\pi V^\pi(s)$$

For the optimal value function, we have the Bellman equation:

$$V^*(s) = R(s) + \max_{a\in A} \gamma \sum_{s'\in S} P_{sa}(s')V^*(s')$$

Definition 4.1.5 (Optimal Policy)**.** The **optimal policy** is

$$\pi^*(s) = \arg\max_{a\in A} \sum_{s'\in S} P_{sa}(s')V^*(s')$$

Notice that

$$V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$$

The strategy to find the optimal policy is first to find V^* and then use the arg max equation to find π^*. So, how do we find V^*?

4.1.1 Value iteration

Algorithm 3: Value Iteration

```
Initialize $V(s) := 0$ for every $s$.
repeat
    for all s do
        $V(s) := R(s) + \max_{a\in A} \gamma \sum_{s'\in S} P_{sa}(s')V(s')$
until convergence;
```

This algorithm makes V converge to V^*.

4.1.2 Policy iteration

Algorithm 4: Policy Iteration

```
Initialize $\pi$ randomly.
Set $V := V^\pi$ (i.e., solve Bellman equation).
repeat
    for all s do
        $\pi(s) := \arg\max_{a\in A} \sum_{s'\in S} P_{sa}(s')V(s')$
until convergence;
```

4.1.3 Learning state probabilities

What to do when $P_{sa}(s')$ is unknown? We estimate

$$P_{sa}(s') = \frac{\text{\# times took action } a \text{ in state } s \text{ and got to } s'}{\text{\# times took action } a \text{ in state } s}$$

or $1/|S|$ if the denominator above is 0.

Algorithm 5: Learning a model for a MDP

```
Initialize $\pi$ randomly.
repeat
    Execute $\pi$ in the MDP for some trials;
    Update estimates of $P_{sa}$ (and possibly $R$);
    Solve Bellman equation using value iteration to get $V$;
    Update $\pi$ to be the greedy policy with respect to $V$.
until convergence;
```

4.1.4 Putting it together

The exploration versus exploitation problem.

In the ε-greedy algorithm, we have $(1-\varepsilon)$ chance of executing π, and ε chance of choosing randomly.

4.2 Continuous State MDPs

4.2.1 Discretization

Suppose our state space is $\mathbf{R}^n$. For $n=2$, we can discretize the space using a grid.

Discretization works well for small dimensions, but it has some problems:

1. Naive representation for V^* and π^*;
2. Curse of dimensionality, i.e., if we discretize each dimension into k values, we get k^n discrete states;

What we can do is to approximate V^* directly, without using discretization, by taking

$$V^*(s) \approx \theta^T \varphi(s)$$

where $\varphi(s)$ are features of the state s.

4.2.2 Models and Simulation

A model is a function that takes the state s and the action a and returns $s' \sim P_{sa}$. One way of building a model is using a physics simulator.

Another way is to learn the model from data. For example, we may collect data from a human flying a helicopter m times and then apply supervised learning. Using linear regression, we have

$$s_{t+1} = As_t + Ba_t$$

by optimizing

$$\min_{A,B} \sum_{i=1}^{m} \sum_{t=0}^{T} \|s_{t+1}^{(i)} - (As_t^{(i)} + Ba_t^{(i)})\|^2$$

A stochastic alternative to $s_{t+1} = As_t + Ba_t$ is

$$s_{t+1} = As_t + Ba_t + \varepsilon_t$$

with $\varepsilon_t \sim N(0, \sigma^2 I)$.

Deep Reinforcement Learning uses neural networks instead of linear regression.

4.2.3 Fitted value iteration

The first step is to choose features $\varphi(s)$ of the state s. We approximate $V^*(s)$ via

$$V(s) = \theta^T \varphi(s)$$

We want to update

$$\begin{aligned} V(s) &= R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s') \\ &= R(s) + \gamma \max_{a \in A} \mathbf{E}_{s' \sim P_{sa}} [V(s')] \end{aligned}$$

Algorithm 6: Fitted Value Iteration

```
Sample $\{s^{(1)}, \dots, s^{(m)}\} \in S$ randomly.
Initialize $\theta := 0$.
**repeat**
    **for** $i = 1, \dots, m$ **do**
        **for** *each action* $a \in A$ **do**
            Sample $s'_1, \dots, s'_k \sim P_{s'a}$ (i.e., using model);
            Set
                $$q(a) = \frac{1}{k} \sum_{j=1}^{k} [R(s^{(i)}) + \gamma V(s'_j)]$$
        Set $y^{(i)} = \max_{a \in A} q(a)$.
    Set
        $$\theta := \arg\min_\theta \frac{1}{2} \sum_{i=1}^{n} (\theta^T \varphi(s^{(i)}) - y^{(i)})^2$$
**until** *convergence*;
```

Notice that $q(a)$ is an estimate of $R(s^{(i)}) + \gamma \mathbf{E}_{s' \sim P_{s^{(i)}a}} [V(s')]$ and that $y^{(i)}$ is an estimate of $R(s^{(i)}) + \gamma \max_{a \in A} \mathbf{E}_{s' \sim P_{s^{(i)}a}} [V(s')]$.

This algorithm gives an approximation to V^*, which implicitly defines π^*, since

$$\pi^*(s) = \arg\max_{a \in A} \mathbf{E}_{s' \sim P_{sa}} [V^*(s')]$$

To compute this expectation, we use a sample $s'_1, \dots, s'_k \sim P_{sa}$.

If the simulator is of the form $s_{t+1} = f(s_t, a_t) + \varepsilon_t$, we can set $\varepsilon_t = 0$ and $k = 1$. That is, pick action

$$\arg\max_{a \in A} V(f(s, a))$$

4.3 State-Action Rewards

An immediate generalization of MDPs is to consider the action as one of the inputs of the reward function, i.e., $R(s, a)$. The Bellman equation becomes

$$V^*(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

and we can use value iteration as before.

The optimal action is

$$\pi^*(s) = \arg\max_a \left(R(s, a) + \gamma \sum_{s'} P_{sa}(s') V^*(s') \right)$$

4.4 Finite-Horizon MDP

We add a horizon time T so the payoff becomes a finite sum

$$R(s_0, a_0) + R(s_1, a_1) + \cdots + R(s_T, a_T)$$

In this case, the discount factor is not necessary. Here, the optimal action may depend on time t, i.e., we have a non-stationary policy $\pi_t^*(s)$.

We can also consider non-stationary state transitions $s_{t+1} \sim P_{s_t a_t}^{(t)}$ or rewards $R(s,a)^{(t)}$.

The expected total payoff starting in state s at time t executing π^* is

$$V_t^*(s) = \mathbf{E}[R(s_t, a_t) + \cdots + R(s_T, a_T) \mid \pi^*, s_0 = s]$$

and the value iteration becomes a dynamic programming algorithm.

We start by computing

$$V_T^*(s) = \max_a R(s,a)$$

which allows us to find

$$V_t^*(s) = \max_a R(s,a) + \max_a \gamma \sum_{s'} P_{sa}(s') V_{t+1}^*(s')$$

Notice that the optimal action is

$$\pi_t^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s') V_{t+1}^*(s') \right)$$

4.5 Linear Quadratic Regulation (LQR)

Suppose we are in a finite-horizon MDP (S, A, P_{sa}, T, R) (it also works for discounted MDPs) where $S = \mathbf{R}^n$ and $A = \mathbf{R}^d$.

We assume that P_{sa} is of the form $s_{t+1} = As_t + Ba_t + w_t$ with $w_t \sim N(0, \Sigma_w)$, $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{n\times d}$.

The reward function is

$$R(s,a) = -(s^T U s + a^T V a)$$

where $U \in \mathbf{R}^{n\times n}$, $V \in \mathbf{R}^{d\times d}$ are positive semi-definite. This means that $s^T U s \geq 0$ and $a^T V a \geq 0$.

If we want, for example, $s \approx 0$, then choosing $U = I$ and $V = I$, we can pick $R(s,a) = -(\|s\|^2 + \|a\|^2)$.

The two key assumptions here are that the state transition probability is a linear function of the previous state, action and some noise, and second that the reward function is a quadratic cost function.

How do we find the matrices A and B? We can, as before, learn from data: collect data and then fit the model

$$\min_{A,B} \sum_{i=1}^{m} \sum_{t=0}^{T-1} \|s_{t+1} - (As_t + Ba_t)\|^2$$

Another approach is to linearize a non-linear model. Fixed a point $\bar{s}_t$, by Taylor expansion,

$$s_{t+1} \approx f(\bar{s}_t) + f'(\bar{s}_t)(s_t - \bar{s}_t)$$

To linearize f around $(\bar{s}_t, \bar{a}_t)$, by Taylor again,

$$s_{t+1} \approx f(\bar{s}_t, \bar{a}_t) + (D_s f(\bar{s}_t, \bar{a}_t))^T (s_t - \bar{s}_t) + (D_a f(\bar{s}_t, \bar{a}_t))^T (a_t - \bar{a}_t)$$

To compute the optimal value function V^*, we use dynamic programming. For the last step, since V is positive semi-definite,

$$V_T^*(s_T) = \max_{a_T} R(s_T, a_T) = \max_{a_T}(-s_T^T U s_T - a_T^T V a_T) = -s_T^T U s_T$$

Thus,

$$\pi_T^*(s_T) = 0$$

Suppose

$$V_{t+1}^*(s_{t+1}) = s_{t+1}^T \Phi_{t+1} s_{t+1} + \Psi_{t+1}$$

where $\Phi_{t+1} \in \mathbf{R}^{n \times n}$ and $\Psi_{t+1} \in \mathbf{R}$.

We update

$$V_t^*(s_t) = \max_{a_t} \left(R(s_t, a_t) + \mathbf{E}_{s_{t+1} \sim P_{s_t a_t}} [V_{t+1}^*(s_{t+1})] \right)$$

Notice that

$$V_t^*(s_t) = \max_{a_t} \left(-s_t^T U s_t - a_t^T V a_t + \mathbf{E}_{s_{t+1} \sim N(As_t + Ba_t, \Sigma_w)} [s_{t+1}^T \Phi_{t+1} s_{t+1} + \Psi_{t+1}] \right)$$

then take derivatives with respect to a_t, set to zero and solve for a_t. This yields

$$a_t = \underbrace{(B^T \Phi_{t+1} B - V)^{-1} B^T \Phi_{t+1} A}_{L_t} s_t$$

and

$$\pi_t^*(s_t) = L_t s_t$$

The optimal action is a linear function of the state s_t.

By simplifying,

$$V_t^*(s_t) = s_t^T \Phi_t s_t + \Psi_t$$

where

$$\Phi_t = A(\Phi_{t+1} - \Phi_{t+1} B_t (B_t^T \Phi_{t+1} B_t - V)^{-1} B_t \Phi_{t+1}) A - U$$

and

$$\Phi_t = -\mathrm{tr}(\Sigma_w \Phi_{t+1}) + \Psi_{t+1}$$

Using these equations, we can go backwards from V_{t+1}^* to V_t^*.

To summarize,

Algorithm 7: Linear Quadratic Regulation

```
Initialize $\Phi_T = -U$ and $\Psi_T = 0$.
for $t = T-1, T-2, \dots, 0$ do
    Recursively compute $\Phi_t, \Psi_t$ using $\Phi_{t+1}, \Psi_{t+1}$.
Calculate $L_t$.
Set $\pi^*(s_t) = L_t s_t$.
```

4.6 Reinforcement Learning Debugging and Diagnostics

See the section Debugging ML Models and Error Analysis.

4.7 Policy search and POMDPs

Instead of finding the optimal value function V^* and obtaining the optimal policy π^*, we can directly search for the policy.

Definition 4.7.1. A **stochastic policy** is a function $\pi : S \times A \longrightarrow \mathbf{R}$ such that $\pi(s, a)$ is the probability of taking the action a at the state s.

For example, the probability that a vehicle takes the action 'accelerate to the right' could be

$$\pi_\theta(s, \text{'right'}) = \frac{1}{1 + e^{-\theta^T s}}$$

and, to the left,

$$\pi_\theta(s, \text{'left'}) = 1 - \frac{1}{1 + e^{-\theta^T s}}$$

Our goal is to find the parameter θ so that when we execute $\pi_\theta(s, a)$ we maximize

$$\max_\theta \mathbf{E}[R(s_0, a_0) + \cdots + R(s_T, a_T) \mid \pi_\theta]$$

Here s_0 is a fixed initial state.
For simplicity, let us take $T = 1$. Then

$$\begin{aligned}
\mathbf{E}[R(s_0, a_0) + R(s_1, a_1) \mid \pi_\theta] &= \sum_{s_0, a_0; s_1, a_1} P(s_0, a_0; s_1, a_1) \underbrace{(R(s_0, a_0) + R(s_1, a_1))}_{\text{payoff}} \\
&= \sum_{s_0, a_0; s_1, a_1} P(s_0) \pi_\theta(s_0, a_0) P_{s_0 a_0}(s_1) \pi_\theta(s_1, a_1)(R(s_0, a_0) + R(s_1, a_1))
\end{aligned}$$

To obtain θ we use the following stochastic gradient ascent algorithm.

```
Algorithm 8: Reinforce
repeat
    Sample $s_0, a_0, s_1, a_1$.
    Compute the payoff $R(s_0, a_0) + R(s_1, a_1)$.
    Update
        $\theta := \theta + \alpha \left[ \frac{D_\theta \pi_\theta(s_0, a_0)}{\pi_\theta(s_0, a_0)} + \frac{D_\theta \pi_\theta(s_1, a_1)}{\pi_\theta(s_1, a_1)} \right] (R(s_0, a_0) + R(s_1, a_1))$
until convergence;
```

Theorem 4.7.1. The reinforce is a gradient ascent algorithm.

Proof. We want to maximize the expected payoff. So we compute the derivative, with respect to θ, of the expected payoff. Reorganizing, we obtain the update rule in the algorithm. □

When to use direct policy search and when to use value iteration? A useful question here is which π^* or V^* is simpler.

Direct policy search works better for partially observable MDPs (POMDPs). In this model, at each step, we get a partial (and potentially noisy) measurement of the state and we have to choose an action a using this partial information.

One way of using direct policy search is to use some estimates like the Kalman filter or probabilistic graphical model to use historical measurements.

Bibliography

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT Press, 2016.

[HTF17] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, 2017. 2

[Mur12] Kevin P. Murphy. *Machine learning: a probabilistic perspective.* MIT Press, 2012.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction.* MIT Press, 2nd edition, 2018.