# Chp3: Probability and Information Theory

## Probability

Probability theory provides a means of quantifying uncertainty. **Probability** is used to represent a **degree of belief**, ranging from (0, 1).

**Frequentist probability** refers to the rates at which events occur. e.g. play a card game and output is $p$, means if repeat the game infinitely many times, and proportion of p would result in same outcome

**Bayesian probability** relates to qualitative levels of certainty. e.g. doctor analyzes the patient and says 40% chance of having a flu (sth not repeatable)

## Random Variables

A **random variable** is a variable that can take on different values randomly.

Random variables can be **discrete** or **continuous**.

## Probability Distributions

Discrete Variables and Probability Mass Function:

A probability distribution over discrete variables can be described using a **probability mass function**, denoted as P(x = $x$) or x ~ P(x).

A probability distribution over many variables is known as **joint probability distribution**, denoted as P(x = $x$, y = $y$) or P($x, y$) in short.

Three properties for probability mass function:

- domain of P cover the set of all possible states of x
- $\forall x \in$ x, $0 \leq P(x) \leq 1$
- **normalized**: $\sum_{x \in x} P(x) = 1$

Continuous Variables and Probability Density Functions:

The probability distribution with continuous random variables can be described using a **probability density function**.

## Expectation, Variance and Covariance

**Expectation** or **expect value** of f(x) with respect to a probability distribution P(x) is the average value $f$ can take on when x is drawn from $P$.

For discrete variables expectation is denoted as $\mathbb{E}_{x \sim P}[f(x)] = \sum_x P(x)f(x)$

For continuous variables expectation is computed as $\mathbb{E}_{x \sim P}[f(x)] = \int P(x)f(x)dx$

Expectations are linear $\mathbb{E}_x[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_x[f(x)] + \beta \mathbb{E}_x[g(x)]$

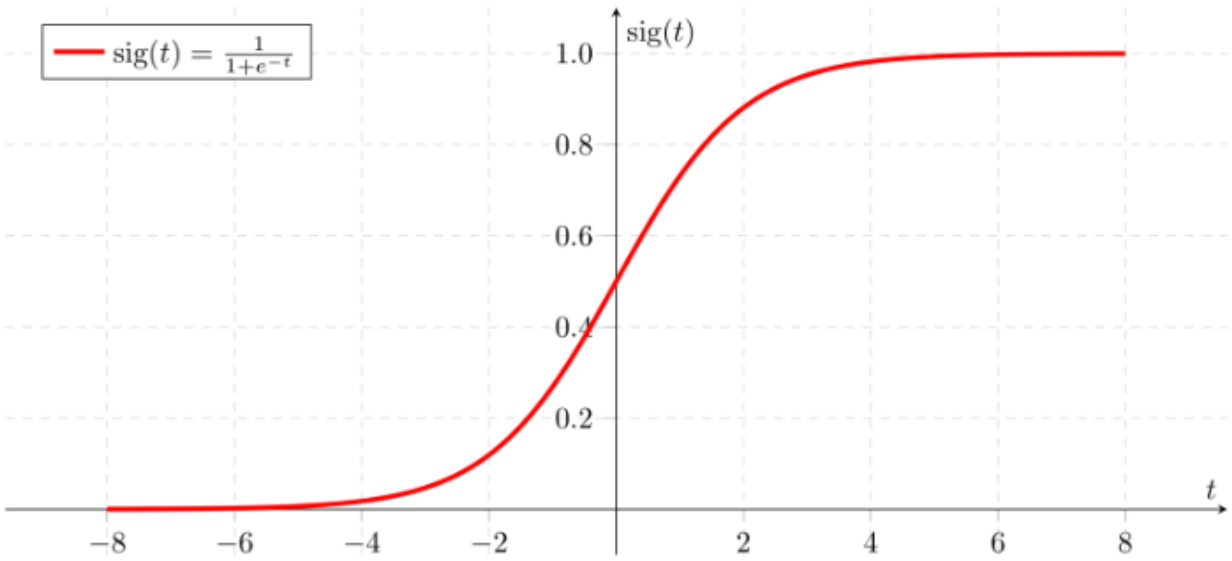
**Variance** measures how much the values x vary as we sample different values of x from its probability distribution. $Var(f(x)) = \mathbb{E}[(f(x) - [f(x)])^2]$. **Standard deviation** is the square root of the variance.

**Covariance** measures how much two values are linearly related to each other.

**Correlation** normalizes the contribution of each variable in order to measure only how much the variables are related.

## Sigmoid

$sig(t) = \frac{1}{1+e^{-t}}$

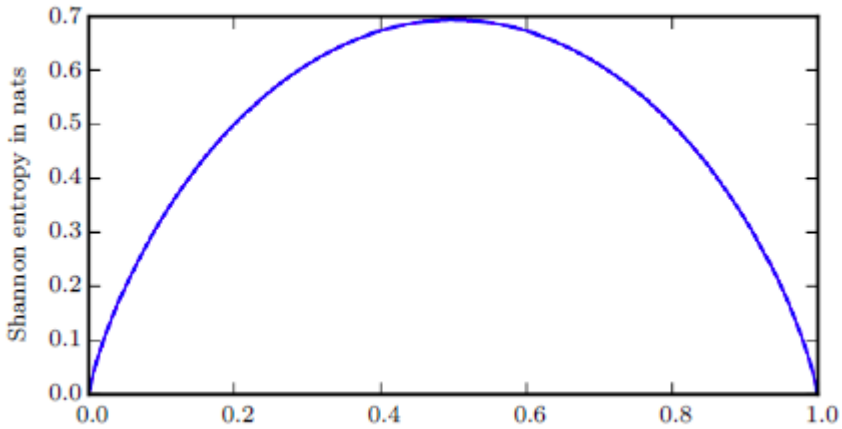



## Information Theory

Information theory assumes that learning an unlikely event has occurred is more informative than learning that a likely event has occurred. In other words, likely events should have low information content, while less likely events should have higher information content.

**Self-information** of an event x = $x$ to be $I(x) = \log P(x)$.

**Shannon entropy** quantifies the amount of uncertainty in an entire probability distribution. $H(x) = \mathbb{E}_{x \sim P}[I(x)] = -\mathbb{E}_{x \sim P}[\log P(x)]$. Distributions that are closer to uniform have high Shannon entropy; while deterministic distributions have low entropy.




If we have two separate probability distributions P(x) and Q(x) over the same random variable x, we can measure how different these two distributions are using the **Kullback-Leibler (KL) divergence**: $D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$. The KL divergence is 0 if and only if P and Q are the same distribution in the case of discrete variables.

**Cross-entropy** $H(P, Q) = H(P) + D_{KL}(P||Q)$. Could be used in machine learning where $P$ is ground truth distribution and $Q$ is estimated distribution.

# Chp4: Numerical Computation

## Overflow and Underflow

The fundamental difficulty in computing math on a computer is that we need to represent infinitely numbers with a finite number of bit patterns.

**Underflow** refers to when numbers near zero are rounded to zero.

Many function behave qualitatively when the argument is zero rather than a small positive number. e.g. some environment will raise an exception or return not-a-number when divided by zero.

**Overflow** refers to when numbers with a large magnitude are approximated as $\infty$ or $-\infty$.

A possible solution could be softmax function. $softmax(x) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_i)}$

## Conditioning

**Conditioning** refers to how rapidly a function changes with respect to small changes in its inputs.

## Gradient based optimization

Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x. Mathematically find $x^* = argmin f(x)$. Usually we call the function we want to minimize **cost function**, or **loss function**, or **error function**.

Gradient descent is the technology that use derivative to minimize a function.

The **derivative** $\frac{d_y}{d_x}$ or $f'(x)$ measures how $f$ changes with $x$. When with multiple inputs, the **partial derivative** $\frac{\partial}{\partial x_i} f(x)$ measures how $f$ changes as only the variable $x_i$ increases at point x. **Gradient** of $f$ is the vector containing all partial derivatives, denoted $\nabla_x f(x)$.

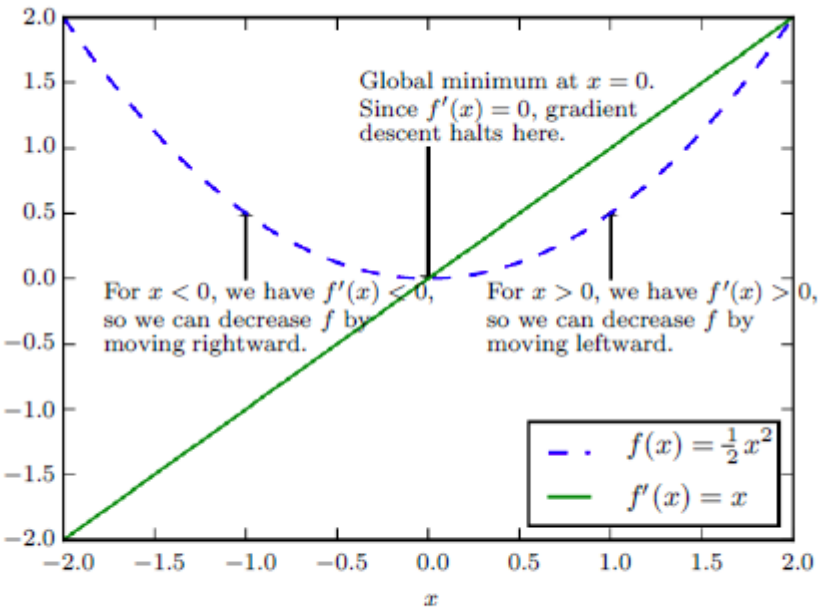
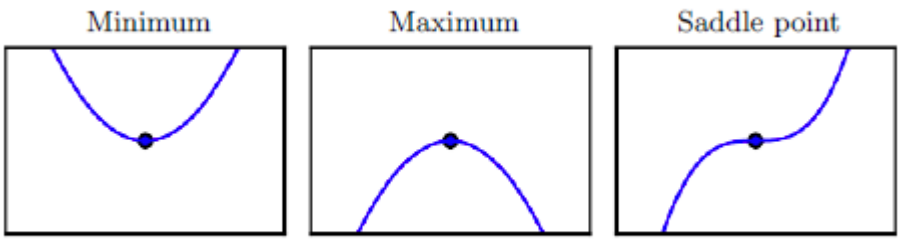



Figure 4.1: An illustration of how the gradient descent algorithm uses the derivatives of a function can be used to follow the function downhill to a minimum.

**Critical points** or **stationary points** are points where $f'(x) = 0$. With multiple inputs, critical points are points where every element of the gradient is zero.




A **local minimum** is a point where $f(x)$ is lower than all neighboring. A **local maximum** is a point where $f(x)$ is higher than all neighboring. **Saddle points** are points neither maxima nor minima. **Global minimum** is a point that obtains the absolute lowest value.

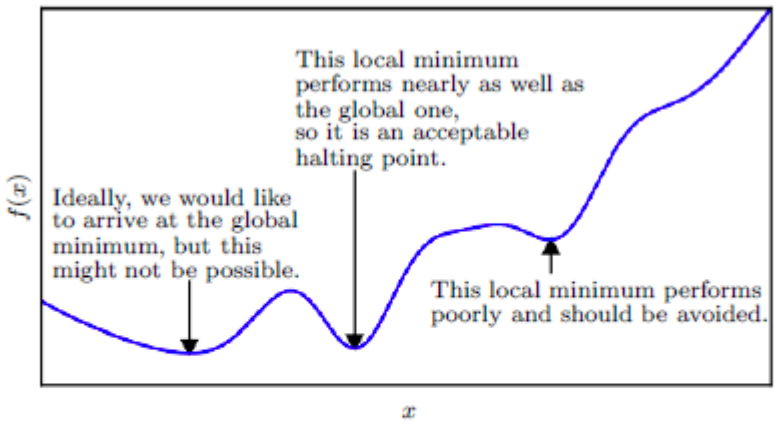



Figure 4.3: Optimization algorithms may fail to find a global minimum when there are multiple local minima or plateaus present. In the context of deep learning, we generally accept such solutions even though they are not truly minimal, so long as they correspond to significantly low values of the cost function.

**Steepest gradient** or **gradient descent** is the method of moving $f$ in the direction of negative gradient. $x' = x - \epsilon \nabla_x f(x)$, where $\epsilon$ is the learning rate, which regulates the size of the step.

# Chp5: Machine Learning Basics

## Learning Algorithm

A machine learning algorithm is an algorithm that is able to learn from data. It is defined as learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$ , as measured by $P$ , improves with experience $E$.

### Task

Task is the means of attaining the ability to perform the task.

Some of the common machine learning tasks include:

- classification: specify which of $k$ categories some input belongs to $f : R^n \rightarrow 1, \ldots, k$
  variant: specify a probability distribution over classes
- classification with missing inputs:
- regression: predict a numerical value given some input $f : R^n \rightarrow R$
- transcription: given a unstructured representation of data, transcribe it into textual form
- machine translation: receive a character sequence in some language and convert it into another language
- anomaly detection: sift through a set of events and flag some of them as unusual
- synthesis and sampling: generate new samples that are similar to training data
- denoising: predict a clear example $x$ from corrupted version $\tilde{x}$

### Performance Measure

A quantitative measure of the performance of machine learning algorithm is needed.

Common evaluation metric:

- accuracy: the proportion of examples the model produces correct output
- error rate: the proportion of examples the model produces incorrect output
- average log-probability

$w$ could be regarded as a set of weights that determine how each feature affects the prediction.

## Maximum Likelihood Estimation

Given a set of $m$ examples $X = x^{(1)}, \dots, x^{(m)}$ drawn from a distribution $p_{data}(x)$, $p_{model}(x;\theta)$ specify the model prediction of $p_{data}(x)$,

the maximum likelihood estimator for $\theta$ is

$$\begin{aligned}\theta_{ML} &= arg\max_{\theta} p_{model}(x;\theta) \\ &= arg\max_{\theta} \prod_{i=1}^{m} p_{model}(x;\theta)\end{aligned}$$

Because product over probability is inconvenient, for example, it is prone to underflow, we transform a product into a sum:

$$\theta_{ML} = arg\max_{\theta} \sum_{i=1}^{m} \log p_{model}(x^{(i)};\theta)$$

Because argmax does not change if we rescale the function, then we divide by $m$ and obtain an expectation: $\hat{P}_{data}$ is the empirical distribution defined by training set

$$\theta_{ML} = arg\max_{\theta} \sum_{i=1}^{m} \mathbb{E}_{x\sim\hat{P}_{data}} \log p_{model}(x^{(i)};\theta)$$

Another way to interpret maximum likelihood estimation is to view it as minimizing the dissimilarity between empirical distribution and the model distribution. One measurement is KL divergence

$$D_{KL}(\hat{P}_{data}||p_{model}) = \mathbb{E}_{x\sim\hat{P}_{data}}[\log \hat{P}_{data}(x) - \log p_{model}(x)]$$

Because the left term $\log \hat{P}_{data}(x)$ is irrelevant with the model, to minimize KL divergence, we only need to minimize $-\mathbb{E}_{x\sim\hat{P}_{data}} \log p_{model}(x)$

Therefore maximum likelihood = minimize log-likelihood = minimize KL divergence

### Supervised Learning Algorithm

### Stochastic gradient descent

A recurring problem in machine learning is that large training sets are necessary for good generalization, but they also means more computationally expensive.

**Stochastic gradient descent** is an extension of gradient descent algorithm. Instead of computing gradients on the whole training set, it use a small set of samples, called **minibatch**. The idea is we fit a training set with billions of examples using updates computed on a hundred samples.

The estimate of the gradient is formed as $g = \frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta)$. Then parameter is updated as $\theta \leftarrow \theta - \epsilon * g$

# Chp6: Feedforward Networks

The **goal** of feedforward network is to approximate some function $f^*()$. For example, for a classifier, $y = f^*(x)$ maps an input $x$ to an output category $y$. A feedforward network could be defined as $y = f(x;\theta)$ , $\theta$ is the parameters.

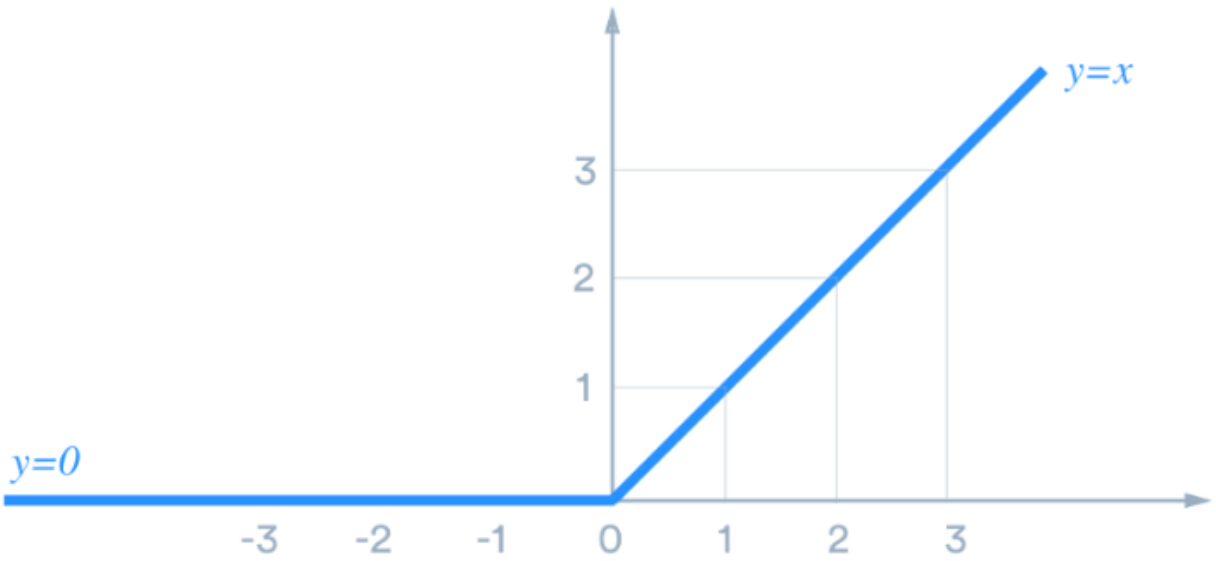
The models are called **feedforward** because there are no feedback connections from output fed into input. Networks with feedback connections are

Networks could be represented as a chain of functions. e.g. $f(x) = f^{(3)}(f^{(2)}(f^{(1)}()))$. The **depth** of model is the length of the chain. In this model, $f^{(1)}$ is **first layer**, $f^{(2)}$ is **second layer**. **Output layer** is $f^{(3)}$ which produces a value that close to $f^*$. **Hidden layers** are layers not directly deal with input and output.

### Activation function

A simple example of neural network is linear models, such as linear regression and logistic regression. But the obvious defect is that due to the limit of linear functions, they cannot understand the interaction between two input variables. Also, the combination of linear functions is also linear function.

So to extend linear models, a non-linear function is needed. **Activation function** is a fixed, not linear function. A typical example is **rectified linear unit (ReLU)**. Mathematically defined as $y = max(0, x)$.




### Gradient base learning

Gradient based learning is sensitive to input so the initialization of feedforward network should be done randomly. In this way, the biases could be initialized to zero or to small numbers.

#### Cost Function

Use cross-entropy between training data and the model's predictions as the cost function.

One requirement for cost function is that the gradient of it must be large and predictable enough to serve as a good guide for learning algorithm. The **negative log-likelihood** helps to solve this problem.

## Chp9: Convolutional Networks

Convolutional network are a specialized neural network for processing grid-like topology, e.g. time-series data (1D), image data (2D). Convolutional network refers to a neural network that use convolution in place of general matrix multiplication at least one layer.

### Convolution

$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$

First parameter(function x) is referred as **input** and second referred as **kernel**. The output is sometimes referred as **feature map**.

In machine learning, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm.

Convolution can be used in more than one axis:

$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$

Convolution is **commutative**:

$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$

**Toeplitz matrix**: each row of the matrix is constrained to be equal to the row above shifted by one element.

## Motivation for CNN

Convolution leverages three important ideas that can help improve learning process: **sparse interactions**, **parameter sharing** and **equivariant representations**.

Convolutional networks usually have **sparse interactions/sparse connectivity/sparse weights**. This is done by making the kernel smaller than the input. For example, the input image might have thousands of pixels, but small and meaningful features such as edges in kernels only occupy tens of pixels.

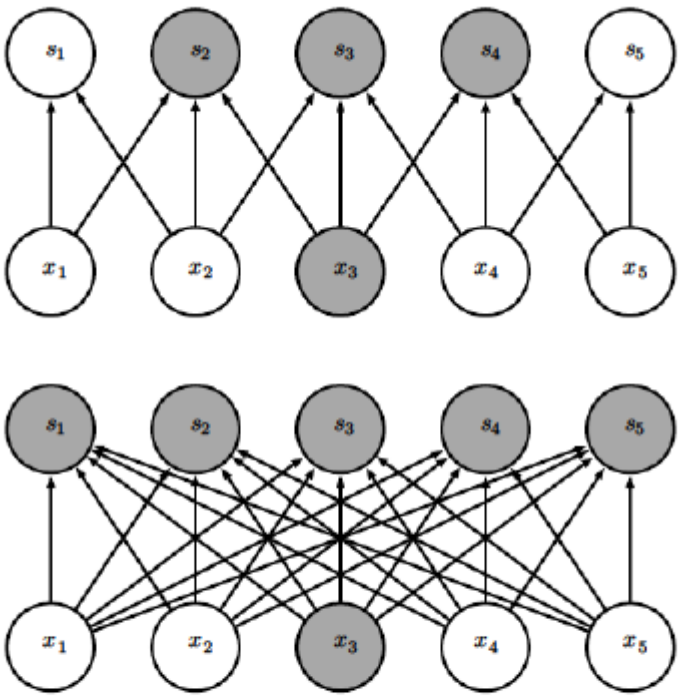



Figure 9.2: *Sparse connectivity, viewed from below:* We highlight one input unit, $x_3$, and also highlight the output units in $\boldsymbol{s}$ that are affected by this unit. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three outputs are affected by $\boldsymbol{x}$. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by $x_3$.

receptive field: the input units that affect specific output unit

Parameter sharing: use the same parameter for more than one function in a model.

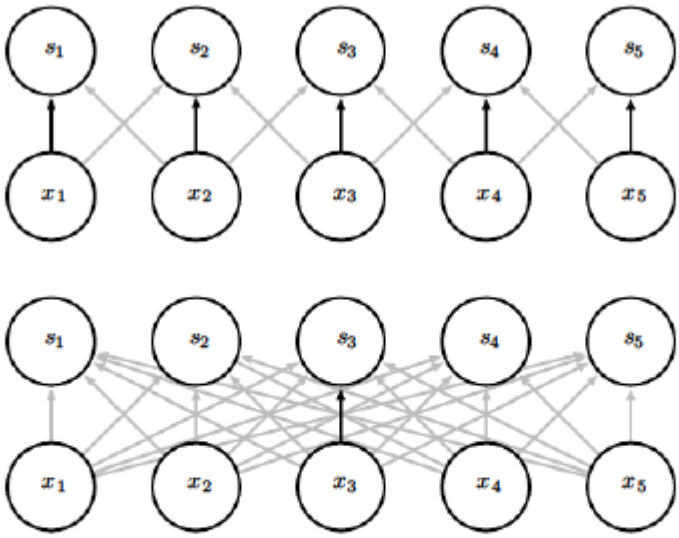



Figure 9.5: Parameter sharing: Black arrows indicate the connections that use a particular parameter in two different models. *(Top)*The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. *(Bottom)*The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

**Equivariance** to translation: if the input changes, the output changes in the same way. In other words, if we are processing time series data, and we move an event layer in time in the input, the exact same representation of it will appear in the output, just later in time.

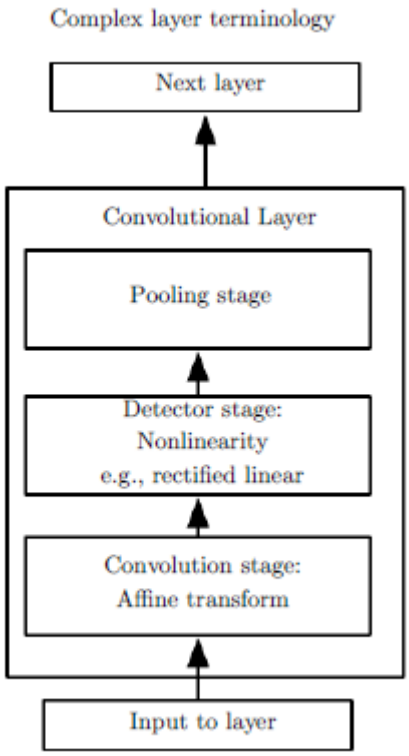
$f(g(x)) = g(f(x))$

But sometimes, we don't want to share parameters across the entire image. For example, if we want to extract different features at different locations (face detection)

## Pooling

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

Pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. It can be a very useful property if we care more about whether some feature is present than exactly where it is.




It is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixel apart.

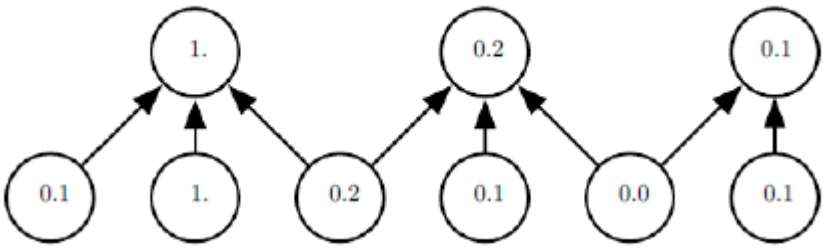
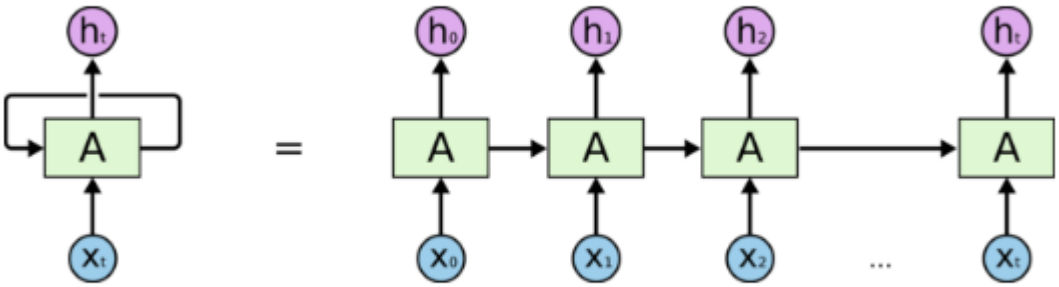



Figure 9.10: *Pooling with downsampling.* Here we use max-pooling with a pool width of three and a stride between pools of two. This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer. Note that the rightmost pooling region has a smaller size, but must be included if we do not want to ignore some of the detector units.

For many tasks, pooling is essential for handling inputs of varying size. For example, if we want to classify images of variable size, the input to the classification layer must have a fixed size.

# Chp10: Sequence Modeling




**An unrolled recurrent neural network.**

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

Convolutional network accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output.(e.g. probabilities of different classes). These models perform this mapping using a fixed amount of computational steps (e.g. the number of layers in the model).

Recurrent networks allow sequences of vectors. *no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.*

By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.

LSTM: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/