1c: Probability, Generalization and Overfitting

Overview

In this lesson, we will briefly review certain topics from probability which are essential for deep learning, and we will introduce the issue of generalization and overfitting in supervised learning.

We will then discuss cross entropy and softmax, which are used for classification tasks as alternatives to the sum squared error loss function. Finally, we will describe weight decay, momentum, and adaptive moment estimation (Adam). Along the way, we will need to introduce the mathematical concepts of Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP) Estimation.

Lesson learning outcomes

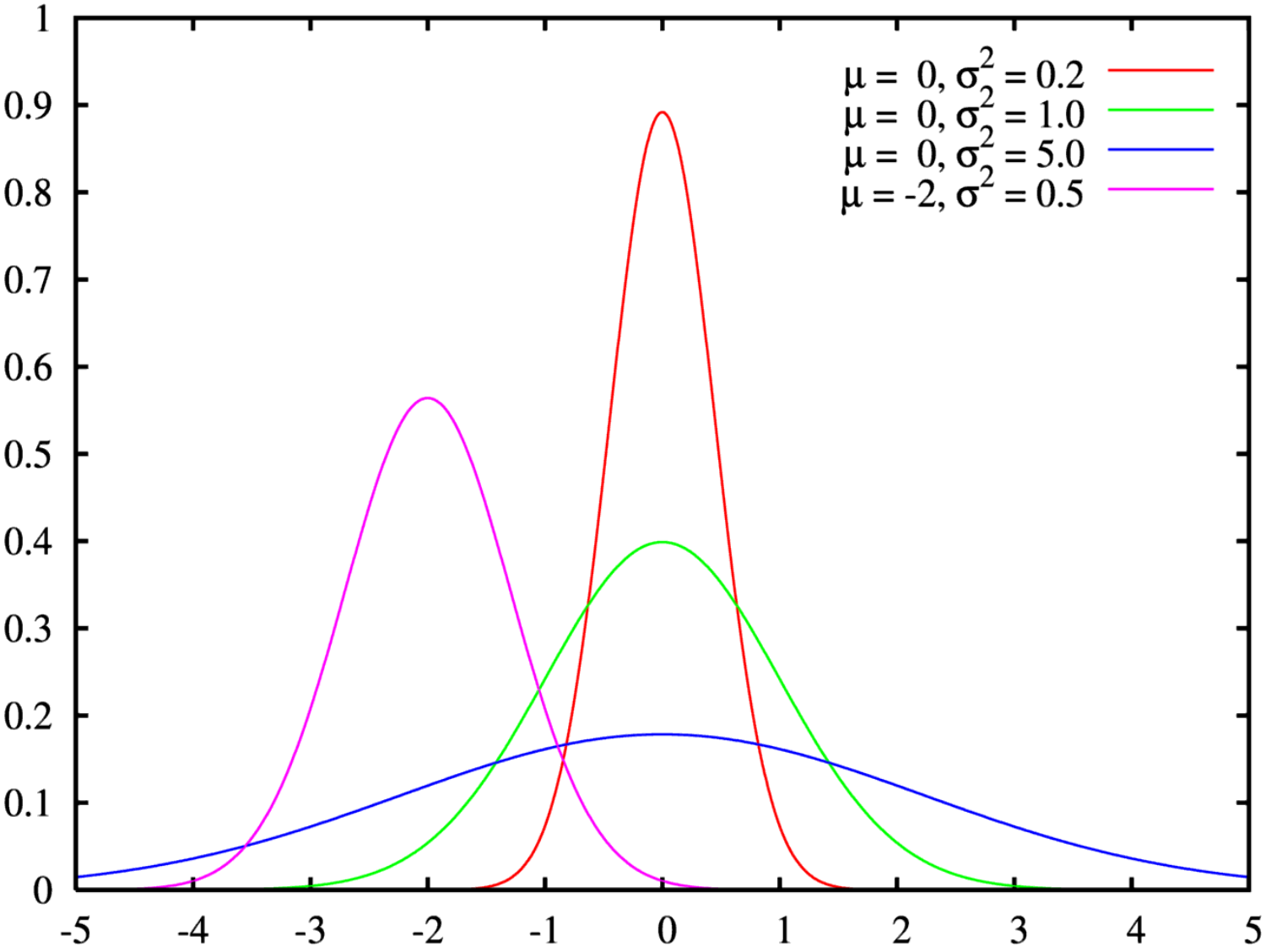
By the end of this lesson, you will be able to:

- compute Entropy and KL-Divergence for discrete probability distributions
- compute probabilities using Bayes' Rule
- describe the basics of supervised learning
- explain how to avoid overfitting in neural networks
- explain and compute cross entropy, softmax
- explain weight decay, momentum

Probability

This section briefly summarizes a few topics from Probability which are fundamental to neural networks and deep learning.

Gaussian Distributions



[image source]

The **Gaussian** distribution with mean μ and standard deviation σ is given by

$$P_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

The d-dimensional **multivariate Gaussian** with mean μ and covariance Σ is given by

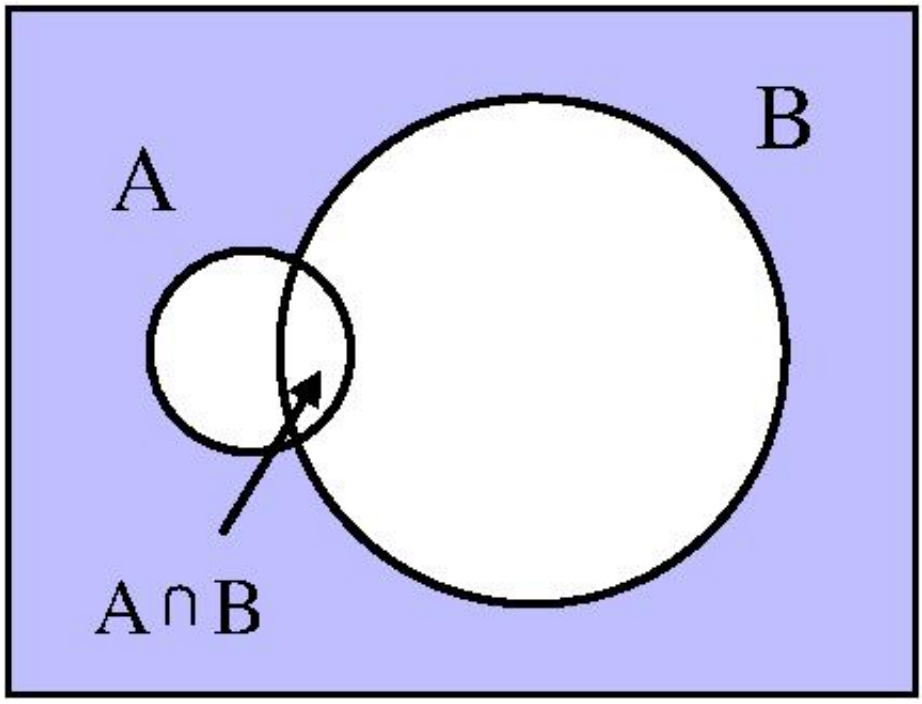
$$P_{\mu,\Sigma}(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)}$$

If $\Sigma = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_d^2)$ is diagonal, the multivariate Gaussian reduces to a product of 1-dimensional Gaussians

$$P_{\mu,\Sigma}(x) = \prod_i P_{\mu_i,\sigma_i}(x_i)$$

The multivariate Gaussian with $\mu = 0$, $\Sigma = \mathrm{I}$ is called the **Standard Normal** distribution.

Conditional Probability



If $P(B) \neq 0$, then the **conditional probability** of A given B is

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Bayes' Rule

The formula for conditional probability can be manipulated to find a relationship when the two variables are swapped:

$$P(A \cap B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

$$\rightarrow \text{Bayes' Rule } P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

This is often useful for assessing the probability of an underlying cause after an effect has been observed:

$$P(\text{Cause} \mid \text{Effect}) = \frac{P(\text{ Effect} \mid \text{Cause }) P(\text{ Cause })}{P(\text{ Effect })}$$

Example: Light Bulb Defects

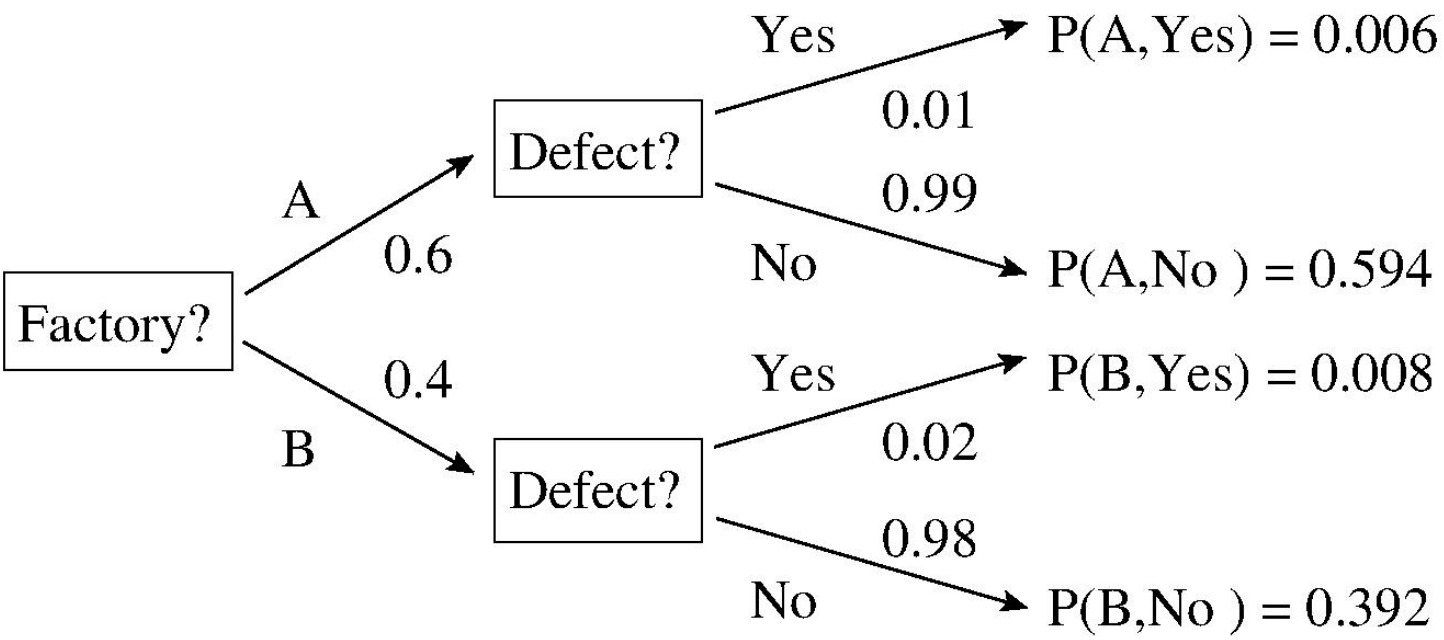
Question: You work for a lighting company which manufactures 60% of its light bulbs in Factory A and 40% in Factory B. One percent of the light bulbs from Factory A are defective, while two percent of those from Factory B are defective. If a random light bulb turns out to be defective, what is the probability that it was manufactured in Factory A?

Answer: There are two random variables: Factory (A or B) and Defect (Yes or No). The prior probabilities (before the bulb has been tested) are:

$$P(A) = 0.6, \quad P(B) = 0.4$$

The conditional probabilities are:

$$P(\text{Defect} \mid \text{A}) = 0.01, \quad \text{and} \quad P(\text{Defect} \mid \text{B}) = 0.02$$



$$P(\text{A} \mid \text{Defect }) = \frac{P(\text{Defect} \mid A)P(A)}{P(\text{Defect})}$$
$$= \frac{0.01 * 0.6}{0.01 * 0.6 + 0.02 * 0.4} = \frac{0.006}{0.006 + 0.008} = \frac{3}{7}$$

Optional video

Entropy and Huffmann Coding

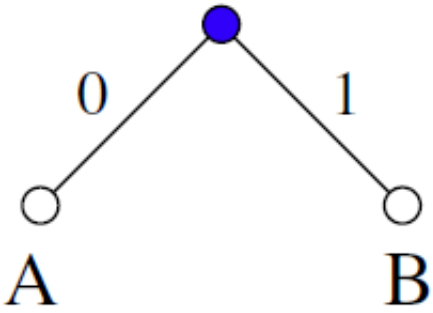
The **entropy** of a discrete probability distribution $p = \langle p_1, \ldots, p_n \rangle$ is

$$H(p) = \sum_{i=1}^{n} p_i(-\log_2 p_i)$$

One way to think of entropy is the number of bits per symbol achieved by a (block) Huffman Coding scheme.

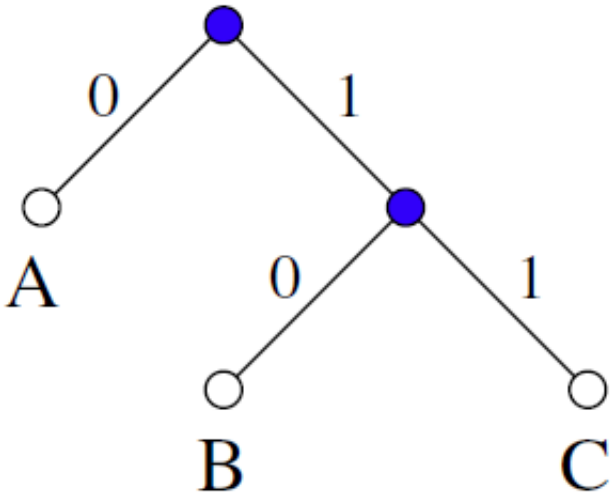
Example 1: $H(\langle 0.5, 0.5 \rangle) = 1 \text{ bit}.$

Suppose we want to encode, in zeros and ones, a long message composed of the two letters A and B, which occur with equal frequency. This can be done efficiently by assigning $A = 0$, $B = 1$. In other words, one bit is needed to encode each letter.



Example 2: $H(\langle 0.5, 0.25, 0.25 \rangle) = 1.5$ bits.

Suppose we need to encode a message consisting of the letters A, B and C, and that B and C occur equally often but A occurs twice as often as the other two letters. In this case, the most efficient code would be $A = 0, B = 10, C = 11$. The average number of bits needed to encode each letter is 1.5.



If the symbols occur in some other proportion, we may need to "block" them together in order to encode them efficiently. But, asymptotically, the average number of bits required by the most efficient coding scheme is given by the entropy

$$H\left(\langle p_1, \ldots, p_n \rangle\right) = \sum_{i=1}^{n} p_i(-\log_2 p_i)$$

KL-Divergence

Given two probability distributions $p = \langle p_1, \ldots, p_n \rangle$ and $q = \langle q_1, \ldots, q_n \rangle$ on the same set Ω, the **Kullback-Leibler Divergence** between p and q is

$$D_{\mathrm{KL}}(p \parallel q) = \sum_{i=1}^{n} p_i(\log_2 p_i - \log_2 q_i)$$

In coding theory, $\mathrm{D_{KL}}(p \parallel q)$ is the number of extra bits we need to transmit if we designed a code

for $q()$ but it turned out that the samples were drawn from $p()$ instead.

KL-Divergence is like a "distance" from one probability distribution to another. However, it is not symmetric.

$$D_{\mathrm{KL}}(p \parallel q) \neq D_{\mathrm{KL}}(q \parallel p)$$

Generally speaking, $D_{\mathrm{KL}}(p \parallel q)$ will be large if there exist some symbol(s) i for which q_i is very small but p_i is large. This is because the code for symbol i will be long, but it will occur frequently.

Entropy and KL-Divergence for Continuous Distributions

The entropy of a continuous distribution $p()$ is

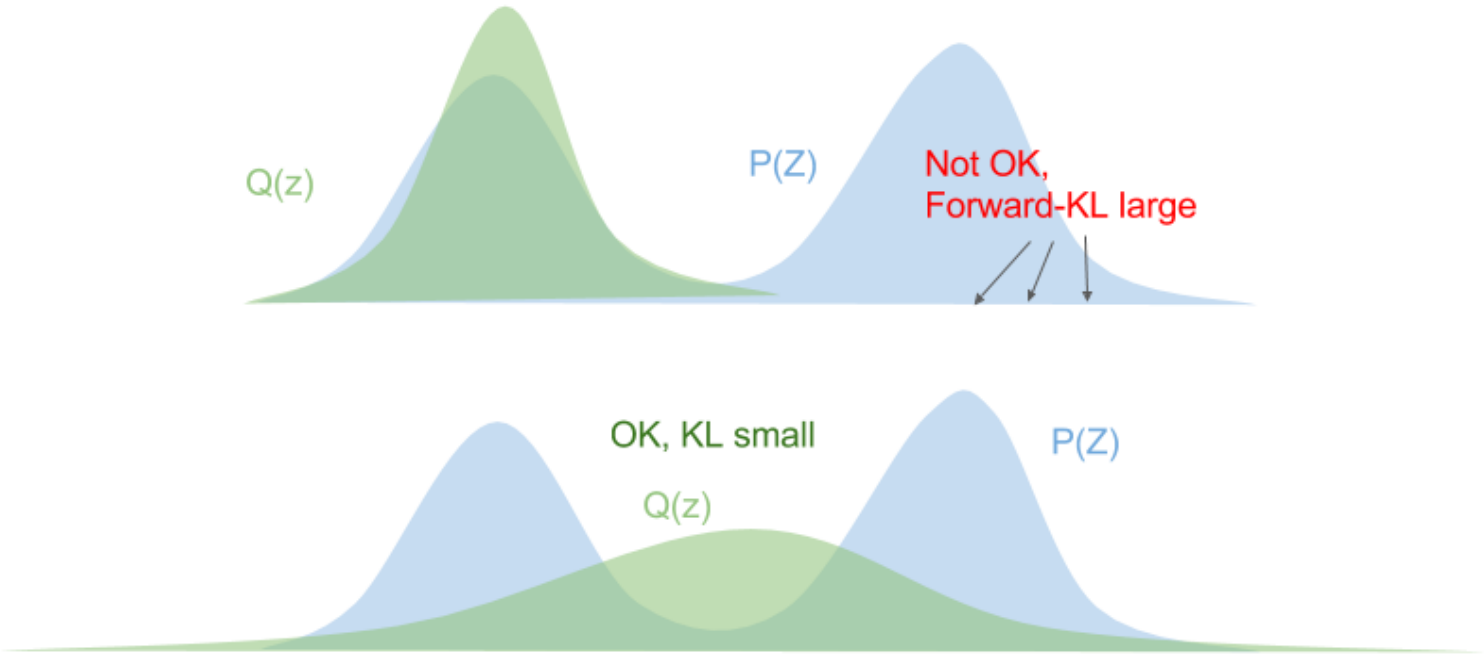
$$H(p) = \int_{\theta} p(\theta)(-\log p(\theta))d\theta$$

The KL-Divergence between two continuous distributions $p()$ and $q()$ is

$$\mathrm{D}_{\mathrm{KL}}(p \parallel q) = \int_{\theta} p(\theta)(\log p(\theta) - \log q(\theta))d\theta$$

Forward KL-Divergence

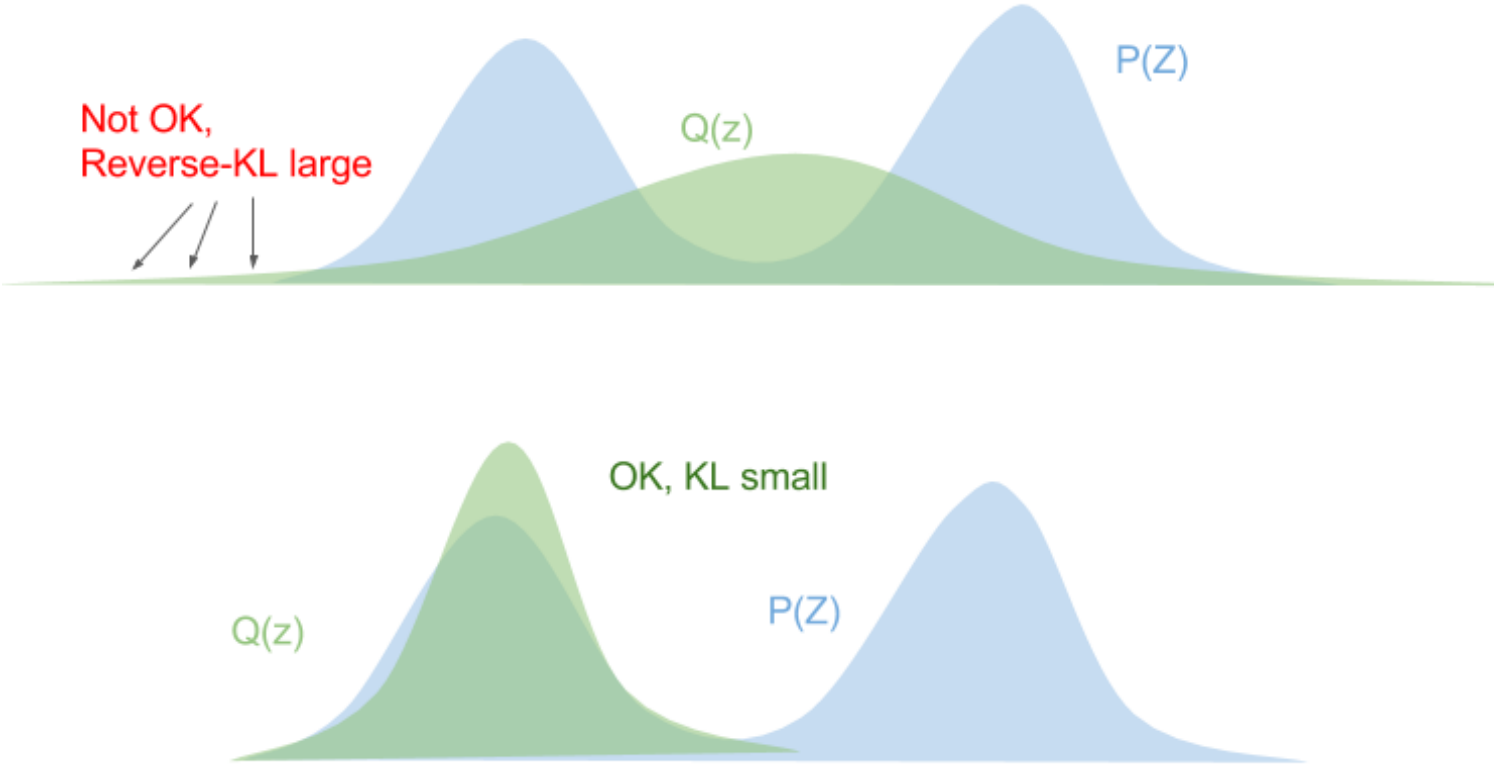
Suppose we are under attack from a drone and we want to track its position. Let's assume the likely location of the drone is given by a probability distribution $P()$ and we want to approximate $P()$ with a Gaussian distribution $Q()$. In this case, we may wish to minimize the **forward** KL-Divergence $\mathrm{D}_{\mathrm{KL}}(P \parallel Q)$. This ensures there will be no place where Q is very small but P is large, so we will not move to that place and get hit by the drone.



[image source: https://blog.evjang.com/2016/08/variational-bayes.html]

Reverse KL-Divergence

Now consider a situation where we are learning to play a video game, and $P()$ is the distribution of actions chosen by a human expert in a particular game state. In this case, we may prefer to minimize the **reverse** KL-divergence $\mathrm{D}_{\mathrm{KL}}(Q \parallel P)$. This ensures there will be no place where P is very small but Q is large, so we will not choose an action which is very unlikely to ever be chosen by the human player.



[image source: https://blog.evjang.com/2016/08/variational-bayes.html]

Optional video

Further Reading

- Gaussian function
- Bayes' Theorem
- Kullback–Leibler divergence
- Probability for machine learning

Textbook Deep Learning (Goodfellow, Bengio, Courville, 2016):

- Probability and Information Theory (Chapter 3)

Generalisation and Overfitting

Supervised, Reinforcement and Unsupervised Learning

Three types of learning will be discussed in this course: Supervised Learning (Weeks 1-4), Reinforcement Learning (Week 5) and Unsupervised Learning (Week 6).

- **Supervised Learning**: system is presented with examples of inputs and their target outputs,
- **Reinforcement Learning**: system is not presented with target outputs, but is given a reward signal, which it aims to maximize,
- **Unsupervised Learning**: system is only presented with the inputs themselves, and aims to find structure in these inputs.

Supervised Learning

For Supervised Learning, we have a **training** set and a **test** set, each consisting of a set of items. Each item specifies a number of input attributes and a target value.

The system is presented with the input and target values for all items in the training set; it goes through some kind of learning procedure, and must then predict the output for each item in the test set.

Various learning paradigms are available for Supervised Learning (Decision Trees, K-Nearest-Neighbors, Support Vector Machine, Random Forest, etc.). In this course, we will concentrate on Neural Networks.

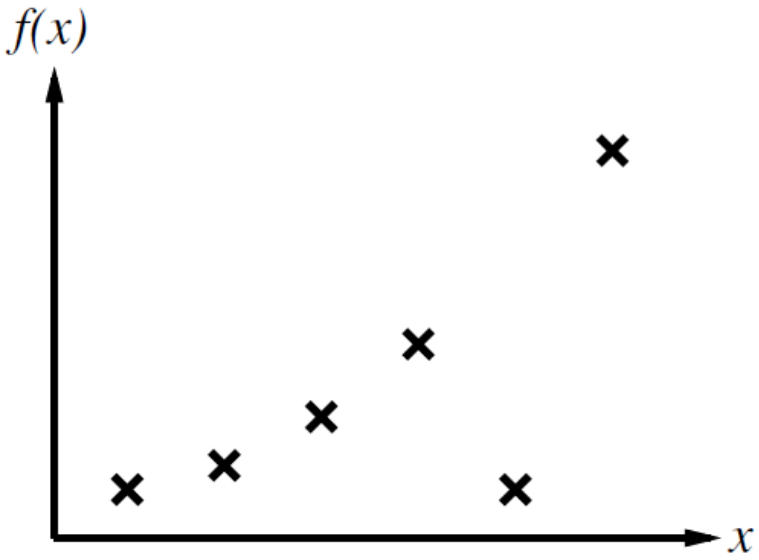
Optional video

Curve Fitting Example

The aim of Supervised Learning is to accurately predict the target value for all items in the **test** set, based on the input attributes.

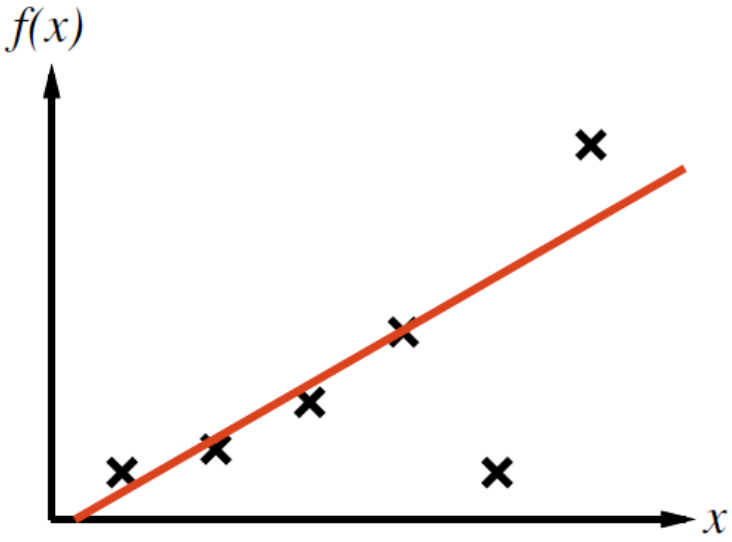
One common mistake that we need to avoid is building a model which fits the **training** set very well, but makes poor predictions on the **test** set. This is known as **overfitting**. In contrast, a system which achieves good accuracy on both the training and test set is said to achieve good **generalization**.

Consider, for example, the task of fitting a curve to the following points:

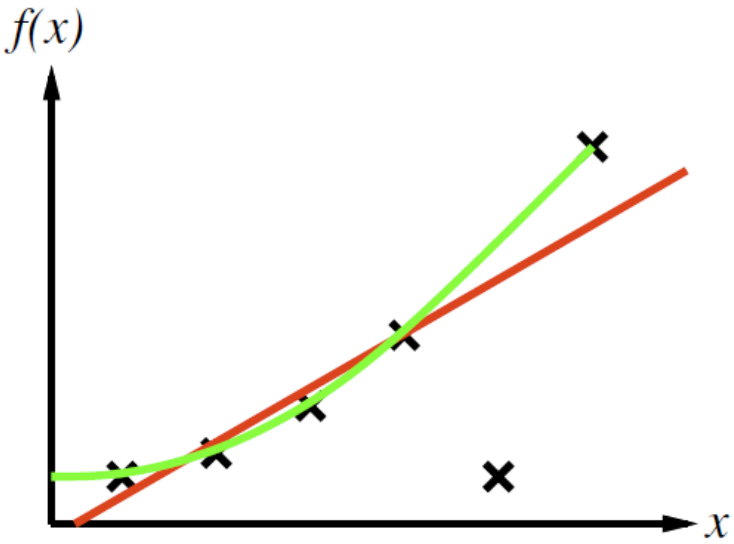


Which curve do you think gives the “best fit” to these data?

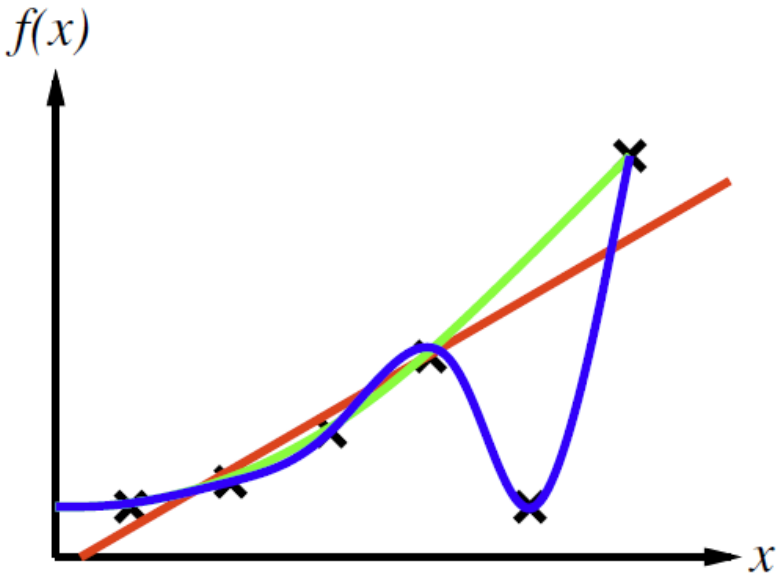
Straight line?



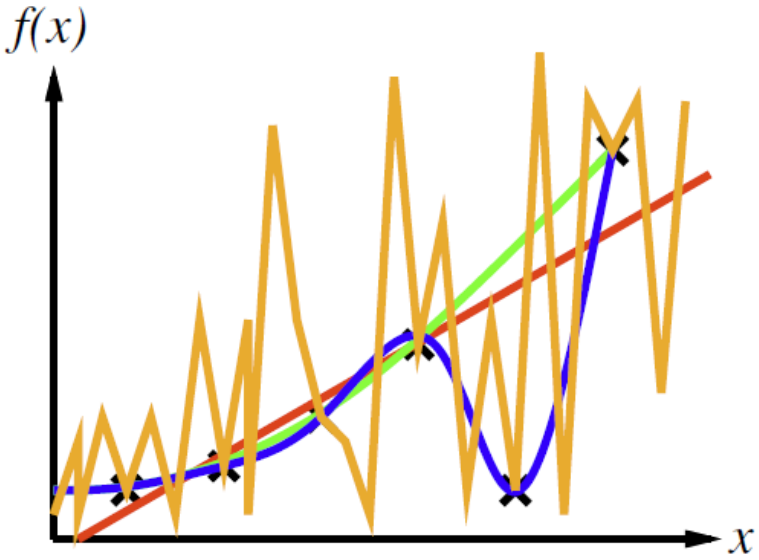
Parabola?



Fourth order polynomial?

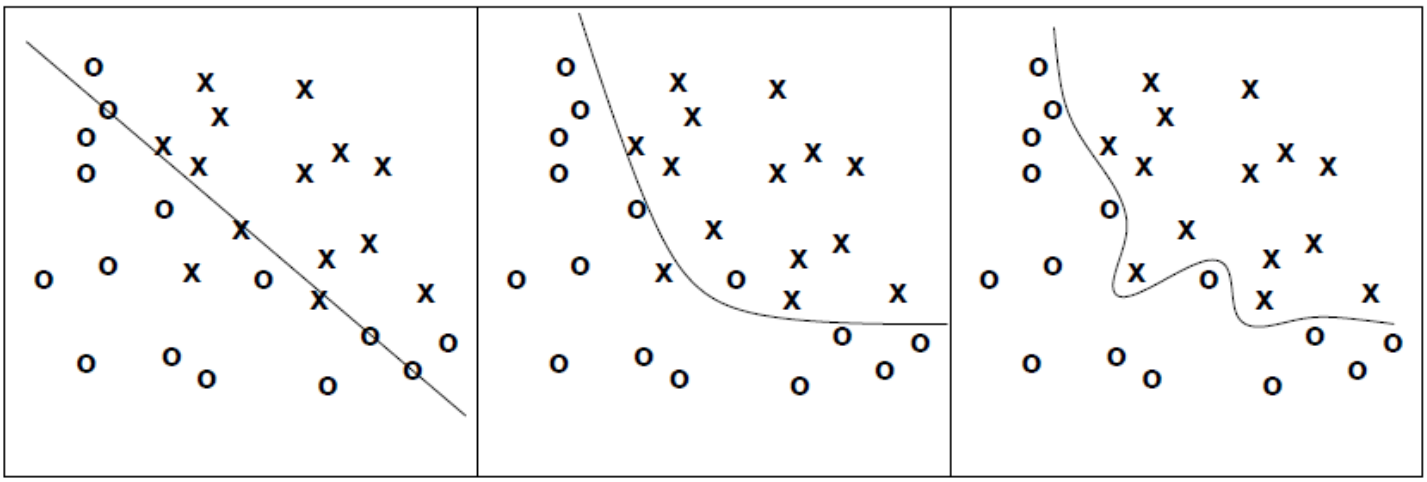


Something else?



Ockham's Razor

Here is another example. The **x**'s and **o**'s represent training items for which the target value is 1 and 0, respectively. The aim is to draw a curve such that test set observations on one side of the curve are predicted to be **x** and on the other side are predicted to be **o**.



inadequate good compromise over-fitting

"The most likely hypothesis is the simplest one consistent with the data."

This principle was popularized by William of Ockham in the thirteenth Century and came to be known as "Ockham's Razor".

Because there can be noise in the measurements, we now understand this principal as a tradeoff between simplicity of the hypothesis and how well it fits the data. In the above example, the straight line on the left side is considered inadequate to fit the data. The curve on the right hand side is seen as being too complicated, thus **overfitting** to the training data. The curve in the middle is seen as achieving a good compromise because it is relatively simple, but is still able to classify almost all the training data correctly.

Noise in the training data

You will notice that if we choose the middle curve in the above example, there are two training items near the boundary which get incorrectly classified. We consider this acceptable because there may be **noise** in the training data. This noise often arises as an accumulation of small changes due to factors which are not included in the model (roundoff errors, friction, air resistance, limitations in measuring equipment, etc.).

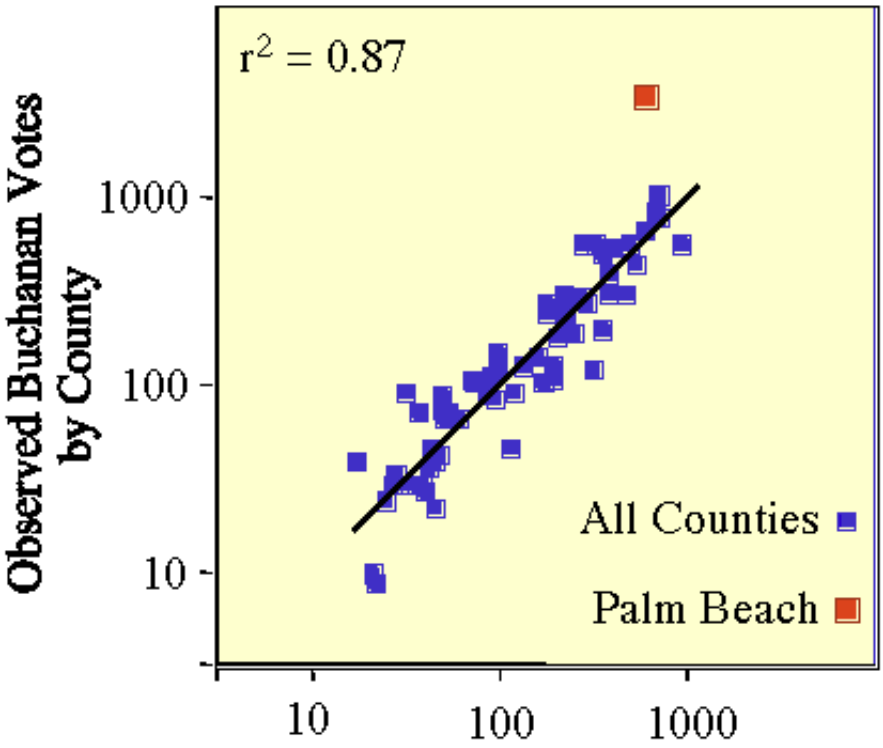
Looking again at the curve fitting example, we see that the 5th data point is correctly predicted by the blue curve but not by the green curve. In this case, the discrepancy is quite large and would more likely be caused by a macroscopic factor such as breakdown of the measuring equipment or human error in preparing the datasets. Our choice of model may therefore depend on how well we trust the data. We would normally prefer the green curve on the assumption that the 5th data point is some kind of anomaly. But, if we feel very confident that the inputs and targets in the training data are precisely correct, we may instead prefer the blue curve, which is more complicated but classifies all the training data correctly.

Outliers

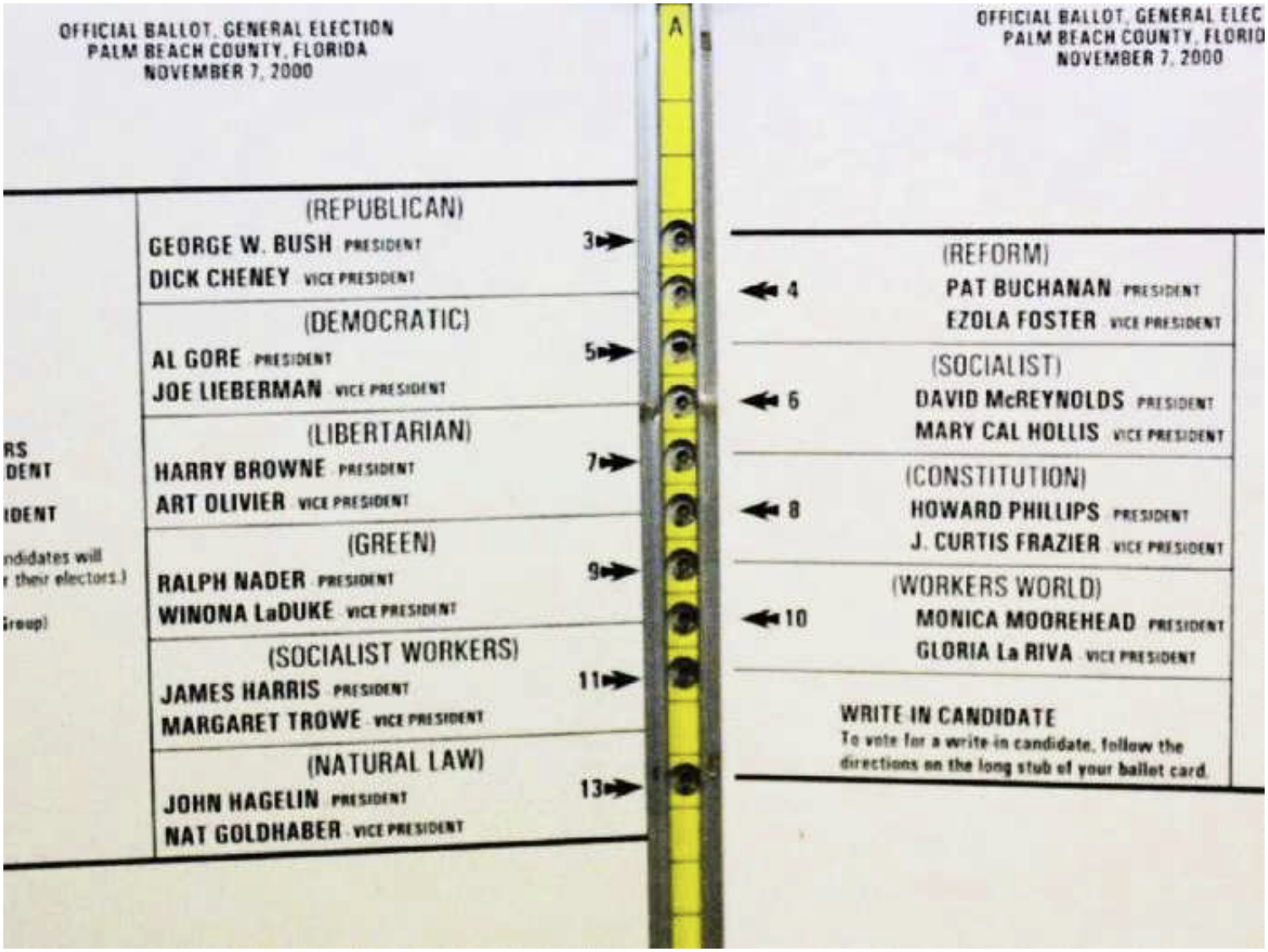
The 5th data point in the curve fitting example is known as an **outlier.**

Another famous example of an outlier is from the results of the 2000 US Presidential Election, in which George W. Bush defeated Al Gore in Florida by fewer than 1000 votes.

This diagram shows the number of votes received by a third party candidate called Pat Buchanan in all Florida counties on the vertical axis, compared to the prediction from a regression model based on total votes cast as well as the proportion of Democratic leaning voters on the horizontal axis. We see that the actual vote in Palm Beach County exceeds the predicted value by 1500 votes (see http://faculty.washington.edu/mtbrett/ for further details).



When we see such a big difference between an actual observation and what was predicted by the model, we should normally look in more detail to see if we can find an alternate explanation. Indeed, many scientific advances have been made in this way, to the point where some have suggested that the most important phrase in scientific discovery is not "Eureka!" but rather "That's funny?".



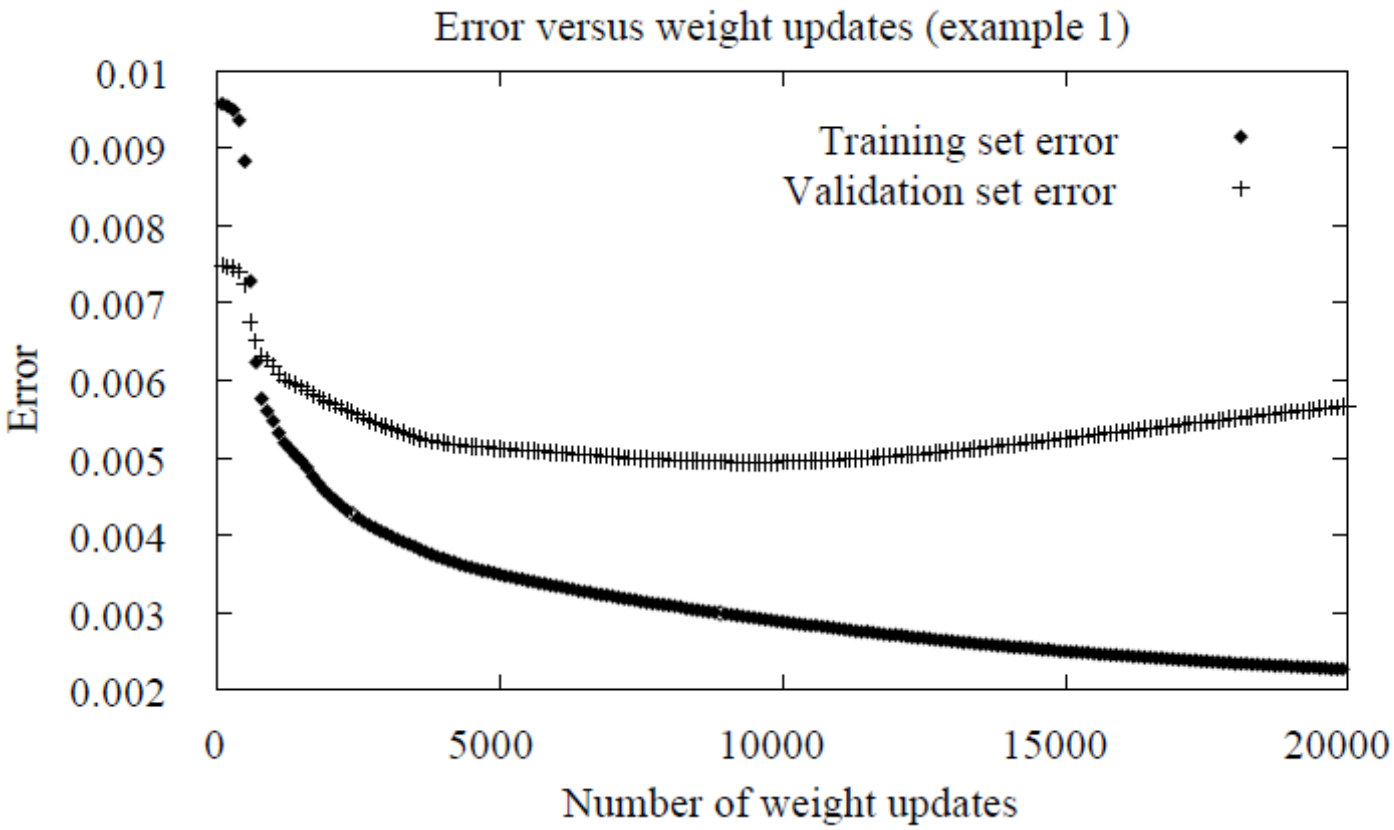
In the case of the Palm Beach voting anomaly, the most likely explanation lies in the design of the electoral ballot, which was known as a "butterfly ballot". It seems that many voters intended to vote for Al Gore, but accidentally voted for Pat Buchanan instead.

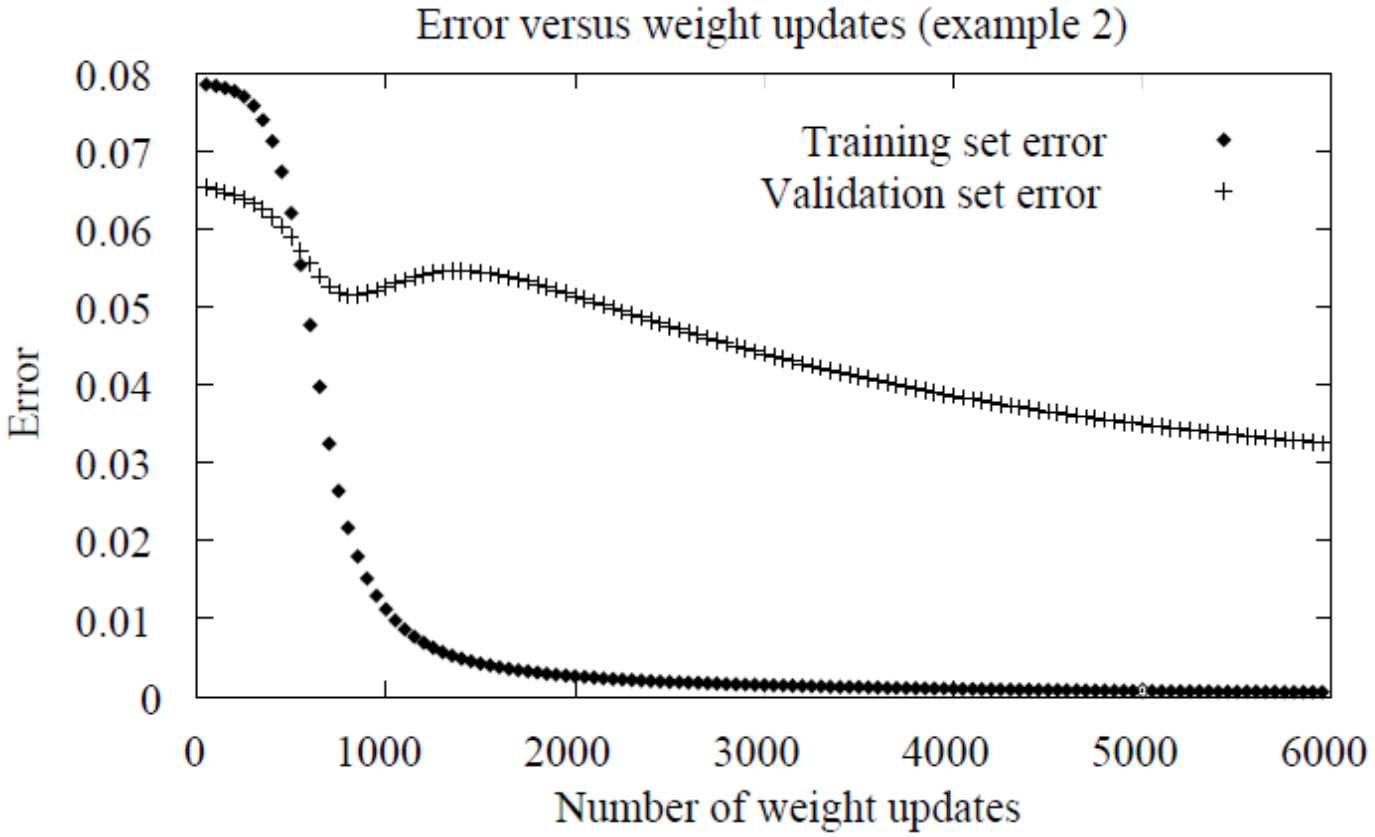
Optional video

Training, validation and test error

One way to avoid overfitting in neural networks is to limit the number of hidden nodes, connections or independent parameters in the network. In order to determine the optimum values, we often divide the data into Training, Validation and Test sets. Typically, as the number of hidden nodes is increased, both the training and validation set error will decrease initially, but after a certain point the training error will continue to decrease but the validation error will plateau or even slightly increase. The (approximate) value for which the validation error is minimal is likely to achieve low test set error as well.

Limiting the number of training epochs has also been tried - although, there is some evidence that this is not necessary when using weight decay (discussed in the next slide). Also, it may happen that the validation error increases temporarily before decreasing further, so you have to train for a sufficient number of epochs to understand what is really happening.



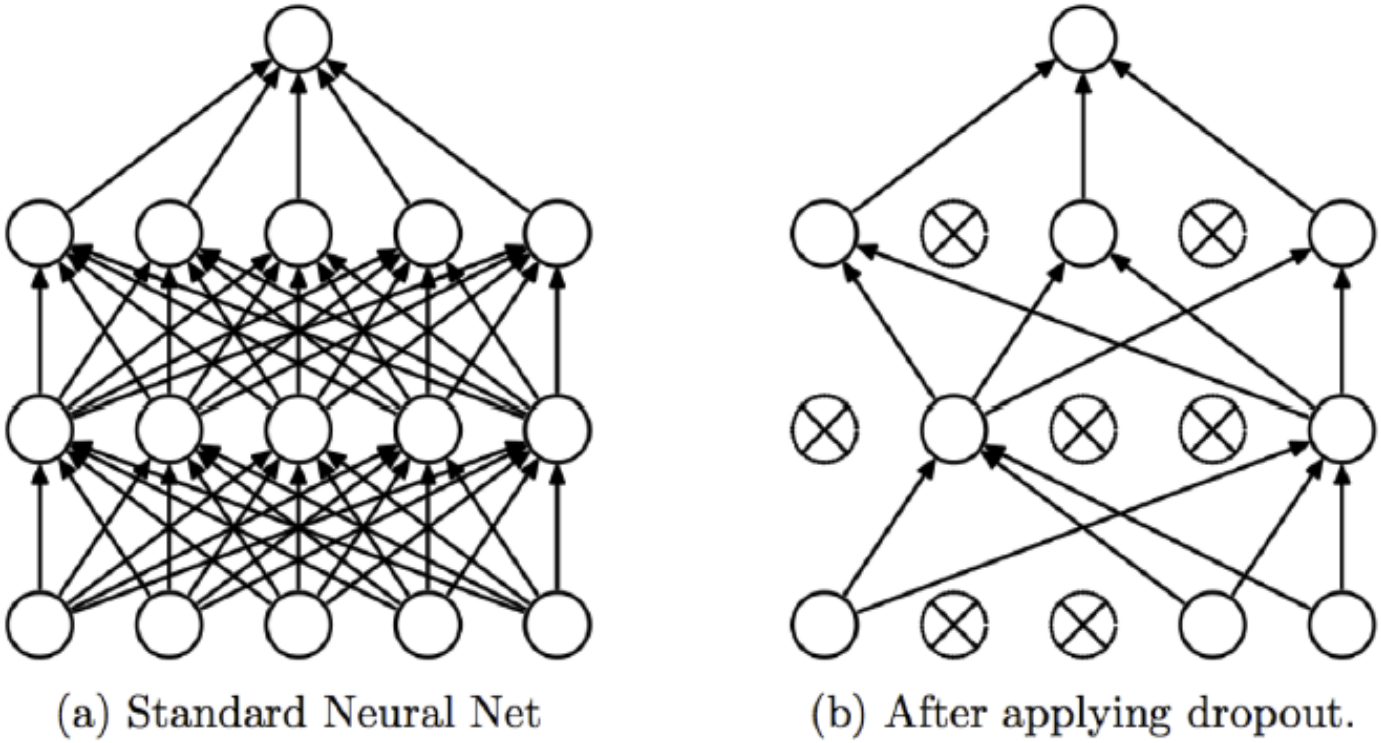


In this example, the validation set error is much larger than the training set error; but, it is still decreasing at epoch 6000.

Optional video

Dropout

Another way to avoid overfitting in neural networks is Dropout.



(a) Standard Neural Net

(b) After applying dropout.

[image source: Srivastava, 2014]

For each minibatch, we randomly choose a subset of nodes which will not be used in the training of that mini batch. Each node is chosen with some fixed probability (usually, one half). When training is finished and the network is deployed, all nodes are used, but their activations are multiplied by the same probability that was used to drop them out during training. Thus, the activation received by each unit during testing is the average of what it would have received during training.

Dropout forces the network to achieve redundancy because it must deal with situations where some features are missing. Another way to view dropout is that it implicitly (and efficiently) simulates an ensemble of different architectures.

Ensembling

Ensembling is a method where a number of different classifiers are trained on the same task, and the final class is decided by "voting" among them. In order to benefit from ensembling, we need to have diversity in the different classifiers. For example, we could train three neural networks with different architectures, three Support Vector Machines with different dimensions and kernels, as well as two other classifiers, and ensemble all of them to produce a final result. (Kaggle Competition entries are often constructed in this way).

Dropout as an Implicit Ensemble

In the case of dropout, a different architecture is created for each minibatch by removing the nodes that are dropped out. The trick of multiplying the output of each node by the probability of dropout implicitly averages the output over all of these different models. Thus, we get the benefit of ensembling but in a very efficient way.

Optional video

References

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929-1958.

Further Reading

Textbook Deep Learning (Goodfellow, Bengio, Courville, 2016):

- Overfitting in Neural Networks (5.2)
- Ensembling (7.11) and Dropout (7.12)

Exercise: Probability

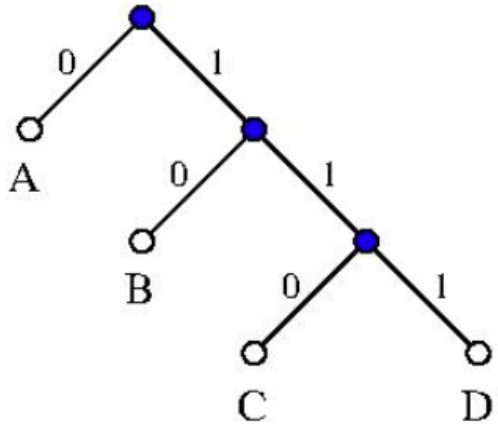
Consider these two probability distributions on the same space $\Omega = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$

$$\begin{aligned} p &= \langle ½, ¼, ⅛, ⅛ \rangle \\ q &= \langle ¼, ⅛, ⅛, ½ \rangle \end{aligned}$$

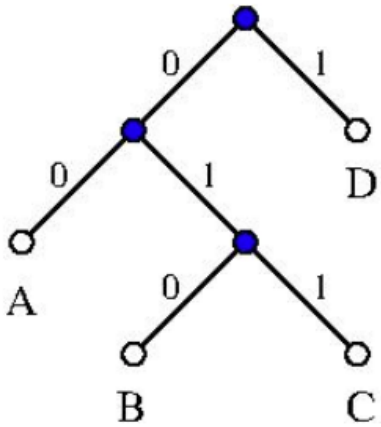
Question 1 *Submitted Sep 2nd 2022 at 7:36:53 pm*

Construct a Huffmann tree for each distribution p and q.

Huffman tree for p :



Huffman tree for q :



Question 2 *Submitted Sep 2nd 2022 at 7:37:32 pm*

Compute the Entropy $\mathrm{H}(p)$.

1.75

Explanation

$$\begin{aligned} \mathrm{H}(p) = \mathrm{H}(q) &= ½(-\log ½) + ¼(-\log ¼) + ⅛(-\log ⅛) + ⅛(-\log ⅛) \\ &= ½(1) + ¼(2) + ⅛(3) + ⅛(3) &= 1.75 \end{aligned}$$

Question 3 *Submitted Sep 2nd 2022 at 7:38:12 pm*

Compute the KL-Divergence in each direction $\mathrm{D}_{\mathrm{KL}}(\,p\,||\,q\,)$ and $\mathrm{D}_{\mathrm{KL}}(\,q\,||\,p\,)$

Which one is larger? Why?

$$\begin{aligned} \mathrm{D}_{\mathrm{KL}}(\,p\,||\,q\,) &= ½(2-1) + ¼(3-2) + ⅛(3-3) + ⅛(1-3) &= 0.5 \\ \mathrm{D}_{\mathrm{KL}}(\,q\,||\,p\,) &= ¼(1-2) + ⅛(2-3) + ⅛(3-3) + ½(3-1) &= 0.625 \end{aligned}$$

$\mathrm{D}_{\mathrm{KL}}(\,q\,||\,p\,)$ is larger, mainly because the frequency of D has increased from ⅛ to ½, so it incurs a cost of $3-1=2$ additional bits every time it occurs (which is often).

Question 4 *Submitted Sep 2nd 2022 at 7:38:59 pm*

One bag contains 2 red balls and 3 white balls. Another bag contains 3 red balls and 2 green balls. One of these bags is chosen at random, and two balls are drawn randomly from that bag, without replacement. Both of the balls turn out to be red. What is the probability that the first bag is the one that was chosen?

$1/4$

Explanation

Let B = first bag is chosen, R = both balls are red. Then

$$\begin{aligned} \mathrm{Prob}(R|B) &= (2/5) \times (1/4) &= 1/10 \\ \mathrm{Prob}(R|\neg B) &= (3/5) \times (2/4) &= 3/10 \\ \mathrm{Prob}(R) &= (1/2) \times (1/10) + (1/2) \times (3/10) &= 1/5 \\ \\ \mathrm{Prob}(B|R) &= \mathrm{Prob}(R|B) \times \mathrm{Prob}(B)/\mathrm{Prob}(R) \\ &= (1/10) \times (1/2)/(1/5) &= 1/4 \end{aligned}$$

Revision 2: Probability

This is a revision quiz to test your understanding of the material from Week 1 on Probability.

You must attempt to answer each question yourself, before looking at the sample answer.

Question 1 *Submitted Sep 2nd 2022 at 7:39:47 pm*

Write the formula for a Gaussian distribution with mean μ and standard deviation σ.

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Question 2 *Submitted Sep 2nd 2022 at 7:40:19 pm*

Write the formula for Bayes' Rule, in terms of a cause A and an effect B.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Question 3 *Submitted Sep 2nd 2022 at 7:40:50 pm*

Write the formula for the Entropy $H(p)$ of a continuous probability distribution $p()$.

$$H(p) = \int_{\theta} p(\theta)\,(-\log\ p(\theta))\, d\theta$$

Question 4 *Submitted Sep 2nd 2022 at 7:41:28 pm*

Write the formula for the Kullback-Leibler Divergence $D_{\mathrm{KL}}(p \mid\mid q)$ between two continuous probability distributions $p()$ and $q()$.

$$D_{KL}\left(p \mid\mid q\right) = \int_{\theta} p\left(\theta\right) \left(\log\ p\left(\theta\right) - \log\ q\left(\theta\right)\right) d\theta$$