3a: Cross Entropy, Softmax and Weight Decay

Week 3 Overview

This week, we will introduce cross entropy and softmax (which are used for classification tasks as alternatives to the sum squared error loss function) and weight decay. Along the way, we will need to introduce the mathematical concepts of Maximum Likelihood Estimation (MLE) and Maximum A Posteriori (MAP) Estimation. We will also learn how to analyze the hidden unit dynamics of neural networks.

Weekly learning outcomes

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

- explain and compute cross entropy, softmax
- explain weight decay
- analyze the geometry of hidden unit activations in neural networks

---

Cross Entropy and Softmax

Loss Functions

In Week 1 we introduced the **sum squared error** (**SSE**) loss function, which is suitable for function approximation tasks.

$$E = \frac{1}{2}\sum_i (t_i - z_i)^2$$

However, for **binary classification** tasks, where the target output is either zero or one, it may be more logical to use the **cross entropy** error:

$$E = \sum_i (-t_i \log(z_i) - (1 - t_i) \log(1 - z_i))$$

In order to explain the motivation for both of these loss functions, we need to introduce the mathematical concept of Maximum Likelihood.

Maximum Likelihood

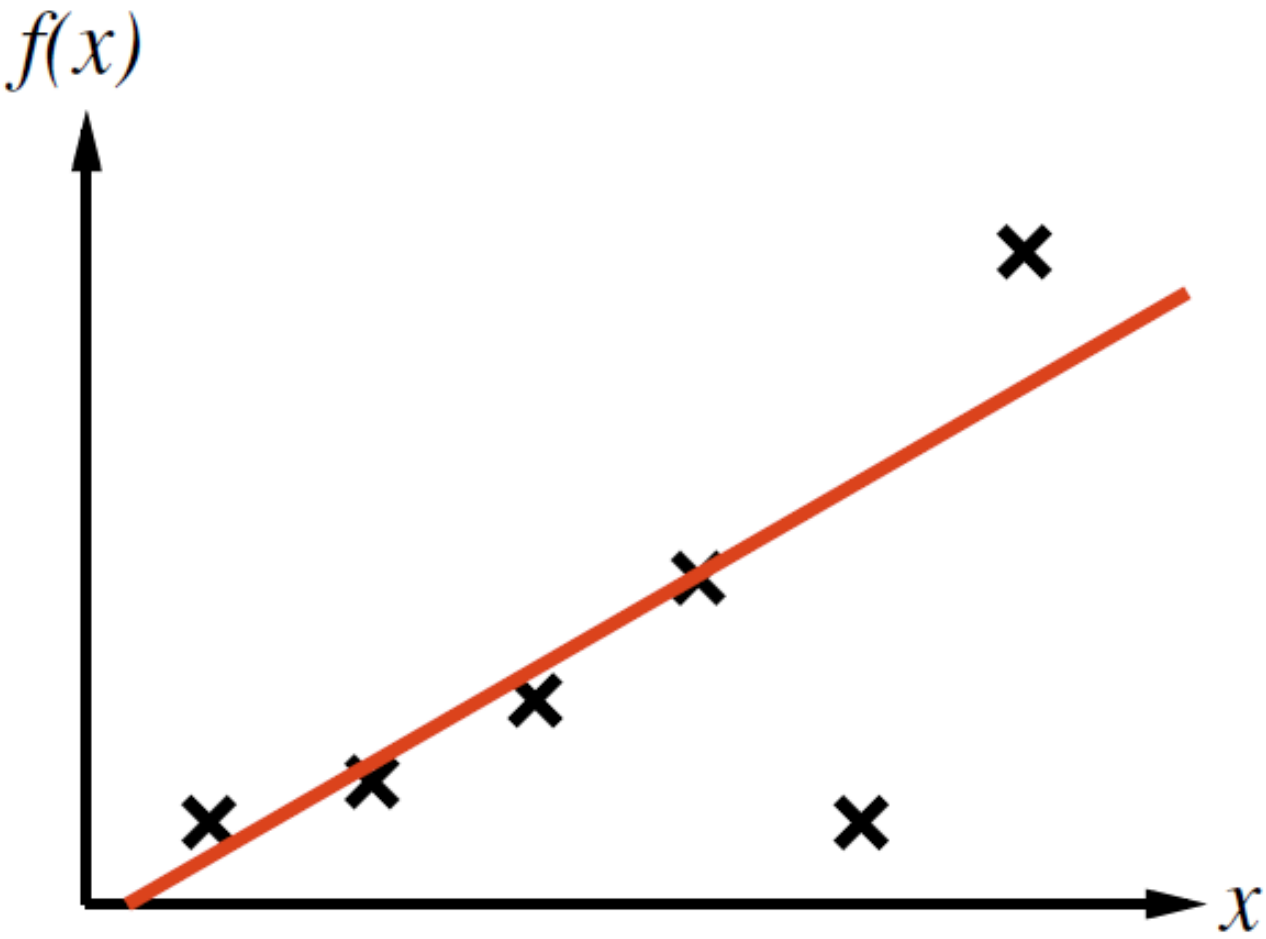
Let H be a class of **hypotheses** for predicting certain **data** D.

Let $\mathrm{Prob}(\mathrm{D}|h)$ be the probability of data D being generated under hypothesis $h \in \mathrm{H}$.

The logarithm of this probability, $\log \mathrm{Prob}(\mathrm{D}|h)$ is called the **likelihood**.

The **Maximum Likelihood Principle** states that we should choose $h \in \mathrm{H}$ which maximizes this likelihood, i.e. maximizes $\mathrm{Prob}(\mathrm{D}|h)$ or, equivalently, maximizes $\log \mathrm{Prob}(\mathrm{D}|h)$.

In our case, the data D consist of a target value t_i for each set of input features x_i in a Supervised Learning task, and we can think of each hypothesis h as a function $f()$ determined by a neural network with specified weights or, to give a simpler example, $f()$ could be a straight line with a specified slope and y-intercept.



Derivation of Least Squares

As previously mentioned, noise in the data is often caused by an accumulation of small errors due to factors which are not captured by the model. The Central Limit Theorem tells us that when a large number of independent random variables are added together, the combined error is well approximated by a Gaussian distribution.

In order to accommodate this kind of noise, let's suppose that our data $\mathrm{D} = \{t_i\}$ are generated by a linear function $f()$ plus noise generated from a Gaussian distribution with mean zero and standard deviation σ. Then

$$\begin{aligned}
\mathrm{Prob}(\,\mathrm{D} \mid h\,) = \mathrm{Prob}(\{t_i\} \mid f\,) &= \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(t_i - f(x_i))^2} \\
\log \mathrm{Prob}(\{t_i\} \mid f\,) &= \sum_i -\frac{1}{2\sigma^2}\left(t_i - f\left(x_i\right)\right)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi) \\
f_{ML} &= \mathrm{argmax}_{f \in \mathrm{H}} \log \mathrm{Prob}(t \mid f) \\
&= \mathrm{argmin}_{f \in \mathrm{H}} \frac{1}{2} \sum_i (t_i - f(x_i))^2
\end{aligned}$$

It is interesting to note that we do not need to know the value of σ.

Derivation of Cross Entropy

For binary classification tasks, the target value t_i is either 0 or 1. It makes sense to interpret the output $f(x_i)$ of the neural network as the **probability** of the true value being 1, i.e.

$$\begin{aligned}\text{Prob}(\,1 \mid f(x_i)) &= f(x_i)\\ \text{Prob}(\,0 \mid f(x_i)) &= 1 - f(x_i)\\ \text{Prob}(\,t_i \mid f(x_i)) &= f(x_i)^{t_i}(1 - f(x_i)^{1-t_i}\end{aligned}$$

Then

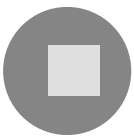
$$\log \text{Prob}(\{t_i\} \mid f) = \sum_i (\,t_i \log f(x_i) + (1 - t_i) \log(1 - f(x_i)))$$

So, according to the Maximum Likelihood principle, we need to maximize the expression on the right hand side (or minimize its negative).

Cross Entropy loss is often used in combination with sigmoid activation at the output node, which guarantees that the output is strictly between 0 and 1, and also makes the backprop computations a bit simpler, as follows:

$$\begin{aligned}E &= \sum_i (-t_i \log(z_i) - (1 - t_i) \log(1 - z_i))\\ \frac{\partial E}{\partial z_i} &= -\frac{t_i}{z_i} + \frac{1 - t_i}{1 - z_i} = \frac{z_i - t_i}{z_i(1 - z_i)}\\ \text{If} \quad z_i = \frac{1}{1 + e^{-s_i}}, \quad \frac{\partial E}{\partial s_i} &= \frac{\partial E}{\partial z_i}\frac{\partial z_i}{\partial s_i} = z_i - t_i\end{aligned}$$

SSE and cross entropy behave a bit differently when it comes to outliers. SSE is more likely to allow outliers in the training set to be misclassified, because the contribution to the loss function for each item is bounded between 0 and 1. Cross Entropy is more likely to keep outliers correctly classified, because the loss function grows logarithmically as the distance between the target and actual value approaches 1. For this reason, Cross Entropy works particularly well for classification tasks that are **unbalanced** in the sense of negative items vastly outnumbering positive ones (or vice versa).



Log Softmax

Some Supervised Learning tasks require data to be classified into more than two classes. If the number of classes is N and we have a neural network with N outputs $z_i, \ldots z_N$, we can make the assumption that the network's estimate for the probability of class j is proportional to $\exp(z_j)$.

Because the probabilities must add up to 1, we need to normalize by dividing by their sum:

$$\mathrm{Prob}(i) = \frac{\exp\left(z_i\right)}{\sum_{j=1}^{N} \exp\left(z_j\right)}$$

$$\log \mathrm{Prob}(i) = z_i - \log \sum_{j=1}^{N} \exp\left(z_j\right)$$

If the correct class is k, we can treat $-\log \mathrm{Prob}(k)$ as our loss function, and the gradient is

$$\frac{d}{dz_i} \log \mathrm{Prob}(k) = \delta_{ik} - \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp\left(z_j\right)} = \delta_{ik} - \mathrm{Prob}(i)$$

where δ_{ik} is the Kronecker delta. This gradient pushes up the correct class $i = k$ in proportion to the distance of its assigned probability from 1, and pushes down the incorrect classes $i \neq k$ in proportion to the probabilities assigned to them.

Softmax compared to Boltzmann Distribution and Sigmoid

If you have studied mathematics or physics, you may be interested to know that Softmax is related to the Boltzmann Distribution, with the negative of output z_i playing the role of the "energy" for "state" i. The familiar Sigmoid function can also be seen as a special case of Softmax, with two classes and one output, as follows: Consider a simplified case where there is a choice between two classes, Class 0 and Class 1. We consider the output z of the network to be associated with Class 1, and we imagine a fixed "output" for Class 0 which is always equal to zero. In this case, the softmax becomes:

$$\text{Prob}(1) = \frac{e^z}{e^z + e^0} = \frac{1}{1 + e^{-z}}$$

Further Reading

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

- Maximum Likelihood (5.5)
- Cross Entropy (3.13)
- Softmax (6.2.2)

Weight Decay

Sometimes a penalty term is added to the loss function which encourages the neural network weights w_j to remain small:

$$E = \frac{1}{2}\sum_i (z_i - t_i)^2 + \frac{\lambda}{2}\sum_j w_j^2$$

(Note: the sum squared error term may be replaced with cross entropy or softmax).

This additional loss term prevents the weights from "saturating" to very high values. It is sometimes referred to as "elastic weights" because it simulates a force on each weight as if there were a spring pulling it back towards the origin according to Hooke's Law. The scaling factor λ needs to be determined from experience, or empirically.

In order to explain the theoretical justification for Weight Decay, we need to introduce Baysian Inference and Maximum A Posteriori (MAP) estimation.

Bayesian Inference and MAP estimation

Recall the Maximum Likelihood principle which selects hypothesis $h \in \mathrm{H}$ for which $\mathrm{Prob}(\mathrm{D} \mid h)$ is maximal.

Bayesian Inference instead seeks to maximize $\mathrm{Prob}(h \mid \mathrm{D})$ i.e. the probability that hypothesis h is correct, given that data D have been observed.

According to Bayes' Theorem:

$$P(h \mid \mathrm{D})P(\mathrm{D}) = P(D \mid h)P(h)$$
$$P(h \mid D) = \frac{P(\mathrm{D} \mid h)P(h)}{P(D)}$$

We do **not** need to know $\mathrm{Prob}(\mathrm{D})$ in order to maximize this expression over h. However, we **do** need to be able to estimate $\mathrm{Prob}(h)$ which is called the **prior** probability of h (i.e. our estimate of the probability **before** the data have been observed). $\mathrm{Prob}(h \mid \mathrm{D})$ is called the **posterior** probability because it is our estimate **after** the data have been observed. For this reason, maximizing $\mathrm{Prob}(h \mid \mathrm{D})$ in this context is sometimes called Maximum A Posteriori or **MAP** estimation.

Note also that $\mathrm{Prob}(h)$ must be a well-defined probability in the sense that its integral over H must be finite and not infinite.

Weight Decay as MAP Estimation

We assume a Gaussian prior distribution for the weights, i.e.

$$
\begin{aligned}
P(w) &= \prod_j \frac{1}{\sqrt{2\pi}\sigma_0} e^{-w_j^2/2\sigma_0^2} \\
P(w \mid t) &= \frac{P(t \mid w)P(w)}{P(t)} = \frac{1}{P(t)} \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(z_i - t_i)^2} \prod_j \frac{1}{\sqrt{2\pi}\sigma_0} e^{-w_j^2/2\sigma_0^2} \\
\log P(w \mid t) &= -\frac{1}{2\sigma^2} \sum_i (z_i - t_i)^2 - \frac{1}{2\sigma_0^2} \sum_j w_j^2 + \text{ constant} \\
w_{\text{MAP}} &= \text{argmax}_{w \in H} \log P(w \mid t) \\
&= \text{argmin}_{w \in H} \left(\frac{1}{2} \sum_i (z_i - t_i)^2 + \frac{\lambda}{2} \sum_j w_j^2 \right)
\end{aligned}
$$

where $\lambda = \sigma^2 / \sigma_0^2$

Further Reading

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

- Weight Decay (5.2.2) and MAP Estimation (5.6.1)

Quiz 3: Backprop Variations

This is a Quiz to test your understanding of the material from Week 2 on Backprop Variations.

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

Question 1

Explain the difference between the following paradigms, in terms of what is presented to the system, and what it aims to achieve:

- Supervised Learning
- Reinforcement Learning
- Unsupervised Learning

No response

Question 2

Explain what is meant by Overfitting in neural networks, and list four different methods for avoiding it.

No response

Question 3

Explain how Dropout is used for neural networks, in both the training and testing phase.

No response

Question 4

Write the formulas for these Loss functions: Squared Error, Cross Entropy, Softmax, Weight Decay (remember to define any variables you use)

No response

Question 5

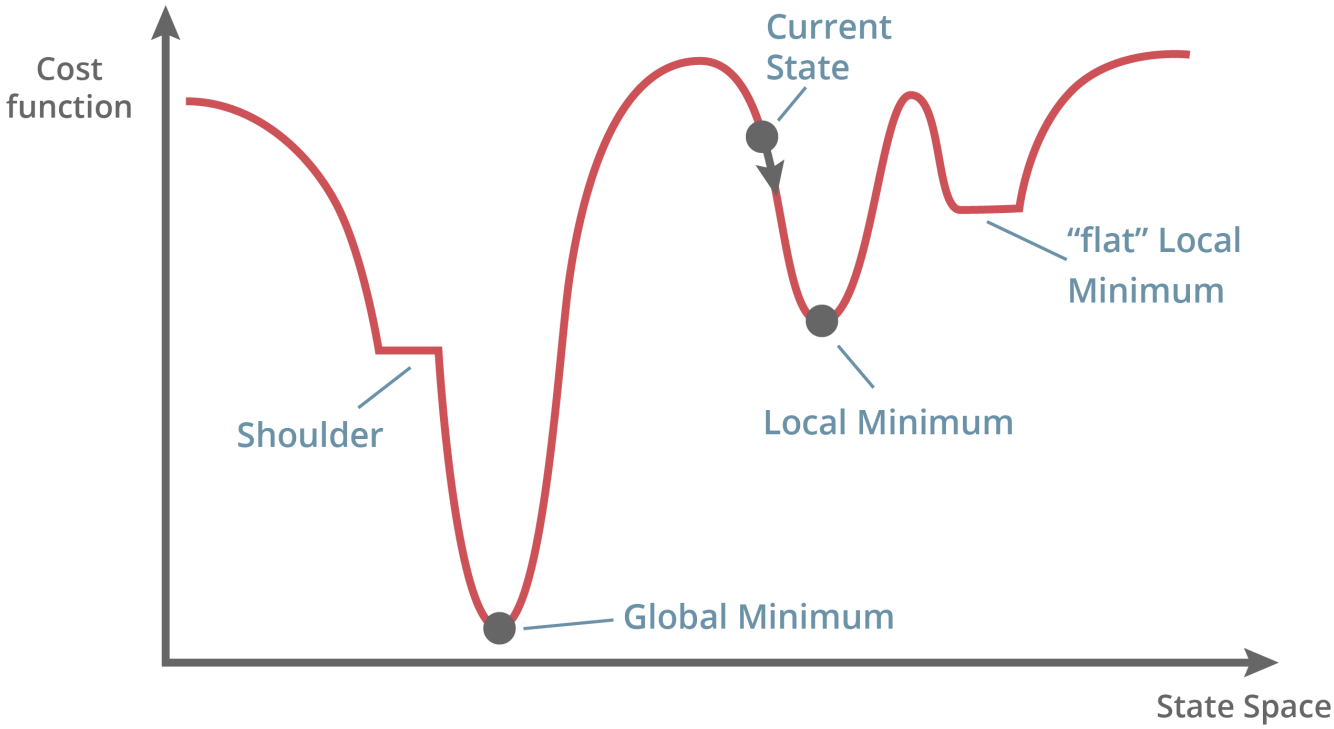
In the context of Supervised Learning, explain the difference between Maximum Likelihood estimation and Bayesian Inference.

No response

Coding Exercise - Gradient Descent with NumPy

Objective

In this exercise, you will learn how to use gradient descent to solve a linear regression problem.



Instructions

Complete the parts of the code marked as "**TODO".**

To run the cell you can press Ctrl-Enter or hit the "Play" button at the top.