CPU EDUCATION

INSPIRED

UNSW

COMP9444 期末班

期末班第二节课讲解

讲师：文杰学长　　**时长：2 小时**

本周内容预览：

1. Bayes 和 KL Divergence 相关公式复习
2. Cross Entropy 和 Soft Max 的复习
3. Gradient Vanishing 和 Exploding 的复习
4. 对应习题讲解
5. 往年真题练习

一、 Online, Batch and Minibatch Learning

Online, Batch and Minibatch Learning

You will notice that the loss function includes a summation over training items:

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

In some cases, the gradients for all training items are computed and added together, and this overall gradient is then used to update the weights in one step. This is known as **Batch Learning**. At the other extreme, we could compute the gradient for one item at a time as they are presented, and update the weights as we go, based on the gradient for each individual item. This is called **Online Learning**. To take advantage of parallel hardware, an intermediate approach known as **Minibatch Learning** is often employed, where the training items are divided randomly into minibatches of roughly equal size. The gradients are computed in parallel for all items in the minibatch, and the weights are updated accordingly. Because each minibatch represents only a portion of the full loss function, its gradient can be thought of as an approximation to the full gradient, with some implicit noise added. For this reason, online and minibatch learning are sometimes collectively referred to as **Stochastic Gradient Descent** (SGD).

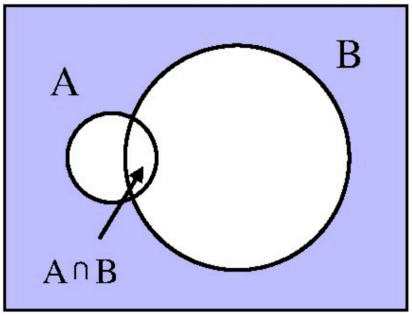
In order for learning to be successful, we should try to avoid **temporal correlations** in the training process. In other words, each minibatch (or a set of consecutively presented items in the case of online learning) should contain a variety of different inputs and corresponding outputs, rather than containing very similar inputs with the same target output.

二、 Probability

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

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

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:

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

Entropy and Huffmann Coding

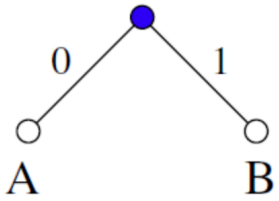
The **entropy** of a discrete probability distribution $p = \langle p_1, \ldots, p_n \rangle$ is

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

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

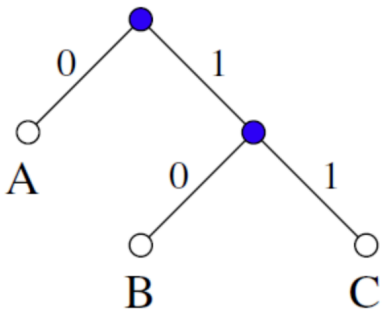
Example 1: $H(\langle 0.5, 0.5 \rangle) = 1 \text{ bit}.$

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: $H(\langle 0.5, 0.25, 0.25 \rangle) = 1.5 \text{ bits}.$

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.



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, $\mathrm{D}_{\mathrm{KL}}(p \parallel q)$ 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

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

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.

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).

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:

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

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

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

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

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.

1. **Softmax**

   Recall the formula for Softmax:

   $\text{Prob}(i) = \exp(z_i) / \Sigma_j \exp(z_j)$

   Consider a neural network being trained on a classification task with three classes $1, 2, 3$. When the network is presented with a particular input, the output values are:

   $z_1 = 1.0, z_2 = 2.0, z_3 = 3.0$

   Suppose the correct class for this input is $\text{Class } 2$. Compute the following, to two decimal places:

   a. $\text{Prob}(i)$, for $i = 1, 2, 3$

   b. $\text{d}(\log \text{Prob}(2))/\text{d}z_j$, for $j = 1, 2, 3$

三、Weight Decay

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

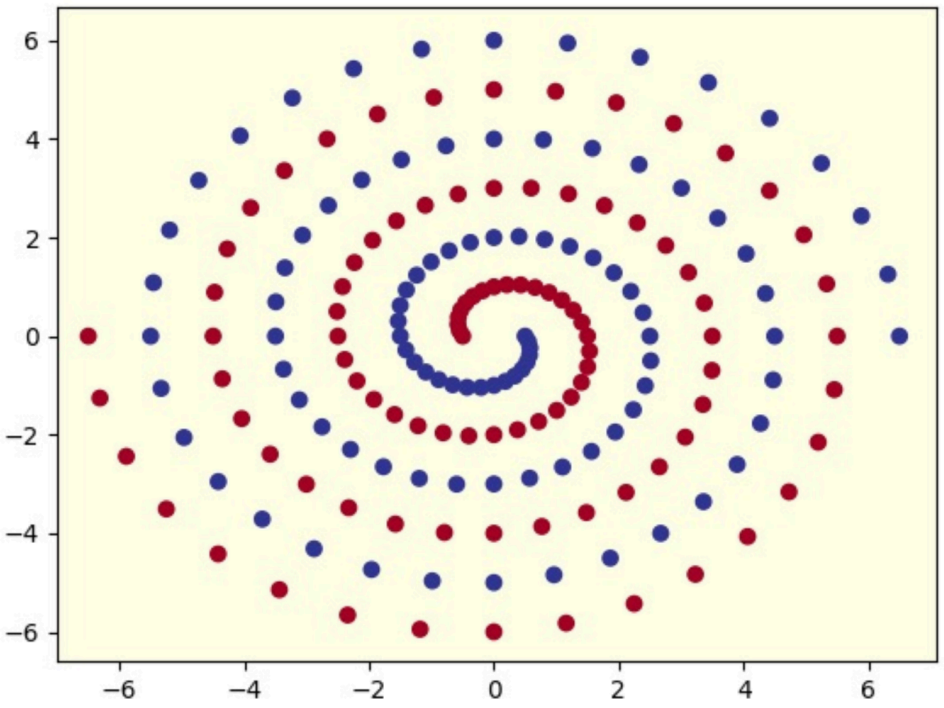
$$E = \frac{1}{2}\sum_i (z_i - t_i)^2 + \frac{\lambda}{2}\sum_j w_j^2$$

(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.

In general, neural networks with more hidden layers are able to implement a wider class of functions.



For example, this Twin Spirals problem cannot be learned with a 2-layer sigmoidal network, but it can be learned with a 3-layer network (Lang & Witbrock, 1988). The first hidden layer learns features that are linearly separable, the second hidden layer learns features that we could informally describe as "convex", and the third (output) layer learns the target function, which we could say has a more "concave" appearance.

It is tempting to think that any function could be learned, simply by adding more hidden layers to the network. However, it turns out that training networks with many hidden layers by backpropagation is not so easy, due to the problem of **Vanishing or Exploding Gradients**.

When the weights are small, the differentials become smaller and smaller as we backpropagate through the layers and end up having no effect. When the weights are large, the activations in the higher layers may saturate to extreme values. As a result, the gradients at those layers would become very small, and would not be propagated to the earlier layers. When the weights have intermediate values, the differentials can sometimes get amplified in places where the transfer function is steep, causing them to blow up to large values.

Dealing with Deep Networks

We will explore a number of enhancements which have been introduced in recent years to allow successful training of deeper networks; the main ones are summarised in this table:

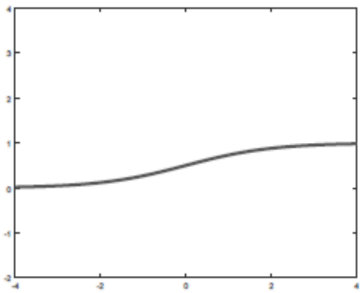
4 - 9 layers:	New Activation Functions (ReLU, SeLU)
10-30 layers:	Weight Initialization, Batch Normalisation
30-100 layers:	Skip Connections (Residual Networks)
more than 100 layers:	Identity Skip Connections, Dense Networks

Dealing with Deep Networks

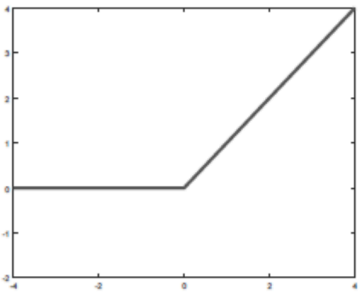
We will explore a number of enhancements which have been introduced in recent years to allow successful training of deeper networks; the main ones are summarised in this table:

4 - 9 layers:	New Activation Functions (ReLU, SeLU)
10-30 layers:	Weight Initialization, Batch Normalisation
30-100 layers:	Skip Connections (Residual Networks)
more than 100 layers:	Identity Skip Connections, Dense Networks

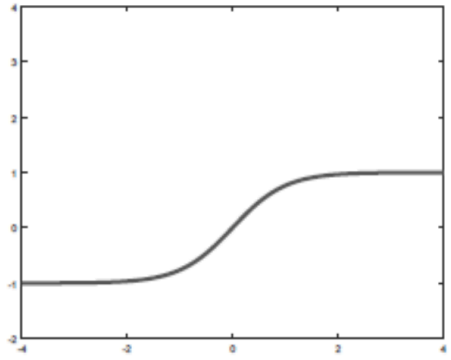
Activation Functions



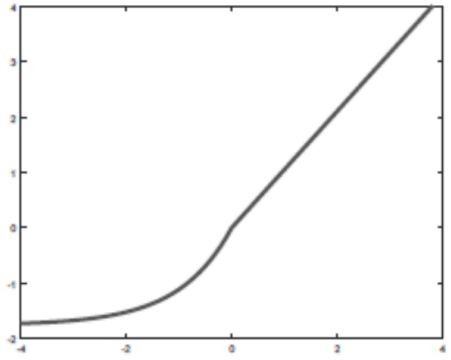
Sigmoid



Rectified Linear Unit (ReLU)



Hyperbolic Tangent



Scaled Exponential Linear Unit (SELU)

The sigmoid and hyperbolic tangent were traditionally used for 2-layer networks, but suffer from the vanishing gradient problem in deeper networks. Rectified Linear Units (ReLUs) have become popular since 2012 for deep networks, including convolutional networks. The gradients are multiplied by either 0 or 1 at each node (depending on the activation) and are therefore less likely to vanish or explode. Scaled Exponential Linear Units (SELUs) are a more recent innovation, which seem to also work well for deep networks.

3. **Hidden Unit Dynamics**

Consider a fully connected feedforward neural network with 6 inputs, 2 hidden units and 3 outputs, using tanh activation at the hidden units and sigmoid at the outputs. Suppose this network is trained on the following data, and that the training is successful.

Item	Inputs	Outputs
	123456	123
1.	100000	000
2.	010000	001
3.	001000	010
4.	000100	100
5.	000010	101
6.	000001	110

Draw a diagram showing:

a. for each input, a point in hidden unit space corresponding to that input, and

b. for each output, a line dividing the hidden unit space into regions for which the value of that output is greater/less than one half.

Question 22
Not yet answered
Marked out of 2.00
Flag question

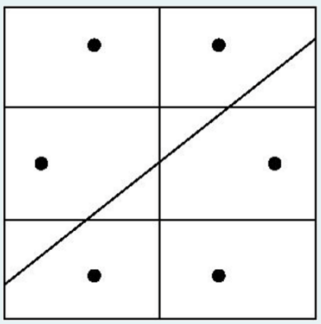
Consider a fully connected feedforward neural network with 6 inputs, 2 hidden units and 4 outputs, using tanh activation at the hidden units and sigmoid at the outputs. Suppose this network is trained on the following data, and that the training is successful.

Item	Inputs	Outputs
	123456	1234
1.	100000	0001
2.	010000	0011
3.	001000	0100
4.	000100	1010
5.	000010	1011
6.	000001	1110

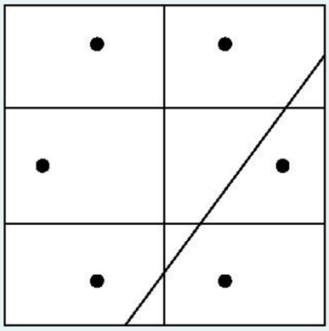
Which of these diagrams correctly shows a point in hidden unit space corresponding to each input, and, for each output, a line dividing the hidden unit space into regions for which the value of that output is greater/less than one half ?

Select one:

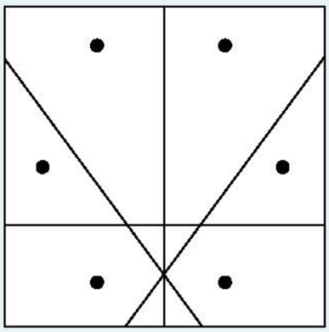
a.



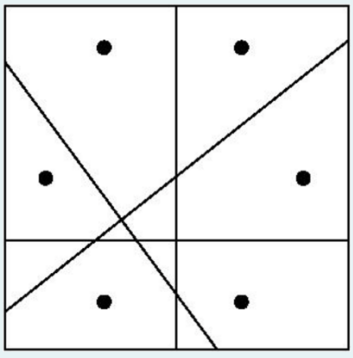
b.



c.



d.



Question **18**
Answer saved
Marked out of 3.00
Flag question

Consider a neural network trained using softmax for a classification task with three classes 1, 2, 3. Suppose a particular input is presented, producing outputs

z_1= 1.3, z_2= 2.1, z_3= 3.1

Assuming the correct class for this input is Class 1, and that Prob(1) is the softmax probability of the network choosing Class 1, compute the following, to two decimal places:

* d(log Prob(1))/dz_1= 0.89
* d(log Prob(1))/dz_2= -0.24
* d(log Prob(1))/dz_3= -0.65

Question **12**
Answer saved
Marked out of 1.00
Flag question

Which of these is NOT a method for dealing with the problem of vanishing or exploding gradients?

Select one:

- a. Batch Normalization
- b. Conjugate Gradients
- c. Weight Initialization
- d. Rectified Linear Unit

Question **15**
Answer saved
Marked out of 1.40
Flag question

Only 40% of the population have been vaccinated against a certain disease. Among those who **are** vaccinated, only 1% of them have the disease. But, among those who are **not** vaccinated, 2% of them have the disease.

If a random person is found to have the disease, what is the probability that they have been vaccinated?

(You can give your answer either as a percentage, or as a number between 0 and 1)

Question **16**
Answer saved
Marked out of 3.00
Flag question

Consider these two probability distributions on the same space Ω = {A, B, C, D, E}

p = ⟨ 1/16, 1/16, 1/8, 1/4, 1/2 ⟩
q = ⟨ 1/4, 1/16, 1/16, 1/2, 1/8 ⟩

Compute (correct to at least two decimal places):

* The Entropy H(p): 1.88
* The KL-Divergence $D_{KL}(p \,||\, q)$: 0.75

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

$$p = \langle \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \rangle, \quad q = \langle \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{2}, \frac{1}{16} \rangle$$

Compute, to two decimal places:

i. (1 mark) the Entropy $H(p)$:

ii. (2 marks) the KL-Divergence $D_{KL}(p \parallel q)$:

Part B: Question 20 (3 marks)

Consider a neural network trained using **softmax** for a classification task with three classes 1, 2, 3. Suppose a particular input is presented, producing outputs

$$z_1 = 2.5, z_2 = 1.5, z_3 = 0$$

If we assume that the correct class for this input is Class 2, and that Prob(2) is the softmax probability of the network choosing Class 2, compute the following (correct to two decimal places):

i. (1 mark) $d(\log \text{Prob}(2))/\, dz_1$:

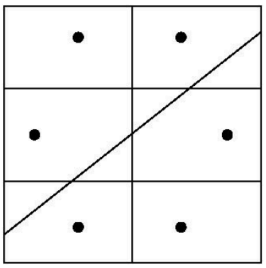
ii. (1 mark) $d(\log \text{Prob}(2))/\, dz_2$:

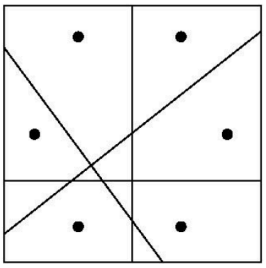
iii. (1 mark) $d(\log \text{Prob}(2))/\, dz_3$:

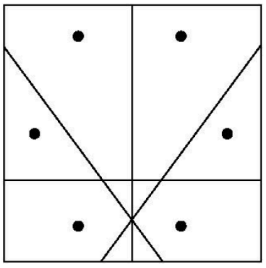
Consider a fully connected feedforward neural network with 6 inputs, 2 hidden units and 4 outputs, using tanh activation at the hidden units and sigmoid at the outputs. Suppose this network is trained on the following data, and that the training is successful.

Item	Inputs	Outputs
	123456	1234
1.	100000	0001
2.	010000	0011
3.	001000	0100
4.	000100	1010
5.	000010	1011
6.	000001	1110

Which of these diagrams correctly shows a point in hidden unit space corresponding to each input, and, for each output, a line dividing the hidden unit space into regions for which the value of that output is greater/less than one half ?

A. 

B. 

C. 

D.