姓名：李千鹏 学号：2021E8014682047 院系：人工智能学院

问题 1.

(1)

隐层到输出层的 BP

$$\begin{aligned}\Delta w_{hj} &= -\eta \cdot \frac{\partial J}{\partial w_{hj}} \\ &= -\eta \cdot \frac{\partial J}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{hj}} \\ &= -\eta \cdot y_h \cdot \frac{\partial J}{\partial net_j}\end{aligned}$$

$\frac{\partial J}{\partial net_j} = \sum_{m=1}^{c} \frac{\partial J}{\partial z_m} \cdot \frac{\partial z_m}{\partial net_j}$

当 $m \neq j$ 时，

$$\begin{aligned}\frac{\partial z_m}{\partial net_j} &= \frac{\partial \frac{e^{net_m}}{\Sigma + e^{net_j}}}{\partial net_j} \\ &= -\frac{e^{net_m}}{(\Sigma + e^{net_j})^2} \cdot e^{net_j} \\ &= -z_m \cdot z_j\end{aligned}$$

当 $m = j$ 时，

$$\begin{aligned}\frac{\partial z_j}{\partial net_j} &= \frac{\partial \frac{e^{net_j}}{\Sigma + e^{net_j}}}{\partial net_j} \\ &= \frac{\partial \left(1 - \frac{\Sigma}{\Sigma + e^{net_j}}\right)}{\partial net_j} \\ &= \frac{\Sigma \cdot e^{net_j}}{(\Sigma + e^{net_j})^2} \\ &= \left(1 - z_j\right) \cdot z_j\end{aligned}$$

由此可得，

$$\begin{aligned}\frac{\partial J}{\partial net_j} &= -(t_j - z_j) \cdot (1 - z_j) \cdot z_j + \sum_{m=1, m\neq j}^{c} (t_m - z_m) \cdot z_m \cdot z_j \\ &= -(t_j - z_j) \cdot z_j + \sum_{m=1}^{c} (t_m - z_m) \cdot z_m \cdot z_j\end{aligned}$$

$\Delta w_{hj} = \eta \cdot \delta_j \cdot y_h$

$\delta_j = (t_j - z_j) \cdot z_j - \sum_{m=1}^{c} (t_m - z_m) \cdot z_m z_j$

输入层到隐层的 BP

$$
\begin{aligned}
\Delta w_{ih} &= -\eta \cdot \frac{\partial J}{\partial w_{ih}} \\
&= -\eta \cdot \frac{\partial J}{\partial net_h} \cdot \frac{\partial net_h}{\partial w_{ih}} \\
&= -\eta \cdot \frac{\partial J}{\partial net_h} \cdot x_i
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J}{\partial net_h} &= \frac{\partial J}{\partial y_h} \cdot \frac{\partial y_h}{\partial net_h} \\
&= \sum_m^c \frac{\partial J}{\partial net_m} \cdot \frac{\partial net_m}{\partial y_h} \cdot (1 - y_h) \cdot y_h \\
&= \sum_m^c -\delta_m \cdot w_{hm} \cdot (1 - y_h) \cdot y_h
\end{aligned}
$$

由此可得,

$\Delta w_{ih} = \eta \cdot \sum_m^c \delta_m \cdot w_{hm} \cdot (1 - y_h) \cdot y_h \cdot x_i = \eta \cdot \delta_h \cdot x_i$

$\delta_h = \sum_m^c \delta_m \cdot w_{hm} \cdot (1 - y_h) \cdot y_h$

(2)

隐层与输出层之间的 Δw 表达式为:

$\Delta w_{hj} = \eta \cdot \delta_j \cdot y_h$

$\delta_j = (t_j - z_j) \cdot z_j - \sum_{m=1}^c (t_m - z_m) \cdot z_m z_j$

输入层与隐层之间的 Δw 表达式为:

$\Delta w_{ih} = \eta \cdot \sum_m^c \delta_m \cdot w_{hm} \cdot (1 - y_h) \cdot y_h \cdot x_i = \eta \cdot \delta_h \cdot x_i$

$\delta_h = \sum_m^c \delta_m \cdot w_{hm} \cdot (1 - y_h) \cdot y_h$

通过上述表达式，可以发现 Δw 均可以表示成 $\Delta w_{ij} = \eta \cdot \delta_j \cdot x_i$

x_i 为 w_{ij} 所连接边的起始节点值，δ_j 为 w_{ij} 所连接边指向结点收集到的加权误差和与激励函数对 net_j 导数的乘积。

问题 2.

设样本为 d 维，输出层有 c 个结点

(1) 计算步骤

1) 随机初始化参数

2) 输入样本并计算样本与输出结点权重的欧式距离平方 $d_c = \sum_{i=1}^{d}(x_i - w_{ic})^2$

3) 寻找到使得 d_c 最小的输出结点，记为 j^*，并找到它的邻接结点

4) 调整 j^* 和它的邻接结点 *(j)* 的权重，按如下公式进行更新

$\Delta w_{ij} = \eta h\left(j, j^*\right)\left(x_i - w_{ij}\right)$

$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$

$h(j, j*) = \exp\left(-\left\|j - j^*\right\|^2 / \sigma^2\right)$

5) 检查是否达到预先设定的要求，若没有，跳转至 *2)*；否则退出

(2) 算法框图

Algorithm 1 SOM Training Algorithm

1: 初始化 η，w，每个输出结点的邻接结点集合 N_j
2: do
3: 输入样本，计算样本与输出结点权重向量的欧氏距离平方 $d_j = \sum_{i=1}^{d}(x_i - w_{ij})^2$，$j = 1, 2, ..., c$
4: 寻找 $argmin(d_j)$，记为 j^*，更新 N_{j*}
5: 调整 j^* 和它的邻接结点 (j) 的权重，按如下公式进行更新
$\Delta w_{ij} = \eta h\left(j, j^*\right)\left(x_i - w_{ij}\right)$
$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$
$h(j, j*) = \exp\left(-\left\|j - j^*\right\|^2 / \sigma^2\right)$
$i = 1, 2, ..., d$，$j \in N_{j^*}$
6: 如果 $h(j, j^*) \leq h_{min}$，$return \quad w$，否则跳转到第 3 步

问题 3.

(1)

no padding, 卷积核 stride 为 1,pooling stride 为 2

由此第一层输出尺寸为 $396*396*20$, 经过激活层和池化层为 $198*198*20$。第二层输出尺寸为 $196*196*30$, 经过激活和池化后为 $98*98*30$。第三层输出尺寸为 $96*96*20$, 经过激活和池化为 $48*48*20$。第四层输出尺寸为 $46*46*10$, 经过激活和池化为 $23*23*10$。全连层输出为 10。

第一层参数: $20*5*5+20=520$

第二层参数: $30*3*3*20+30=5,430$

第三层参数: $20*3*3*30+20=5,420$

第四层参数: $10*3*3*20+10=1,810$

全连层参数: $23*23*10*10+10=52,910$

本网络以及全连接、参数不共享网络的参数对比如表 1

表 1: 参数对比表

	当前层权重数	全连与非权值共享	减少量
第一层卷积	520.00	501,814,336,320.00	501,814,335,800.00
第二层卷积	5,430.00	903,637,670,880.00	903,637,665,450.00
第三层卷积	5,420.00	53,106,462,720.00	53,106,457,300.00
第四层卷积	1,810.00	975,073,960.00	975,072,150.00
全连层	52,910.00	52,910.00	0.00

(2)

由于采用的是 $max \quad pooling$, 需要统计池化层每个元素对应上一层的位置。当误差传递到池化层时, 上一层标记位置的误差为池化层对应的误差, 其余位置的误差都为 0

假设某一池化层误差为 $\sigma_{pooling}=\begin{matrix}1 & 2\\ 3 & 4\end{matrix}$

如果上一层的输出为 $\begin{matrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 1 \\ 0 & 1 & 2 & 1 \\ 1 & 2 & 3 & 1 \end{matrix}$，那么上一层的误差就为 $\begin{matrix} 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 \end{matrix}$

(3)

可以将最后一层全连接改成多层全连接

激活函数使用 *tanh*、*ReLu*

添加 *batch normalization* 层

polling 采用平均池化或者最大池化

添加 *dropout* 层

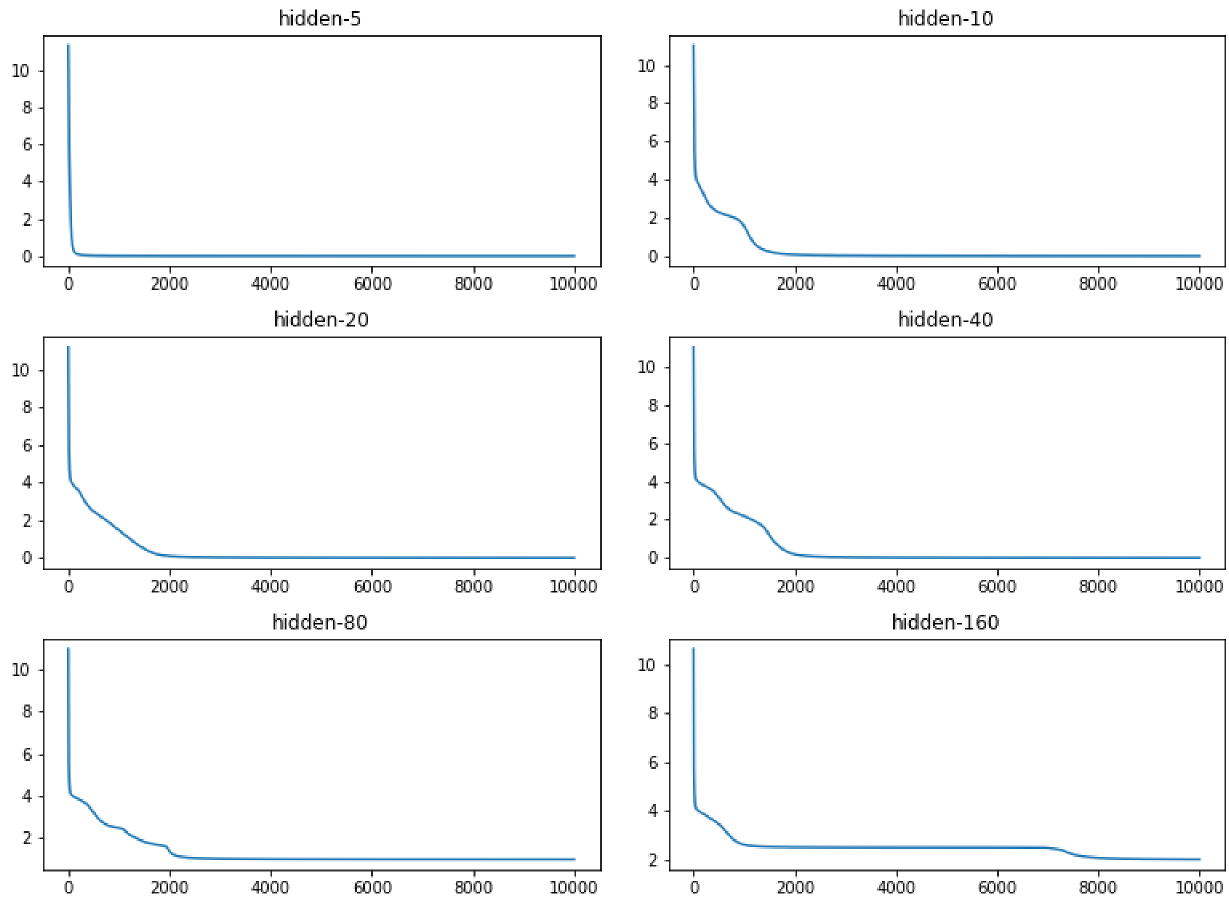
在上述操作的基础上，构建 *triplet network* 或者 *siamese network*

问题 4.

题目共给出 *30* 个数据，*batchsize* = 1 为单样本方式更新，*batchsize* = 30 为用所有数据进行更新权重，代码见 *PR4.ipynb*

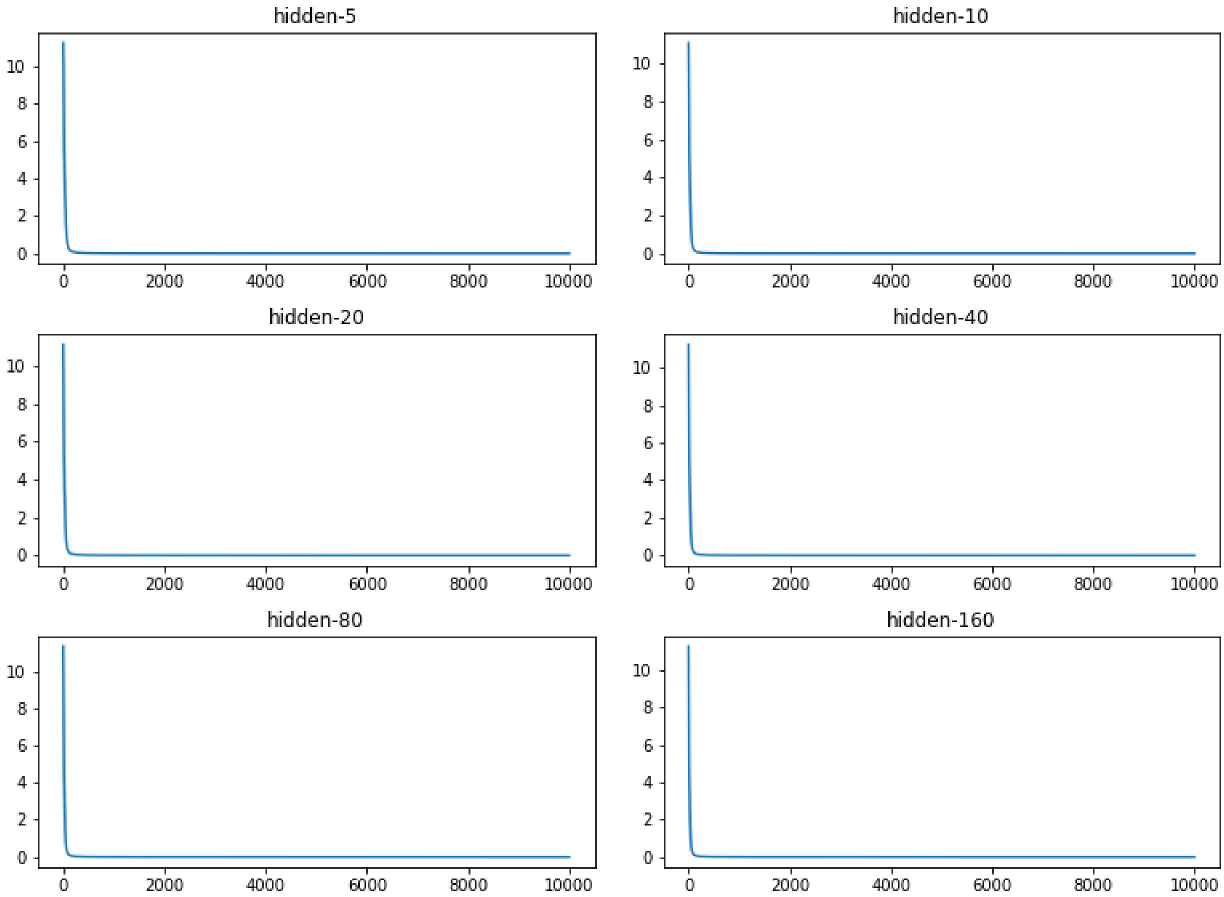
2.a

设置隐层结点分别为 *5*，*10*，*20*，*40*，*80*，*160*；学习率 *0.1*；最大 *epoch* 为 *10000*；下图分别为 *batchsize* 为 *1*、*30* 的所有数据 *loss* 累加和随 *epoch* 变化的曲线以及 *acc*



5_1： 1.0
10_1： 1.0
20_1： 1.0
40_1： 1.0
80_1： 0.9333333333333333
160_1： 0.8666666666666667

batchsize=1 的 *loss* 曲线以及 *acc*，*5—1* 代表 *5* 个隐层结点，*batchsize=1*



5_30： 1.0
10_30： 1.0
20_30： 1.0
40_30： 1.0
80_30： 1.0
160_30： 1.0

batchsize=30 的 *loss* 曲线以及 *acc*，*5—30* 代表 *5* 个隐层结点，*batchsize=30*

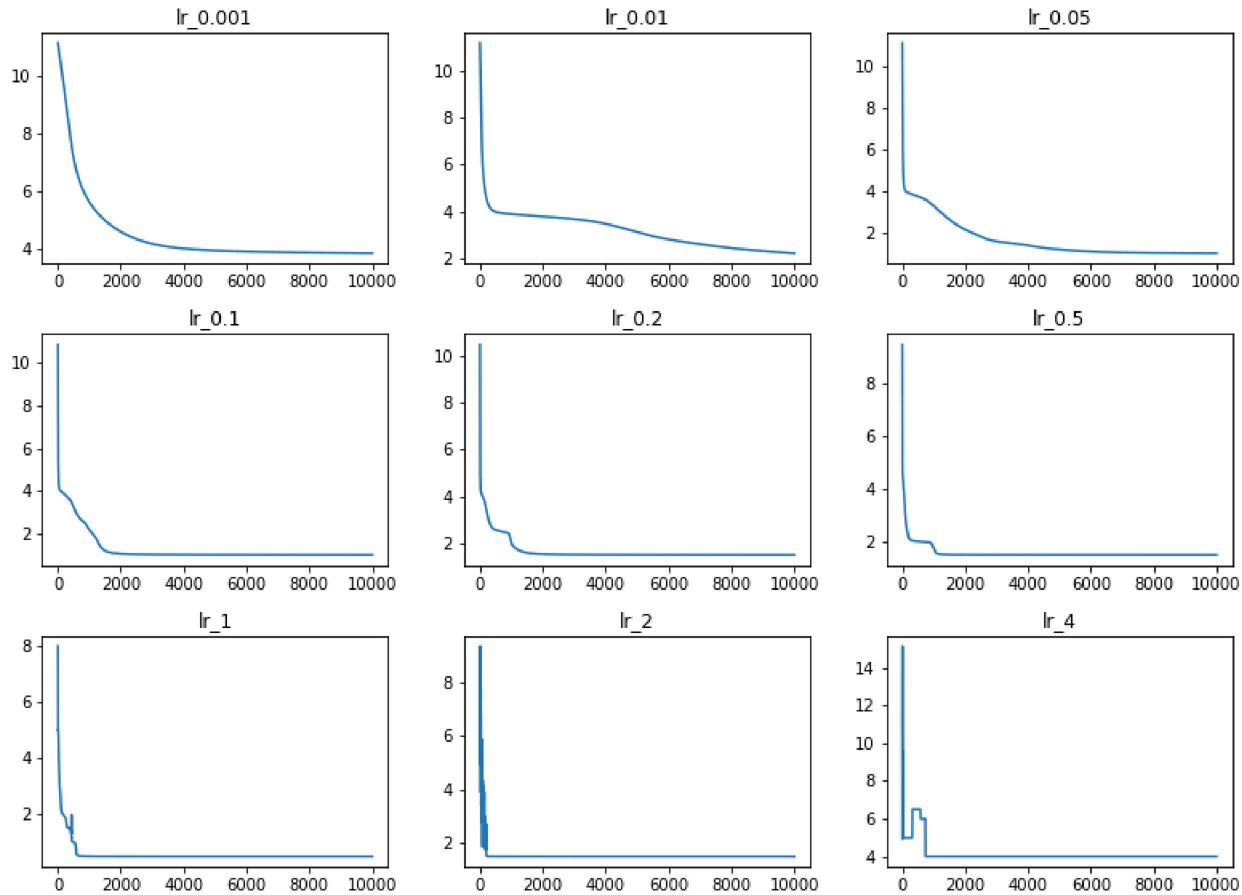
在单样本更新权重的 *loss* 曲线里，可以看到随着隐层结点数的增加，*loss* 曲线下降的越来越缓慢，并且 *loss* 值也越来越大，导致最终可能不收敛。而对于批处理更新权重的 *loss* 曲线，可以看到随着隐层结点数目的增加，*loss* 变化的非常快，很快就可以收敛。单样本更新和批处理更新展现了完全不同的结果，是因为批处理更新权重时考虑了 *batchsize* 个样本，而单样本更新只考虑了当前样本，导致更新后的网络适合当前样本但

不一定适合其余样本。

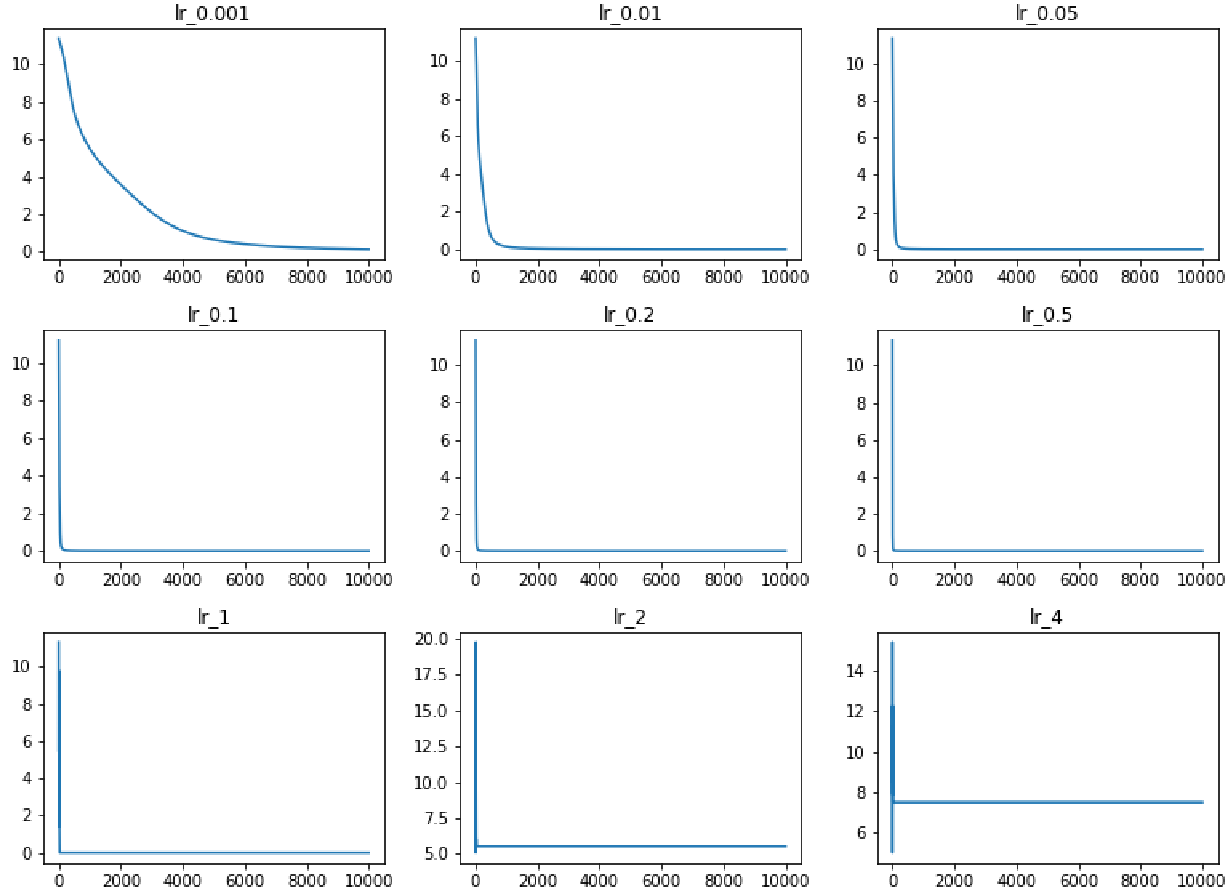
对于单样本更新而言，随着隐藏节点数增加，权重增多，使得每次更新时，权重的更新过分依赖当前样本，减缓收敛速度，最终可能导致不收敛。对于批处理而言，随着隐层结点数目的增加，由于更新考虑大量样本，使得更新权重后的网络能更好地拟合原来的样本，加快收敛。

2.b

设置学习率为 *0.001,0.01,0.05,0.1,0.2,0.5,1,2,4*；*batchsize* 为 *1*、*30*；隐层结点为 *80;epochmax* 为 *10000*；所有样本 *loss* 累加和随 *epoch* 曲线如图所示



batchsize=1 的 *loss* 曲线

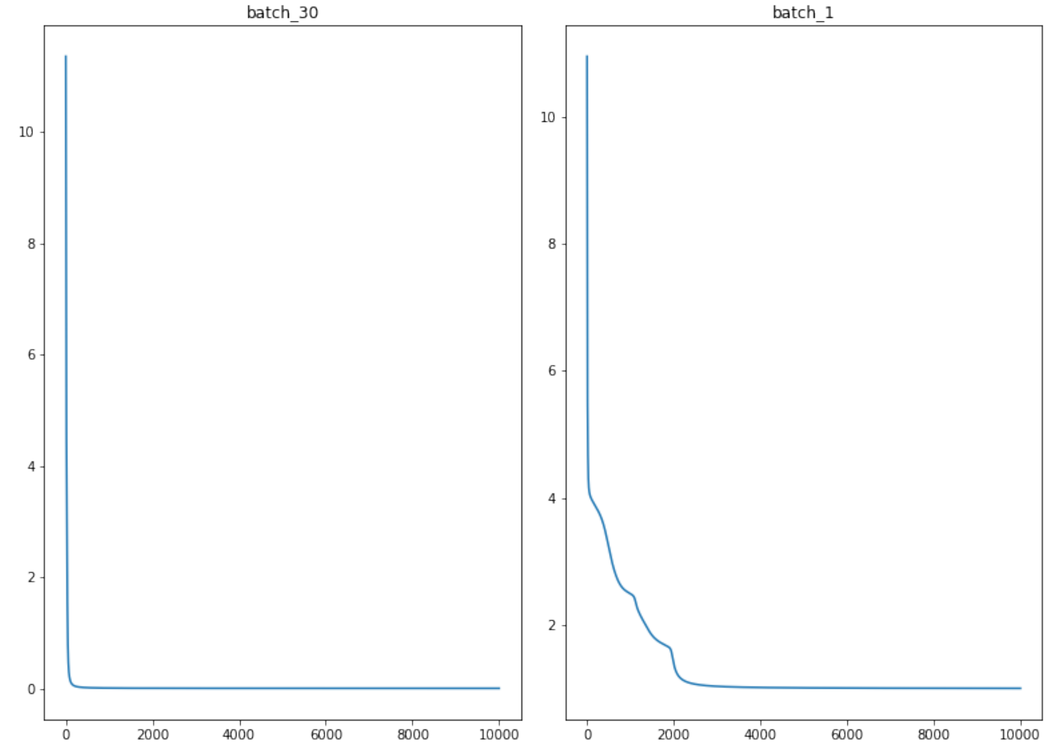


batchsize=30 的 *loss* 曲线

对于单样本和批处理更新权重而言，在有限的 *epoch* 条件下，当学习率从 *0.0001* 变化到 *4* 时，*loss* 变化速度先变快，后变慢。并且最终的 *loss* 先变小，后变大。这是因为当学习率比较小的时候，更新一次权重虽然能使得网络朝着 *loss* 最小的方向演化，但是变化的速度太慢，要想收敛需要增大 *epoch* 次数。随着学习率的增加，收敛的速度加快。但是当学习率增大到一定程度后，再增加学习率，就会使得权重的变化量过大，始终远离最优解，最终的 *loss* 也会变大。

2.c

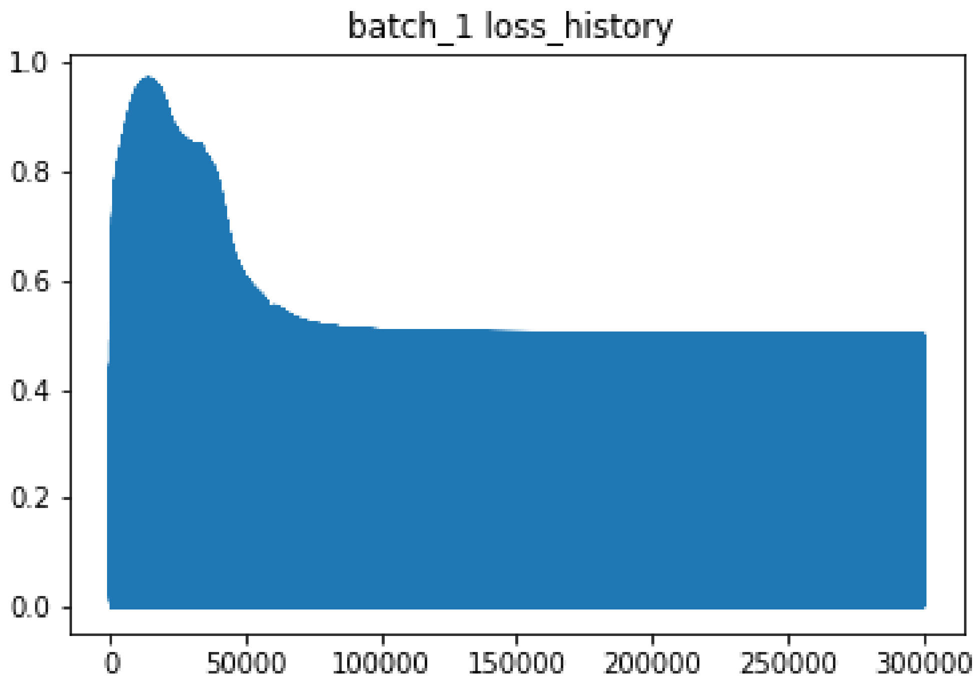
epoch 为 *10000*，*batchsize* 为 *1*、*30*，*lr* 为 *0.1*，隐层节点数为 *80*，所有样本 *loss* 累加和随 *epoch* 变化如下图



batchsize=1、*30* 的 *loss* 曲线

从 *loss* 曲线可以看到，当网络结构固定时，批处理更新权重能够很快将网络收敛，而单样本更新 *loss* 变化很慢并且不能够收敛。出现这样的原因在 *2.a* 中已经阐述，是因为批处理更新权重时考虑了 *batchsize* 个样本，而单样本更新只考虑了当前样本，导致更新后的网络适合当前样本但不一定适合其余样本。为了佐证我的想法，绘制出单样本每一次更新权重时的 *loss* 曲线 (单个样本的 *loss*)。

从下图中可以看到单样本更新时，每个样本对应的 *loss* 都不一样，有些可能很小，有些可能比较大。这正是因为前一次更新满足了上个样本，但是不一定能够满足其他的样本。而批处理更新考虑的样本比较多，不会出现这样的问题。



batchsize=1 单个样本的 *loss* 曲线