Task A1: handwritten digits recognition

2020212267 刘帅

---

0 - 准备之前

在我们处理之前，文件夹结构应按如下方式组织。

```
A1
├── A1_from_scratch.py  #利用numpy从头实现多层感知机网络
├── dataprepocessing.py #数据处理，将ubyte数据转换为可供numpy读取的npy文件
├── A1_torch.py         #pytorch implementation
|   ├── t10k-images.idx3-ubyte
|   ├── t10k-labels.idx1-ubyte
|   ├── train-images.idx3-ubyte
|   ├── train-labels.idx1-ubyte
|   ├── test_data.npy
|   ├── test_label.npy
|   ├── train_data.npy
|   ├── train_label.npy
```

0.1 - dataprepocessing.py

将ubyte数据转换为可供numpy读取的npy文件

```python
import numpy as np
import struct


def loadImageSet(filename):
    binfile = open(filename, 'rb')  # 读取二进制文件
    buffers = binfile.read()

    head = struct.unpack_from('>IIII', buffers, 0)  # 取前4个整数，返回一个元组

    offset = struct.calcsize('>IIII')  # 定位到data开始的位置
    imgNum = head[1]
    width = head[2]
    height = head[3]

    bits = imgNum * width * height  # data一共有60000*28*28个像素值
    bitsString = '>' + str(bits) + 'B'  # fmt格式：'>47040000B'

    imgs = struct.unpack_from(bitsString, buffers, offset)  # 取data数据，返回一个元
组

    binfile.close()
    imgs = np.reshape(imgs, [imgNum, width * height])  # reshape为[60000,784]型数
组

    return imgs, head
```

```python
def loadLabelSet(filename):
    binfile = open(filename, 'rb')  # 读二进制文件
    buffers = binfile.read()

    head = struct.unpack_from('>II', buffers, 0)  # 取label文件前2个整形数

    labelNum = head[1]
    offset = struct.calcsize('>II')  # 定位到label数据开始的位置

    numString = '>' + str(labelNum) + "B"  # fmt格式: '>60000B'
    labels = struct.unpack_from(numString, buffers, offset)  # 取label数据

    binfile.close()
    labels = np.reshape(labels, [labelNum])  # 转型为列表(一维数组)

    return labels, head


if __name__ == "__main__":
    file1 = './data/train-images.idx3-ubyte'
    file2 = './data/train-labels.idx1-ubyte'
    file3='./data/t10k-images.idx3-ubyte'
    file4='./data/t10k-labels.idx1-ubyte'


    imgs, data_head = loadImageSet(file1)
    labels, labels_head = loadLabelSet(file2)
    np.save("data/train_data.npy", imgs)
    np.save("data/train_label.npy", labels)

    imgs, data_head = loadImageSet(file3)
    labels, labels_head = loadLabelSet(file4)
    np.save("data/test_data.npy", imgs)
    np.save("data/test_label.npy", labels)
```

1 - 调用库&数据预处理

1.1 - numpy库、结果可视化及训练进度可视化

```python
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

1.2 - tensorboard

```python
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
```

1.3 - 转换成onehot编码

```
def one_hot(y):
    res = np.zeros([y.shape[0], 10])

    for i in range(y.shape[0]):
        res[i][y[i]] = 1
    return res
```

2 - 数据处理

```
def load_data():
    train_set_x_orig = np.load('./data/train_data.npy', encoding='bytes')  # 
train set features (60000,784)
    train_set_y_orig = np.load('./data/train_label.npy', encoding='bytes')  # 
train set labels (60000,1)
    test_set_x_orig = np.load('./data/test_data.npy', encoding='bytes')  # test 
set features (10000,784)
    test_set_y_orig = np.load('./data/test_label.npy', encoding='bytes')  # test 
set labels (10000,1)
    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig
```

3 - 多层感知机模型

3.1 - 线性层

对线性层进行kaiming normal初始化,对linear层调用call函数，从而使实例化对象可调用，后面可以进行layer的嵌套。

```
class Linear(object):#定义线性运算
    def __init__(self, n_in, n_out):
        self.input = None
        # kaiming normal
        self.W = np.random.normal(
            loc=0,  #均值
            scale=2 / n_in,  #标准差
            size=(n_in, n_out)) #输出的shape

        self.b = np.zeros(n_out, )

        self.dW = np.zeros(self.W.shape)
        self.db = np.zeros(self.b.shape)

        #optimizer的参数
        self.m_W = self.dW
        self.m_b = self.db
        self.v_W =self.dW
        self.v_b = self.db

    def forward(self, input):
        self.input = input
        lin_output = np.dot(input, self.W) + self.b  #x*W+b
        self.z = lin_output
```

```
        return self.Z

    __call__ = forward

    def backward(self, dZ, output_layer=False):#输出的轻微扰动，即（y-y_hat）
        self.dW = np.atleast_2d(self.input).T.dot(np.atleast_2d(dZ)) /
dZ.shape[0]
        self.db = np.mean(dZ)
        self.dA = dZ.dot(self.W.T)
        return self.dA
```

3.2 - 激活层

3.2.1 - ReLU()

```
class ReLU():
    def __init__(self):
        self.A = None
        self.Z = None

    def forward(self, Z):
        self.A = np.maximum(0,Z)
        assert(self.A.shape == Z.shape)
        self.Z = Z
        return self.A
    __call__ = forward

    def backward(self, dA):
        dZ = np.array(dA, copy=True) # just converting dz to a correct object.
        # When z <= 0, set dz to 0 as well.
        dZ[self.Z <= 0] = 0
        return dZ
```

3.2.2 - Softmax()

$$e^{Zi}/\Sigma_{j=1}^{len(t)}e^{z_j}$$

```
class Softmax():
    def forward(self, Z):
        t = np.sum(np.exp(Z), axis=1)
        AL = np.exp(Z) / t[:,np.newaxis]
        return AL
    __call__ = forward
```

3.3 - dropout()

```
class Dropout():
    def __init__(self, dropout_rate=0.5, is_training=True):
        dropout_rate = 1 - dropout_rate
        self.dropout_rate = dropout_rate
        self.is_training = is_training

    def forward(self, A):
        self.A = np.array(A, copy=True)
```

```
        if self.is_training:
            self.binary_scaled_mask = np.random.binomial(1, self.dropout_rate,
                                                         size=self.A.shape) /
self.dropout_rate
            #相当于一次trail中，留下概率为droout_rate
            self.A *= self.binary_scaled_mask
        return self.A

    __call__ = forward

    def backward(self, dA):
        dA *= self.binary_scaled_mask
        return dA
```

3.4 - Batch Normalization()

$\mu_B \leftarrow \frac{1}{m}\Sigma_{i=1}^{m}x_i$

$\sigma_B^2 \leftarrow \frac{1}{m}\Sigma_{i=1}^{m}(x_i-\mu_B)^2$

$\widehat{x}_i \leftarrow \frac{x_i-\mu_B}{\sqrt{(\sigma_B^2+\varepsilon)}}$

$y_i \leftarrow \gamma\widehat{x}_i+\beta$

查看pytorch源码，发现pytorch官方实现BN过程中加入了EMA 有:

$\mu_{EMA} \leftarrow \lambda\mu_{EMA}+(1-\lambda)\mu_B$

绘制出Batchnorm的计算图并进行反向传播 有:

$\frac{\partial L}{\partial\gamma}=\Sigma_{i=1}^{m}\frac{\partial L}{\partial y_i}*\widehat{xi}$

$\frac{\partial L}{\partial\beta}=\Sigma_{i=1}^{m}\frac{\partial L}{\partial y_i}$

$\frac{\partial L}{\partial\widehat{x}_i}=\frac{\partial L}{\partial y_i}*\gamma$

$\frac{\partial L}{\partial\sigma^2}=\Sigma_{i=1}^{m}\frac{\partial L}{\partial\widehat{x}_i}*(x_i-\mu)*\frac{-(\sigma^2+\epsilon)^{-3/2}}{2}$

$\frac{\partial L}{\partial\mu}=\Sigma_{i=1}^{m}\frac{\partial L}{\partial\widehat{x}_i}*\frac{-1}{\sqrt{\sigma^2+\epsilon}}+\frac{\partial L}{\partial\sigma^2}*\frac{\partial\sigma^2}{\partial\mu}$

综上，有:

$\frac{\partial L}{\partial x_i}=\frac{\partial L}{\partial\widehat{x}_i}*\frac{1}{\sqrt{\sigma^2+\epsilon}}+\frac{\partial L}{\partial\mu}*\frac{1}{m}+\frac{\partial L}{\partial\sigma^2}*\frac{2(x_i-\mu)}{m}$

```
class BN():

    def __init__(self, n_out, momentum_BN = 0.9):

        self.gamma = np.ones(n_out)
        self.beta = np.zeros(n_out)

        self.dgamma = np.zeros(self.gamma.shape)
        self.dbeta = np.zeros(self.beta.shape)
        self.m_gamma = np.zeros(self.gamma.shape)
        self.v_gamma = np.zeros(self.gamma.shape)
        self.m_beta = np.zeros(self.beta.shape)
        self.v_beta = np.zeros(self.beta.shape)
        self.momentum_BN = momentum_BN
```

```
        self.mean=0.
        self.var=0.
        self.mean_avg = self.mean
        self.var_avg = self.var
        self.epsilon = 1e-7

        self.is_training = True

    def forward(self, Z):
        self.Z = Z
        mean = Z.mean(axis=0)
        self.mean = mean #mu
        self.mean_avg = (1. - self.momentum_BN) * self.mean_avg +
self.momentum_BN * mean
        var = Z.var(axis=0)
        self.var = var #sigma
        self.var_avg = (1. - self.momentum_BN) * self.var_avg + self.momentum_BN
* var

        if self.is_training:
            self.Z_hat = (self.Z - mean) / np.sqrt(var + self.epsilon)
        else:
            self.Z_hat = (self.Z - self.mean_avg) / np.sqrt(self.var_avg +
self.epsilon)
        output = self.gamma * self.Z_hat + self.beta
        return output
    __call__ = forward


    def backward(self,dA):
        self.dgamma = np.sum(dA * self.Z_hat, axis = 0)
        self.dbeta = np.sum(dA, axis = 0)
        dZ_hat = dA * self.gamma
        dsigma = -0.5*np.power(self.var + self.epsilon, -1.5)*np.sum(dZ_hat*
(self.Z - self.mean),axis=0)
        dmu = -np.sum(dZ_hat/np.sqrt(self.var+ self.epsilon), axis=0) -
2*dsigma*np.sum(self.Z-self.mean,axis=0)/self.Z.shape[0]
        dA = dZ_hat/np.sqrt(self.var+self.epsilon) + 2.*dsigma*(self.Z-
self.mean)/self.Z.shape[0] + dmu/self.Z.shape[0]
        return dA
```

3.5 - 网络结构

3.5.1 MLP

```
class nn_MLP:
    def __init__(self, layers, activations, config):
        #config的两个参数分别表示dropout和BN层的超参
        self.layers = []
        for i in range(len(layers) - 1):
            self.layers.append(Linear(layers[i], layers[i + 1]))
            if config[1]:
                self.layers.append(BN(layers[i + 1], config[1]))
            if activations[i + 1].lower() == 'relu':
                act_layer = ReLU()
```

```
            elif activations[i + 1].lower() == 'softmax':
                act_layer = Softmax()
            self.layers.append(act_layer)
            if config[0]:
                self.layers.append(Dropout(config[0]))
        if config[0]:
            self.layers.pop(-1)
        # print(layers)

    def forward(self, input, mode):
        output = None
        for layer in self.layers:
            if not isinstance(layer, Linear):
                if mode == "test":
                    layer.is_training = False
                else:
                    layer.is_training = True
            output = layer(input)
            input = output
        AL = output
        return AL

    __call__ = forward
```

3.6 - 损失函数

CrossEntropyloss表示为

$$H(p,q) = -\Sigma_x(p(x)logq(x))$$

```
    class CrossEntropyLoss:
    def __init__(self, model):
        self.model = model
        return

    def loss(self, y, AL):

        self.y = one_hot(y)
        self.AL = AL
        id0 = range(y.shape[0])
        cost = -np.mean(np.sum((self.y * np.log(self.AL + 1e-6)), axis=1,
keepdims=True))
        cost = np.squeeze(cost)
        return cost

    def backward(self):
        dZ = self.AL - self.y
        d = dZ
        for layer in reversed(self.model.layers[:-1]):

            d = layer.backward(d)#倒着逆向求每层的backward
```

3.7 - 优化器

3.7.1 - SGD

同时引入动量momentum，防止sgd下降过程中出现局部最优。

```
class SGDOptimizer:
    def __init__(self, model, lr=0.001, momentum=0.9, weight_decay=0.01):
        self.lr = lr
        self.model = model
        self.momentum = momentum
        self.weight_decay = weight_decay

    def step(self):
        for layer in self.model.layers:
            if isinstance(layer, Linear):
                v_W = self.momentum * layer.v_W + self.lr * layer.dW
                v_b = self.momentum * layer.v_b + self.lr * layer.db
            #更新参数
                layer.W -= v_W
                layer.W -= self.lr * layer.W * self.weight_decay
                layer.b -= v_b

                layer.v_W = v_W
                layer.v_b = v_b

            elif isinstance(layer, BN):
                v_gamma = self.momentum * layer.v_gamma + self.lr * layer.dgamma
                v_beta = self.momentum * layer.v_beta + self.lr * layer.dbeta

                layer.gamma -= v_gamma
                layer.gamma -= self.lr * layer.gamma * self.weight_decay
                layer.beta = layer.beta - v_beta

                layer.v_gamma = v_gamma
                layer.v_beta = v_beta
```

3.7.2 - Adam

```
class Adam:
    def __init__(self, model, lr=0.001, betas=(0.9, 0.999), epsilon=1e-8):
        self.lr = lr
        self.model = model
        self.beta1 = betas[0]
        self.beta2 = betas[1]
        self.epsilon = epsilon
        self.iter = 0

    def step(self):
        for layer in self.model.layers:
            if isinstance(layer, Linear):
                self.iter += 1
                layer.m_W = (self.beta1 * layer.m_W + (1 - self.beta1) *
layer.dW)#一阶动量
```

```
                layer.v_W = (self.beta2 * layer.v_W + (1 - self.beta2) *
(layer.dW ** 2))#二阶动量
                m_hat_W = layer.m_W / (1 - self.beta1 ** self.iter)
                v_hat_W = layer.v_W / (1 - self.beta2 ** self.iter)
                layer.W -= (self.lr / (np.sqrt(v_hat_W + self.epsilon)) *
m_hat_W)#更新参数

                layer.m_b = (self.beta1 * layer.m_b + (1 - self.beta1) *
layer.db)
                layer.v_b = (self.beta2 * layer.v_b + (1 - self.beta2) *
(layer.db ** 2))
                m_hat_b = layer.m_b / (1 - self.beta1 ** self.iter)
                v_hat_b = layer.v_b / (1 - self.beta2 ** self.iter)
                layer.b -= (self.lr / (np.sqrt(v_hat_b + self.epsilon)) *
m_hat_b)
            elif isinstance(layer, BN):
                layer.m_gamma = (self.beta1 * layer.m_gamma + (1 - self.beta1) *
layer.dgamma)
                layer.v_gamma = (self.beta2 * layer.v_gamma + (1 - self.beta2) *
(layer.dgamma ** 2))
                m_hat_gamma = layer.m_gamma / (1 - self.beta1 ** self.iter)
                v_hat_gamma = layer.v_gamma / (1 - self.beta2 ** self.iter)
                layer.gamma -= (self.lr / (np.sqrt(v_hat_gamma + self.epsilon))
* m_hat_gamma)

                layer.m_beta = (self.beta1 * layer.m_beta + (1 - self.beta1) *
layer.dbeta)
                layer.v_beta = (self.beta2 * layer.v_beta + (1 - self.beta2) *
(layer.dbeta ** 2))
                m_hat_beta = layer.m_beta / (1 - self.beta1 ** self.iter)
                v_hat_beta = layer.v_beta / (1 - self.beta2 ** self.iter)
                layer.beta -= (self.lr / (np.sqrt(v_hat_beta + self.epsilon)) *
m_hat_beta)
```

4 - 训练过程

4.1 - 数据读取

```
train_x, train_y, test_x, test_y = load_data()
```

4.2 - 训练

```
def train_epoch(model, loss_fn, batch_size, epoch, opt, model_name):
    N = train_x.shape[0]
    mix_ids = np.random.permutation(N)  # mix data
    nbatches = int(np.ceil(float(N) / batch_size))
    loss = np.zeros(nbatches)
    for beg_i in tqdm(range(nbatches), desc="Epoch {} for model {}".format(epoch
+ 1, model_name)):
        # get the i-th batch
        batch_ids = mix_ids[batch_size * beg_i:min(batch_size * (beg_i + 1), N)]
        x_batch, y_batch = train_x[batch_ids], train_y[batch_ids]

        # forward pass
```

```
        y_hat = model(x_batch, "train")
        # backward pass
        loss[beg_i] = loss_fn.loss(y_batch, y_hat)
        loss_fn.backward()

        # update
        opt.step()
        # if beg_i % 100 == 0:
        #     print("Loss:{:7.4f} [{}/{}]".format(loss[beg_i], beg_i *
len(x_batch), train_x.shape[0]))
        #     # time.sleep(0.1)
    return loss
```

```
def test_val(model, loss_fn, batch_size, epoch):
    correct = 0
    loss = 0
    nbatches = int(np.ceil(float(test_x.shape[0]) / batch_size))
    losses = np.zeros(nbatches)
    ids = np.arange(test_x.shape[0])
    for it in range(nbatches):
        batch_ids = ids[batch_size * it:min(batch_size * (it + 1),
test_x.shape[0])]
        X_batch, y_batch = test_x[batch_ids], test_y[batch_ids]
        outputs = model(X_batch, "test")
        loss += loss_fn.loss(y_batch, outputs)
        losses[it] = loss_fn.loss(y_batch, outputs)
        correct += (outputs.argmax(1) ==
one_hot(y_batch).argmax(1)).astype(int).sum()
    loss /= nbatches

    print("Evaluation on testing set:\n      Accuracy:{:4.2f}%, Avg loss:
{:10.7f}".format(correct / test_x.shape[0] * 100, loss))

    return correct / test_x.shape[0] * 100, losses
```

4.3 - 参数设置

```
learning_rate = 1e-3
epochs = 25
batch_size = 32
```

4.4 - tensorboard写入和plt可视化

```
writer.add_scalars("num_layers/val_loss", {"1 layer": val_loss1.mean(), "2
layers": val_loss2.mean(), "3 layers": val_loss3.mean(), "5 layers":
val_loss4.mean()}, k)
    writer.add_scalars("num_layers/val_accuracy", {"1 layer": acc_val1, "2
layers": acc_val2, "3 layers":acc_val3, "5 layers": acc_val4}, k)

# flush the records
writer.flush()
writer.close()
```

```
print("Done!")

# %load_ext tensorboard
# %tensorboard --logdir=runs


# print losses
plt.figure(figsize=(6,3))
plt.title('Loss Plot')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(hidden_loss1, label="1 layer")
plt.plot(hidden_loss2, label="2 layers")
plt.plot(hidden_loss3, label="3 layers")
plt.plot(hidden_loss4, label="5 layers")
plt.legend(loc="upper left")
plt.grid()
plt.show()

# print accuracy
plt.figure(figsize=(6,3))
plt.title('Accuracy Plot')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.plot(hidden_acc1, label="1 layer")
plt.plot(hidden_acc2, label="2 layers")
plt.plot(hidden_acc3, label="3 layers")
plt.plot(hidden_acc4, label="5 layers")
plt.legend(loc="upper left")
plt.grid()
plt.show()
```

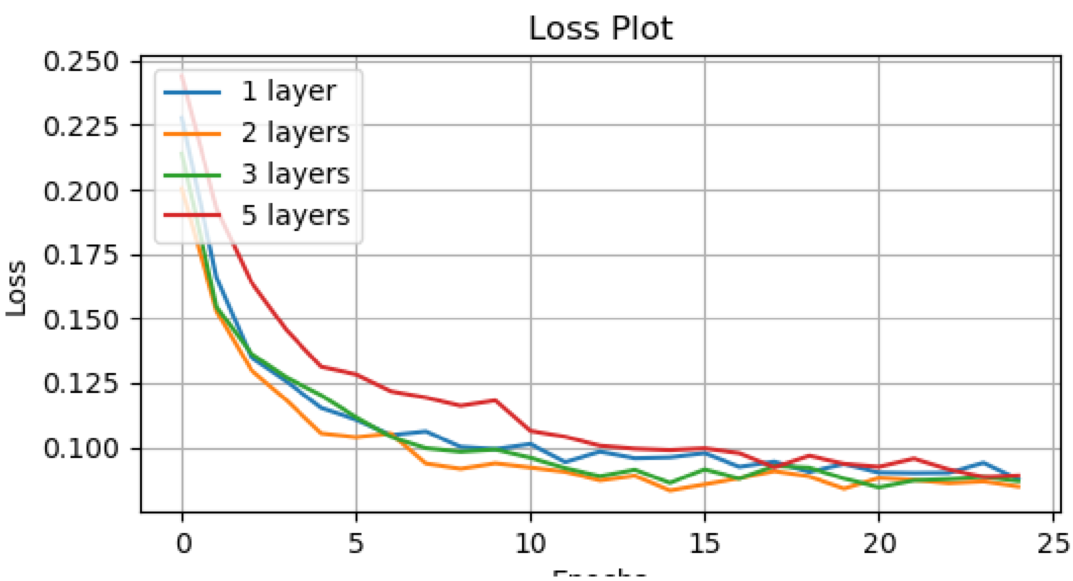
4.4 - 对比试验（只展示训练结果）

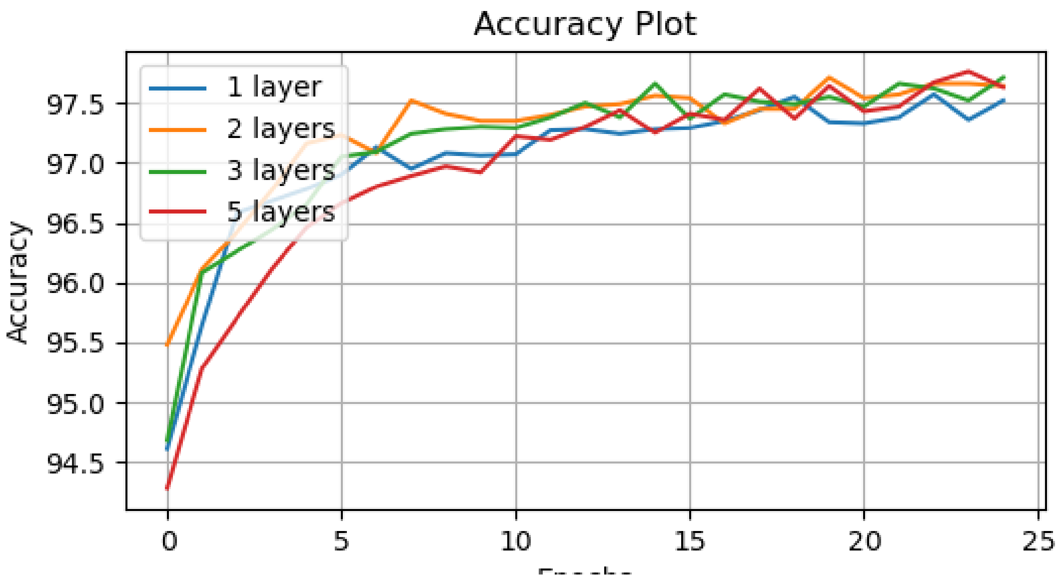
训练过程截图

```
Epoch 12 for model 1: 100%|██████████| 1875/1875 [00:05<00:00, 356.79it/s]
Evaluation on training set:
      Accuracy:98.58%, Avg loss: 0.0498986
Evaluation on testing set:
      Accuracy:97.27%, Avg loss: 0.0943006
Epoch 12 for model 2: 100%|██████████| 1875/1875 [00:06<00:00, 310.02it/s]
Evaluation on training set:
      Accuracy:98.67%, Avg loss: 0.0457131
Epoch 12 for model 3:   0%|          | 0/1875 [00:00<?, ?it/s]Evaluation on testing set:
      Accuracy:97.40%, Avg loss: 0.0905449
Epoch 12 for model 3: 100%|██████████| 1875/1875 [00:07<00:00, 240.60it/s]
Evaluation on training set:
      Accuracy:98.63%, Avg loss: 0.0467538
Evaluation on testing set:
      Accuracy:97.38%, Avg loss: 0.0919666
Epoch 12 for model 4: 100%|██████████| 1875/1875 [00:11<00:00, 158.72it/s]
Evaluation on training set:
      Accuracy:98.35%, Avg loss: 0.0580696
Epoch 13 for model 1:   0%|          | 0/1875 [00:00<?, ?it/s]Evaluation on testing set:
      Accuracy:97.19%, Avg loss: 0.1040984
Epoch 13 for model 1: 100%|██████████| 1875/1875 [00:04<00:00, 424.38it/s]
Evaluation on training set:
      Accuracy:98.60%, Avg loss: 0.0481795
Evaluation on testing set:
      Accuracy:97.28%, Avg loss: 0.0983180
```

由于numpy只支持在cpu上进行计算，因此为提高训练效率，将epochs设为3，（除了第一组）主要观察不同超参和不同结构下的训练结果。

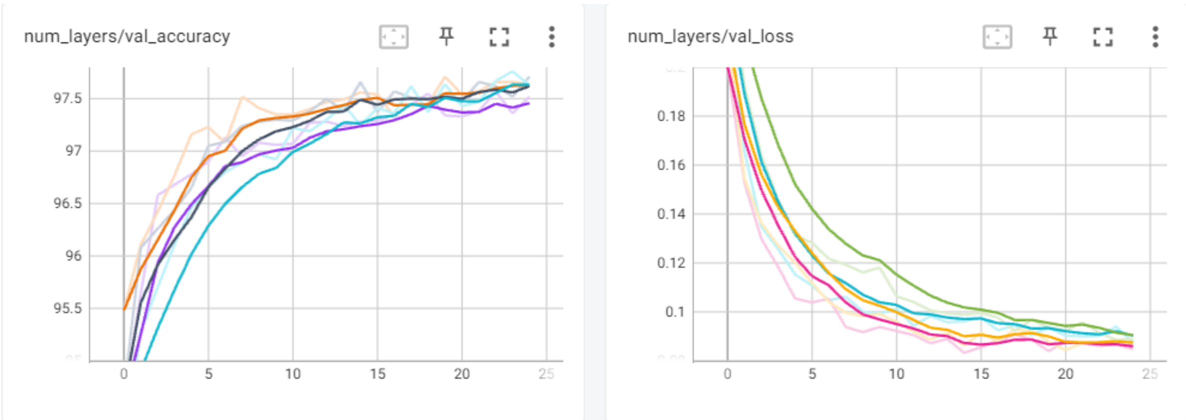
4.4.1 改变层数





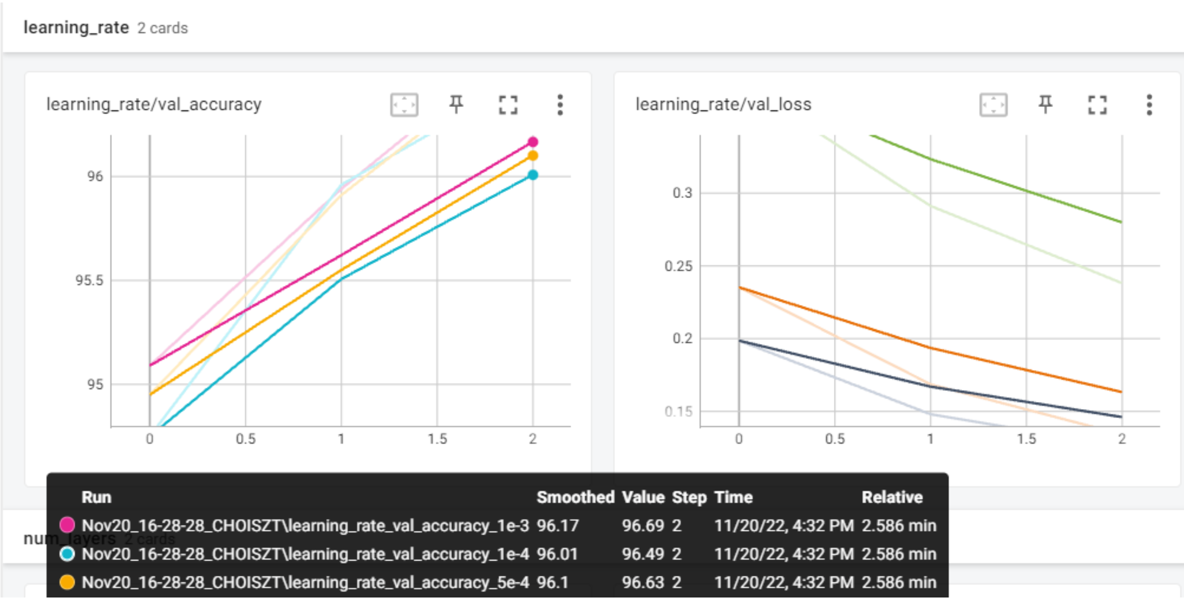
在tensorboard中进行可视化：

```
$ tensorboard --logdir=../runs --port 8561
TensorFlow installation not found - running with reduced feature set.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.11.0 at http://localhost:8561/ (Press CTRL+C to quit)
```

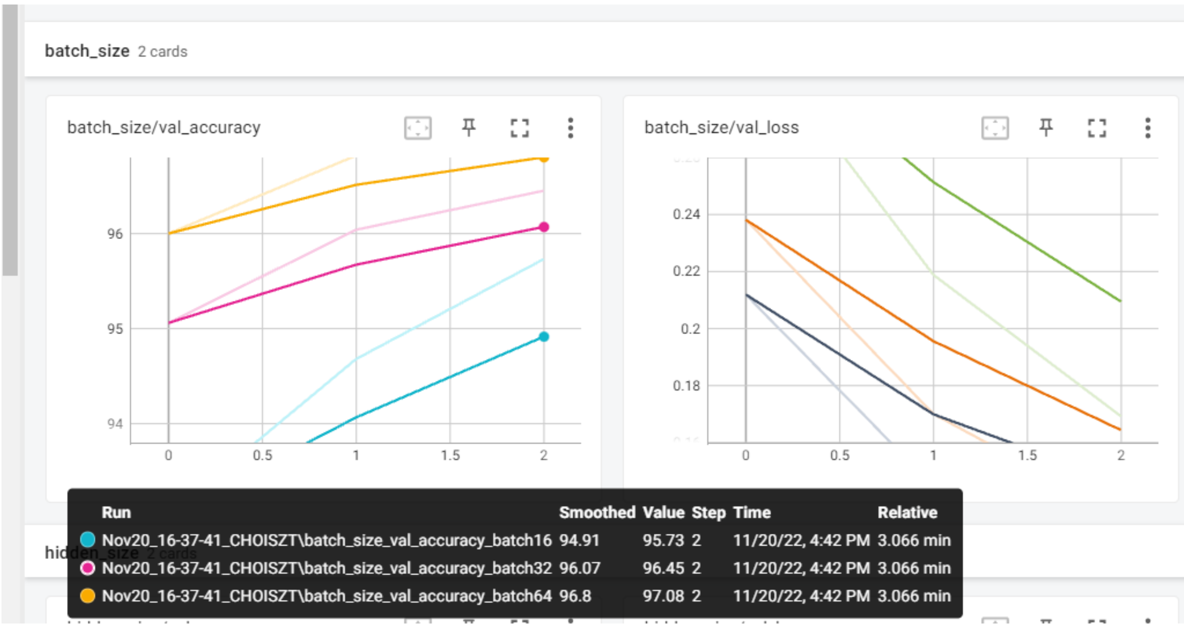


4.4.2 改变隐藏层神经元数

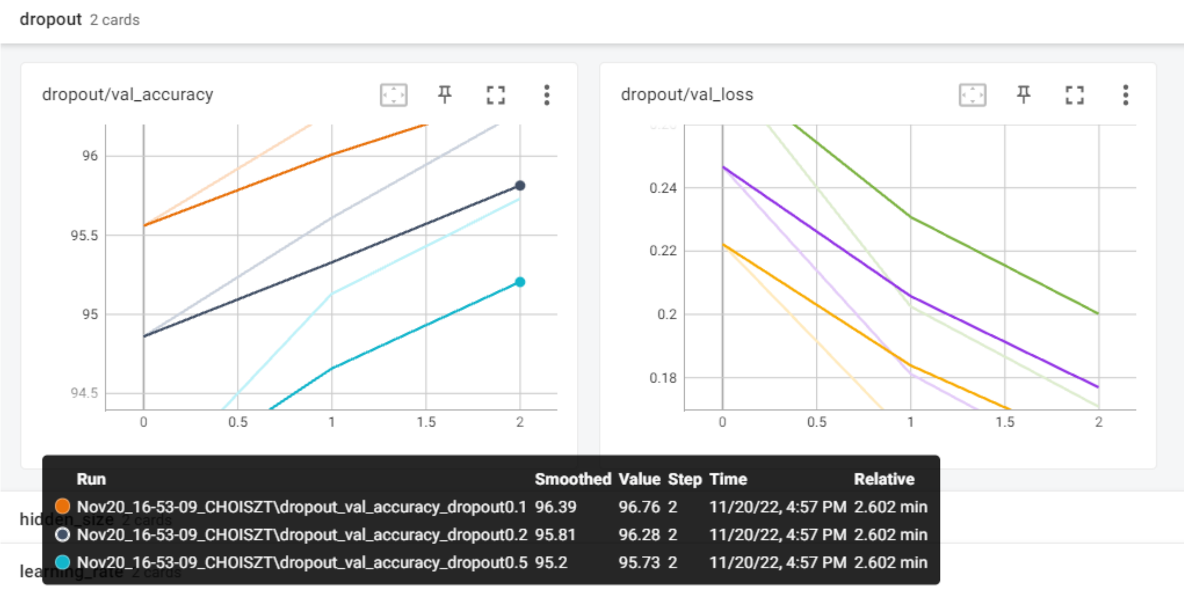
4.4.3 - 改变学习率



4.4.4 - 改变batchsize



4.4.5 - 改变dropout



4.4.6 - optimizer

4.5 消融实验