به نام خدا

درس یادگیری عمیق

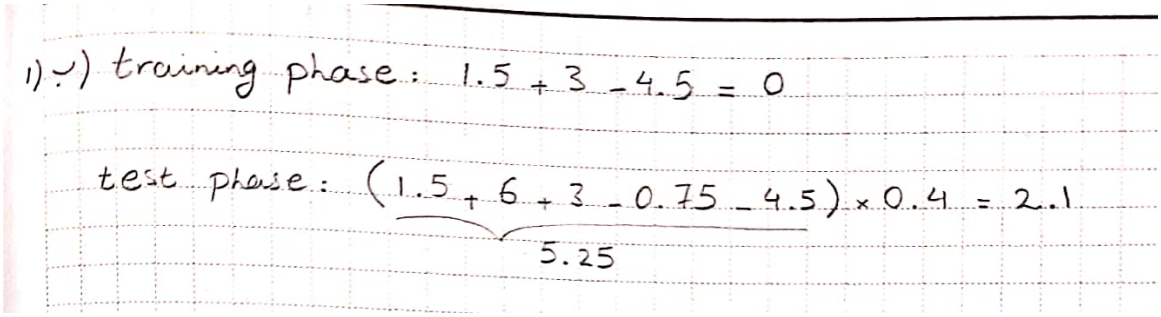
تمرین دوم

غزل زمانی نژاد

401722244

1. الف) با استفاده کردن از توابع فعال سازی میتوانیم non-linearity به شبکه اضافه کنیم. MLP بدون توابع فعال سازی، فرمول خطی دارد و استفاده کردن از چندین لایه خطی پشت هم بدون تابع فعال سازی، gain خاصی ندارد. علاوه بر آن رابطه خطی توانایی حل بسیاری از مسائل را ندارد ولی در صورت استفاده از تابع فعال سازی میتوانیم مسائل پیچیده ای حل کنیم.

ب)



ج) در اینجا یک مسئله دو کلاسه داریم. به طور کلی، تعداد نورون های مورد نیاز به میزان پیچیدگی دیتاست نیز بستگی دارد اما در حالت اول (اضافه کردن ۲ نورون) احتمالا مدل نتواند به خوبی ویژگی ها را استخراج کند و underfit می شود و حتی روی دیتاست آموزش نیز نمی تواند به دقت بالایی برسد. در حالت دوم (اضافه کردن ۲۵۶ نورون) مدل می تواند ویژگی های مناسبی از تصویر استخراج کند و به حالت best fit برسد. در حالت سوم (اضافه کردن ۱ میلیون نورون) این تعداد نورون برای دسته بندی یک مسئله باینری بسیار زیاد است و در صورت استفاده نکردن از روش های regularization، احتمالا دچار overfit می شویم. یعنی مدل روی داده های آموزش به دقت بالا (حتی ۱۰۰٪) خواهد رسید ولی نمی تواند روی داده آزمون به دقت بالایی برسد و generalization خوبی نخواهد داشت.

د) در یک شبکه عمیق در back propagation باید مشتق ها را در هم ضرب کنیم. اگر مقدار عددی مشتق ها بزرگ باشد شبکه دچار exploding gradients و اگر مشتق ها بسیار کوچک باشند دچار vanishing gradients خواهد شد. در موارد زیر ممکن است مدل دچار انفجار گرادیان شده باشد:

- مدل روی داده‌های آموزشی چیز زیادی یاد نگرفته بنابراین منجر به خطای ضعیف شود.
- به دلیل ناپایداری مدل ها در هر آپدیت، تغییرات زیادی در خطا داشته باشیم.
- خطای مدل ها در طول آموزش NaN شده باشد.

- وزن های مدل به صورت تصاعدی رشد کرده و هنگام آموزش مدل بسیار بزرگ شوند.
- مقدار مشتق ها ثابت بماند.

برای حل این مشکل می توانیم از راه های زیر استفاده کنیم:

- کاهش تعداد لایه های مدل: یکی از راه های جلوگیری از vanishing/exploding gradient کم کردن تعداد لایه های شبکه است اما با این کار پیچیدگی مدل کمتر می شود و ممکن است نتواند مسائل پیچیده را حل کند.
- Gradient clipping: کلیپ کردن مقدار گرادیان ها و کاهش اندازه آنها یکی از راه های جلوگیری از انفجار گرادیان است.
- مقداردهی اولیه وزن ها: بهتر است به جای اینکه در ابتدا وزن ها را به صورت تصادفی مقداردهی کنیم، از روش هایی مثل Xavier یا He استفاده کنیم.

2. الف) درست؛ استفاده از منظم سازی باعث کاهش پیچیدگی مدل می شود و ممکن است اگر بیش از حد از این روش ها استفاده شود، مدل نتواند به خوبی ویژگی ها را یاد بگیرد و underfit شود.

ب) غلط؛ هرچه تعداد ویژگی های داده بیشتر شود، پیچیدگی مدل در حین آموزش بیشتر خواهد شد و در نتیجه بیشتر منجر به overfit می شود.

ج) غلط؛ مطابق فرمول با زیاد کردن ضریب، مجموع loss بیشتر خواهد شد و مدل سعی میکند بیشتر مقدار خطا را کاهش دهد و ممکن است underfit شود.

3. .

3) الف)

$$H_3 = x_1 w_{13} + x_2 w_{23} = \underbrace{0.35 \times 0.1}_{0.035} + \underbrace{0.9 \times 0.8}_{0.72} = 0.755$$

$$y_3 = \sigma(0.755) = 0.680$$

$$H_4 = x_1 w_{14} + x_2 w_{24} = \underbrace{0.35 \times 0.4}_{0.14} + \underbrace{0.9 \times 0.6}_{0.72} = 0.86$$

$$y_4 = \sigma(H_4) = 0.702$$

$$O_5 = y_3 w_{35} + y_4 w_{45} = 0.68 \times 0.3 + 0.702 \times 0.9 = 0.835$$

$$y_5 = \sigma(O_5) = 0.697$$

(ب)

ب) Consider using MSE loss,

$$E = \sum \frac{1}{2}(y - y_5)^2$$

$$= \frac{1}{2}(0.5 - 0.697)^2 = 0.019$$

$$\frac{\partial E}{\partial w_{35}} = \frac{\partial E}{\partial y_5} \cdot \frac{\partial y_5}{\partial o_5} \cdot \frac{\partial o_5}{\partial w_{35}} = -(y - y_5) \cdot \sigma(o_5)(1 - \sigma(o_5)) \cdot y_3$$

$$= -(0.5 - 0.697) \times 0.697(1 - 0.697) \times 0.68 = \boxed{0.028}$$

$$\frac{\partial E}{\partial w_{45}} = \frac{\partial E}{\partial y_5} \cdot \frac{\partial y_5}{\partial o_5} \cdot \frac{\partial o_5}{\partial w_{45}} = -(y - y_5) \cdot \sigma(o_5)(1 - \sigma(o_5)) \cdot y_4$$

$$= -(0.5 - 0.697) \times 0.697 \times (1 - 0.697) \times 0.702 = \boxed{0.029}$$

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial o_5} \cdot \frac{\partial o_5}{\partial y_3} \cdot \frac{\partial y_3}{\partial H_3} \cdot \frac{\partial H_3}{\partial w_{13}} = \frac{\partial E}{\partial o_5} \cdot w_{35} \cdot \sigma(H_3)(1 - \sigma(H_3)) \cdot x_1$$

$\left(\frac{\partial E}{\partial o_5} = 0.041\right)$

$$= 0.041 \times 0.3 \times 0.68 \times (1 - 0.68) \times 0.35 = 0.0009$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial o_5} \cdot \frac{\partial o_5}{\partial y_3} \cdot \frac{\partial y_3}{\partial H_3} \cdot \frac{\partial H_3}{\partial w_{23}} = \frac{\partial E}{\partial o_5} \cdot w_{35} \cdot \sigma(H_3)(1 - \sigma(H_3)) \cdot x_2$$

$$= 0.041 \times 0.3 \times 0.68(1 - 0.68) \times 0.9 = 0.0024$$

$$\frac{\partial E}{\partial w_{14}} = \frac{\partial E}{\partial o_5} \cdot \frac{\partial o_5}{\partial y_4} \cdot \frac{\partial y_4}{\partial H_4} \cdot \frac{\partial H_4}{\partial w_{14}} = \frac{\partial E}{\partial o_5} \cdot w_{45} \cdot G(H_4)(1 - G(H_4)) \cdot x_1$$

$$= 0.041 \times 0.9 \times 0.702(1 - 0.702) \times 0.35 = 0.0027$$

$$\frac{\partial E}{\partial w_{24}} = \frac{\partial E}{\partial o_5} \cdot \frac{\partial o_5}{\partial y_4} \cdot \frac{\partial y_4}{\partial H_4} \cdot \frac{\partial H_4}{\partial w_{24}} = \frac{\partial E}{\partial o_5} \cdot w_{45} \cdot G(H_4)(1 - G(H_4)) \cdot x_2$$

$$= 0.041 \times 0.9 \times 0.702(1 - 0.702) \times 0.9 = 0.0069$$

$$w_i \leftarrow w_i - \alpha \cdot \frac{\partial E}{\partial w_i} \qquad \alpha = 1$$

$$w_{13} = 0.1 - 0.0009 = \boxed{0.0991}$$

$$w_{14} = 0.4 - 0.0027 = \boxed{0.3973}$$

$$w_{23} = 0.8 - 0.0024 = \boxed{0.7976}$$

$$w_{24} = 0.6 - 0.0069 = \boxed{0.5931}$$

$$w_{35} = 0.3 - 0.028 = \boxed{0.272}$$

$$w_{45} = 0.9 - 0.029 = \boxed{0.871}$$

.4

ابتدا داده MNIST را لود می کنیم و با با استفاده از data loader، داده را به batchهای ۶۴تایی تقسیم می کنیم. تعدادی از داده ها را به همراه برچسب شان چاپ می کنیم. سپس یک داده را با جزئیات بیشتر چاپ می کنیم.

در بخش بعد یک شبکه با ۲ لایه مخفی (اولی دارای ۵۱۲ و دومی دارای ۱۲۸ نورون) می سازیم. در لایه آخر هم ۱۰ نورون برای دسته بندی قرار می دهیم. از لایه dropout با نرخ ۰.۲ نیز استفاده می کنیم تا از overfit شدن مدل جلوگیری کنیم.

برحسب ورودی activation، نوع تابع فعال سازی را مشخص می کنیم. در اینجا تابع فعال سازی می تواند یکی از sigmoid، Relu یا tanh باشد.

در تابع forward، در صورتی که تابع ضرر negative log likelihood باشد، در لایه آخر log softmax را اعمال می کنیم. و در صورتی که تابع ضرر cross entropy باشد نیازی به استفاده از softmax نیست (مطابق pytroch documentation)

```python
import torch.nn as nn
import torch.nn.functional as F

## TODO: Define the NN architecture
# different activation functions: relu, sigmoid, tanh

class Net(nn.Module):
    def __init__(self, in_features=28*28, hidden=[512, 128], num_classes=10, activation='relu', criterion='cross-entropy'):
        super().__init__()
        self.criterion = criterion

        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(in_features, hidden[0])

        if activation == 'sigmoid':
            self.activation = nn.Sigmoid()
        elif activation == 'tanh':
            self.activation = nn.Tanh()
        else: self.activation = nn.ReLU()

        self.linear2 = nn.Linear(hidden[0], hidden[1])
        self.dropout = nn.Dropout(0.2)
        self.linear3 = nn.Linear(hidden[1], num_classes)

    def forward(self, x):
        out = self.flatten(x)
        out = self.activation(self.linear1(out))
        out = self.dropout(out)
        out = self.activation(self.linear2(out))
        out = self.dropout(out)
        out = self.linear3(out)

        # apply logsoftmax when using nll as loss
        if self.criterion == 'nll':
            out = F.log_softmax(out, dim=-1)

        return out
```

در قسمت بعدی تابع train را پیاده سازی می کنیم. مقدار دیفالت نرخ آموزش ۰.۰۱ و تعداد ایپاک ها را

۱۵ قرار می دهیم. بهینه ساز می تواند یکی از این ۳ الگوریتم باشد: Adam, RMSProp, SGD و تابع ضرر می تواند یکی از این دو تابع باشد: Cross entropy, Negative log likelihood. در یک حلقه به تعداد ایپاک iterate می کنیم. سپس مدل را در حالت آموزش قرار می دهیم و یک batch از داده می خوانیم. در forward پیش بینی مدل را محاسبه می کنیم و سپس مقدار خطا را بدست می آوریم. در back propagation پارامترهای مدل را آپدیت می کنیم. میزان دقت مدل را بدست می آوریم و به همراه خطا، آنها را در یک لیست ذخیره می کنیم.

```python
def train(model, train_loader, criterion, my_optimizer, learning_rate=0.01, num_epochs=15):

    ## TODO: specify optimizer
    if my_optimizer == 'adam':
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    elif my_optimizer == 'rms-prop':
        optimizer = optim.RMSprop(model.parameters(), lr=learning_rate)
    else: optimizer = optim.SGD(model.parameters(), lr=learning_rate)

    ## TODO: specify loss function
    loss_fn = nn.NLLLoss() if criterion == 'nll' else nn.CrossEntropyLoss()

    losses = []
    accuracies = []

    size = len(train_loader.dataset)
    # training
    for epoch in range(num_epochs):

        # iterate on batches
        print("Epoch", epoch)
        epoch_loss = 0
        correct = 0

        model.train()
        for batch, (x, y) in enumerate(train_loader):

            # forward propagation
            pred = model(x)
            # calculate loss
            loss = loss_fn(pred, y)
            # loss.item() gives the average loss of the batch
            # so multiply it by batch size to get the sum
            epoch_loss +=  pred.shape[0] * loss.item()

            # backward propagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if batch % 300 == 0:
                loss, current = loss.item(), (batch + 1) * len(x)
                print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

            # calculate accuracy
            pred_classes = torch.argmax(pred, dim=1)
            correct += (pred_classes == y).float().sum()

        epoch_loss /= size
        correct = correct * 100 / size

        print(f"epoch loss: {epoch_loss:.4f}")
        print(f"epoch accuracy: {correct:.4f}")

        losses.append(epoch_loss)
        accuracies.append(correct)

        print("*"*25)

    return losses, accuracies
```

از تابع plot برای رسم مقدار خطا و دقت مدل روی داده آموزشی در ایپاک های مختلف استفاده می کنیم.

```python
def plot(losses, accuracies, batch_size=64, learning_rate=0.01):
    # plotting
    plt.title("Training Curve (batch_size={}, lr={})".format(batch_size, learning_rate))
    plt.plot(losses, label="Train")
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.show()

    plt.title("Training Curve (batch_size={}, lr={})".format(batch_size, learning_rate))
    plt.plot(accuracies, label="Train")
    plt.xlabel("Iterations")
    plt.ylabel("Accuracy")
    plt.legend(loc='best')
    plt.show()
```

از تابع predict برای پیش بینی مدل روی داده آزمون و محاسبه میزان خطا استفاده می کنیم. برای این کار ابتدا torch.no_grad را صدا می زنیم تا پارامترهای مدل را آپدیت نکند. سپس پیش بینی مدل را بدست می آوریم و با argmax کلاسی که مدل برای هر داده پیش بینی می کند در یک لیست ذخیره می کنیم.

```python
def predict(model, test_loader, criterion):

    loss_fn = nn.NLLLoss() if criterion == 'nll' else nn.CrossEntropyLoss()
    with torch.no_grad():
        predictions = []
        ## TODO: calculate and print avg test loss
        loss = 0

        for data in test_loader:
            images, labels = data
            pred = model(images)
            loss += pred.shape[0] * loss_fn(pred, labels).item()

            pred_classes = torch.argmax(pred, dim=1)
            predictions.append(pred_classes)

        loss /= len(test_loader.dataset)
        print('Test Loss: {:.6f}\n'.format(loss))

    return torch.cat(predictions)
```

از تابع visualize برای رسم پیش بینی مدل بر روی تعدادی از داده های آزمون استفاده می کنیم.

از تابع metrics برای محاسبه میزان دقت مدل، confusion matrix و امتیاز f1 برای هر کلاس استفاده می کنیم.

با استفاده از تابع roc، منحنی roc برای هر کلاس را رسم می کنیم و در نهایت منحنی roc میانگین تمامی کلاس ها را نیز رسم می کنیم. مقدار auc هریک از پلات ها را نیز محاسبه می کنیم.

در پایان از تابع test برای صدا زدن تمامی توابع مربوط به داده آزمون بهره می بریم.

در ۱۰ آزمایش مختلف (ترکیبی از توابع فعال سازی، تابع ضرر و الگوریتم بهینه سازی) مدل را آموزش داده و سپس با داده آزمون آن را تست می کنیم.

Experiment 1:

ReLU, NLL, SGD

Train loss: 0.1804

Train accuracy: 0.9473

Test Loss: 0.171226

Test Accuracy: 0.9507

f1_score for class 0 is: 0.9702471003530005

f1_score for class 1 is: 0.9771528998242531

f1_score for class 2 is: 0.9512670565302145

f1_score for class 3 is: 0.9422885572139302

f1_score for class 4 is: 0.9499489274770174

f1_score for class 5 is: 0.9400560224089636

f1_score for class 6 is: 0.9564315352697095

f1_score for class 7 is: 0.9478811495372625

f1_score for class 8 is: 0.9336078229541945

f1_score for class 9 is: 0.9333996023856859

Experiment 2:

ReLU, NLL, RMSProp

Train loss: 0.2184

Train accuracy: 0.9549

Test Loss: 0.365624

Test Accuracy: 0.9493

f1_score for class 0 is: 0.9780724120346762

f1_score for class 1 is: 0.9806678383128294

f1_score for class 2 is: 0.9530266343825667

f1_score for class 3 is: 0.9381995133819951

f1_score for class 4 is: 0.9438909281594128

f1_score for class 5 is: 0.9350348027842227

f1_score for class 6 is: 0.9614775725593667

f1_score for class 7 is: 0.9519650655021834

f1_score for class 8 is: 0.9232303090727817

f1_score for class 9 is: 0.9229268292682927

Experiment 3:

ReLU, CE, Adam

Train loss: 0.1666

Train accuracy: 0.9586

Test Loss: 0.307096

Test Accuracy: 0.9505

f1_score for class 0 is: 0.9677093844601412

f1_score for class 1 is: 0.982174688057041

f1_score for class 2 is: 0.9577464788732395

f1_score for class 3 is: 0.948301329394387

f1_score for class 4 is: 0.9544757033248081

f1_score for class 5 is: 0.9371174179187534

f1_score for class 6 is: 0.9599578503688092

f1_score for class 7 is: 0.9561752988047808

f1_score for class 8 is: 0.9081735620585267

f1_score for class 9 is: 0.9285714285714285

Experiment 4:

ReLU, CE, SGD

Train loss: 0.1776

Train accuracy: 0.9487

Test Loss: 0.171901

Test Accuracy: 0.9496

f1_score for class 0 is: 0.970336852689794

f1_score for class 1 is: 0.9806848112379281

f1_score for class 2 is: 0.9496824621397167

f1_score for class 3 is: 0.9398422090729782

f1_score for class 4 is: 0.9505354411014788

f1_score for class 5 is: 0.9319842608206858

f1_score for class 6 is: 0.9558899844317591

f1_score for class 7 is: 0.945721271393643

f1_score for class 8 is: 0.9309807991696938

f1_score for class 9 is: 0.9341258048538881

Experiment 5:

Tanh, CE, SGD

Train loss: 0.3317

Train accuracy: 0.9037

Test Loss: 0.305115

Test Accuracy: 0.9116

f1_score for class 0 is: 0.9373072970195272

f1_score for class 1 is: 0.9667386609071275

f1_score for class 2 is: 0.9199803632793323

f1_score for class 3 is: 0.9015338941118258

f1_score for class 4 is: 0.9140225179119754

f1_score for class 5 is: 0.8625486922648858

f1_score for class 6 is: 0.9341825902335457

f1_score for class 7 is: 0.9200385356454721

f1_score for class 8 is: 0.8713605082053997

f1_score for class 9 is: 0.8755406054781354

Experiment 6:

Tanh, NLL, SGD

Train loss: 0.2602

Train accuracy: 0.9243

Test Loss: 0.254482

Test Accuracy: 0.9279

f1_score for class 0 is: 0.962556165751373

f1_score for class 1 is: 0.9737072743207713

f1_score for class 2 is: 0.9205722742969906

f1_score for class 3 is: 0.9104403760514597

f1_score for class 4 is: 0.9310168625447113

f1_score for class 5 is: 0.8958097395243488

f1_score for class 6 is: 0.9490114464099896

f1_score for class 7 is: 0.9269249632172633

f1_score for class 8 is: 0.8879795396419436

f1_score for class 9 is: 0.9112426035502958

Experiment 7:

Tanh, NLL, RMSProp

Train loss: 0.2474

Train accuracy: 0.9296

Test Loss: 0.313152

Test Accuracy: 0.9118

f1_score for class 0 is: 0.9590036325895174

f1_score for class 1 is: 0.9713292788879235

f1_score for class 2 is: 0.9130850047755492

f1_score for class 3 is: 0.9192483494159471

f1_score for class 4 is: 0.8992416034669556

f1_score for class 5 is: 0.8792710706150341

f1_score for class 6 is: 0.9392207792207793

f1_score for class 7 is: 0.9106116048092002

f1_score for class 8 is: 0.8944930519814719

f1_score for class 9 is: 0.8335483870967741

Experiment 8:

Sigmoid, CE, Adam

Train loss: 0.0832

Train accuracy: 0.9749

Test Loss: 0.129283

Test Accuracy: 0.9645

f1_score for class 0 is: 0.9743073047858942

f1_score for class 1 is: 0.9833916083916083

f1_score for class 2 is: 0.9643201542912246

f1_score for class 3 is: 0.9565656565656565

f1_score for class 4 is: 0.9679089026915113

f1_score for class 5 is: 0.9562363238512035

f1_score for class 6 is: 0.9701414353064434

f1_score for class 7 is: 0.9636184857423796

f1_score for class 8 is: 0.9553014553014553

f1_score for class 9 is: 0.9501466275659824

Experiment 9:

Sigmoid, CE, RMSProp

Train loss: 0.0651

Train accuracy: 0.9811

Test Loss: 0.137099

Test Accuracy: 0.9698

f1_score for class 0 is: 0.9826530612244898

f1_score for class 1 is: 0.9863975427819218

f1_score for class 2 is: 0.9716796875

f1_score for class 3 is: 0.965925925925926

f1_score for class 4 is: 0.9665167416291854

f1_score for class 5 is: 0.9638688160088937

f1_score for class 6 is: 0.9671704012506515

f1_score for class 7 is: 0.9686581782566112

f1_score for class 8 is: 0.9636174636174636

f1_score for class 9 is: 0.9585621567648527

Experiment 10:

Sigmoid, NLL, SGD

Train loss: 0.8543

Train accuracy: 0.7163

Test Loss: 0.814859

Test Accuracy: 0.7355

f1_score for class 0 is: 0.8702064896755161

f1_score for class 1 is: 0.9052544476623914

f1_score for class 2 is: 0.723209751142712

f1_score for class 3 is: 0.7097661623108666

f1_score for class 4 is: 0.6965416463711641

f1_score for class 5 is: 0.5642946317103621

f1_score for class 6 is: 0.8291431503335043

f1_score for class 7 is: 0.7804646752015174

f1_score for class 8 is: 0.5950226244343891

f1_score for class 9 is: 0.5901981230448384

تحلیل: مقایسه مقدار خطا بین ۱۰ آزمایش نمی تواند درست باشد زیرا از توابع خطای گوناگون استفاده کرده ایم (میتوانیم میزان خطای آزمایش هایی که از یک تابع ضرر استفاده کرده اند را با یکدیگر مقایسه کنیم). اما در مقایسه میزان دقت روی داده آزمون، بیشترین دقت مربوط به آزمایش ۹ (0.9698) و کمترین دقت مربوط به آزمایش ۱۰ (0.7355) است. آزمایش ها را بر اساس میزان دقت آزمون به صورت زیر خواهند بود:

9. ('Sigmoid, CE, RMSProp', 0.9698)

8. ('Sigmoid, CE, Adam', 0.9645)

1. ('ReLU, NLL, SGD', 0.9507)

3. ('ReLU, CE, Adam', 0.9505)

4. ('ReLU, CE, SGD', 0.9496)

2. ('ReLU, NLL, RMSProp', 0.9493)

6. ('Tanh, NLL, SGD', 0.9279)

7. ('Tanh, NLL, RMSProp', 0.9118)

5. ('Tanh, CE, SGD', 0.9116)

10. ('Sigmoid, NLL, SGD', 0.7355)

به طور کلی میزان دقت در آزمایش هایی که الگوریتم بهینه سازی Adam بوده، بالاتر است (آزمایش ۸ با اختلاف کم در جایگاه دوم قرار گرفته است). الگوریتم آدام در مقایسه با sgd سریعتر می تواند converge کند.مثلا در آزمایش ۱۰ که از الگوریتم sgd استفاده شده است، مدل دچار underfit شده و نتوانسته در ۱۵ ایپاک به دقت بالایی (نسبت به سایر آزمایش ها) روی داده آموزشی برسد.

در مقایسه توابع فعال سازی، به طور کلی آزمایش هایی که از تابع tanh استفاده کرده اند، نسبت به سایر آزمایش ها در جایگاه پایین تری قرار گرفته اند. انتظار میرفت relu بهترین نتایج را کسب کند زیرا توانسته مشکلات sigmoid (بسیاری از مقادیر گرادیان نزدیک به صفر، خروجی همواره مثبت، مشتقات هم علامت و حرکت زیگزاگی) را حل کند اما دو آزمایش sigmoid بالاتر از relu قرار گرفته اند.

برای یک مسئله طبقه بندی چند کلاسه مطابق با maximum likelihood estimation، می توانیم از کراس آنتروپی یا NLL استفاده کنیم. در مقایسه توابع ضرر، استفاده کردن از یک لایه log softmax و سپس تابع ضرر NLL مشابه با استفاده از تابع ضرر Cross Entropy است (مطابق با این لینک).

در این آزمایش ها overfit اتفاق نیفتاده زیرا میزان دقت داده آموزشی و داده آزمون اختلاف چندانی ندارند. تنها آزمایش ۱۰ دچار underfit شده است و برای حل این مشکل می توانیم مدل را با تعداد ایپاک بیشتر آموزش دهیم.

نمودارهای هر یک از آزمایش ها نیز در نوت بوک تمرین موجود است.

منابع

https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/