

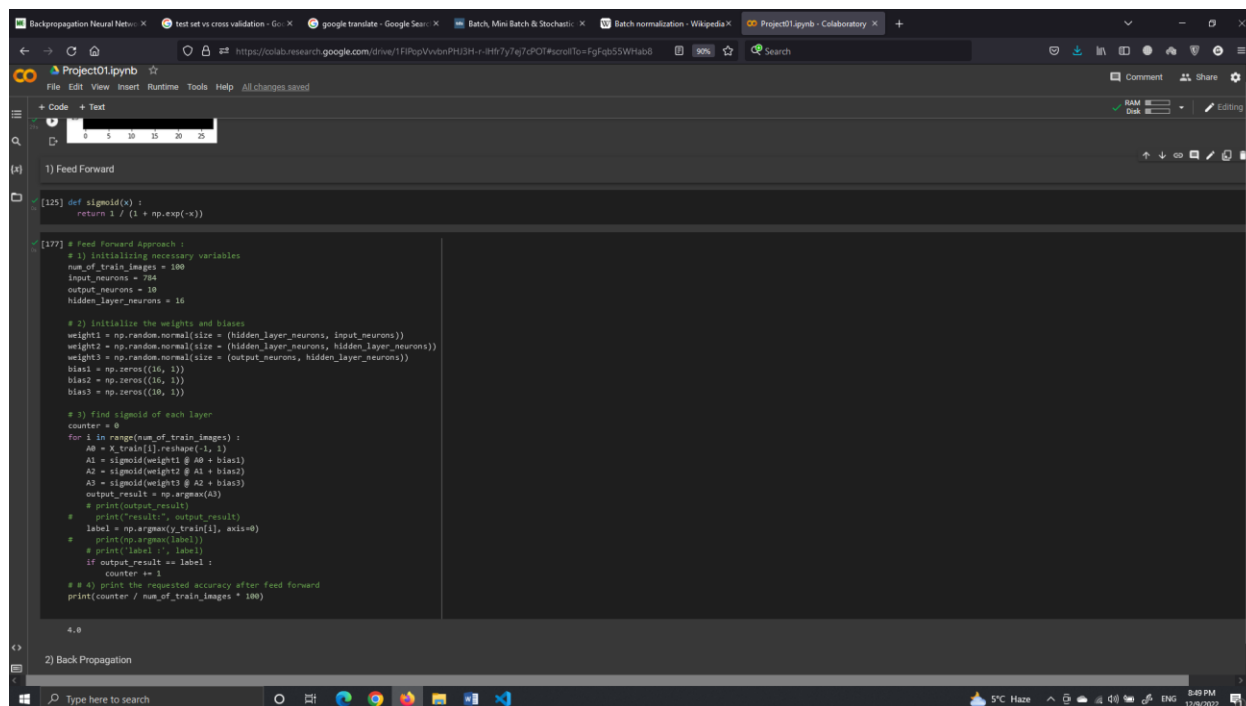
((گزارش کار پروژه شبکه های خواسته شده))

گردآورنده : عرفان ماجدی 9831099

مبانی هوش محاسباتی

دکتر عبادزاده

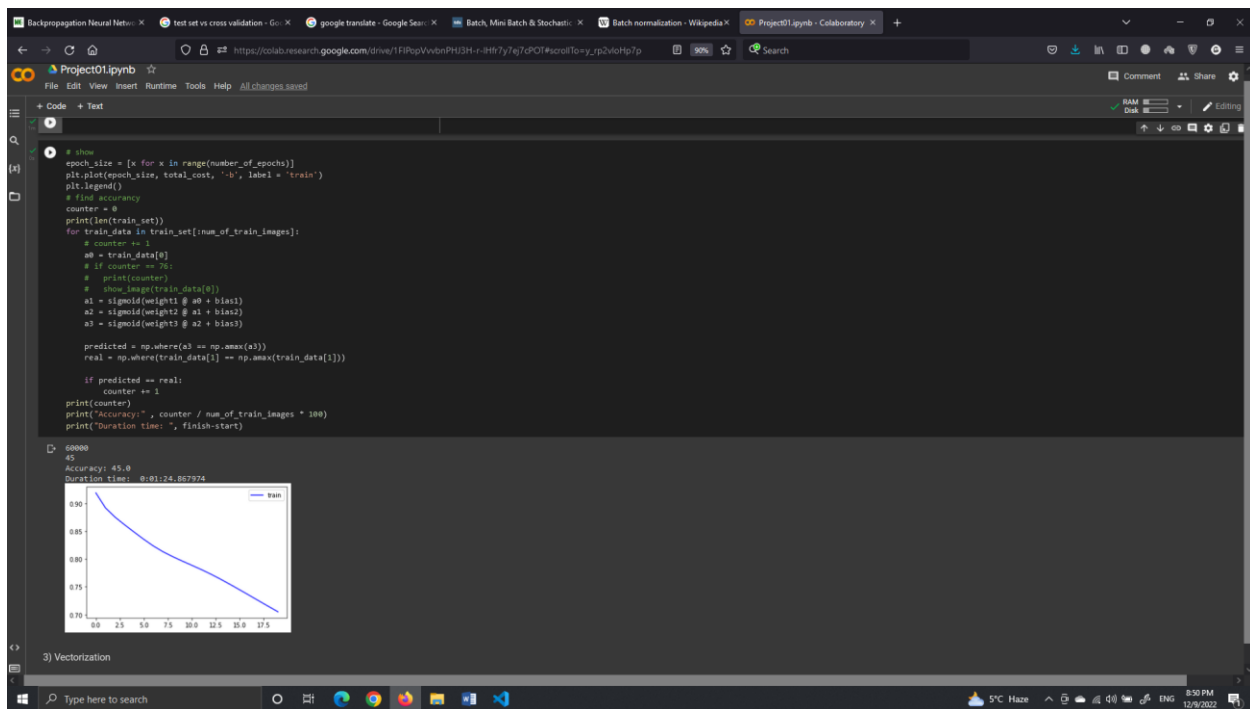
موارد خواسته شده برای گزارش)



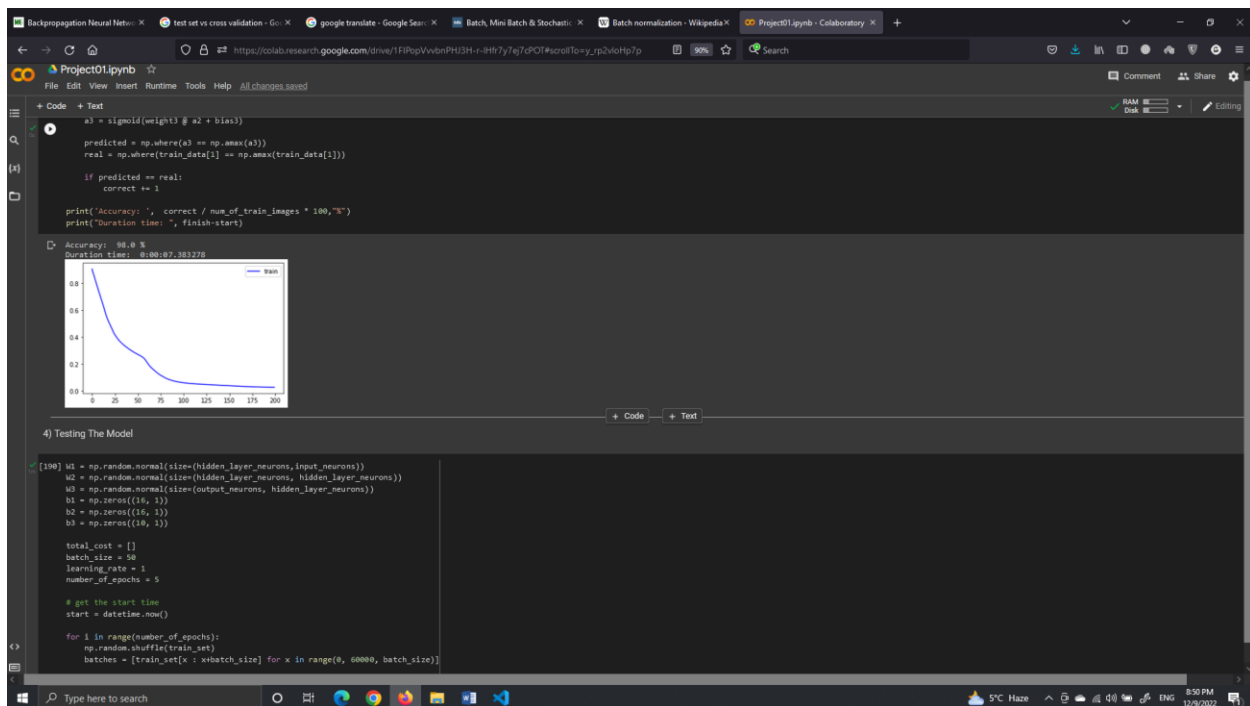
The screenshot shows a Google Colab notebook titled "Project01.ipynb". The code is written in Python and implements a feedforward neural network. It starts with a "Feed Forward" section where the sigmoid function is defined. Then, it initializes variables for the number of training images, input, output, and hidden layer neurons. It also initializes weights and biases for the hidden and output layers. The main loop iterates over the training images, calculates the output of the network, and prints the result. Finally, it prints the accuracy of the network.

```
[125] def sigmoid(x):  
    return 1 / (1 + np.exp(-x))  
  
[177] # Feed Forward Approach :  
# 1) Initializing necessary variables  
num_of_train_images = 100  
input_neurons = 784  
output_neurons = 10  
hidden_layer_neurons = 16  
  
# 2) Initialize the weights and biases  
weight1 = np.random.normal(size = (hidden_layer_neurons, input_neurons))  
weight2 = np.random.normal(size = (hidden_layer_neurons, hidden_layer_neurons))  
weight3 = np.random.normal(size = (output_neurons, hidden_layer_neurons))  
bias1 = np.zeros((16, 1))  
bias2 = np.zeros((16, 1))  
bias3 = np.zeros((10, 1))  
  
# 3) Find sigmoid of each layer  
counter = 0  
for i in range(num_of_train_images):  
    A0 = X_train[i].reshape(-1, 1)  
    A1 = sigmoid(weight1 @ A0 + bias1)  
    A2 = sigmoid(weight2 @ A1 + bias2)  
    A3 = sigmoid(weight3 @ A2 + bias3)  
    output_result = np.argmax(A3)  
    # print(output_result)  
    # print("result:", output_result)  
    label = np.argmax(y_train[i], axis=0)  
    # print(np.argmax(label))  
    # print("label :", label)  
    if output_result == label:  
        counter += 1  
  
# 4) print the requested accuracy after feed forward  
print(counter / num_of_train_images * 100)  
  
4.0  
  
2) Back Propagation
```

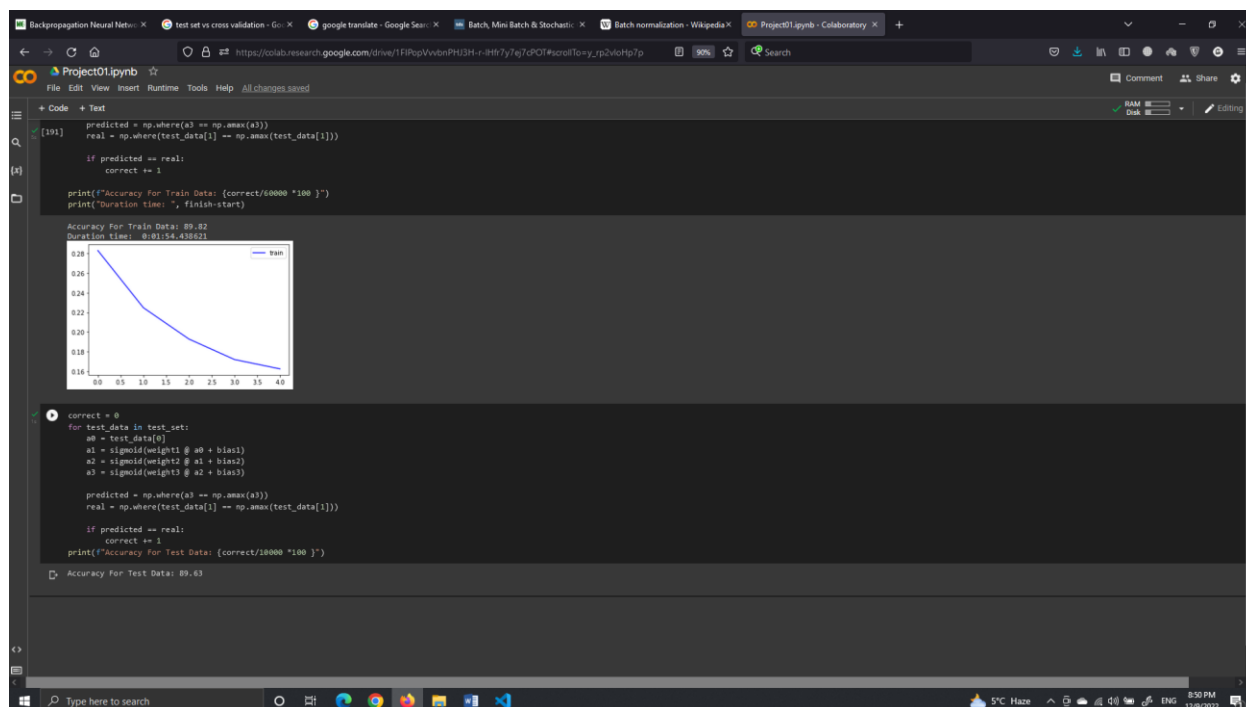
دقت feed forward



نمودار، accuracy و مقدار تایم اجرای الگوریتم back propagation



نمودار، accuracy و مقدار تایم اجرای الگوریتم vectorization



نمودار بعد از تست کردن مدل و دقت داده های تست در `test_set`

جواب سوالات تحقیقاتی امتیازی)

جواب سوال یک :

Cross validation تکنیکی برای ارزیابی مدل های ML با آموزش چندین مدل ML بر روی زیرمجموعه های داده های ورودی موجود و ارزیابی آن ها بر روی زیر مجموعه داده های مکمل است. از **cross validation** برای تشخیص بیش از حد برازش ، به عنوان مثال ، شکست در تعمیم یک الگو استفاده کنید. **مجموعه اعتبار سنجی**: مجموعه ای از مثال هایی که برای تنظیم پارامترهای یک طبقه بندی کننده، به عنوان مثال برای انتخاب تعداد واحدهای پنهان در یک شبکه عصبی استفاده می شود. — **مجموعه تست**: مجموعه ای از مثال ها که فقط برای ارزیابی عملکرد یک طبقه بندی کننده کاملاً مشخص استفاده می شود.

جواب سوال دو :

در **Batch Gradient Descent** ، تمام داده های آموزشی برای برداشتن یک مرحله در نظر گرفته می شود. میانگین گرادیان تمام مثال های آموزشی را می گیریم و سپس از آن گرادیان میانگین برای به روزرسانی پارامترهایمان استفاده می کنیم. بنابراین این فقط یک مرحله از شیب نزول در یک دوره است.

Batch Gradient Descent برای منیفولدهای خطای محدب یا نسبتاً صاف عالی است. در این مورد، ما تا حدودی مستقیماً به سمت یک راه حل بهینه حرکت می کنیم. نمودار هزینه در مقابل دوره ها نیز کاملاً صاف است زیرا ما در حال میانگین گیری از تمام گرادیان های داده های آموزشی برای یک مرحله واحد هستیم. هزینه در طول دوره ها کاهش می یابد.

در **Batch Gradient Descent**، تمام مثال ها را برای هر مرحله از **Gradient Descent** در نظر می گرفتیم. اما چه می شود اگر مجموعه داده ما بسیار بزرگ باشد. مدل های یادگیری عمیق مشتاق داده هستند. هر چه داده ها بیشتر باشد، شانس یک مدل برای خوب بودن بیشتر است. فرض کنید مجموعه داده ما دارای 5 میلیون نمونه باشد، سپس فقط برای برداشتن یک مرحله، مدل باید گرادیان تمام 5 میلیون نمونه را محاسبه کند. این روش کارآمدی به نظر نمی رسد. برای مقابله با این مشکل ما **Stochastic Gradient Descent** داریم. در نزول گرادیان تصادفی (SGD)، ما در هر زمان فقط یک مثال را برای برداشتن یک گام در نظر می گیریم. از آنجایی که ما در هر زمان فقط یک مثال را در نظر می گیریم، هزینه بر روی نمونه های آموزشی همچنین از آنجایی که هزینه آن بسیار متغیر است، هرگز به حداقل نمی رسد، اما به نوسان در اطراف آن ادامه می دهد.

SGD را می توان برای مجموعه داده های بزرگتر استفاده کرد. وقتی مجموعه داده بزرگ باشد سریعتر همگرا می شود زیرا باعث به روز رسانی بیشتر پارامترها می شود. ر نوسان است و لزوماً کاهش نمی یابد. اما در دراز مدت با نوسانات شاهد کاهش هزینه خواهید بود. ما **Batch Gradient Descent** را دیده ایم. ما همچنین شاهد نزول گرادیان تصادفی هستیم **Batch Gradient Descent**، را می توان برای منحنی های صاف تر استفاده کرد. زمانی که مجموعه داده بزرگ باشد می توان از SGD استفاده کرد **Batch Gradient Descent**. مستقیماً به حداقل همگرا می شود SGD. برای مجموعه داده های بزرگتر سریعتر همگرا می شود. اما، از آنجایی که در SGD ما در هر زمان فقط از یک مثال استفاده می کنیم، نمی توانیم پیاده سازی برداری شده را روی آن پیاده سازی کنیم. این می تواند محاسبات را کند کند. برای مقابله با این مشکل، ترکیبی از **Batch Gradient Descent** و SGD استفاده می شود.

نه ما از همه مجموعه داده به طور همزمان استفاده می کنیم و نه از مثال واحد در یک زمان استفاده می کنیم. ما از مجموعه ای از تعداد ثابت نمونه های آموزشی استفاده می کنیم که کمتر از مجموعه داده واقعی است و آن را یک دسته کوچک می نامیم. انجام این کار به ما کمک می کند تا به مزایای هر دو نوع قبلی که دیدیم دست پیدا کنیم. بنابراین، پس از ایجاد دسته های کوچک با اندازه ثابت، مراحل زیر را در یک دوره انجام می دهیم:

یک مینی دسته انتخاب کنید

آن را به شبکه عصبی تغذیه کنید

میانگین گرادیان دسته کوچک را محاسبه کنید

از گرادیان میانگینی که در مرحله 3 محاسبه کردیم برای به روز رسانی وزن ها استفاده کنید

مراحل 1-4 را برای مینی بچ هایی که ایجاد کردیم تکرار کنید

دقیقاً مانند SGD، میانگین هزینه در طول دوره ها در نزول گرادیان دسته ای کوچک نوسان دارد، زیرا ما تعداد کمی از نمونه ها را در یک زمان میانگین می گیریم.

جواب سوال سه :

نرمال سازی دسته ای (همچنین به عنوان هنجار دسته ای شناخته می شود) روشی است که برای آموزش شبکه های عصبی مصنوعی سریعتر و پایدارتر از طریق عادی سازی ورودی لایه ها با مرکزیت مجدد و مقیاس گذاری مجدد استفاده می شود. این توسط سرگئی آیوف و کریستین سگدی در سال 2015 پیشنهاد شد.

در حالی که اثر عادی سازی دسته ای مشهود است، دلایل اثربخشی آن همچنان مورد بحث است. اعتقاد بر این بود که می تواند مشکل تغییر متغیر داخلی را کاهش دهد، که در آن مقدار اولیه پارامتر و تغییرات در توزیع ورودی های هر لایه بر نرخ یادگیری شبکه تأثیر می گذارد. [1] اخیراً، برخی از محققان استدلال کرده اند که نرمال سازی دسته ای تغییر متغیر کمی داخلی را کاهش نمی دهد، بلکه تابع هدف را هموار می کند، که به نوبه خود عملکرد را بهبود می بخشد. با این حال، در زمان اولیه، نرمال سازی دسته ای در واقع باعث انفجار شدید گرادیان در شبکه های عمیق می شود، که تنها با اتصالات پرش در شبکه های باقیمانده کاهش می یابد. برخی دیگر معتقدند که نرمال سازی دسته ای به جداسازی جهت طول می رسد و در نتیجه شبکه های عصبی را تسریع می بخشد. اخیراً یک تکنیک برش گرادیان عادی و تنظیم فراپارامتر هوشمند در شبکه های بدون نرمالیزور معرفی شده است که به نام «NF-Nets» نامیده می شود که نیاز به نرمال سازی دسته ای را کاهش می دهد.

جواب سوال چهار :

وظیفه ی لایه ی pooling در CNN ، feature map ها را کوچک تر می کند. یک روش استفاده از آن max pooling است که در هر پنجره مقدار ماکزیمم آن را برمیگرداند. روش دیگر average pooling است که در هر پنجره مقدار میانگین آن را برمیگرداند. به دو دلیل CNN بر شبکه ساده ارجحیت دارد : (1) در CNN با استفاده از کرنل هایی که وجود دارد وابستگی محلی میان پیکسل ها در نظر گرفته می شود یعنی شبکه همسایگی بین پیکسل ها را در نظر می گیرد و براساس آن ها ویژگی های جدید ایجاد می کند. (2) به دلیل مشترک بودن یک کرنل برای همسایگی های مختلف وزن ها به اشتراک گذاشته می شوند بنابراین تعداد پارامتر های شبکه به طرز قابل توجهی کاهش پیدا می کند.