1η εργασία στα Νευρωνικά δίκτυα

Ιωακείμ Ευαγγελινός
ΑΕΜ 4187

Περίληψη—Κατά την εκπόνηση της παρούσας εργασία, εκπαιδεύονται και συγκρίνονται 8 μοντέλα συνελικτικών νευρωνικών δικτύων, το καθένα με διαφορετικές υπέρ-παραμέτρους, στο σύνολο δεδομένων **CIFAR10** [1]. Συγκεκριμένα, συγκρίνονται τα ποσοστά επιτυχίας στην Ακρίβεια (**Accuracy**) στα σύνολα δεδομένων εκπαίδευσης και δοκιμής, οι χρόνοι εκπαίδευσης. Έπειτα, συγκρίνονται με τους κατηγοριοποιητές της ενδιάμεσης εργασίας.

***Index Terms*—**Κατηγοριοποιητής, **CIFAR10**, Κατηγοριοποιητής n-πλησιέστερων γειτόνων, Κατηγοριοποιητής πλησιέστερου κέντρου, **Convolutional Neural Network**, **Neural Network**, **Adam**, **Stochastic Gradient Descent**, **Data Augmentation**, **PyTorch**

I . Εισαγωγή

Αυτή η εργασία προσδιορίζεται ως τεκμηρίωση της 1ης εργασίας που υλοποιεί ένα συνελικτικό νευρωνικό δίκτυο εμπρόσθιας τροφοδότησης, στο σύνολο δεδομένων CIFAR10. Το νευρωνικό μοντέλο που δημιούργησα με χρήση της βιβλιοθήκης PyTorch δέχεται ως είσοδο μια (ή περισσότερες) εικόνα και προβλέπει σε ποια από τις πιθανές κατηγορίες ανήκει.

II . Υπόβαθρο

Το CIFAR10 είναι ένα σύνολο δεδομένων 60000 RGB εικόνων με διαστάσεις $32x32$. Το σύνολο δεδομένων χωρίζεται σε σύνολο δεδομένων εκπαίδευση (50000 εικόνες) και δοκιμής (10000 εικόνες). Χρησιμοποιείται εκτενώς για εκπαιδευτικούς σκοπούς με στόχο την σωστή κατηγοροποίηση μιας εικόνας στην σωστή κατηγορία.
Το PyTorch είναι μια πολύ χρήσιμη βιβλιοθήκη της Python για την δημιουργία νευρωνικών δικτύων.

III . Αλγόριθμος

1. Μοντέλο:
Για την τελική σχεδίαση του μοντέλου, έκανα σύγκριση μοντέλα διάφορων μεγέθους κρυφά επίπεδα και επέλεξα το μοντέλο με την καλύτερη απόδοση (Σημείωση: Όταν έκανα την σύγκριση, είχα κάνει μόνο Normalization από Data Augmentation οπότε τα αποτελέσματα ίσως διαφέρουν με τον πιο πρόσφατο κώδικα).
Μοντέλο 1)
$Covolution \rightarrow ReLu \rightarrow Dropout \rightarrow MaxPooling \rightarrow Covolution \rightarrow ReLu \rightarrow Dropout \rightarrow Linear \rightarrow ReLu \rightarrow Dropout \rightarrow Linear \rightarrow ReLu \rightarrow Dropout \rightarrow Linear \rightarrow ReLu \rightarrow Dropout \rightarrow Linear \rightarrow Softmax$
Η δομή του μοντέλου είναι να περνώ την εικόνα μέσα από ένα επίπεδο συνέλιξης (Convolutional Layer), μετά από ένα επίπεδο επίπεδο συγκέντρωσης μεγίστων (Max pooling layer), μετά πάλι από ένα επίπεδο συνέλιξης και στην συνέχεια μέσα από τέσσερα γραμμικά επίπεδα (linear layer), όπου ενδιάμεσα από κάθε επίπεδο υπάρχει μια συνάρτηση ενεργοποίησης (ReLU) και ένα επίπεδο περιορισμού ενεργοποίησης (dropout layer). Η έξοδος περνάει και αυτήν από μια συνάρτηση ενεργοποίησης softmax.
Μοντέλο 2)
$Covolution \rightarrow ReLu \rightarrow Dropout \rightarrow MaxPooling \rightarrow Covolution \rightarrow ReLu \rightarrow Dropout \rightarrow Linear \rightarrow Softmax$
Η δομή του μοντέλου είναι να περνώ την εικόνα μέσα από ένα επίπεδο συνέλιξης (Convolutional Layer), μετά από ένα επίπεδο επίπεδο συγκέντρωσης μεγίστων (Max pooling layer), μετά πάλι από ένα επίπεδο συνέλιξης και στην συνέχεια μέσα από ένα γραμμικό επίπεδο (linear layer), όπου ενδιάμεσα από κάθε επίπεδο υπάρχει μια συνάρτηση ενεργοποίησης (ReLU) και ένα επίπεδο περιορισμού ενεργοποίησης (dropout layer). Η έξοδος περνάει και αυτήν από μια συνάρτηση ενεργοποίησης softmax.
Παρακάτω βρίσκονται τα αποτελέσματα στα σύνολα εκπαίδευσης και δοκιμής των παραπάνω μοντέλων για 100 εποχές με πιθανότητα περιορισμού ενεργοποίησης 0% (Σημείωση: Το 2ο μοντέλο έκανε overfit και σταμάτησε στην εποχή 76).

Μοντέλο	Επιτυχία % εκπαίδευσης	Επιτυχία % δοκιμής
1	69.25199627876282	63.87999653816223
2	77.52999663352966	65.45000076293945

Βλέποντας ότι το 2ο μοντέλο έχει καλύτερη επίδοση, αποφάσισα να χρησιμοποιήσω το 2ο και αργότερα το απλοποίησα στο παρακάτω μοντέλο: $Covolution \rightarrow ReLu \rightarrow MaxPooling \rightarrow Covolution \rightarrow ReLu \rightarrow Dropout \rightarrow Linear \rightarrow Softmax$.

2. Εκπαίδευση: Για την εκπαίδευση, δημιούργησα μια συνάρτηση όπου για κάθε εποχή, για όλα τα batches, υπολογίζω κόστος του κάθε batch με βάση το criterion CrossEntropyLoss() της PyTorch, και, στη συνέχεια, ενημερώνω τα βάρη με βάση το κόστος με back-propagation χρησιμοποιώντας τον optimizer που επέλεξα. Ακολουθεί ένα απόσπασμα της συνάρτησης που υλοπιεί την παραπάνω διαδικασία:

```
    for epoch in range
    ↪  (continue_from_epoch,
    ↪  continue_from_epoch + epoches):
        subloss=0
```

```
        total_loss=0
        start_time_epoch=time.time()
        start_time = time.time()
        for batch_number,(x_train,y_train) in enumerate(dataloader):
            x_train, y_train= x_train.to(device), y_train.to(device)
            y_pred=model(x_train)

            loss=criterion(y_pred,y_train)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            #subloss+=loss.item()
            total_loss+=loss.item()
```

3. Δοκιμή:
Για την δοκιμή του μοντέλου, απλώς υπολογίζω την ακρίβεια του μοντέλου στο σύνολο δεδομένων δοκιμής. Ακολουθεί η συνάρτηση που υλοποιεί την παραπάνω διαδικασία:

```
def test_model(model,dataloader, limit_batches=-1):
    samples_tested=0
    correct=0
    with no_grad():
        for batch_number,(x_test,y_test) in enumerate(dataloader):
            x_test,y_test= x_test.to(device), y_test.to(device)
            y_pred=model(x_test)


            samples_tested+=y_test.size(0)
            prediction= torch.max(y_pred,1)[1]
            correct+= (prediction==y_test).sum()
            #if (batch_number%100==0):
            #    print(f"Batch number: {batch_number}, Accuraccy: {correct/samples_tested}")
            #if (limit_batches == batch_number):
            #    break
    accuracy= correct/samples_tested
    return (accuracy)
```

IV. Προεπεξεργασία εικόνων

Κατά την εισαγωγή των εικόνων, τις κανονικοποιώ, τις αντιστρέφω τυχαία τα χρώματα με 20% πιθανότητα και τις αναστρέφω τυχαία οριζόντια με 50% πιθανότητα.

Ο κώδικας μετασχηματισμών εικόνας:

```
mean=[0.4914, 0.4822, 0.4465]
std=[0.2023, 0.1994, 0.2010]

transformations=transforms.Compose([
    transforms.ToTensor(),
    transforms.RandomInvert(p=0.2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.Normalize(mean= mean,std= std),
    ])
```

Για την κανονικοποίηση, υπολογίζω την τυπική απόκλιση και το μέσο με τον παρακάτω κώδικα στο αρχικό αμετάβλητο σύνολο εκπαίδευσης:

```
mean = 0.
std = 0.
nb_samples = 0.
for data, _ in dataloader_train:
    batch_samples = data.size(0)
    data = data.view(batch_samples, data.size(1), -1)
    mean += data.mean(2).sum(0)
    std += data.std(2).sum(0)
    nb_samples += batch_samples

mean /= nb_samples
std /= nb_samples
```

V . Σύγκριση μοντέλων

Παρακάτω συγκρίνω τα μοντέλα (με 20% πιθανότητα περιορισμού ενεργοποίησης και batch size = 50) για 100 εποχές με διαφορετικούς ρυθμούς μάθησης (0.001 και 0.0001), βελτιστοποιητές (Adam και Stochastic gradient descent), και αριθμό νευρώνων (στο 1ο και 2ο κρυφό επίπεδο) ([100,70],[36,50]) ως προς τον χρόνο εκπαίδευσης σε δευτερόλεπτα και τα ποσοστά επιτυχίας στα σύνολα εκπαίδευσης και δοκιμής:

h1,h2	lr	optim	secs	test, train%
100,70	0.001	adam	912.9	61.05, 63.37
100,70	0.0001	adam	885.9	68.76, 74.19
36,50	0.001	adam	886.4	61.77, 63.72
36,50	0.0001	adam	882.6	65.46, 69.72
100,70	0.001	sgd	891.6	54.11, 55.26
100,70	0.0001	sgd	897.6	28.75, 28.35
36,50	0.001	sgd	888.3	48.25, 48.33
36,50	0.0001	sgd	890.9	24.38, 24.04

Από τα παραπάνω αποτελέσματα, παρατηρώ ότι με Stochastic gradient descent, τα μοντέλα δεν προλαβαίνουν να εκπαιδευτούν πλήρως (με 200 εκπαιδεύονται κανονικά), σε αντίθεση με τον Adam. Παρατηρώ επίσης ότι με ρυθμός μάθησης 0.0001, τα μοντέλα με Adam έχουν καλύτερη επίδοση από τα αντίστοιχα μοντέλα με ρυθμό μάθησης 0.001. Οι χρόνοι εκπαίδευσης είναι σχετικά ίδιοι σε όλα τα μοντέλα. Το καλύτερο μοντέλο από όλα αυτά, και αυτό που θα χρησιμοποιούσα, είναι το 2ο (100,70 & 0.0001 & adam). Για το 2ο μοντέλο, τα ποσοστά ακρίβειας σωστής κατηγοριοποίησης για κάθε κλάση είναι τα εξής:

Κλάση 0: 0.7912 Κλάση 1: 0.8570 Κλάση 2: 0.6330 Κλάση 3: 0.5378 Κλάση 4: 0.7360 Κλάση 5: 0.7028 Κλάση 6: 0.7978 Κλάση 7: 0.7582 Κλάση 8: 0.8332 Κλάση 9: 0.7928

VI . Σύγκριση αποτελεσμάτων με τα αποτελέσματα ενδιάμεσης εργασίας

Από την ενδιάμεση εργασία, έχουμε τα εξής αποτελέσματα:

Clf	Train time	Test time	Acc	Mem
1-NN	0.303	104.171	0.3545	614800785
3-NN	0.275	104.166	0.3302	614800785
NC	0.501	0.26370	0.2775	246278

Το νευρωνικό μοντέλο έχει πολύ καλύτερη απόδοση στην ακρίβεια αλλά πολύ χειρότερο χρόνο εκπαίδευσης από τους κατηγοριοποιητές n-πλησιέστερων γειτόνων και τον κατηγοριοποιητή πλησιέστερου κέντρου. Ο χρόνος εκπαίδευσης μπορεί να μειωθεί εκπαιδεύοντας το μοντέλο για λιγότερες εποχές. Σε κάθε περίπτωση χρήσης στον πραγματικό κόσμο, θα επέλεγα το νευρωνικό μοντέλο.

VII . Πηγαίος κώδικας

Ο πηγαίος κώδικας βρίσκεται στο αρχείο $CNN\_CIFAR10.ipynb$

Αναφορές

[1] *Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009*. https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf