IE598 Deep Learning and Neural Networks

Pramod Srinivasan(psrnvsn2)

HW3 - Part 3 : Action Recognition in Short Clips

In Parts 1 and 2 of the homework, we have been able to extract visual features from each frame of the video by using a deep convolution neural network. If we stop one layer before softmax, we can collect a set of features from the fully connected layer. These 1024-size features are now sequentially fed into a Long Short Term Memory (LSTM) model in order to be able to learn the temporal dynamics from the time-varying inputs. One important observation is that Convolutional Neural Networks learn more generic features on the bottom of the network (such as edges, local shapes) and more intricate, dataset-specific features near the top of the network. Since there is not enough data to train CNN from scratch, we resort to training only the top layers. The model takes up lots of memory (small batch size=slow training). We use a pre-trained network from ImageNet dataset to finetune it based on the UCF-101 data.

We now present the implementation details and subsequent experimental findings to improve the performance of the given VGG-NET for CIFAR-10 dataset. Specifically, we have covered the following aspects:

1. Number of LSTM Layers
2. Regularization by Weight Decay
3. Interlayer Batch Normalization
4. Increase in Number of Epochs
5. Gradient Clipping
6. Dropout
7. Learning Rate
8. Batch Size
9. Truncated BackPropagation
10. Loss Scaling
11. Inference Technique

1 Number of LSTM Layers

The aim of this experiment is to understand the how the LSTM network depth affects the accuracy in the large-scale action recognition setting.

Number of stacked layers	Single layer	Two layer stacking	Three layer stacking
Train Accuracy	0.958	0.974	0.987
Batch Loss	0.549	0.432	0.375

Number of stacked layers	Single layer	Two layer stacking	Three layer stacking
Top1	**0.6565**	0.640	0.636
Top5	**0.8835**	0.883	0.868
Top10	**0.9335**	0.935	0.930

Observations

- We observe that there is no improvement in the train/test accuracies with the increase in number of layers.
- Deeper LSTMs may tend to perform equally well if not better. Clearly, the results indicate that having no linear layer loses on expressivity and having 2 linear layers overfits the training data, thus keeping our previously found best configuration of 1 linear layer and 1 output layer to be still optimal.
- Thus, we choose the single layer LSTM.

2 Regularization Method : Weight Decay

The optimizer hook function for weight decay adds a scaled parameter to the corresponding gradient. This can be used for regularization purposes.

```
optimizer.add_hook(chainer.optimizer.WeightDecay(0.0005))
```

2.1 Results

Test Accuracy Metric	Without Weight Decay	With Weight Decay
Top1	0.6565	**0.6655**
Top5	0.883	**0.88247**
Top10	0.9435	**0.9348**

Observations

- This is a 5% improvement over the original.
- Weight Decay is an ideal candidate for regularization.

3 Batch Normalization

Batch normalization is a simple and effective way to improve the performance of a neural network. It is established that batch normalization enables the use of higher learning rates besides acting as a regularizer thus achieving a speed-up in the training process. We observed the positive effect of batch normalization as we increase the number of weight layers. Chainer provides features for certain parameters which can be learnt during the backpropagation to unlearn the normalization and thereby recover the identity mapping.

An important observation is that during the test time, BatchNormalization layer functions differently. The mean and variance are not computed based on batch instead a single fixed empirical mean of activations during training is used.

Test Accuracy Metric	BatchNormalization	NoBatchNormalization
Top1	**0.6565**	0.643
Top5	**0.8835**	0.876
Top10	**0.9335**	0.940

3.1 Observation

- Batch normalization is chosen for better test accuracy as well as the purpose of regularization during training process.

4 Optimization Methods

The following optimization techniques were employed : Adam, MomentumSGD and RMSProp. The results are summarized in the table below.

Train Accuracy Metric	MomentumSGD	Adam	RMSProp
Train Accuracy	0.958	**0.9994**	0.974
Batch Loss	0.549	**0.020**	0.329

Test Accuracy Metric	MomentumSGD	Adam	RMSProp
Top1	**0.6565**	0.628	0.636
Top5	**0.8835**	0.8392	0.845
Top10	**0.9335**	0.9039	0.911

4.1 Observations

- There is a 3% and 2% improvement over the original when Adam and RMSProp are employed respectively
- Training accuracies of Adam and RMSProp are better than MomentumSGD – possibly *overfitting*.

- MomentumSGD achieves a 5% improvement over baseline.
- The learning rate of the optimizer has to be fine-tuned for best results.

5 Learning Rate

Initially, we started training using MomentumSGD with an aggressive learning rate of 1e-3 This was found to trouble learning because the learning rate was too high and the loss fluctuated heavily. After testing learning rates, we found that around 1e-6 worked decently well. Because MomentumSGD was showing relatively slow progress in the training, we implemented Adam, the recommended adaptive learning rate method, into our training to speed up the learning. However, the training and test accuracies were below the MomentumSGD performance. although we observed that with Adam implemented, we found that the loss decreased much more than SGD after about half an epoch.

5.1 Train Accuracy Results (after 25 epochs)

Train Accuracy Metrics	$\eta = 3e-4$	$\eta = 5e-4$	$\eta = 7e-4$	$\eta = 9e-4$	$\eta = 1e-3$
Train Accuracy	0.885	0.956	0.990	0.990	**0.9961**
Batch Loss	1.048	0.549	0.325	0.213	**0.162**

5.2 Test Accuracy Results

Train Accuracy Metrics	$\eta = 3e-4$	$\eta = 5e-4$	$\eta = 7e-4$	$\eta = 9e-4$	$\eta = 1e-3$
Top1	0.630	0.6565	0.657	0.665	**0.6669**
Top5	0.869	**0.8835**	0.887	0.885	0.8774
Top10	0.934	**0.9435**	0.941	0.935	0.9385

5.3 Observations

- A common heuristic is to set the Learning rate low enough so that SGD is guaranteed to make converge towards a good solution, however in practice that would take a very long time.
- The best learning rate achieves a 5% improvement over the baseline Top-1 accuracy as well as achieving comparable performance in the Top-5 and Top-10 accuracies.
- We observe that setting the learning rate to $\eta = 1e-3$ would be favorable for the best optimizer, MomentumSGD

6 Increase in Number of Epochs

We varied the number of epochs and we observed that an increase in the training timewas determined by computing performance after each training epoch and stopping training if no

significant improvement has been found over the last 25 epochs. The results are summarized in the following tables.

6.1 Train Accuracy Results (after 25 epochs)

Number of Epochs	E = 25	E = 50	E = 100
Train Accuracy	0.956	0.978	0.999
Batch Loss	0.549	0.450	0.040

6.2 Test Accuracy Results

Number of Epochs	E = 25	E = 50	E = 100
Top1	0.6566	0.6656	0.664
Top5	0.8835	0.899	0.888
Top10	0.9334	0.941	0.9390

6.3 Observation

- Increasing the number of epochs improves the train accuracy/batch loss but does not cause the test accuracies to improve - overfitting
- We will choose the number of epochs to be 25 as it gives the reasonable learning performance as well as time.

7 Regularization Method : Gradient Clipping

Gradient clipping needs to happen after computing the gradients, but before applying them to update the model's parameters. The optimizer hook function scales all gradient arrays to fit to the defined L2 norm threshold. This operation is typically used to clip gradients before applying them with an optimizer.

```
optimizer.add_hook(chainer.optimizer.GradientClipping(5.0))
```

7.1 Train Accuracy Results (after 25 epochs)

Clip Value	MaxClipValue = 5	MaxClipValue = 7	MaxClipValue = 9	MaxClipValue = 11
Train Accuracy	0.956	0.984	0.982	**0.990**
Batch Loss	0.549	0.360	0.284	**0.223**

7.2 Test Accuracy Results

Clip Value	MaxClipValue = 5	MaxClipValue = 7	MaxClipValue = 9	MaxClipValue = 11
Top1	0.6566	0.652	0.651	**0.661**
Top5	0.8835	0.882	0.888	**0.884**
Top10	0.9334	0.940	0.941	**0.940**

Observations

- All the experiments clipped the derivative of the loss with respect to the network inputs to the LSTM layers
- Clipping the output gradients proved vital for **numerical stability**.
- It was observed that the networks sometimes had numerical problems late on in training, after they had started overfitting on the training data.

8 Sequence Length

One intuitive observation is that if the number of layers in the shortest path between time step 1 and time step t increases, then it usually becomes harder for the network to learn to remember information from the distant past. In this series of experiments, we vary the input sequence length and note the results here as follows:

8.1 Train Accuracy Results (after 25 epochs)

Length of sequence	30	60	120	150
Batch Loss	1.998	1.118	0.549	0.411
Train Accuracy	0.703	0.866	0.958	0.977

8.2 Test Accuracy

Length of sequence	30	60	120	150	180
Top1 Accuracy	0.55876	0.6263	0.6565	**0.672**	0.644
Top5 Accuracy	0.813	0.8651	0.958	**0.8835**	0.881
Top10 Accuracy	0.889	0.92521	0.9334	**0.9431**	0.935

Observations

- The hypothesis is that a higher sequence length may improve the test accuracy
- We observed that for a sequence length ≥ 180, the accuracy started to decline.
- This was probably because the final few frames of the video do not have any significant material to learn, hence limiting the sequence length to the average video frame length yields best results.

- If the video length is too small, then repeat sequence in reverse direction to maintain smooth motion until its long enough. If the video is too big, select random segment of length 120.
- Since the length of sequence is longer, the model should be able to learn more time-consuming actions.
- Surprisingly, optimizing for full length predictions hardly improves the performance.

9 Dropout

The retention probability p is varied to observe the change in performance. For instance, a dropout of 0.5 would mean at test time, the **mean network** that contains all of the hidden units but with their outgoing weights halved to compensate for the fact that twice as many of them are active.

9.1 Train Accuracy Results (after 25 epochs)

Retention p	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$	$p = 0.6$
Train Accuracy	1.00	0.956	0.945	0.956	0.896
Batch Loss	0.078	0.730	0.668	0.549	1.041

9.2 Test Accuracy Results

Retention p	$p = 0.2$	$p = 0.3$	$p = 0.4$	$p = 0.5$	$p = 0.6$
Top1 Accuracy	0.665	.6533	0.6570	0.6565	0.622
Top5 Accuracy	0.879	0.8811	0.8776	0.88354	0.868
Top10 Accuracy	0.934	0.9356	0.9361	0.9334	0.928

9.3 Observations

- There is visible increase in training error after having incorporated dropout.
- The main purpose of dropout is the regularization which can prevent overfitting – for a value of $p = 0.5$, we observe the model performs better than the one with dropout.
- No dropout converges faster than dropout but not necessarily to the global optimum.
- The ideal dropout retention probability, $p = 0.3$ model showed the highest recognition rate for the test and sequence data.

10 Scaling

We now understand the effect of scaling the loss during the training procedure on the learning pattern of the LSTM.

10.1 Scaling Accuracy Results (after 25 epochs)

Scaling Factor	Simple addition	Scale Factor
Train Accuracy	0.975	0.958
Batch Loss	0.446	0.549

10.2 Test Accuracy Results

Scaling Factor	Simple addition	Scale Factor
Top1 Accuracy	0.659	0.6566
Top5 Accuracy	0.886	0.8835
Top10 Accuracy	0.9372	0.9335

10.3 Observations

- Although the simple addition results in higher train accuracies, we observe it tends to overfit.
- Although both scaling and non-scaling give comparable test accuracies, we will continue to employ the **scaling technique** in our final model.

11 Backpropogation with Truncation

Truncated BackPropogation Through Time (BPTT) processes the sequence one timestep at a time, and every k_1 timesteps, it runs BPTT for k_2 timesteps, so a parameter update can be cheap if k_2 is small. Consequently, its hidden states have been exposed to many timesteps and so may contain useful information about the far past, which would be opportunistically exploited.

11.1 Train Accuracy Results (after 25 epochs)

Truncation Length	Length = 30	Length = 50	Length = 80	Length = 120
Train Accuracy	0.958	0.999	0.972	0.786
Batch Loss	0.549	0.222	0.454	1.898

11.2 Test Accuracy Results

Truncation Length	Length = 30	Length = 50	Length = 80	Length = 120
Top1 Accuracy	0.656	**0.668**	0.656	0.5750
Top5 Accuracy	0.8835	**0.892**	0.881	0.8269
Top10 Accuracy	0.934	**0.941**	0.935	0.9054

11.3 Observations

- The ideal truncated backprop length should be a **good fraction** of the overall sequence length.
- The training time changed from around 34 min to 45min.
- From our experiments with training $seq_length = 120$, we observe that the best backprop length is 50, which is about 40% of the total sequence length.

12 Inference Scaling

In order to capture the spatial movement of some specific features, we employ the following inference technique. For each mini-batch, we choose a stride of 30, the first 60 frame clips from each video and average the prediction across these clips. The following code snippet demonstrates the inference technique applied for each mini-batch.

```
pred = np.zeros((batch_size,101),np.float32)
stride = 30
length_of_sequence = 60
for test_run in xrange(test_runs):
    rnn_input_data = rnn_input_data_raw[:]
    start = test_run * stride
    end = start    + length_of_sequence

    for j in xrange(len(rnn_input_data)):
        if(rnn_input_data[j].shape[0]<1.0*length_of_sequence):
            rnn_input_data[j] = repeatSequence(rnn_input_data[j],1.0*end)

    for j in xrange(len(rnn_input_data)):
        nFrames = rnn_input_data[j].shape[0]
        if(end <= nFrames):
            rnn_input_data[j] = rnn_input_data[j][start:end,:]

    rnn_input_data_np = np.asarray(rnn_input_data)

    rnn.reset_state()
    rnn.zerograds()
    sequence_length = float(length_of_sequence-1)
    for j in xrange(length_of_sequence-1):
        scale_factor = float(j+1)/length_of_sequence # 1.0

        x = rnn_input_data_np[:,j,:]
        x = cuda.to_gpu(x)
```

```
current_prediction = rnn(x,y,is_train=False,bn_bool=True)

current_prediction.to_cpu()

pred[:,:] += scale_factor*np.log(current_prediction.data)
```

12.1 Test Accuracy Results

Truncation Length	Inference Technique	Majority Voting
Top1 Accuracy	0.6367	0.659
Top5 Accuracy	0.8766	0.881
Top10 Accuracy	0.935	0.940

13 Summary

The final model I used is a sequence length of 150 with 50 truncated backprop. The optimization used in MomentumSGD with Regularization as Weight Decay. I also employed the inference scaling technique and increased the number of training epochs to 50.

Accuracy

```
Top 1 : 0.668
Top 5 : 0.879
Top 10 : 0.94
```

The confusion matrix is also plotted here :

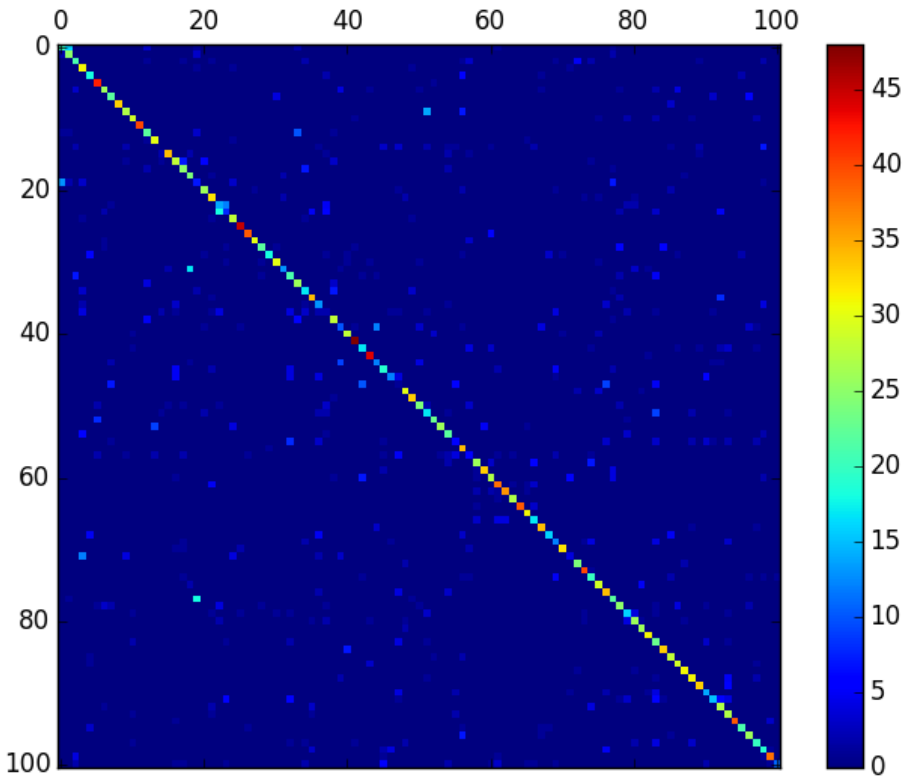
We give an overview of the obvious, reasonable and random misclassifications made by the model in Part 1 and 2 and understand how these classifications improved over time.

- ApplyEyeMakeup is classified as itself only 8 times, whereas it is misclassified as ApplyLipstick 19 times, and BlowDryHair 13 times
- ApplyLipstick is confused with BrushingTeeth 7 times.
- BlowDryHair is confused with HairCut 8 times and HeadMassage 4 times.
- HighJump is confused with FloorGymnatics 14 times

Some still reasonable errors :

- Brushing Teeth is confused with ApplyEyeMakeup 10 times
- CricketShot is confused with CricketBowling 14 times

Figure 1: Confusion matrix for the 101 classes



- FrontCrawl is confused with BreastStroke 27 times
- JumpRope is confused with Basketball 7 times
- Pushups are confused with BabyCrawling 8 times

Some arbitrary errors :

- BoxingPunchingBag is confused with HeadMassage 10 times
- FieldHockeyPenalty is confused with JavelinThrow 7 time
- Nunchuks is confused with diving 7 times, and TrampolinJump 6 times

13.1 Best Classifications

The following are the classes with 100% accuracy:

- FrisbeeCatch

- SoccerPenalty
- Basketball
- Billiards
- BlowDryHair

13.2 Performance Improvements

- Accuracy for ApplyEyeMakeup goes up from 18% to 36%
- Accuracy for HighJump goes up from 18.9% to 33%
- Accuracy for Basketball goes up from 84% to 100%
- Accuracy for BoxingPunchingBag goes up from 36.7% to 58%

13.3 Performance Deterioration

- Accuracy for CricketShot goes down as its confusion with CliffDiving goes up.
- Accuracy for BodyWeightSquats goes down from 6.7% to 0%

13.4 No effect on Performance

- Accuracy for JumpRope remains at 0%.
- Accuracy for BodyWeightSquats goes down from 6.7% to 0%

The classes with best improvement are *JavelinThrow* and *Shotput*. *Skiing* continues to prove to be hard to classify.