AN EMPIRICAL STUDY OF EXAMPLE FORGETTING DURING DEEP NEURAL NETWORK LEARNING

CS6313 - Machine Learning
Fall 2019

Reproduced by
Marco A De Souza Azevedo
Oom Pattarabanjird

Motivation

Inspired by the phenomenon *catastrophic forgetting*:

*Phenomenon that neural network forgets previously learned information when trained for the new task*

Applied to this current work:

*Catastrophic forgetting can cause a problem with mini batch SGD optimization since each batch can be regarded similarly to the new task and SGD optimization is a situation of continuous learning*

**Hypothesis:** shift on input distribution (e.g.: lack of common factors in input lead to different convergent solutions)
Background and Definition

Forgetting event
  Example is misclassified after being correctly classified

Learning event
  Example is classified correctly for the first time

Unforgettable examples
  Learned at some point and never misclassified
Related Work

Curriculum Learning
- Learning with increased difficulty helps minimize task loss
- Safe to remove *unforgettable* examples

Deep Generalization
- Does not depend on complexity of the model
- Overparameterized model can reach low test error
- Generalization is maintained when removing substantial amount of training examples
Claim/Target Task

Goal 1:
Gain insight into the optimization process and the training examples

Goal 2:
Determine if forgetting statistics can be used to identify important samples and outliers
An Intuitive Figure Showing Why Claim

Unforgettable
Clear features
High contrast with sky

Forgettable
Less visible features
Small contrast with background
Proposed Solution

Empirical Analysis:

Train classifier on dataset when sampled in current mini-batch

Sort dataset examples based on number of forgetting samples

MNIST, permuted MNIST and CIFAR-10
Implementation

Number of forgetting events

Histogram of forgetting events of MNIST, permuted MNIST and CIFAR-10

Forgetting by chance

Analyze distribution of forgetting events
Implementation

First learning events
Order of examples during learning

Detection of noisy examples
Atypical characteristics

Distribution of forgetting events in CIFAR-10 when labels are changed
Implementation

Removing most forgettable events

Generalization performance of ResNet18 Increasing number of elements are removed from training set
Load and train MNIST data

Step 1: Load MNIST

```python
[ ] ### M: Load data
    trainset = datasets.MNIST(
        root='/tmp/data', train=True, download=True, transform=transform)
    testset = datasets.MNIST(
        root='/tmp/data', train=False, download=True, transform=transform)
```

Step 2: Train MNIST with parameters below

```python
args_dict = {'dataset': 'mnist', 'batch_size': 64, 'epochs': 200, 'lr': 0.01,
             'momentum': 0.5, 'no_cuda': True, 'seed': 99, 'sorting_file': 'none',
             'remove_n': 0, 'keep_lowest_n': 0, 'no_dropout': True,
             'input_dir': 'mnist_results/', 'output_dir': 'mnist_results'}
```
Load and train MNIST data

Step 3: Call train() on each epoch

```python
elapsed_time = 0
for epoch in range(args.epochs):
    start_time = time.time()

    train(args, model, device, trainset, optimizer, epoch, example_stats)
    test(args, model, device, testset, example_stats)
```

Within train():
Permutate samples;
On each minibatch, get indices of each samples (more on next):

```python
for batch_idx, batch_start_ind in enumerate(
    range(0, len(trainset.targets), batch_size)):

    # Get trainset indices for batch
    # M: Get indices for minibatch (a subset of the permuted indices)
    batch_inds = trainset_permutation_inds[batch_start_ind:
                                          batch_start_ind + batch_size]
```
Load and train MNIST data

Compute outputs, loss, and get predicted class

```python
# Forward propagation, compute loss, get predictions
optimizer.zero_grad()
# M: outputs is the prediction
outputs = model(inputs)
# M: Calculate cross-entropy loss; will only be stored later
loss = criterion(outputs, targets)
# M: The code below get the index of the max in data (a Tensor)
# along 1 dimension
_, predicted = torch.max(outputs.data, 1)
```

Get accuracy and for each sample in mini batch and compute statistics (see next):

```python
# Update statistics and loss
# M: Get list of True and False
acc = predicted == targets
# M: iterate over each index in the mini-batch
for j, index in enumerate(batch_inds):
    # Get index in original dataset (not sorted by forgetting)
    index_in_original_dataset = train_index[index]
    # Compute misclassification margin
    # M: Get the output (log probability) for the correct expected class
    output_correct_class = outputs.data[j, targets[j].item()]
```
Load and train MNIST data

Append statistics (loss, accuracy and margin) to example stats and write to pkl file

```python
# Add the statistics of the current training example to dictionary
# M: the get below defaults to [[]][[],[]]
index_stats = example_stats.get(index_in_original_dataset,
    [[], [], []])
index_stats[0].append(loss[j].item())
# M: each element will be either 1 or 0
index_stats[1].append(acc[j].sum().item())
index_stats[2].append(margin)
example_stats[index_in_original_dataset] = index_stats
```
Load and train CIFAR10 data

Step 1: CIFAR10 was first trained with no sorting, no sample removal, no data augmentation and no cutout

```python
args_dict = { 'dataset': 'cifar10', 'data_augmentation': False, 'cutout': False, 'sorting_file': 'none', 'input_dir': 'cifar10_results', 'output_dir': 'cifar10_results', 'seed': 1, 'remove_n': 0, 'keep_lowest_n': 0, 'remove_subsample': 0, 'noise_percent_labels': 0, 'noise_percent_pixels': 0, 'noise_std_pixels': 0, 'no_cuda': False, 'model': 'resnet18', 'optimizer': 'adam', 'epochs': 100, 'batch_size': 128}
```

For each event, calculate loss and predict output

```python
model_optimizer.zero_grad()
inputs = inputs.cuda()
outputs = model(inputs)
# print(outputs)
loss = criterion(outputs, targets)
# print(loss)
_, predicted = torch.max(outputs.data, 1)
```

Update accuracy, loss stats of each event and save it in example_stats

```python
# Add the statistics of the current training example to dictionary
index_stats = example_stats.get(index_in_original_dataset, [[]], [], []])
# print(index_stats)
index_stats[0].append(loss[j].item())
index_stats[1].append(acc[j].sum().item())
index_stats[2].append(margin)
example_stats[index_in_original_dataset] = index_stats
```
Sort dataset based on forgettable events

Step 1: Use accuracy value after training the CIFAR data to compute whether event is learned or unlearned or forgettable
- Learned event: event with accuracy of 1
- Unlearned event: event with accuracy of 0
- Forgettable event: event with accuracy dropping from 1 to 0

```python
def compute_forgetting_statistics(diag_stats, npresentations):
    presentations_needed_to_learn = {}  # event that accuracy is still 0
    unlearned_per_presentation = {}     # forgettable event
    margins_per_presentation = {}       # misclassified margin for each event
    first_learned = {}                  # event that accuracy = 1 indicating learned event

    unlearned_per_presentation[example_id] = np.where(transitions == -1)[0] + 2
    # Find all presentations when forgetting occurs
    if len(np.where(transitions == -1)[0]) > 0:
        # accuracy drops from 1 to 0 indicates forgettable event
        unlearned_per_presentation[example_id] = {}

    first_learned = {}                  # event that accuracy = 1 indicating learned event
    if len(np.where(transitions == -1)[0]) > 0:
        presentations_needed_to_learn[example_id] = np.where(transitions == -1)[0][-1] + 1
    # Find number of presentations needed to learn example,
    # e.g. last presentation when acc is 0
    if len(np.where(presentation_acc == 0)[0]) > 0:
        presentations_needed_to_learn[example_id] = np.where(presentation_acc == 0)[0][-1] + 1
    else:
        presentations_needed_to_learn[example_id] = 0

    # Find the misclassification margin for each presentation of the example
    margins_per_presentation = np.array(example_stats[2][npresentations])

    # Find the presentation at which the example was first learned,
    # e.g. first presentation when acc is 1
    if len(np.where(presentation_acc == 1)[0]) > 0:
        first_learned[example_id] = np.where(presentation_acc == 1)[0][0]
    else:
        first_learned[example_id] = np.nan
```
Sort dataset based on forgettable events

Step 2: Sort the example_stats to rank the sample from the highest forgetting count to the lowest forgetting count

```python
print('Number of unforgettable examples: {}'.format(
    len(np.where(np.array(example_stats) == 0)[0])))
return np.array(example_original_order)[np.argsort(
    example_stats)], np.sort(example_stats)
```

Step 3: Save the sorted file with a stat of sample ID and sample values

```python
# Sort examples by forgetting counts in ascending order, over one or more training runs
ordered_examples, ordered_values = sort_examples_by_forgetting(
    unlearned_per_presentation_all, first_learned_all, args.epochs)
print(ordered_examples)
print(ordered_values)

# Save sorted output
if args.output_name.endswith('.pkl'):
    with open(os.path.join(args.output_dir, args.output_name),
              'wb') as fout:
        pickle.dump(
            {
                'indices': ordered_examples,
                'forgetting counts': ordered_values
            }, fout)
```
Train CIFAR10 with random data removal

```python
args_dict = {'dataset': 'cifar10', 'data_augmentation': False, 'cutout': False, 'sorting_file': 'none', 'input_dir': 'cifar10_results', 'output_dir': 'cifar10_results', 'seed': 1, 'remove_n': 20000, 'keep_lowest_n': -1, 'remove_subsample': 0, 'noise_percent_labels': 0, 'noise_percent_pixels': 0, 'noise_std_pixels': 0, 'no_cuda': False, 'model': 'resnet18', 'optimizer': 'adam', 'epochs': 50, 'batch_size':}

# Random removal: permute the training data and remove samples
if args.keep_lowest_n < 0:
    # Remove remove_n number of examples from the train set at random
    train_index = npr.permutation(np.arange(len(
        train_dataset.train_labels)))[len(train_dataset.train_labels):len(train_dataset.train_labels) - args.remove_n]

Call the example_stats for accuracy

test_acc_remove_20000 = max(example_stats['test'][1])
print(test_acc_remove_20000)
```
Train CIFAR10 with sorted data removal

```
args_dict = {'dataset': 'cifar10', 'data_augmentation': False, 'cutout': False, 'sorting_file': 'cifar10_sorted', 'input_dir': 'cifar10_results', 'output_dir': 'cifar10_results', 'seed': 1, 'remove_n': 20000, 'keep_lowest_n': 0, 'remove_subsample': 0, 'noise_percent_labels': 0, 'noise_percent_pixels': 0, 'noise_std_pixels': 0, 'no_cuda': False, 'model': 'resnet18', 'optimizer': 'adam', 'epochs': 50, 'batch_size': 128}

if args.sorting_file == 'none':
    train_indx = np.array(range(len(train_dataset.train_labels)))
    train_indx = np.array(range(len(train_dataset.targets)))
else:
    try:
        with open(os.path.join(args.input_dir, args.sorting_file) + '.pkl', 'rb') as fin:
            ordered_indx = pickle.load(fin)['indices']
    except IOError:
        with open(os.path.join(args.input_dir, args.sorting_file), 'rb') as fin:
            ordered_indx = pickle.load(fin)['indices']

    # Get the indices to remove from training
    # 0: number of remove_n
    elements_to_remove = np.array(ordered_indx)[args.keep_lowest_n:args.keep_lowest_n + args.remove_n]

    # Remove the corresponding elements
    train_indx = np.setdiff1d(range(len(train_dataset.train_labels)), elements_to_remove)
```

Sorted removal: Used the sorted file output and remove the samples with the highest forgettable events by using ordered_indx
Load and train CIFAR10 data with noisy labels

Compute number of labels to change

```python
# Compute number of labels to change
nlabels = len(train_dataset.targets)
nlabels_to_change = int(args.noise_percent_labels * nlabels / 100)
nclasses = len(np.unique(train_dataset.targets))
print('flipping ' + str(nlabels_to_change) + ' labels')
```

For each label, introduce noise by changing to another label

```python
# Flip each of the randomly chosen labels
for l, label_ind_to_change in enumerate(labels_inds_to_change):
    # Possible choices for new label
    label_choices = np.arange(nclasses)
    # Get true label to remove it from the choices
    true_label = train_dataset.targets[label_ind_to_change]
    # Remove true label from choices
    label_choices = np.delete(
        label_choices,
        true_label)  # the label is the same as the index of the label
    # Get new label and relabel the example with it
    noisy_label = npr.choice(label_choices, 1)
    train_dataset.targets[label_ind_to_change] = noisy_label[0]
```
Graph examples with noisy labels

Get indices of examples with noisy labels from file

```python
noisy_labels = []
with open('"/Users/marcouzevedo/Documents/UVa/Fa"
  ) as label_file:
    for line in label_file:
        number = int(line.split()[0])
        noisy_labels.append(number)
```

Get index of noisy sample from ordered example

```python
new_indices = np.array([])
for n in noisy_labels:  # for each noisy index
    new_index = np.where(ordered_examples == n)  # get index of noisy index in ordered_examples
    new_indices = np.append(new_indices, new_index)  # append index to list
new_indices = new_indices.astype(int)  # cast as int

labels, values = zip(*Counter(ordered_values[new_indices]).items())
width = 1
arr = np.zeros(20)
for i, _ in enumerate(labels):
    arr[i] = values[i]

indexes = np.arange(len(arr))  # FOR MNIST
# plt.bar(indexes, values, width)
# plt.xticks(indexes + width * 0.5, labels)
# FOR CIFAR
plt.bar(indexes, arr.tolist(), width)
plt.xticks(indexes + width * 0.5, indexes)
plt.ylabel('number of forgetting events')
plt.xlabel('samples in training data')
plt.show()
```
Experimental Results

Number of forgetting events vs. number of samples in MNIST (regular and log respectively)
Experimental Results

TRAINING DETAILS

CIFAR10 dataset trained with ResNet18

Training iteration was reduced from 200 epochs in the paper to 50 epochs due to computational cost.

After 50 iterations, highest test accuracy was 77.45%

Selected removed: samples were sorted based on forgettable events and most forgettable events were removed first

Random removed: randomly selected samples to be removed

Note: removing most forgettable examples first does not hurt performance as much as randomly removed samples
Experimental Results

Examples with noisy labels are more likely to be forgotten

TRAINING DETAILS

CIFAR10 dataset trained with ResNet18

Training with 100 epochs (~1 hour on Colab)

Randomly selected 20% of examples to change labels
Experimental Analysis and Conclusion and Future Work

There exists a large set of unforgettable examples

Examples with noisy labels and uncommon features are the most forgettable

Removing a large fraction of forgettable examples does not compromise performance of the neural network

Future work:
The theory behind forgetting is needed to be further investigated

Understand forgetting phenomena within other forms of learning (e.g. speech or text)
Work Split

Marco

Loaded MNIST data
Trained CIFAR10 data to generate figure that compare forgettable events between normal CIFAR10 and noisy CIFAR10
Made slides for paper review

Oom

Used the loaded MNIST data to generate figure showing forgettable events of each sample in the data
Loaded and trained CIFAR10 dataset
Generated the figure showing test accuracy with and without forgettable samples removed
Made slides for the result section
References


References


References


