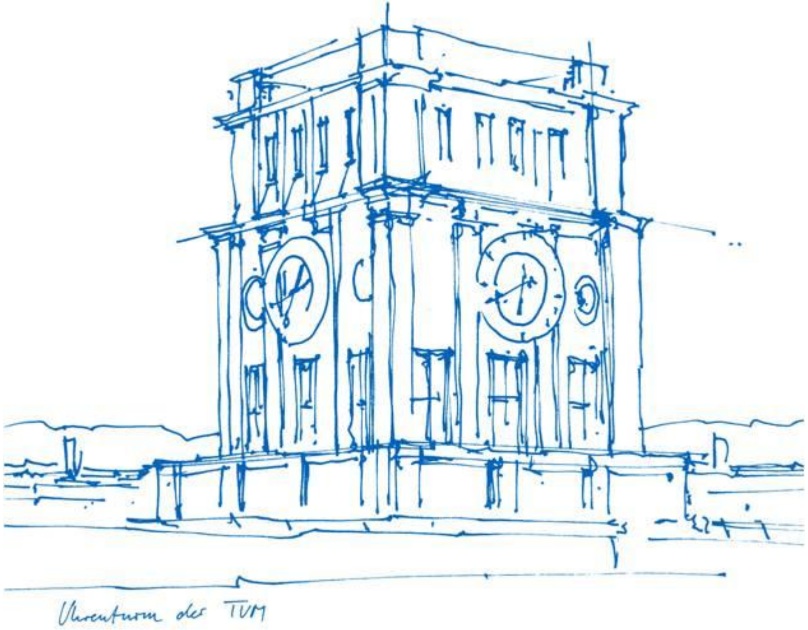
SELECTIVE PARAMETER UPDATING - WEEK 1



Coşku Barış Coşlu

Mato Gudelj

Baykam Say

03 May 2023

Contents

Background Research - Recommended Readings

- Selective Backprop [1]
  - Perform forward pass (every nth epoch)
  - Skip backward pass for low difficulty examples (low loss)
- Weight Update Skipping [2]
  - Skip updating weights and only update bias
  - Go back to normal training if no improvements occur
  - Only update biases for the last few layers

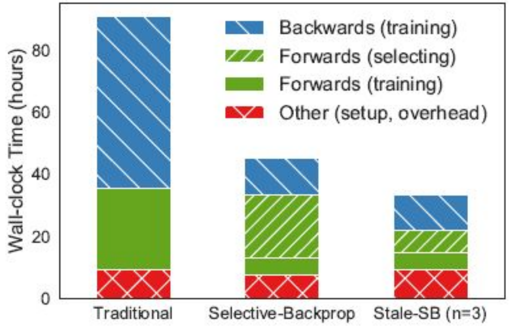


Figure 1: Comparison of selective-backprop approaches [1]

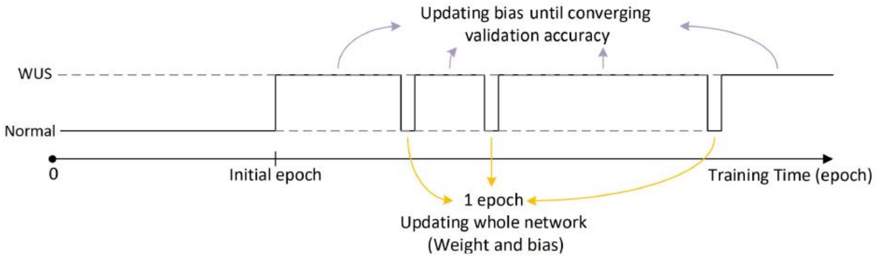


Figure 2: Weight update skipping training cycle [2]

Background Research - Additional Literature

- Adapters [3]
  - Injecting new layers to the network
  - Near identity initialization, trained afterwards
- Estimating Example Difficulty using VoG [4]
  - Could be used in similar setting as [1]
- BitFit [5]
  - Similar to weight update skipping
  - No need for normal training step since only doing fine-tuning
  - Update only a small section of biases
- (IA)^3 [6]
  - Scales activations by learned vectors
- When Do Curricula Work? [7]
  - Explores when exactly difficulty ordering works, empirically
  - Result: Clear benefit in time constrained situations

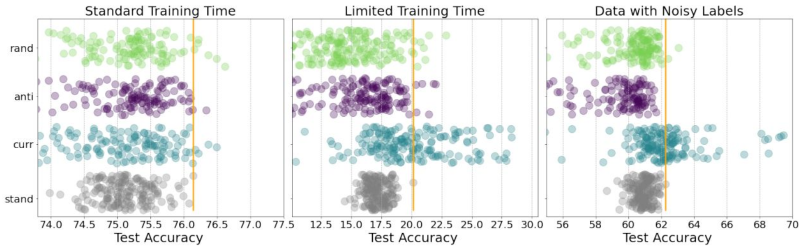


Figure 3: Curricula learning across scenarios - no benefit in unconstrained training, large benefit for time-limited or noisy training

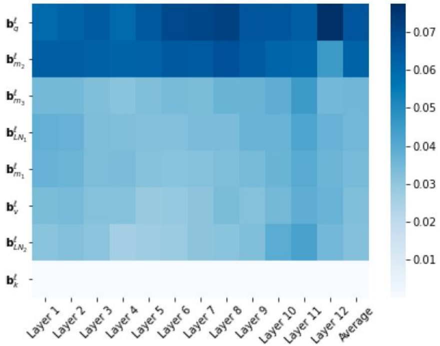


Figure 4: Change in bias components in BitFit [5]

Downstream Task & Dataset

- Sentiment Analysis
  - SST-2 Binary Classification Dataset (GLUE) [8]
- Question Answering
  - SQuAD [9]

Training Loop

- Data preprocessing / tokenization

```python
def load_dataset(dataset = "squad", split = None):
    dataset = huggingface_datasets.load_dataset(dataset, split=split)
    return dataset
```

```python
def tokenize(dataset, tokenizer, max_length):
    # If we need to truncate, truncate the context instead of the question
    tokenized_inputs = tokenizer(
        dataset["question"],
        dataset["context"],
        max_length=max_length,
        truncation="only_second",
        padding="max_length",
        return_offsets_mapping=True
    )

    offset_mapping = tokenized_inputs.pop("offset_mapping")
    start_positions = []
    end_positions = []

    # Update the start and end positions
    for i, offsets in enumerate(offset_mapping):
        answer = dataset["answers"][i]
        # Start/end character index of the answer in the text
        start_char = answer["answer_start"][0]
        end_char = start_char + len(answer["text"][0])

        sequence_ids = tokenized_inputs.sequence_ids(i)
```

```python
        # String of 1s in sequence_ids[i] is the context, find first and last
        context_start = sequence_ids.index(1)
        context_end = len(sequence_ids) - 1 - sequence_ids[::-1].index(1)

        # If the answer is out of the span (in the question) or after the context, set to 0,0
        if end_char < offsets[context_start][0] or start_char > offsets[context_end][1]:
            start_positions.append(0)
            end_positions.append(0)
        else:
            idx = context_start

            while offsets[idx][0] <= start_char and idx < context_end:
                idx += 1
            start_positions.append(idx - 1)

            while idx >= context_start and offsets[idx][1] >= end_char:
                idx -= 1
            end_positions.append(idx + 1)

    tokenized_inputs["start_positions"] = start_positions
    tokenized_inputs["end_positions"] = end_positions

    return tokenized_inputs
```

Training Loop

- Layer freezing

```python
def freeze(model):
    # Set requires_grad to False for all parameters
    for param in model.parameters():
        param.requires_grad = False


def unfreeze_last(model, unfreeze_n = 1):
    # Unfreeze the last n layers
    if "distilbert" in model.name_or_path:
        # DistilBERT
        for param in model.distilbert.transformer.layer[-unfreeze_n:].parameters():
            param.requires_grad = True
    elif "bert" in model.name_or_path:
        # BERT
        for param in model.bert.encoder.layer[-unfreeze_n:].parameters():
            param.requires_grad = True

    # QA head
    for param in model.qa_outputs.parameters():
        param.requires_grad = True
```

Training Loop

- Training on the full dataset or a subset

```python
def train(n_train = None, n_val = None, model_name = "distilbert-base-cased", freeze = True, unfreeze_n = 1):
    # ========= MODEL ========= #
    # Load model
    model = AutoModelForQuestionAnswering.from_pretrained(model_name)
    # Freeze all but last layer + QA head
    if freeze:
        freeze(model)
        unfreeze_last(model, unfreeze_n=unfreeze_n)


    # ========= DATA ========= #
    dataset = load_dataset()
    data_collator = DefaultDataCollator()

    # Take subset of dataset if specified
    if n_train:
        dataset["train"] = dataset["train"].select(range(n_train))
    if n_val:
        dataset["validation"] = dataset["validation"].select(range(n_val))

    # Tokenize the dataset with our tokenization function
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    max_length = model.config.max_position_embeddings
    tokenize_partial = partial(tokenize, tokenizer=tokenizer, max_length=max_length)
    tokenized_dataset = dataset.map(tokenize_partial, batched=True, remove_columns=dataset["train"].column_names)

```

```python

    # ========= TRAINING ========= #
    training_args = TrainingArguments(
        output_dir="results",
        evaluation_strategy="epoch",
        learning_rate=2e-5,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        num_train_epochs=5,
        weight_decay=0.01
    )

    trainer = Trainer(
        model,
        training_args,
        train_dataset=tokenized_dataset["train"],
        eval_dataset=tokenized_dataset["validation"],
        data_collator=data_collator,
        tokenizer=tokenizer
    )

    # Perform validation before training
    print("Evaluating before training (epoch 0)...")
    metrics = trainer.evaluate()
    print(metrics)

    trainer.train()


if __name__ == "__main__":
    train(5000, 500, model_name = "distilbert-base-cased", freeze=False)
```

Generic Update Policy Hook

- UpdatePolicy class
- apply() to enable/disable parameter updates
- Disable parameter by name or freeze entire layers
- Can be extended

```python
class UpdatePolicy:
    def __init__(self, model: Module):
        self.model = model

    best_eval_loss = math.inf
    unfreeze_n = 1

    # Applies the update policy by setting requires_grad on parameters
    def apply(self, epoch=0, metrics=None):
        # Update all parameters
        for param in self.model.parameters():
            param.requires_grad = True

        # Disable updates on individual parameter after the first 2 epochs
        parameter = self.model.get_parameter("distilbert.embeddings.word_embeddings.weight")
        parameter.requires_grad = epoch in range(2)

        # Freeze all layers but the last one if eval_loss is better than best_eval_loss by a margin
        if metrics["eval_loss"] < self.best_eval_loss - 1:
            self.freeze()
            self.unfreeze_last()

        self.best_eval_loss = min(self.best_eval_loss, metrics["eval_loss"])
```

Generic Update Policy Hook

- Manual training loop that injects update policy

```python
# Perform training
for epoch in range(int(training_args.num_train_epochs)):
    # Apply the update policy before each epoch
    update_policy.apply(epoch, metrics)

    for batch in train_dataset.iter(training_args.per_device_train_batch_size):
        batch = {k: torch.tensor(v, dtype=torch.long) for k, v in batch.items()}
        trainer.training_step(model, batch)
        trainer.optimizer.step()
        trainer.optimizer.zero_grad()
        trainer.lr_scheduler.step()
        progress_bar.update(1)

    # Evaluate after each epoch
    print(f"Evaluating epoch {epoch + 1}...")
    metrics = trainer.evaluate()
    print(metrics)
```

Open Issues & Solutions

- Issues
  - Which dataset to choose?
    - SQuAD or SST
  - Which model to use?
    - BERT? DistilBERT?
- Solutions
  - For both
    - Look at relevant literature
    - Pick model and dataset with most relevant datapoints
    - Goal: Easy and fair comparisons with existing methods

Plan for the Next Two Weeks

- Train a full baseline, compare with known numbers to validate training code
- Extend training code with proper logging (TensorBoard)
- Extend training code with an extensible config system
- Brainstorm potential approaches
  - By next meeting shortlist a few and implement at least one
- …keep reading literature

References

[1] A. H. Jiang *et al.*, ‘Accelerating Deep Learning by Focusing on the Biggest Losers’, *arXiv [cs.LG]*. 2019.

[2] P. Safayenikoo and I. Akturk, ‘Weight Update Skipping: Reducing Training Time for Artificial Neural Networks’, *arXiv [cs.LG]*. 2020.

[3] N. Houlsby *et al.*, ‘Parameter-Efficient Transfer Learning for NLP’, *arXiv [cs.LG]*. 2019.

[4] Agarwal, C., & Hooker, S. (2020). Estimating Example Difficulty using Variance of Gradients. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10358-10368.

[5] E. B. Zaken, S. Ravfogel, and Y. Goldberg, ‘BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models’, *arXiv [cs.LG]*. 2022.

[6] H. Liu *et al.*, ‘Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning’, *arXiv [cs.LG]*. 2022.

[7] Wu, X., Dyer, E., & Neyshabur, B. (2020). When Do Curricula Work? ArXiv, abs/2012.03107.

[8] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, ‘GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding’, *arXiv [cs.CL]*. 2019.

[9] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, ‘SQuAD: 100,000+ Questions for Machine Comprehension of Text’, *arXiv [cs.CL]*. 2016.