HyperSched

Deadline-aware Scheduler for Model Development

Richard Liaw, Romil Bhardwaj, Lisa Dunlap, Yitian Zou,
Joseph E. Gonzalez, Ion Stoica, Alexey Tumanov
Data Science @ Boogle Inc.
Learning Rate?
Momentum??
Network Size?
Preprocessing Parameters???
Featurization?????
Learning Rate?
Momentum??
Network Size?
Preprocessing Parameters???
Featurization?????
How to optimize?
Try Random Search
How to optimize?
Try Random Search
Terri is faced with the decision choosing the right level of parallelism.

Trials (sets of hyperparameters to evaluate)
Terri is faced with the decision choosing the right level of parallelism

Trials (sets of hyperparameters to evaluate)
Terri is faced with the decision choosing the right level of parallelism.
Terri is faced with the decision choosing the right level of parallelism.
Terri is faced with the decision choosing the right level of parallelism.
Terri is faced with the decision choosing the right level of parallelism.
DEADLINES EXIST

Scheduling Problem?
DEADLINES EXIST

Scheduling Problem?
Given finite time and compute resources, scheduling problem

Instead of increasing

- DL cluster efficiency
  [OSDI 2018]
- Job Completion Time
  [NSDI 2019, EuroSys 2018]
Given finite time and compute resources, scheduling problem evaluate many random trials (configurations)  

Instead of increasing  
- DL cluster efficiency  
  [OSDI 2018]  
- Job Completion Time  
  [NSDI 2019, EuroSys 2018]
Given finite time and compute resources, the Scheduling Problem evaluate many random trials (configurations) to obtain the best trained model. Instead of increasing:

- DL cluster efficiency [OSDI 2018]
- Job Completion Time [NSDI 2019, EuroSys 2018]
HyperSched is an application-level scheduler for model development.
HyperSched is an application-level scheduler for model development.

- Balances *explore* and *exploit* by adaptively allocating resources based on:
HyperSched is an application-level scheduler for model development.

- Balances **explore** and **exploit** by adaptively allocating resources based on:
  - Awareness of resource constraints
HyperSched is an application-level scheduler for model development.

- Balances **explore** and **exploit** by adaptively allocating resources based on:
  - Awareness of resource constraints
  - Awareness of training objectives
Properties/Assumptions of model development workloads
Properties/Assumptions of model development workloads

Model development consists of evaluating many trials.
Properties/Assumptions of model development workloads

Model development consists of evaluating many trials.

• Each trial is iterative and returns intermediate results
Properties/Assumptions of model development workloads

Model development consists of evaluating many trials.

- Each trial is iterative and returns intermediate results
- Trials can be checkpointed during training.
Properties/Assumptions of model development workloads

Model development consists of evaluating many trials.

- Each trial is iterative and returns intermediate results
- Trials can be checkpointed during training.
- All trials share the same objective. Care only about 1 model.
Properties/Assumptions of model development workloads

Model development consists of evaluating many trials.

- Each trial is iterative and returns intermediate results.
- Trials can be checkpointed during training.
- All trials share the same objective. Care only about 1 model.
- Model training can be accelerated by parallelizing/distributing its workload (data parallelism).
How to use allocation for *exploration* and *exploitation*
Naive Approach: Static Space/Time Allocation
Naive Approach: Static Space/Time Allocation

Exploration

TIME

# GPU
Naive Approach: Static Space/Time Allocation
Naive Approach: Static Space/Time Allocation
Problem: Initial Performance is a weak proxy of final behavior
Naive Solution: Static Space/Time Allocation

Underallocate exploration...
Naive Solution: Static Space/Time Allocation

... or underallocate exploitation
Naive Solution: Static Space/Time Allocation

Main problem: Cannot rely on initial performance.
Better Solution:
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]
Better Solution:
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- Distributed hyperparameter tuning algorithm based off optimal resource allocation.
Better Solution:
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- Distributed hyperparameter tuning algorithm based off optimal resource allocation.
- SOTA results over other existing algorithms
Better Solution:  
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- Distributed hyperparameter tuning algorithm based off optimal resource allocation.
- SOTA results over other existing algorithms
- Deployed on many AutoML offerings today
Better Solution:
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- $r$: min. epoch
- $R$: max epoch
- $\eta$ (eta): Balance explore/exploit
- Intuition: Progressively allocate more resources to promising trials

*Simplified representation*
Better Solution: Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- $r$: min. epoch
- $R$: max epoch
- $\eta$ (eta): Balance explore/exploit
- Intuition: Progressively allocate more resources to promising trials

LIMIT = $r$

```
while trial.iter < R:
    trial.run_one_epoch()
```
Better Solution:
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- \( r \): min. epoch
- \( R \): max epoch
- \( \eta \) (eta): Balance explore/exploit
- **Intuition**: Progressively allocate more resources to promising trials

\[
\text{LIMIT} = r
\]

while trial.iter < R:
    trial.run_one_epoch()
if trial.iter == LIMIT:
- $r$: min. epoch
- $R$: max epoch
- $\eta$ (eta): Balance explore/exploit
- **Intuition**: Progressively allocate more resources to promising trials

**Better Solution:**
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

```python
LIMIT = r
while trial.iter < R:
    trial.run_one_epoch()
    if trial.iter == LIMIT:
        if is_top(trial, LIMIT, 1/\eta):
```

* Simplified representation
Better Solution: Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- \( r \): min. epoch
- \( R \): max epoch
- \( \eta \) (eta): Balance explore/exploit
- **Intuition:** Progressively allocate more resources to promising trials

\[
\text{LIMIT} = r
\]

\[
\text{while trial.iter < R:}
\]
\[
\text{trial.run_one_epoch()}
\]
\[
\text{if trial.iter == LIMIT:}
\]
\[
\text{if is_top(trial, LIMIT, 1/\eta):}
\]
\[
\text{LIMIT *= \eta}
\]

* Simplified representation
- $r$: min. epoch
- $R$: max epoch
- $\eta$ (eta): Balance explore/exploit
- **Intuition**: Progressively allocate more resources to promising trials

![Graph showing accuracy over time with simplified representation.]

**Better Solution:**
Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

```
LIMIT = r
while trial.iter < R:
    trial.run_one_epoch()
    if trial.iter == LIMIT:
        if is_top(trial, LIMIT, 1/\eta):
            LIMIT *= \eta
    else:
```

* Simplified representation
Better Solution: Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

- \( r \): min. epoch
- \( R \): max epoch
- \( \eta \) (eta): Balance explore/exploit
- Intuition: Progressively allocate more resources to promising trials

\[
\text{LIMIT} = r
\]

\[
\text{while trial.iter < R:}
\]
\[
\text{trial.run_one_epoch()}
\]
\[
\text{if trial.iter == LIMIT:}
\]
\[
\text{if is_top(trial, LIMIT, 1/\eta):}
\]
\[
\text{LIMIT *= } \eta
\]
\[
\text{else:}
\]
\[
\text{# allow new trials to start}
\]
\[
\text{trial.pause(); break}
\]

* Simplified representation
Better Solution: Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

Benefit: Mitigate noisy initial performance by adaptive allocation
Better Solution: Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

Benefit: Mitigate noisy initial performance by adaptive allocation

How to improve?
Better Solution: Asynchronous Successive Halving Algorithm (ASHA) [Li2018]

Benefit: Mitigate noisy initial performance by adaptive allocation

How to improve?
HyperSched Solution
HyperSched Solution

1. Build on ASHA’s adaptive allocation
HyperSched Solution

1. Build on ASHA’s adaptive allocation
2. Avoid starting trials close to deadline
HyperSched Solution

1. Build on ASHA’s adaptive allocation
2. Avoid starting trials close to deadline
3. Consolidate parallel resources to top trial near deadline to maximize accuracy
HyperSched: Early Termination
Build on ASHA’s adaptive allocation.

From ASHA:
- Evaluate trials for min. epoch $r$ - up to max epoch $R$
- Balance explore/exploit with parameter $\eta$
- Mitigate problem of noisy initial performance
HyperSched: Admission Policy

Avoid starting trials close to deadline

# GPU

TIME

Accuracy

TIME
HyperSched: Admission Policy

Avoid starting trials close to deadline

- R: max epoch
- η: Explore/exploit parameter
HyperSched: Admission Policy

Avoid starting trials close to deadline

- **R**: max epoch
- **\(\eta\)**: Explore/exploit parameter
- **Intuition**: Only start trials if they have a chance of beating incumbent

```python
def should_start_trial():
    return T_left > min( 
        furthest_trial().time * \(\eta\), 
        base_epoch_time*R)
```

```python
Accuracy
```

```python
TIME
```

```python
# GPU
TIME
```

```python
def should_start_trial():
    return T_left > min( 
        furthest_trial().time * \(\eta\), 
        base_epoch_time*R)
```
HyperSched: Resource Reallocation

Dynamically allocate parallel resources to final trials

![Diagram showing the allocation of # GPU over time and the corresponding accuracy over time.](image)
HyperSched: Resource Reallocation

Dynamically allocate parallel resources to final trials

- Uniform Allocation of available resources

```python
def on_result(trial):
    if should_stop(trial):
        update_allocation()
    return
```
HyperSched: Resource Reallocation

Dynamically allocate parallel resources to final trials

- Uniform Allocation of available resources
- Resize by checkpointing and starting again with more parallel workers

```python
def on_result(trial):
    if should_stop(trial):
        update_allocation()
        return
    elif should_resize(trial):
        ckpt = trial.checkpoint()
        set_allocation(trial)
        trial.restart(ckpt)
```
HyperSched Implementation
HyperSched leverages Ray Tune’s scheduler API

http://tune.io/
HyperSched Implementation
HyperSched Implementation

- Trials return intermediate information (performance, overhead)
HyperSched Implementation

- Trials return intermediate information (performance, overhead)
- Maintains internal allocation mapping and deadline timer
HyperSched Implementation

- Trials return intermediate information (performance, overhead)
- Maintains internal allocation mapping and deadline timer
- Uses Tune Scheduling APIs for execution (resizing, checkpointing, pausing, etc).

Trials return intermediate information (performance, overhead)

- Maintains internal allocation mapping and deadline timer
- Uses Tune Scheduling APIs for execution (resizing, checkpointing, pausing, etc).
HyperSched Implementation

- Trials return intermediate information (performance, overhead)
- Maintains internal allocation mapping and deadline timer
- Uses Tune Scheduling APIs for execution (resizing, checkpointing, pausing, etc).
- Does not manage physical placement decisions
Overview of HyperSched Results

For more results, see paper + poster.
CIFAR10 Experiment

Setup:
- 1 hour deadline, 8 GPUs (V100)
- Resnet50 on CIFAR10
- 144 different hyperparameter configurations

https://github.com/kuangliu/pytorch-cifar (2.3k stars)
CIFAR10 Experiment

Setup:
- 1 hour deadline, 8 GPUs (V100)
- Resnet50 on CIFAR10
- 144 different hyperparameter configurations

https://github.com/kuangliu/pytorch-cifar (2.3k stars)
CIFAR10 Experiment

Setup:
- 1 hour deadline, 8 GPUs (V100)
- Resnet50 on CIFAR10
- 144 different hyperparameter configurations

Accuracy vs Time

GPUs Allocated vs Time

https://github.com/kuangliu/pytorch-cifar (2.3k stars)
CIFAR10 Experiment

Setup:
- 1 hour deadline, 8 GPUs (V100)
- ResNet50 on CIFAR10
- 144 different hyperparameter configurations

https://github.com/kuangliu/pytorch-cifar (2.3k stars)
CIFAR10 Experiment

Setup:
- 1 hour deadline, 8 GPUs (V100)
- Resnet50 on CIFAR10
- 144 different hyperparameter configurations

Accuracy vs Time

Mitigates noisy initial performance

GPUs Allocated vs Time

https://github.com/kuangliu/pytorch-cifar (2.3k stars)
CIFAR10 Experiment

Setup:
- 1 hour deadline, 8 GPUs (V100)
- Resnet50 on CIFAR10
- 144 different hyperparameter configurations

Mitigates noisy initial performance

Achieve 93.84% Val (original repo 93.57%)

https://github.com/kuangliu/pytorch-cifar (2.3k stars)
Performance across deadlines

- ResNet50 model on CIFAR10, (8 V100 GPUs)
- 144 different configurations
Performance across deadlines

- ResNet50 model on CIFAR10, (8 V100 GPUs)
- 144 different configurations
Performance across deadlines

- ResNet50 model on CIFAR10, (8 V100 GPUs)
- 144 different configurations
Performance across deadlines

HyperSched outperforms ASHA across a variety of deadlines by evaluating less trials and exploiting existing trials

<table>
<thead>
<tr>
<th>Deadline (s)</th>
<th>ASHA</th>
<th>HyperSched</th>
</tr>
</thead>
<tbody>
<tr>
<td>900 (s)</td>
<td>0.7</td>
<td>0.85</td>
</tr>
<tr>
<td>1800 (s)</td>
<td>0.75</td>
<td>0.85</td>
</tr>
<tr>
<td>3600 (s)</td>
<td>0.85</td>
<td>0.925</td>
</tr>
</tbody>
</table>

Accuracy

<table>
<thead>
<tr>
<th>Deadline (s)</th>
<th>Trials Evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>900 (s)</td>
<td>22.5</td>
</tr>
<tr>
<td>1800 (s)</td>
<td>45</td>
</tr>
<tr>
<td>3600 (s)</td>
<td>90</td>
</tr>
</tbody>
</table>

Max Accuracy

HyperSched outperforms ASHA across a variety of deadlines by evaluating less trials and exploiting existing trials.
HyperSched Summary

- HyperSched is an application-level scheduler for deadline-based model development

- HyperSched uses constraint-awareness and is informed by application-level objectives to increase model accuracy

- Our evaluation shows HyperSched outperforms state-of-the-art parameter tuning algorithms
HyperSched Summary

- HyperSched is an application-level scheduler for deadline-based model development.
- HyperSched uses constraint-awareness and is informed by application-level objectives to increase model accuracy.
- Our evaluation shows HyperSched outperforms state-of-the-art parameter tuning algorithms.

Thank you! Questions?