LinkedIn

Carnegie Mellon University

LinkedIntern
Class of 2019
Summer

Scalable Automatic Machine Learning

Learning to Learn on the Cloud

Yuwei Qiu
AI Algorithm Foundation Team

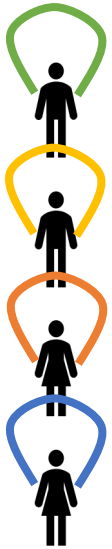
Agenda

- Motivation
- Problem
- Algorithm
- Design
- Result

Motivation

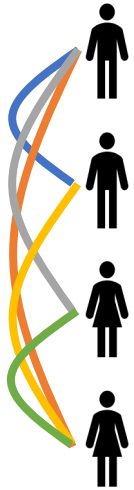
What does in do everyday?

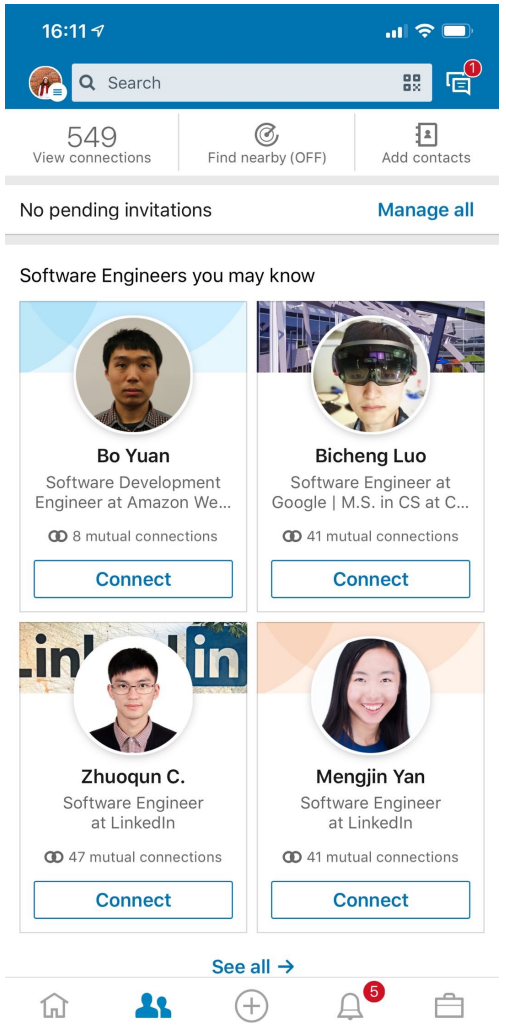
- **Connection!**
  - Connection between **member and themselves**
    - Feed!



What does in do everyday?

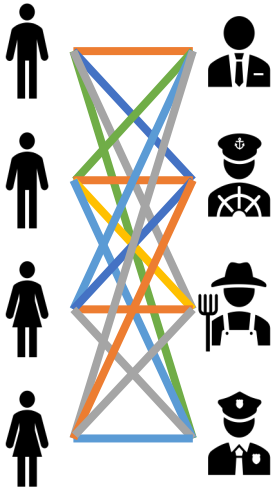
- **Connection!**
  - Connection between **member and member**
    - People You May Know!

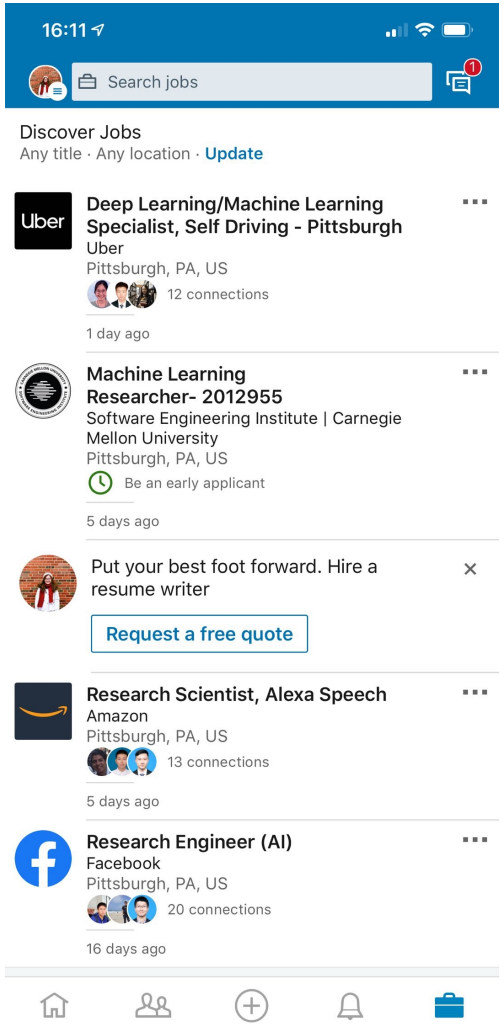




What does in do everyday?

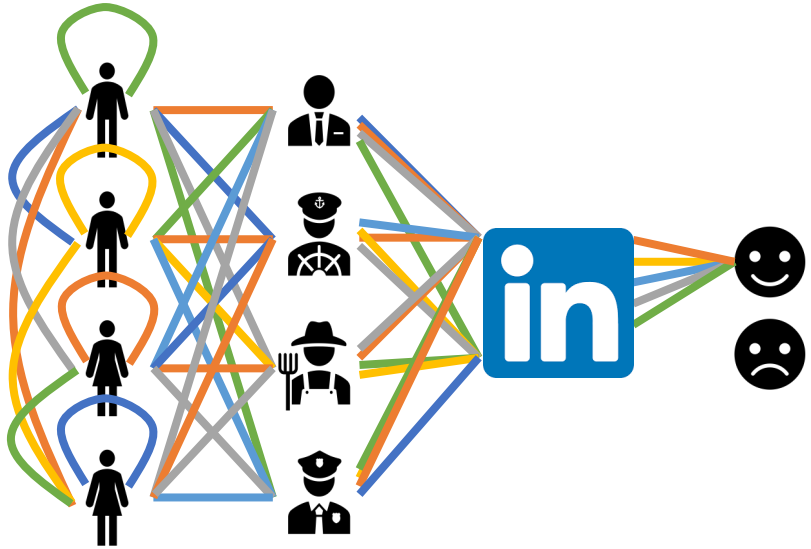
- **Connection!**
  - Connection between **member and job**
    - Job You May Be Interested In!





What does in do everyday?

- in filters out unpleasant connections.
- And all of these connections are operated in our engineering teams in the form of **Data.**



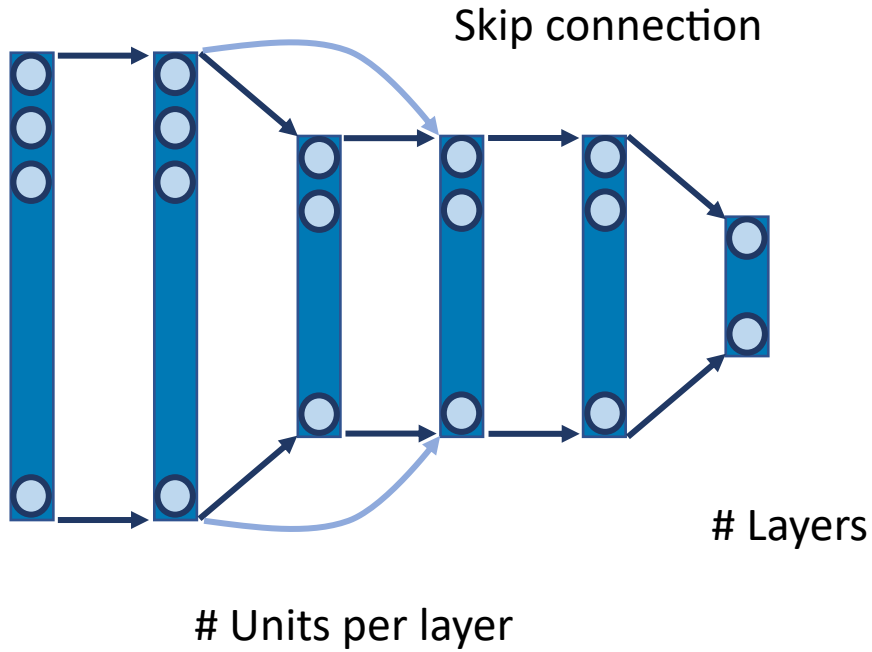
What do 's engineers do everyday?

- We build platforms/algorithms/tools to **learn** the data.
- Data come and go, tasks come and go, but **learning is forever**.

One problem of Learning

- Performance is very **sensitive** to many **parameters**
- Architectural hyperparameters & Others

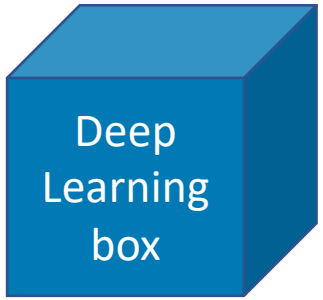
(e.g. Feedforward Network)



Optimization algorithm,
Learning rates,
Momentum,
Batch normalization,
Batch sizes,
Dropout rates,
Weight decay,
Data augmentation,
Activation function
...

Traditional Deep Learning pipeline

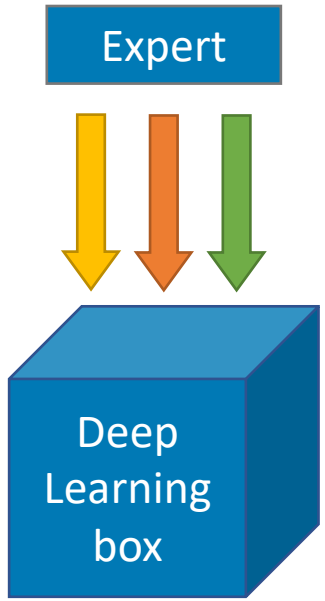
- Clean & preprocess the data
- Select better features
- Select a model family
- Select the optimizers
- …
- …
- Training and validation



Deep learning **end-to-end**

Traditional Deep Learning pipeline

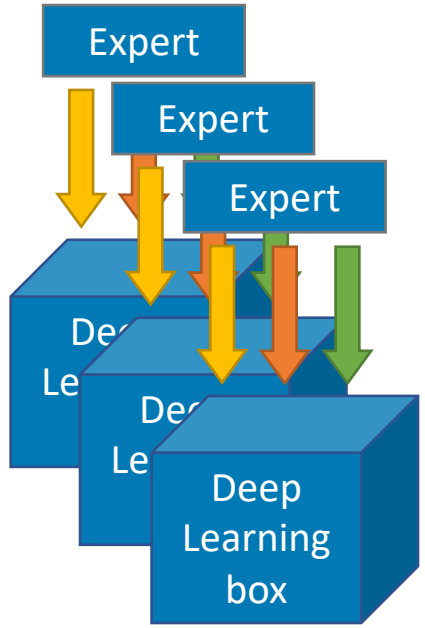
- Clean & preprocess the data
- Select better features
- Select a model family
- Select the optimizers
- …
- …
- Training and validation



Deep learning **“end-to-end”**

Traditional Deep Learning pipeline

- *While* ***True****:*
  - Clean & preprocess the **new data**
  - Select better features
  - Select a model family
  - Select the optimizers
  - …
  - …
  - Training and validation



…

- Whenever new data/datasets come, we must re-do all these steps.

Current:

Solution =

Expertise + Data + N × Hours

Scalable
Automatic Tuning

But can we turn this into:

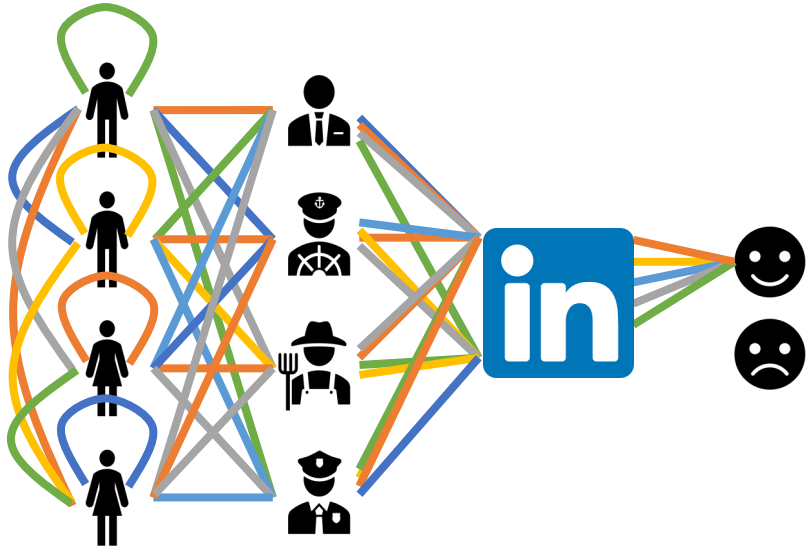
Solution =

Data + N × Hours (+ Sacrifice of Performance)

LinkedIntern
Yuwei Qiu

What is it inside in ?

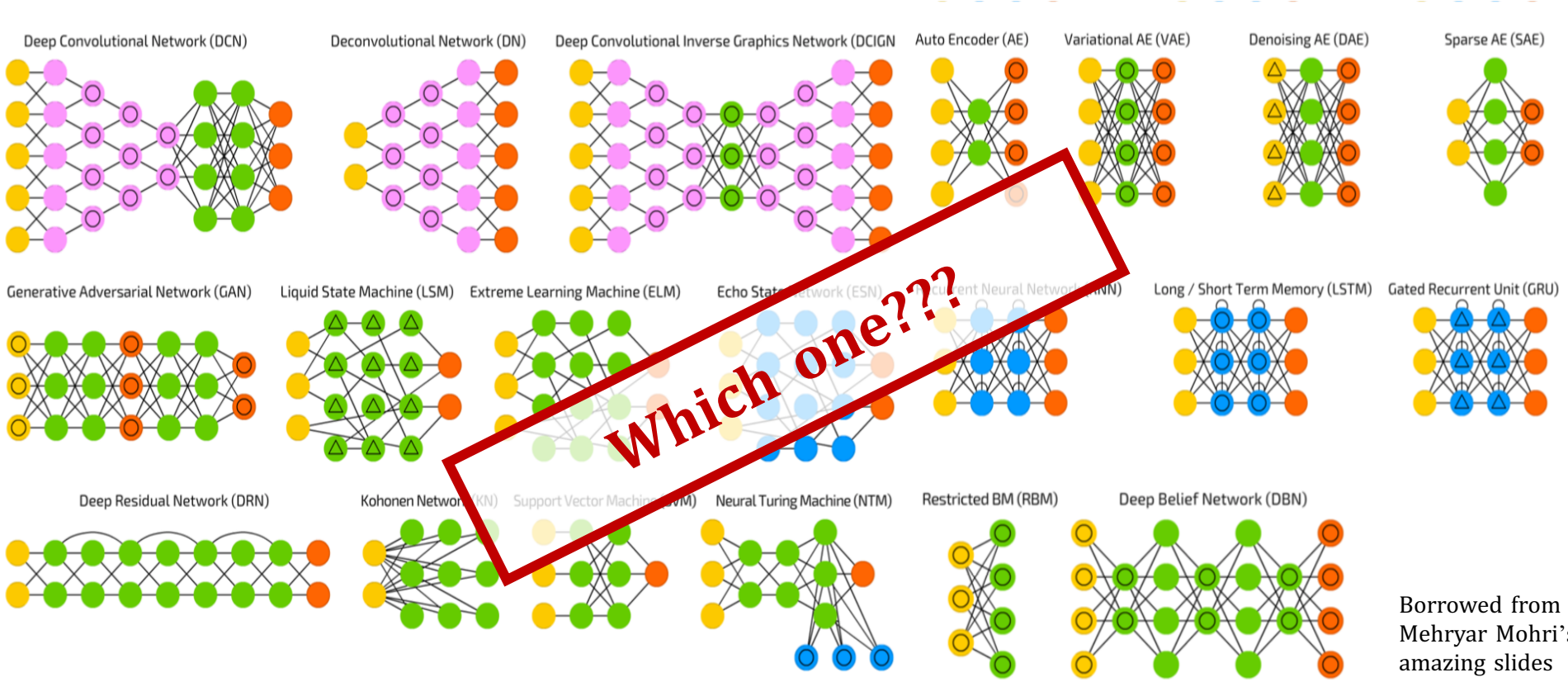
- Theoretically, **how to find it?** —*Algorithm*
- Engineeringly, **how to deploy it?** —*Design*



Algorithm

How to find in ?

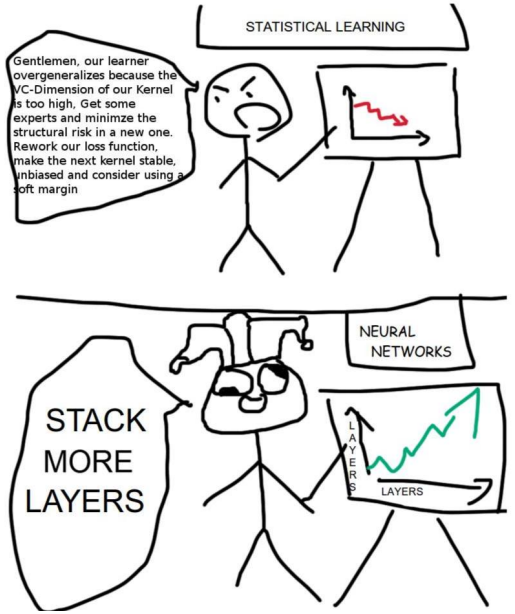
- Some possible answers…



Borrowed from Mehryar Mohri's amazing slides

How to find in ?

- Find Deep Learning Experts!
  - **Manual Tune**



How to find in ?

Method	Expertise	Exploring Time	Resources	Accuracy
Manual Tune	Yes	Depends…	Normal	Depends

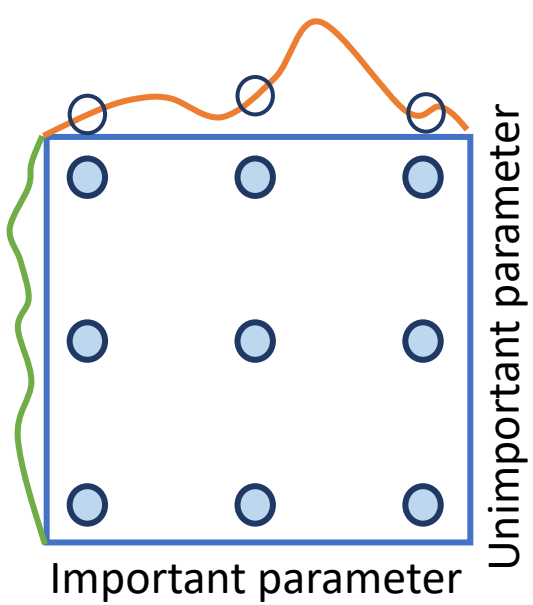
- Manual Tune
  - Expertise in Deep Learning is a requirement
  - Speed and Accuracy, somehow, depend on personal ability…

How to find in ?

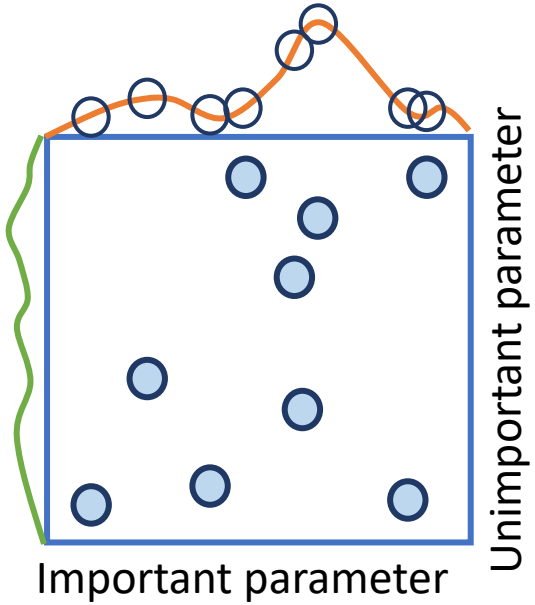
- Find Deep Learning Experts!
  - **Manual Tune**
- Well, just try out enough sets of hyperparameter and structures
  - **Grid/Random search**

Grid Search and Random Search

- Shoot all of these structures, all hyperparameters. Then find the best set of parameters
- Random search handles unimportant dimensions better



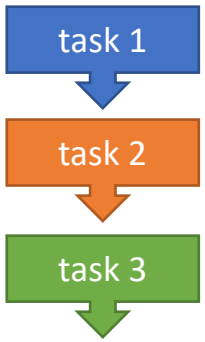
Grid Layout



Random Layout

Grid Search and Random Search

- If you want to grid search or random search on N set of parameters…
  - Typically **much more time or much more resources**.



Vertically: N times more time

Horizontally: N times more resources

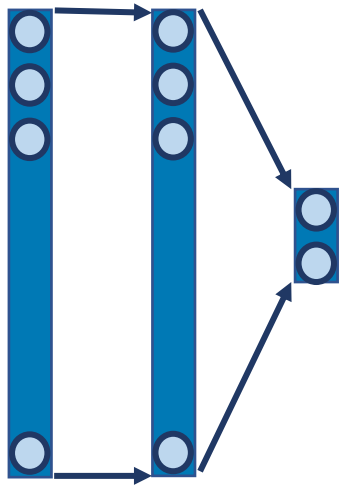
How to find in ?

Method	Expertise	Exploring Time	Resources	Accuracy
Manual Tune	Yes	Depends…	Normal	Depends
Grid Search	No	Very Very Long	Very Very Much	Good!
Random Search	No	Very Long	Very Much	Good!

- Grid Search and Random Search
  - Most of the time a fairly good enough solution can be reached.
  - Sacrifice the resources and training/validation time.
    - The overhead of **heavyweight structure exploration** is inevitable.
    - This is **disastrous** in online services.

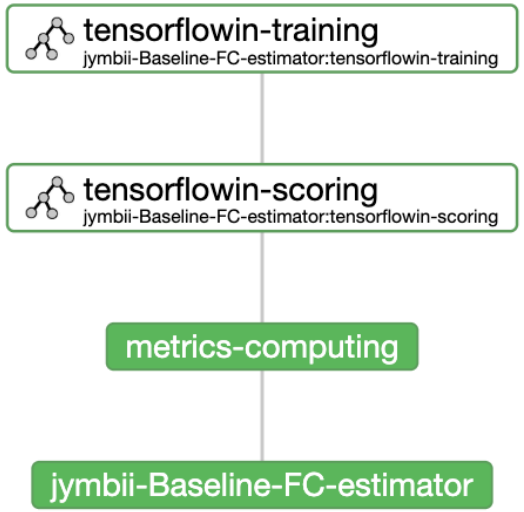
Disastrous Heavyweight Structure Exploration

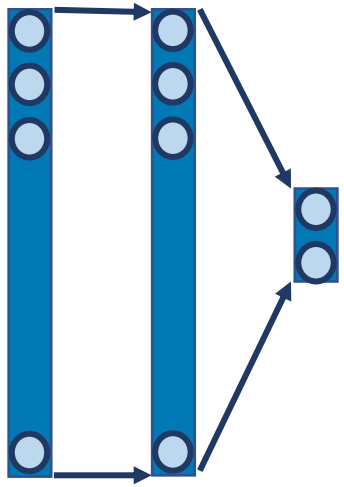
- What does it cost to train a heavyweight structure over and over again on a large-scale dataset?
- And what does it mean in real-world services?



On jymbii
Feature dim ~5000
1 hidden layer dim =5000
Output dim =2
20 workers

Disastrous Heavyweight Structure Exploration





On jymbii
Feature dim ~5000
1 hidden layer dim =5000
Output dim =2
20 workers

Name	Type	Timeline	Start Time	End Time	Elapsed	Status	Details
tensorflowin-training	flow		2019-07-10 16:36 19s	2019-07-10 18:21 45s	1h 45m 27s	Success	
tensorflowin-scoring	flow		2019-07-10 18:21 45s	2019-07-10 18:47 32s	25m 45s	Success	
metrics-computing	spark		2019-07-10 18:47 32s	2019-07-10 18:54 38s	7m 6s	Success	Log
jymbii-Baseline-FC-estimator	noop		2019-07-10 18:54 38s	2019-07-10 18:54 38s	0 sec	Success	Log

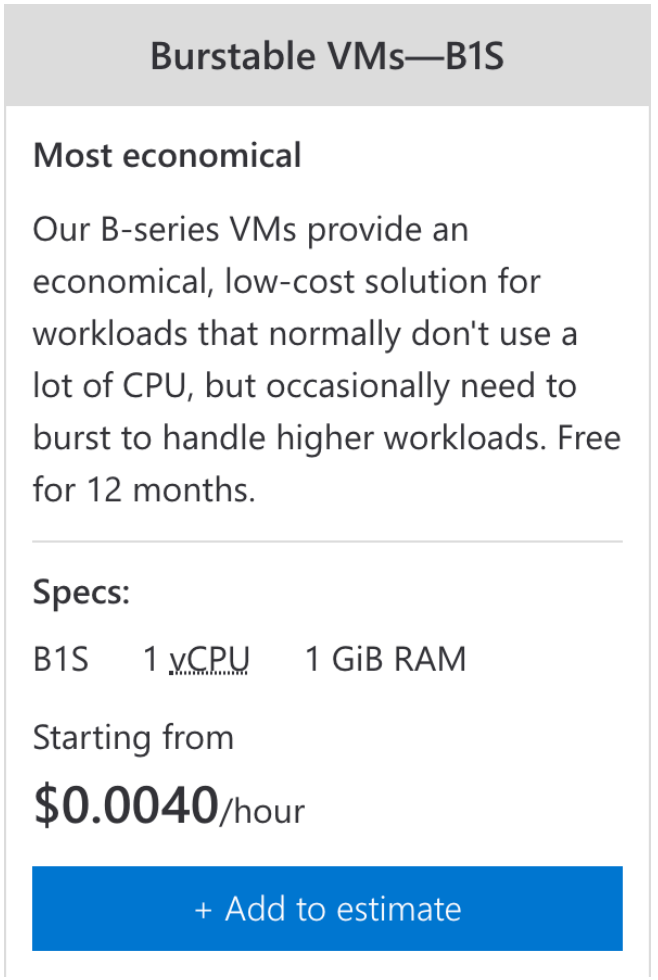
Disastrous Heavyweight Structure Exploration

- 1 hidden-layer vanilla FCN:
  - Training plus Validation
    - **~ 2 hours / 20 workers**
- What does it mean in real-world services?

Most Economical, Low-Cost solution:

1 vCPU, 1 GiB RAM, 1 hour

= $0.0040



Disastrous Heavyweight Structure Exploration

- Suppose you have:
  - 20 workers (each one with 4vCPU and 32GB RAM).
- The dataset is updated weekly
  - which means you need to **REDO all explorations** weekly.

Disastrous Heavyweight Structure Exploration

- Just based on the statistics of previous experiments, let's do the math.
  - If you, a Deep Learning expert, manually tuned the networks
    - You spend $2(\text{trials}) \times 2(\text{hour}) \times 20 \times 4 \times 32 \times \$0.0040 = \$40.96$ per week
  - If you use Bayesian Optimization:
    - you spend $8(\text{trials}) \times 2(\text{hour}) \times 20 \times 4 \times 32 \times \$0.0040 = \$163.84$ per week
  - If you use Grid/Random Search:
    - you spend $12(\text{trials}) \times 2(\text{hour}) \times 20 \times 4 \times 32 \times \$0.0040 = \$245.78$ per week
  - If you, a Deep Learning new bird, manually tuned the networks
    - ……

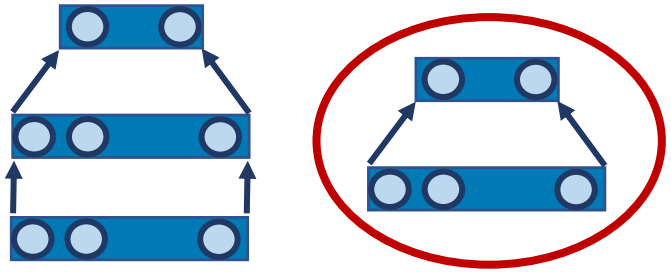
How to find in ?

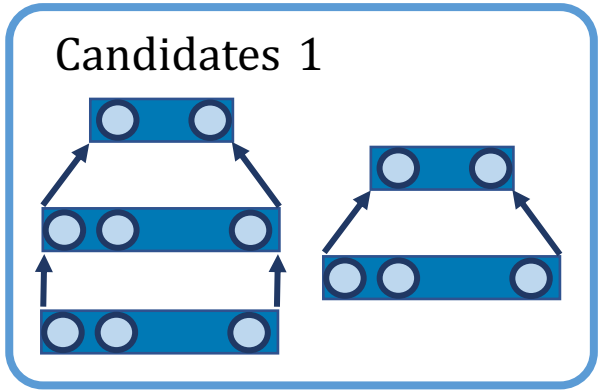
- Find Deep Learning Experts!
  - **Manual Tune**
- Well, just try out different sets of hyperparameter and structures
  - **Grid/Random search**
- Emmm… Is there a way not to explore heavy networks?
  - **AdaNet!**

AdaNet

- **“Instead of start with heavyweight structure, AdaNet grows with lightweight subnetworks.”**
- Presented in “*AdaNet: Adaptive Structural Learning of Artificial Neural Networks*” at *ICML 2017*

Iteration 1

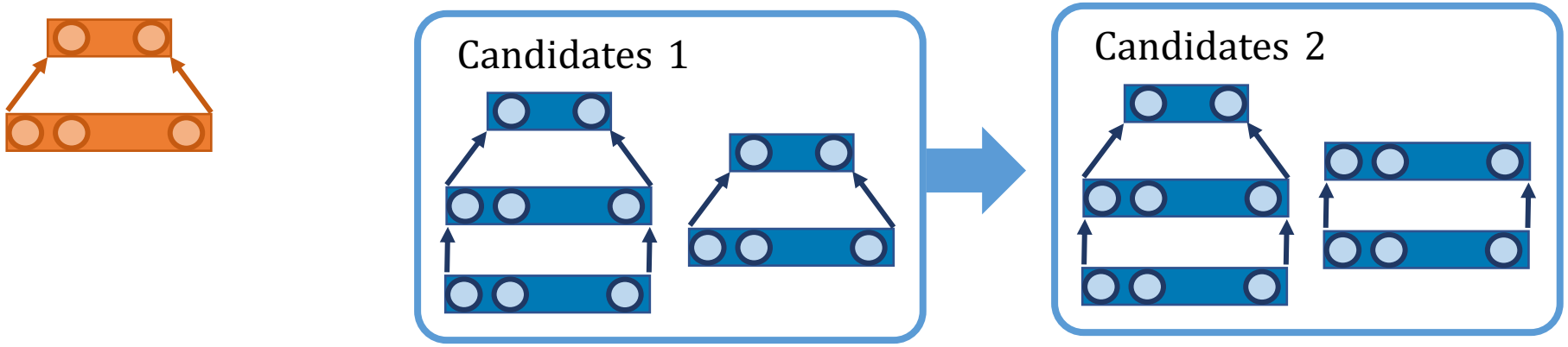




- Start with random small subnetworks and train them **separately**
- Choose the one leading to the best reduction of **evaluation metric function**.

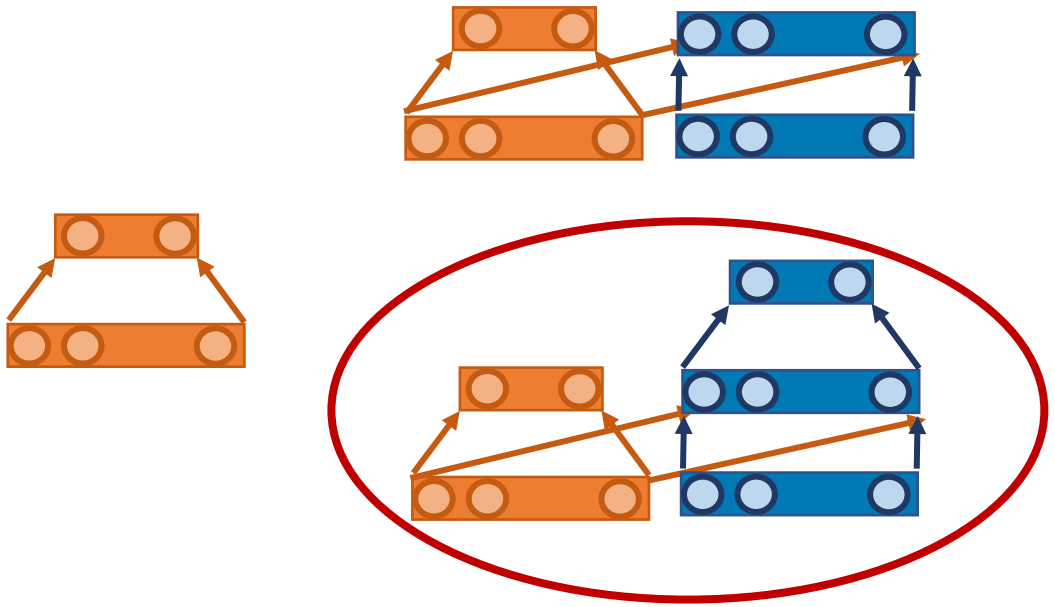
Iteration 1

Example “Specific Rules”
1- Same depth but different size
2- Same size but one more layer

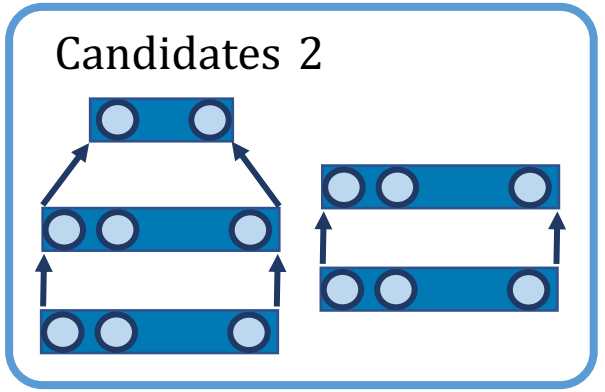


- Based on the evaluation results of these subnetwork, expand candidates pool under **specific rules**.
- **Train and re-train** the subnetworks in the expanded candidates pool.

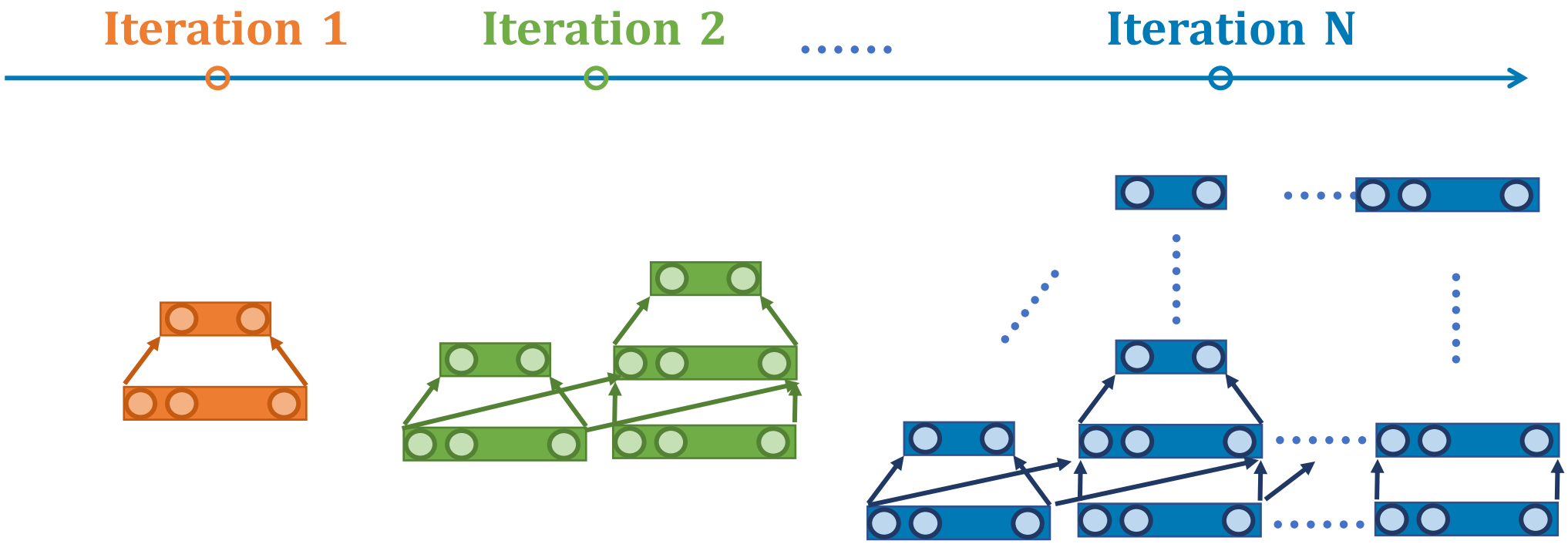
Iteration 1 — Iteration 2



Example “Specific Rules”
1- Same depth but different size
2- Same size but one more layer



- Augment candidates with the current network.
- Then **add and train new connections** between subnetworks.
- Choose the one leading to the best reduction of **evaluation metric function**.



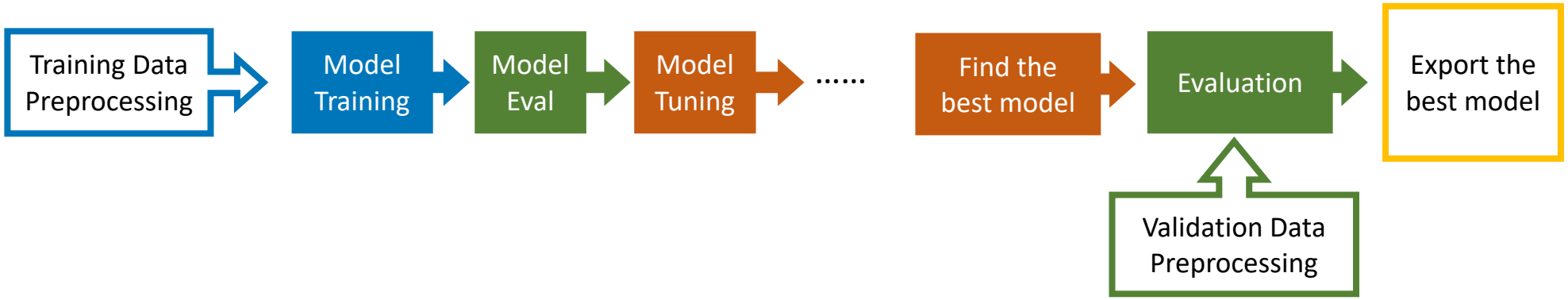
- Repeatedly adjust the architecture of the neural networks with new subnetworks.
- Until reach the maximum iterations or convergence.

How to find in ?

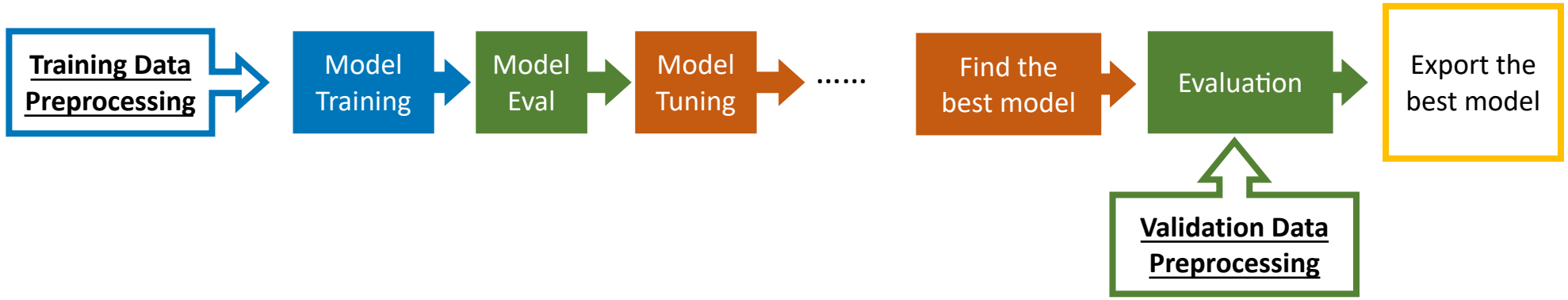
Method	Expertise	Exploring Time	Resources	Accuracy
Manual Tune	Yes	Depends…	Normal	Depends
Grid Search	No	Very Very Long	Very Very Much	Mostly Good!
Random Search	No	Very Long	Very Much	Mostly Good!
AdaNet	No	Short!	Normal	Good!

Design

How to deploy in ?

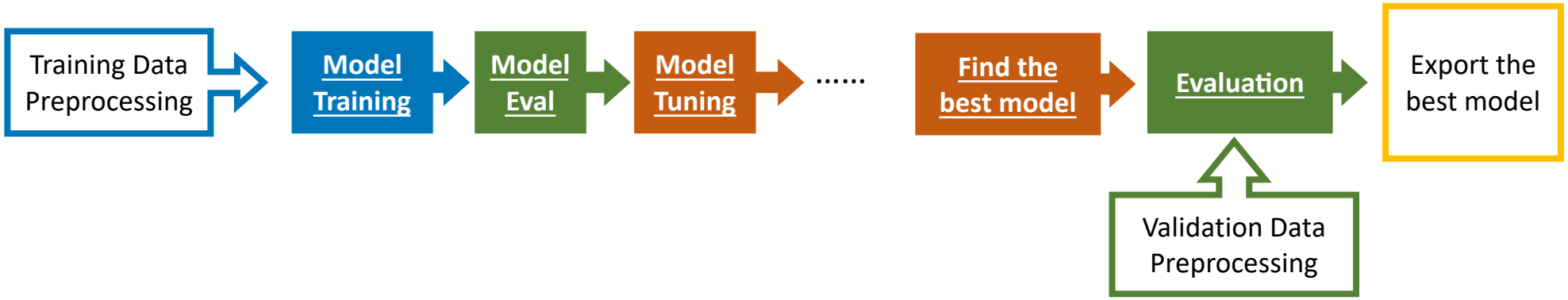


How to deploy in ?



- Data Preprocessing: Spark Job (Avro2TF job)
  - transform dense ***<member, job>*** concatenated pairs to sparse ***{key: value}*** input features

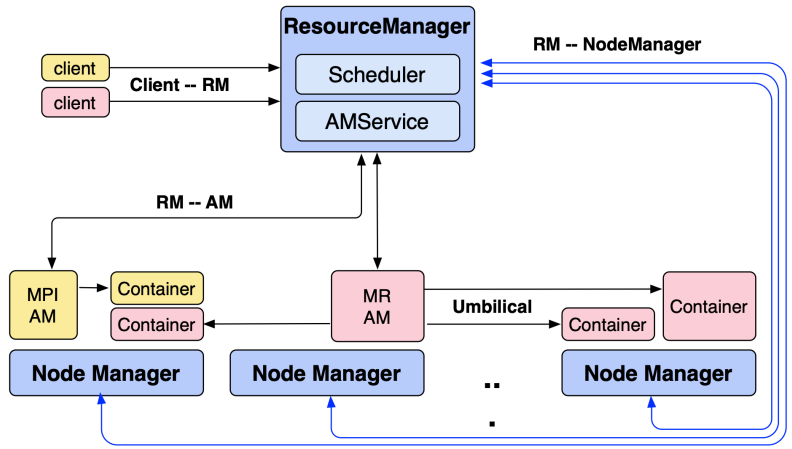
How to deploy in ?



- Data Preprocessing: Spark Job (Avro2TF job)
  - transform dense ***<member, job>*** concatenated pairs to sparse ***{key: value}*** input features
- **Model Training, Tuning, Evaluation: Tensorflow Job (On TonY)**
  - **How to cope with different tuning strategies?**

Tensorflow on Yarn (TonY)

- First, let's know our cluster well:
  - 1 Chief
    - Model training, management of extra work, e.g., checkpoint saving and restoring or writing summaries.
  - 1 Evaluator
    - A special task that is not part of the training cluster
  - N Worker
    - Model Training

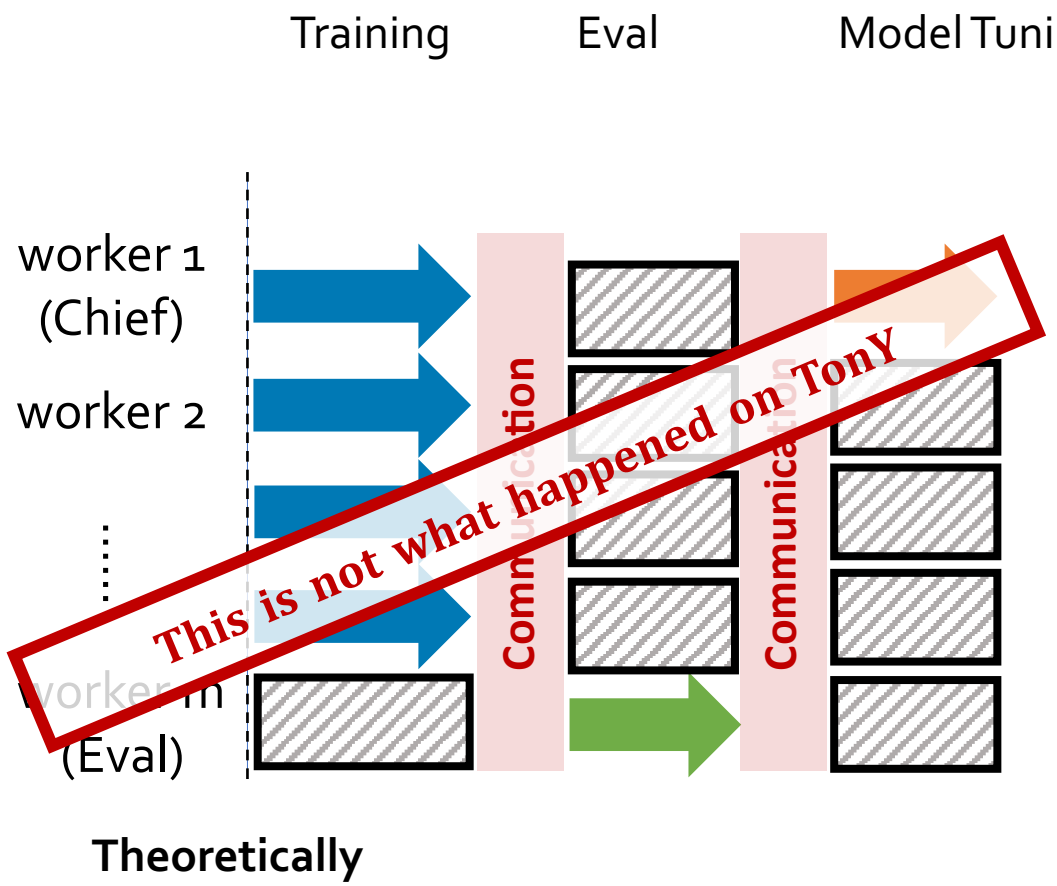


Picture credit to *Apache Hadoop YARN: Yet Another Resource Negotiator*

How to deploy in ?

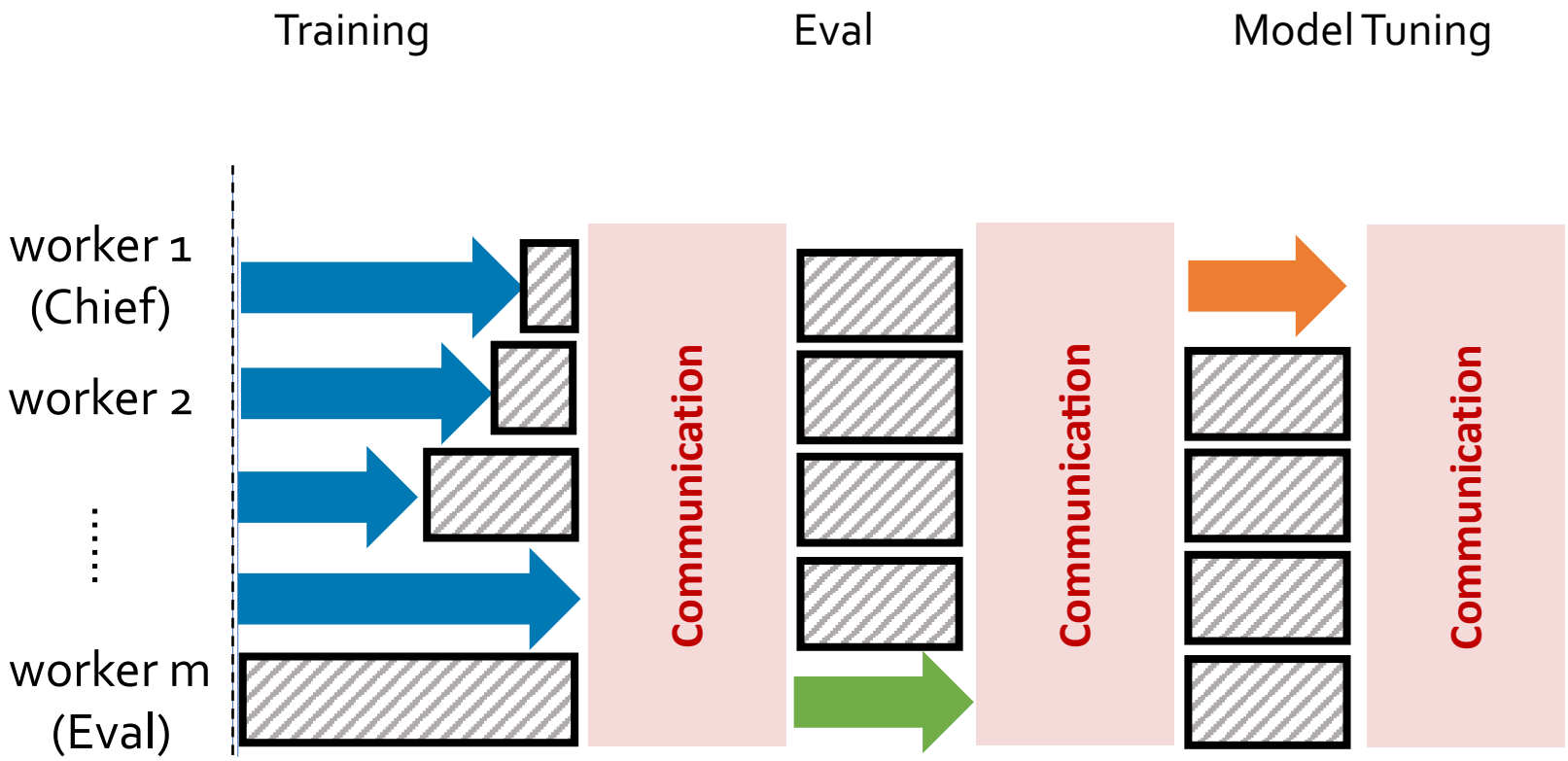
- Treat every worker equally and run trials either vertically or horizontally.
  - **Manual Tune**
  - **Grid/Random search**

Manual Tune and Grid/Random Search



Theoretically

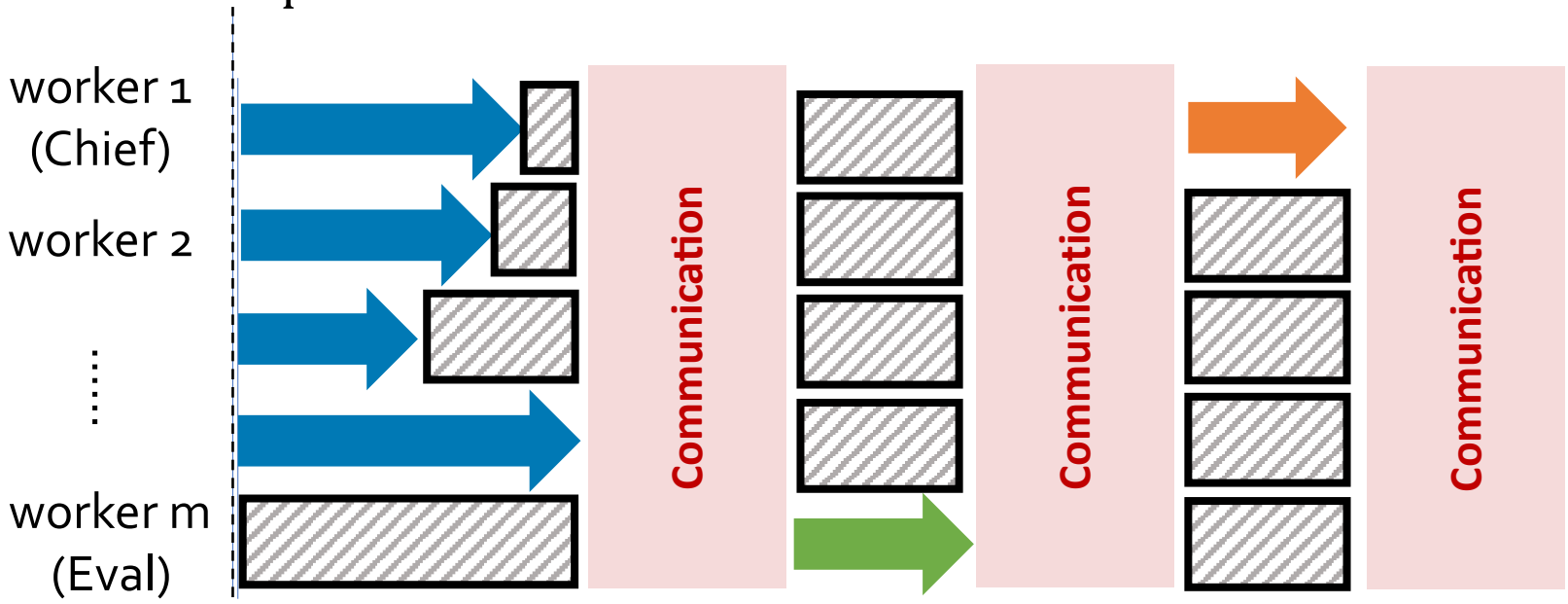
Manual Tune and Grid/Random Search



Practically: Straggler Effect and Communication Latency

Straggler Effect and Communication Latency

- Straggler is still working on its share of the parallelized computation when most or all of the other processors have completed their shares.
- Communication time can be even larger or be multiple times than the computation time.



Practically

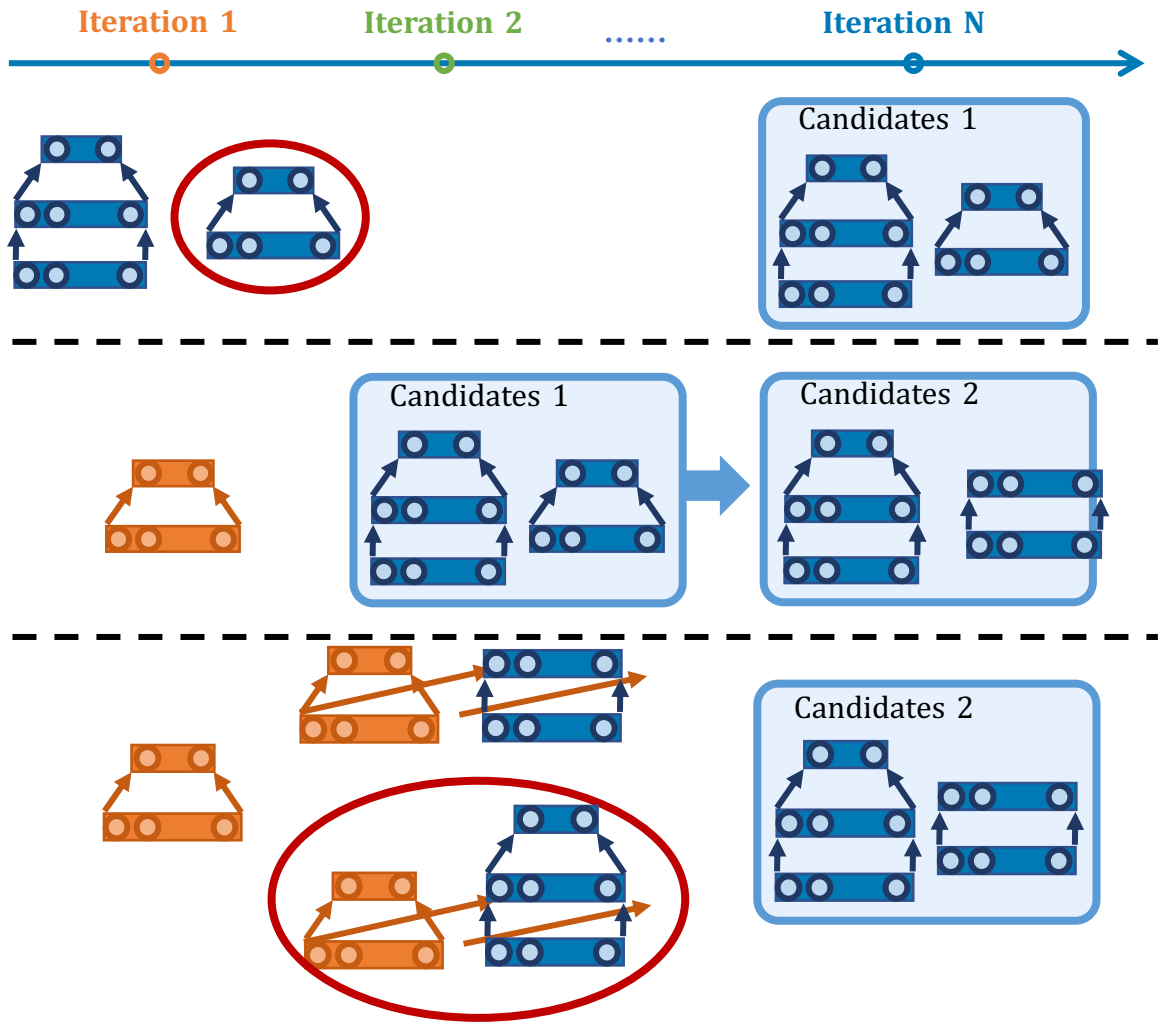
How to deploy in ?

- Treat every worker equally and run trials either vertically or horizontally.
  - **Manual Tune**
  - **Grid/Random search**
- Things do not work out for **AdaNet**, why?
  - Straggler effect and communication latency are the bottleneck.
    - Since training of **enormous** lightweight networks is inevitable.

Challenging Distributed Training of AdaNet

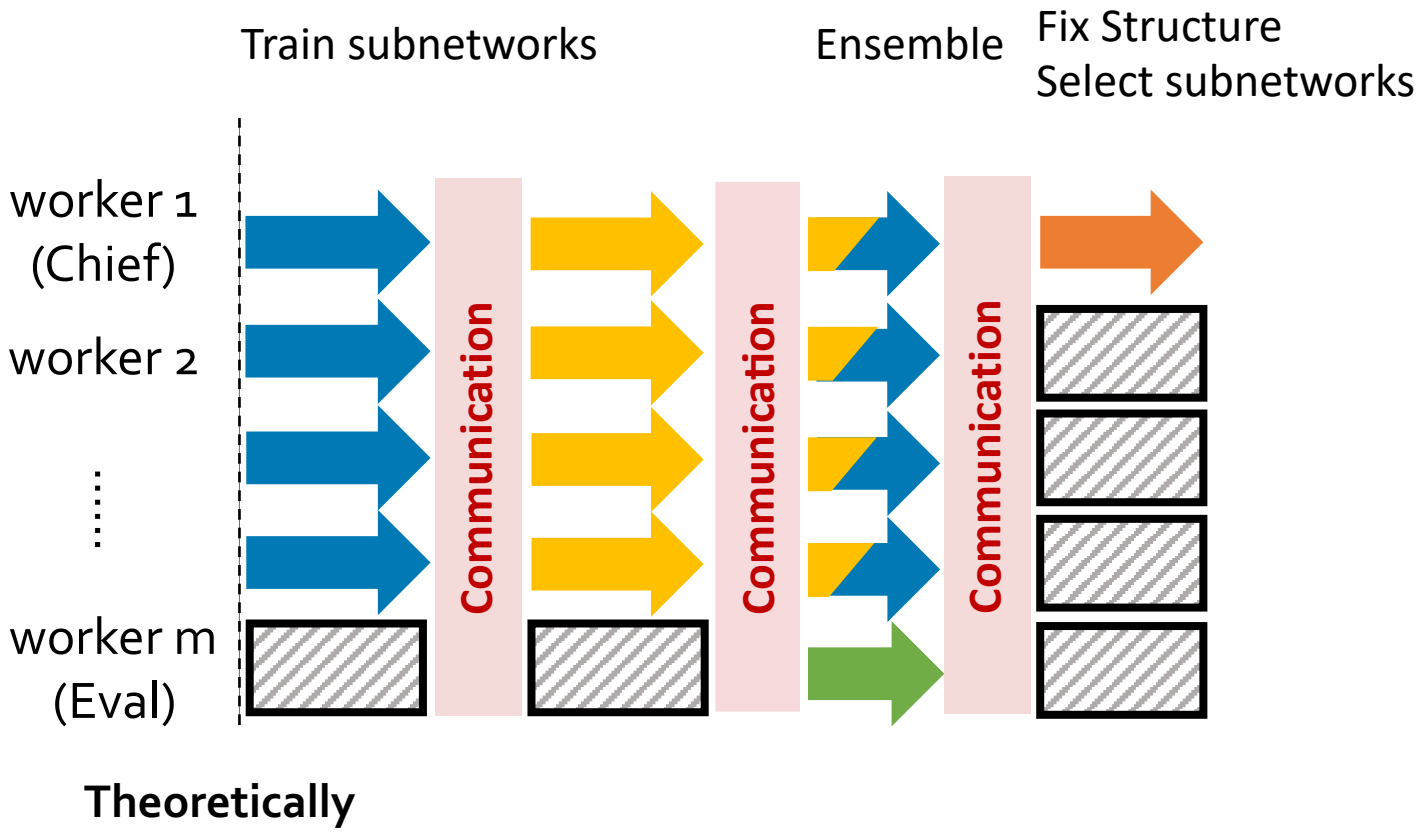
- (Remind): Algorithm Guideline
- Initialize the subnetwork pool
- LOOP{
  - **Train subnetworks**
  - **Try different ensembles**
  - **Fix the structure**
  - **Select next subnetworks**

  }
- Export the final structure



Challenging Distributed Training of AdaNet

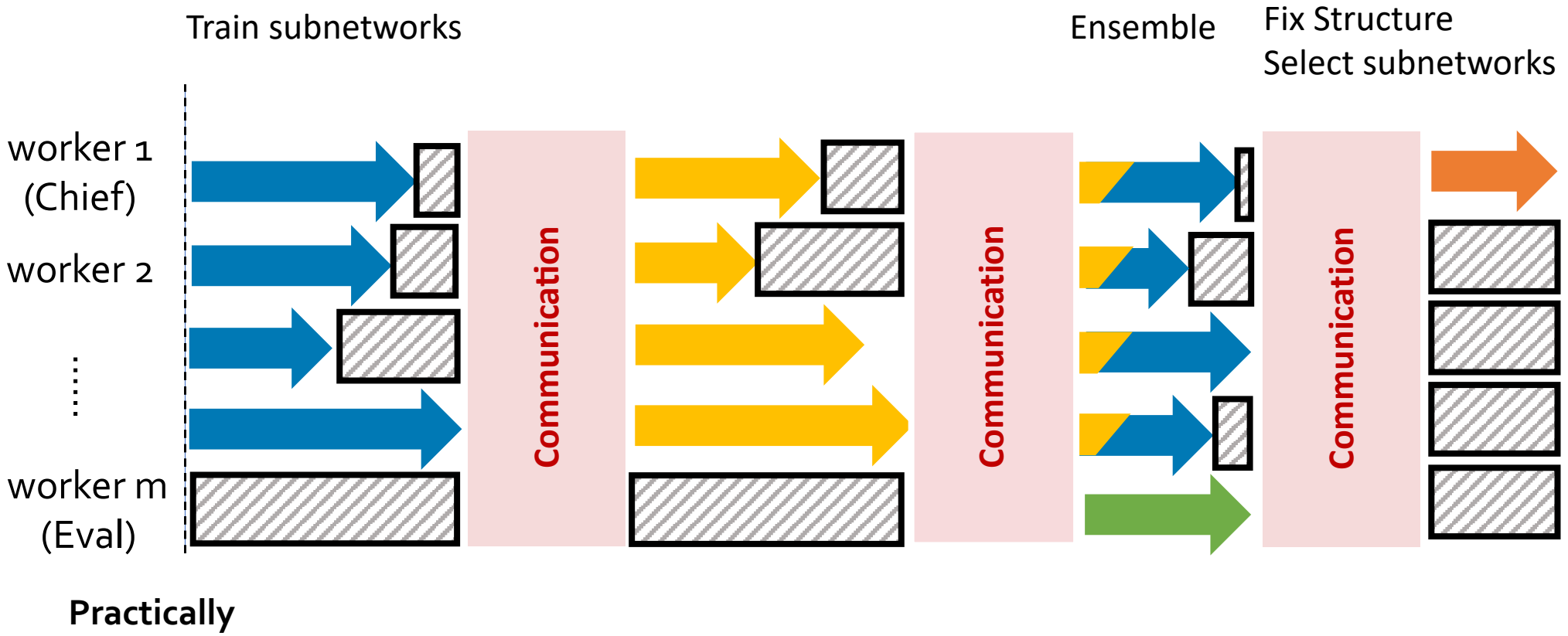
- Suppose each time we try 2 different subnetworks (lightweight).
- And we only try 1 ensemble one iteration.



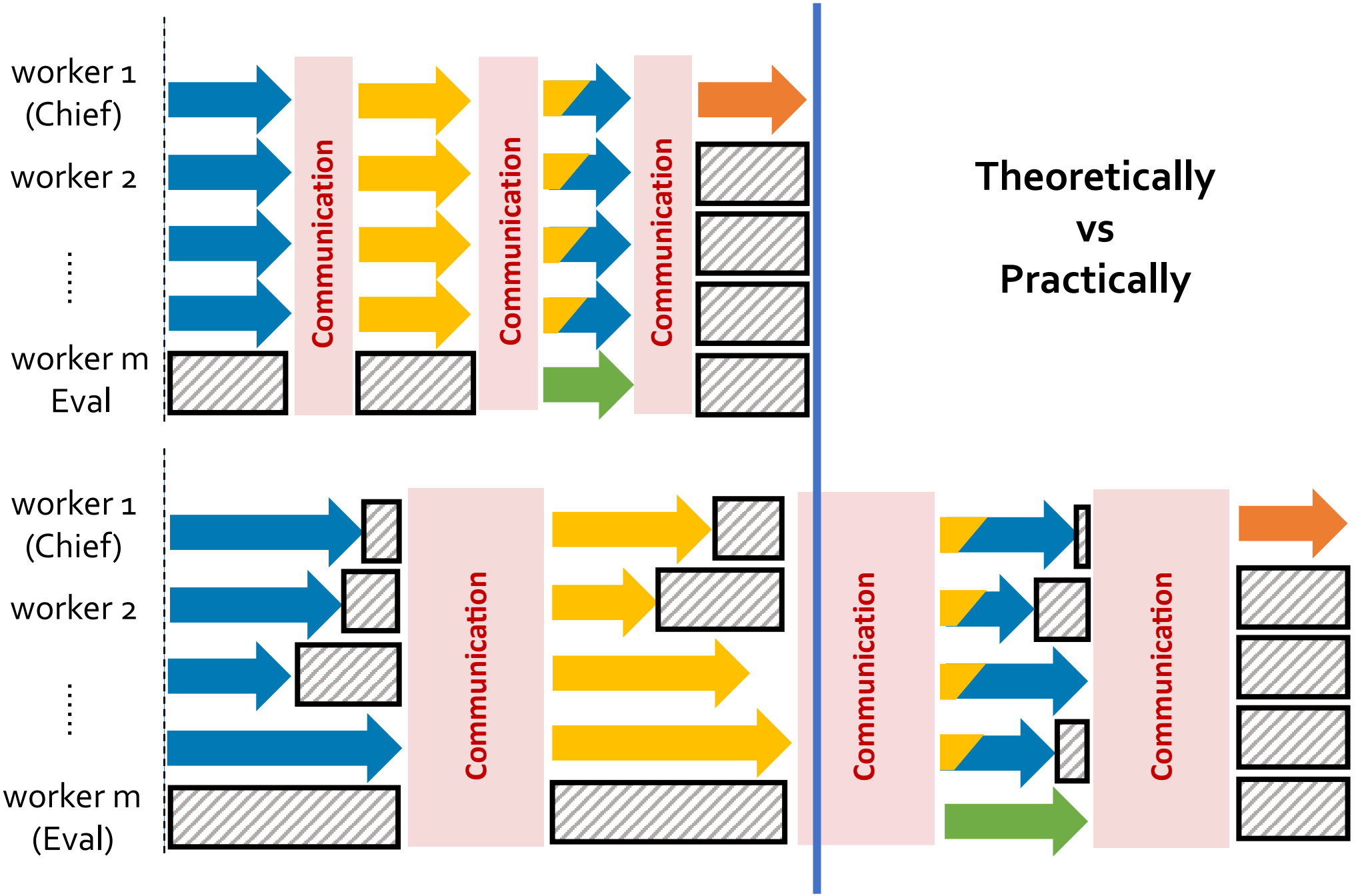
Theoretically

Challenging Distributed Training of AdaNet

- Suppose each time we try 2 different subnetworks (lightweight).
- And we only try 1 ensemble one iteration.



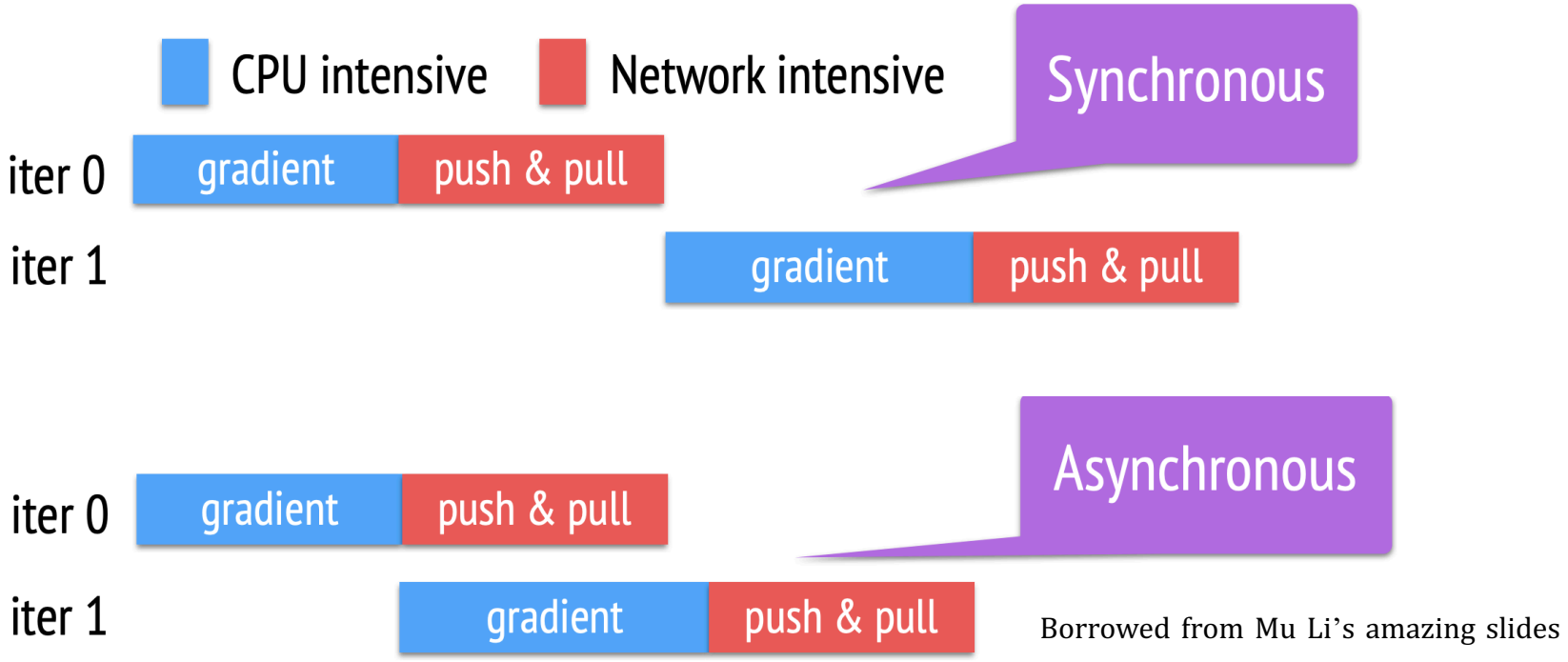
Practically


worker 1
(Chief)
worker 2
worker m
Eval
Communication
Communication
Communication
Theoretically
vs
Practically
worker 1
(Chief)
worker 2
worker m
(Eval)
Communication
Communication
Communication

How to deploy in ?

- Treat every worker equally and run trials either vertically or horizontally.
  - **Manual Tune**
  - **Grid/Random search**
- Things do not work out for **AdaNet**, why?
  - Straggler effect and communication latency are the bottleneck.
    - Since training of enormous light weight networks is inevitable.
  - Quick question:
    - **How to deal with straggler effect and communication latency?**

Distributed Strategy



Borrowed from Mu Li's amazing slides

Use Async instead of Sync

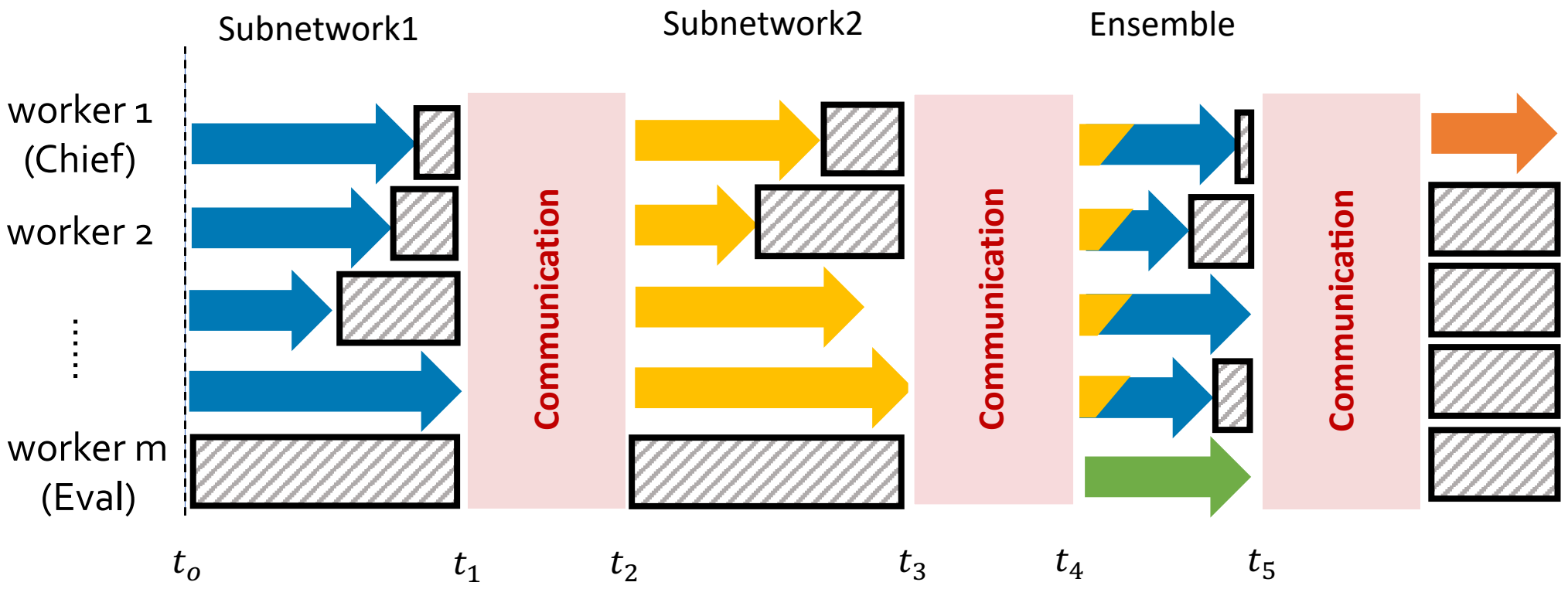
-- The easiest method to wipe out communication traffic **in sacrifice of performance.**

How to deploy in ?

- Treat every worker equally and run trials either vertically or horizontally.
  - **Manual Tune**
  - **Grid/Random search**
- Things do not work out for **AdaNet**, why?
  - Straggler effect and communication latency are the bottleneck.
    - Since training of enormous light weight networks is inevitable.
- Now we know we can't use asynchronous distributed strategy to train **one** subnetwork.
  - However, can we use asynchronous distributed strategy to train **a lot of** subnetworks?

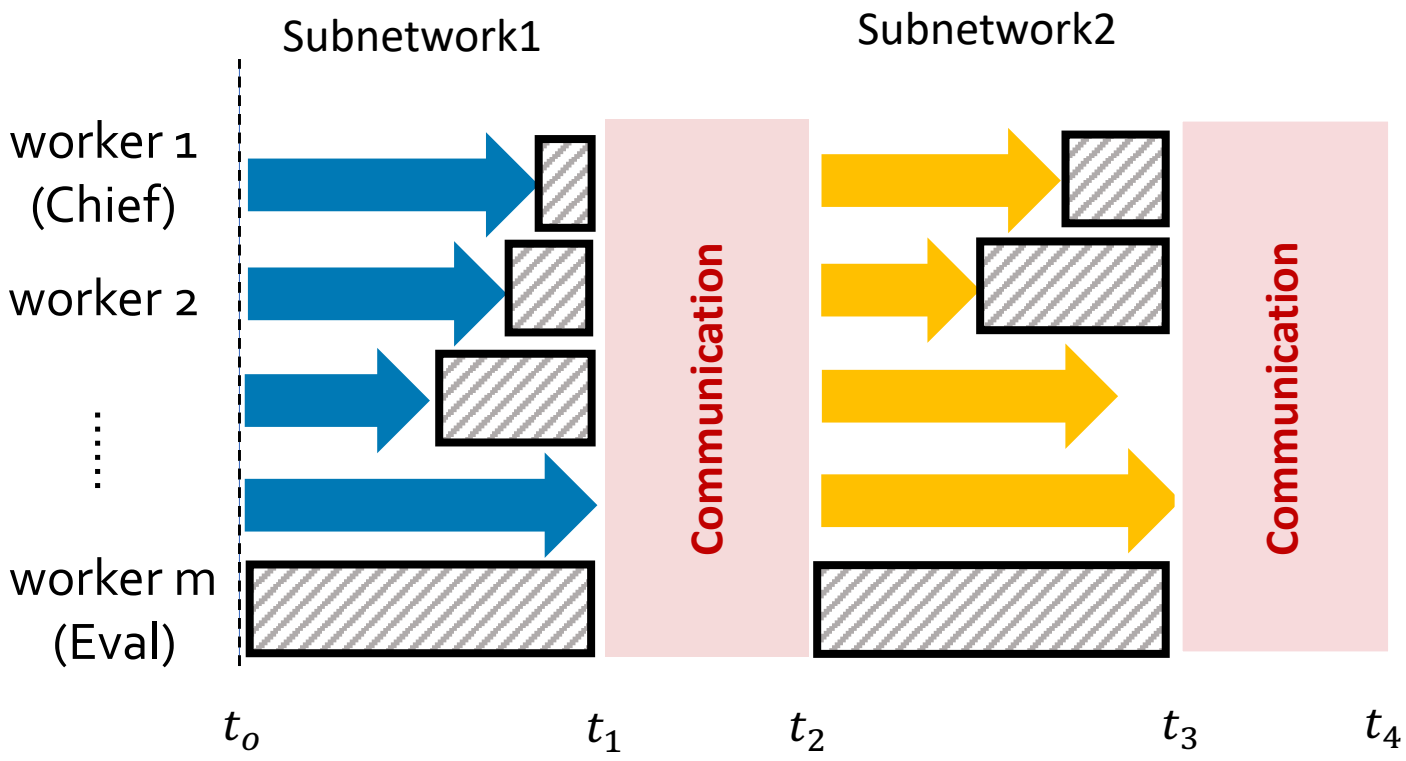
Asynchronous Model Training in AdaNet

- We train subnetworks **asynchronously**!



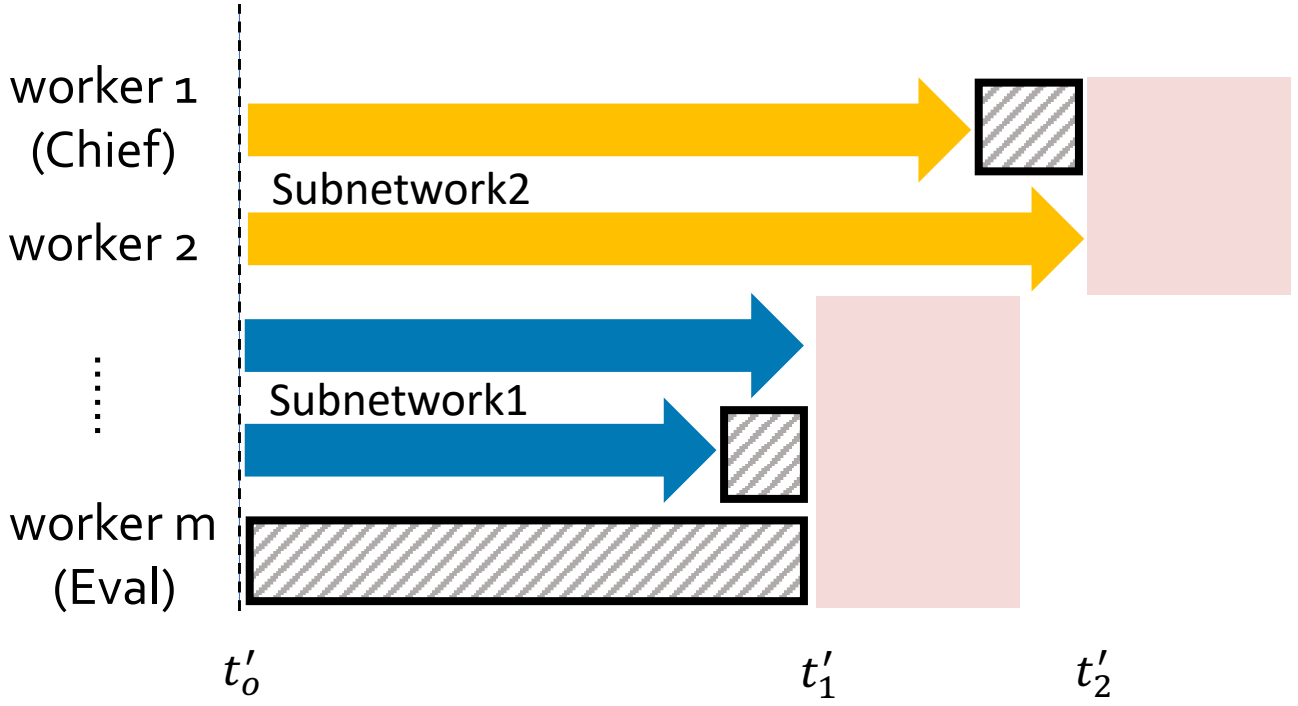
Asynchronous Model Training in **AdaNet**

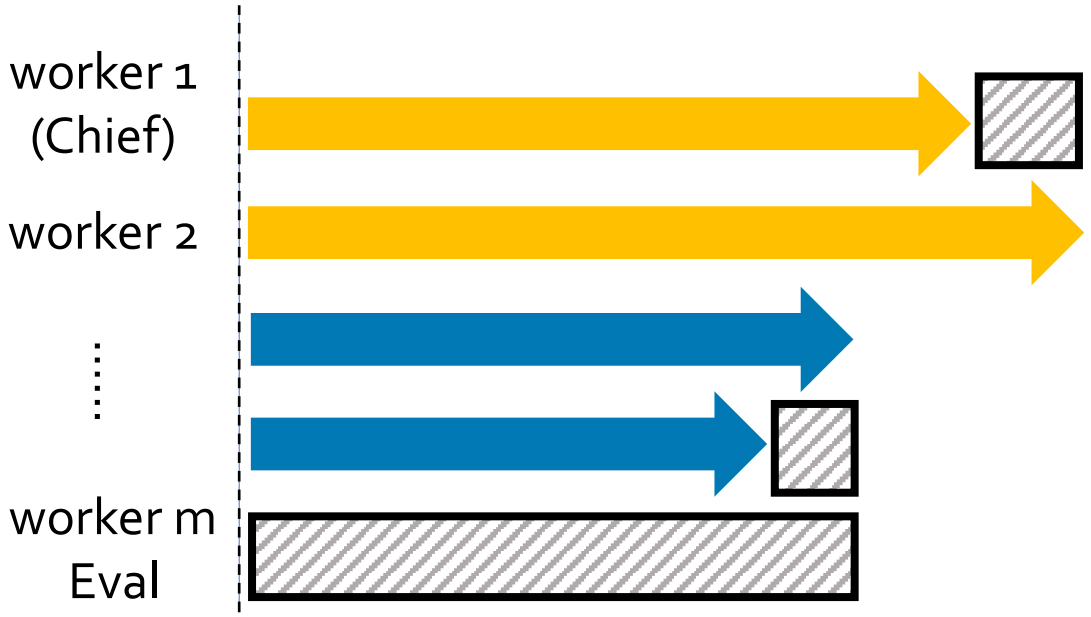
- Subnetwork 1 occupies m **workers from** t_o **to** t_1
- Subnetwork 2 occupies m **workers from** t_2 **to** t_3



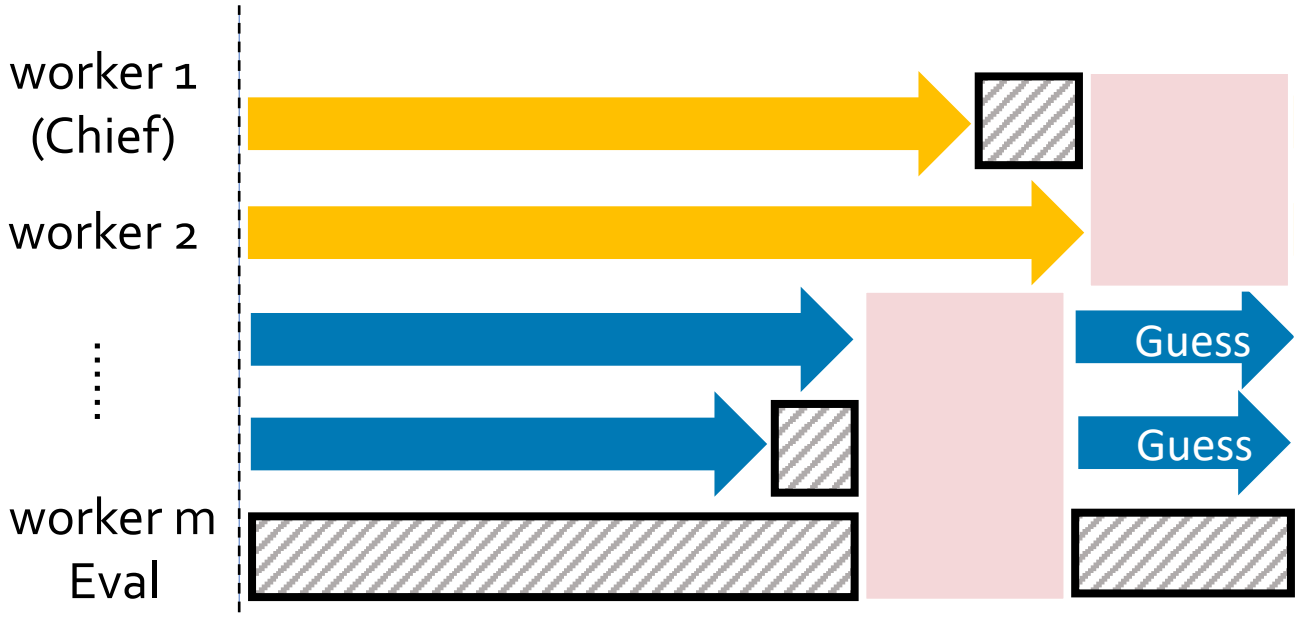
Asynchronous Model Training in AdaNet

- Subnetwork 1 occupies $m/2$ workers from t'_o to t'_1
- Subnetwork 2 occupies $m/2$ workers from t'_o to t'_2

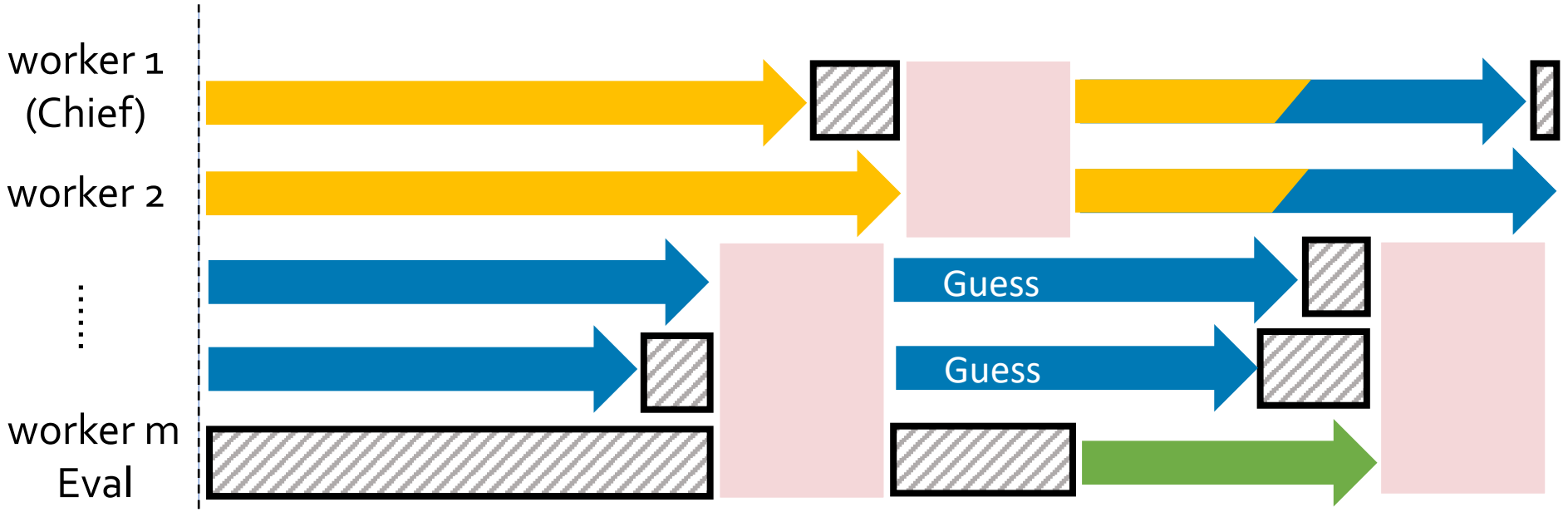




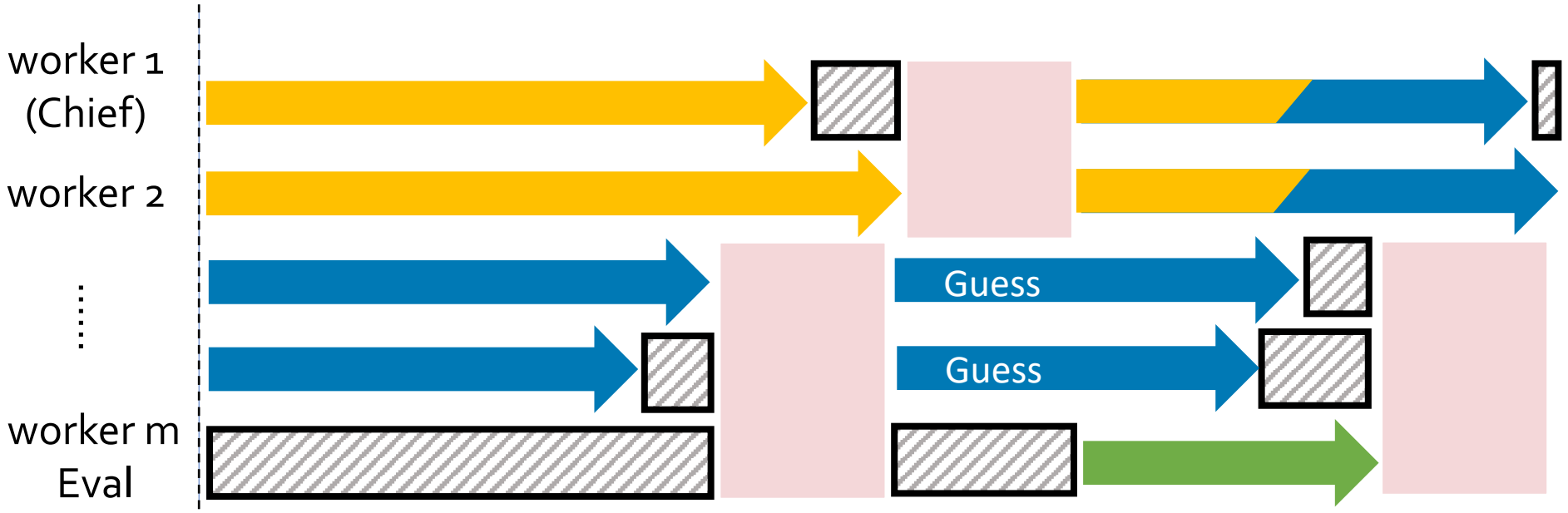
- Start training subnetworks separately.
  - Each subnetwork occupies one share of all resources.

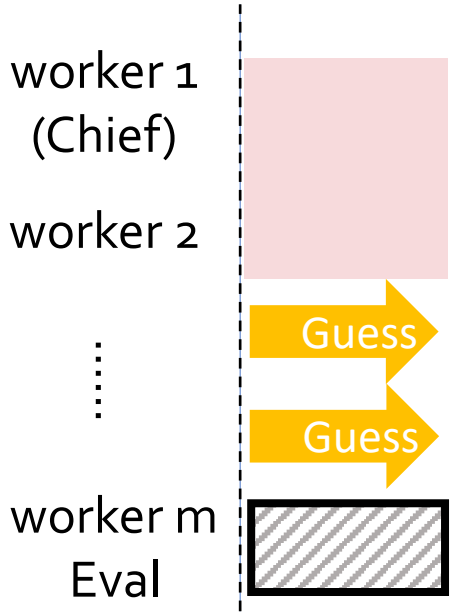


- Once finished, start to re-train of current subnetwork
  - **Assume this subnetwork will remain in the pool.** **(Forecasting)**

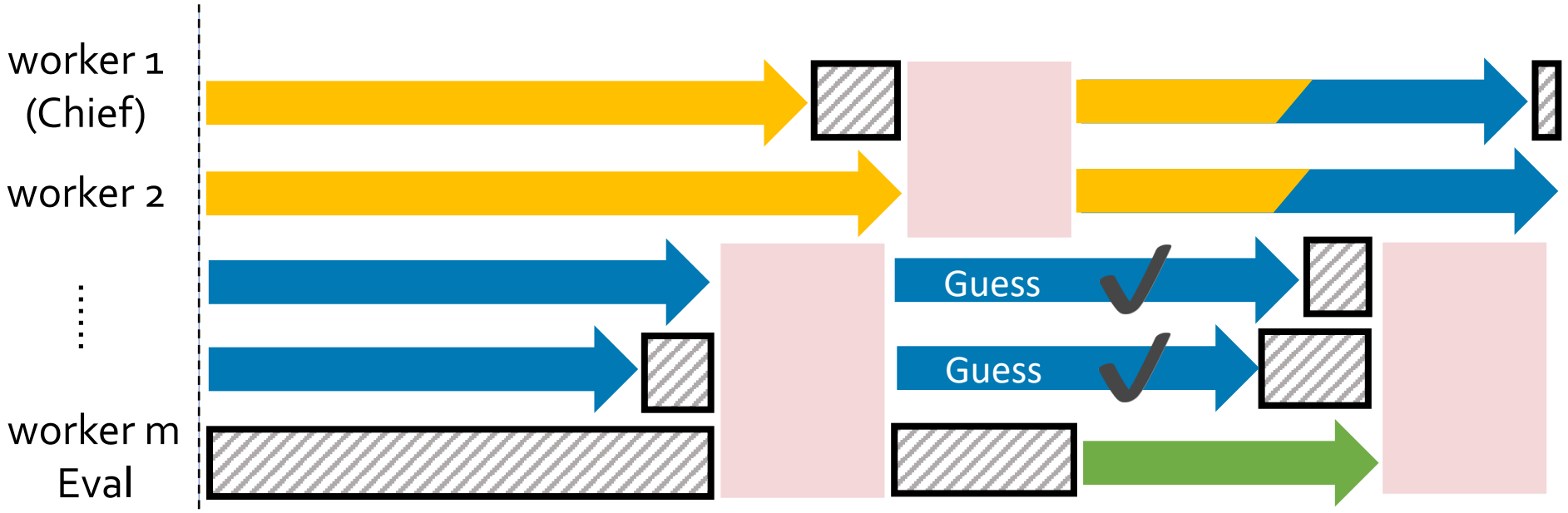


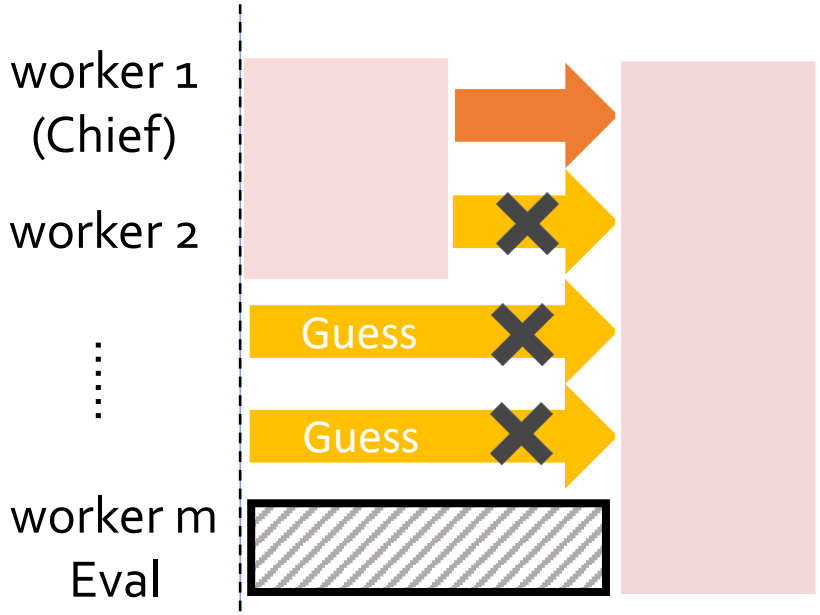
- When all subnetworks of current iteration are finished, start ensemble using all remaining available resources.
  - Previously occupied nodes continued on their previous work.





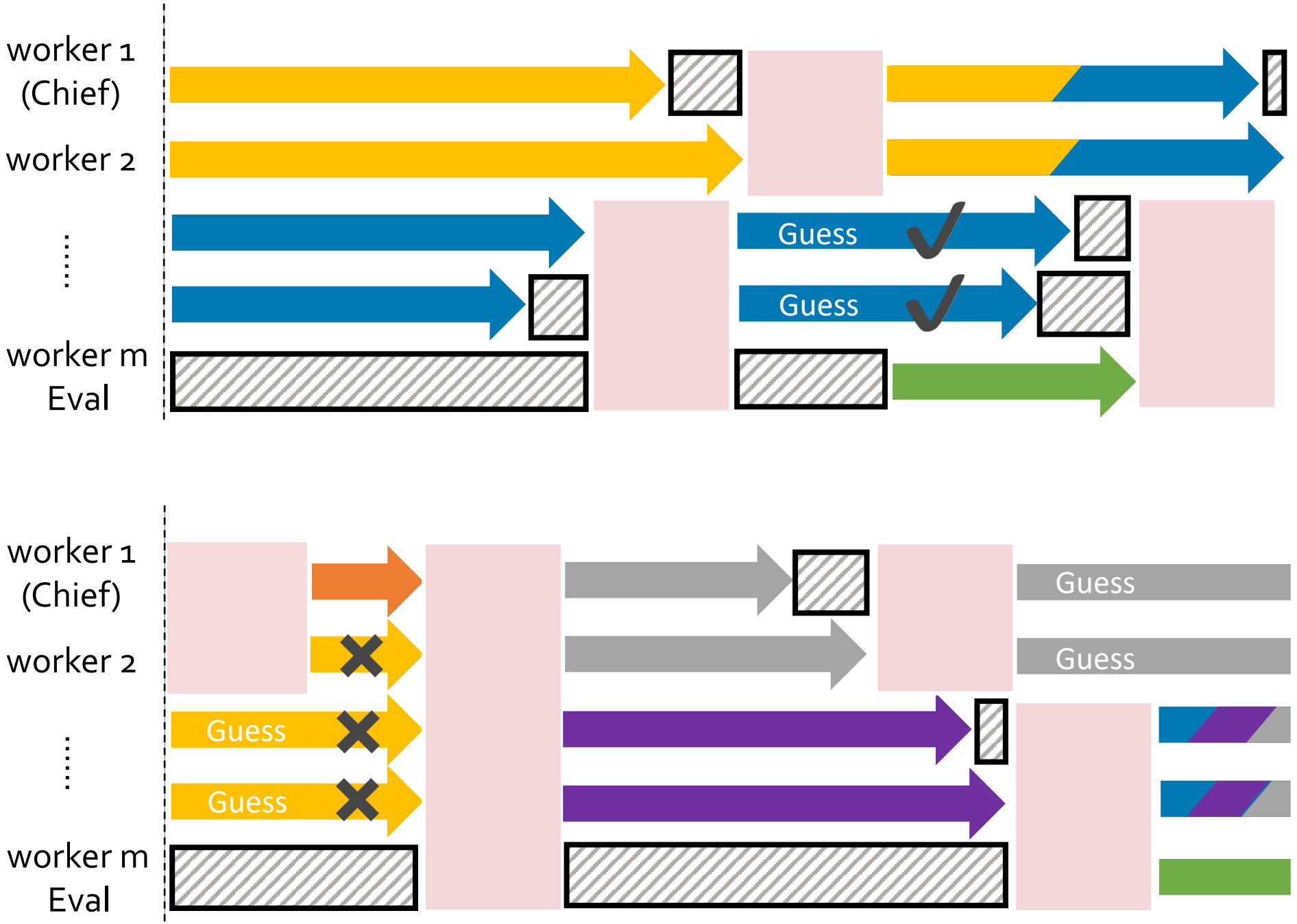
- During ensemble, if any occupied resources are freed, put all of them into another subnetwork training.
  - **Assume this subnetwork will remain in the pool.**
    (Forecasting)





- After evaluation, Chief decides subnetworks for next iteration.
  - **Keep luck guess**
  - **Throw away bad guess**

Scalable
Automatic Tuning



LinkedIntern
Yuwei Qiu

Asynchronous Model Training in **AdaNet**

- We have these observations in Sync model training:
  - Many workers involved in communication during training of subnetworks.
    - Heavier straggler effect.
  - Long waiting time during evaluation and model adjusting.
    - Waste of resources.

- Based on that, we have the following refinements in Async model training:
  - Train the subnetworks separately with different shares of resources
    - A smaller amount of workers are involved in one subnetwork.
  - Forecast subnetworks of next iteration and start training early.
    - Guess with risk that the current subnetworks will be kept in the next iteration

Results

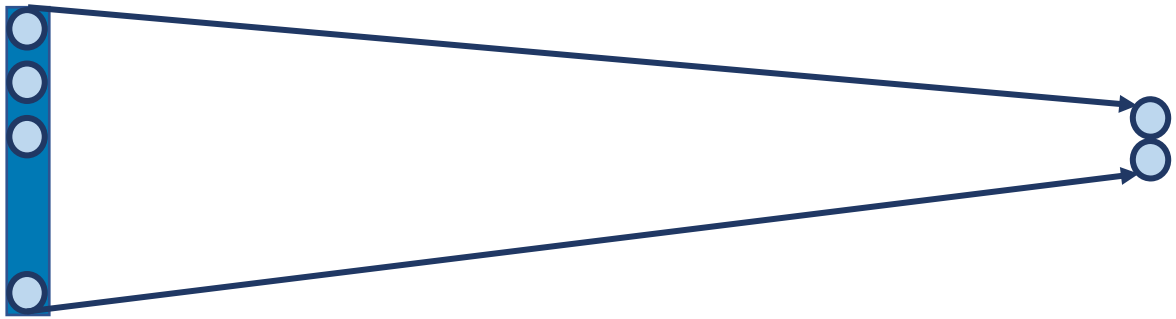
Resources

- All of the following experiments are done on ***TonY*** by:
  - Training with 20 workers, each with 32g
  - Validation with 20 workers, each with 32g

Baseline – Logistic Regression

Model	Training Steps	Training Time	Validation Time	AUC
Logistic Reg	120,232	1 hour 45 mins	20 mins	66.3%

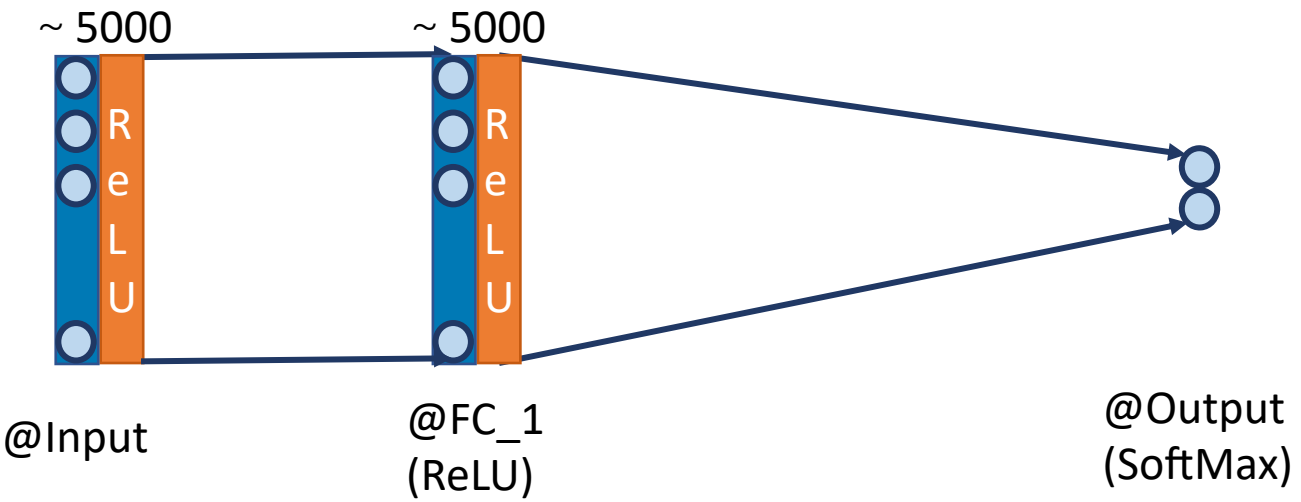
Opt: 1e-1 Decay LR SGD
Reg: L2, 1e-4



Manual Tuned – 1 layer fully connected NN

Model	Training Steps	Training Time	Validation Time	AUC
Logistic Reg	120,232	1 hour 45 mins	20 mins	66.3%
Manual Tuned	120,232	2 hours 05 mins	30 mins	66.8%

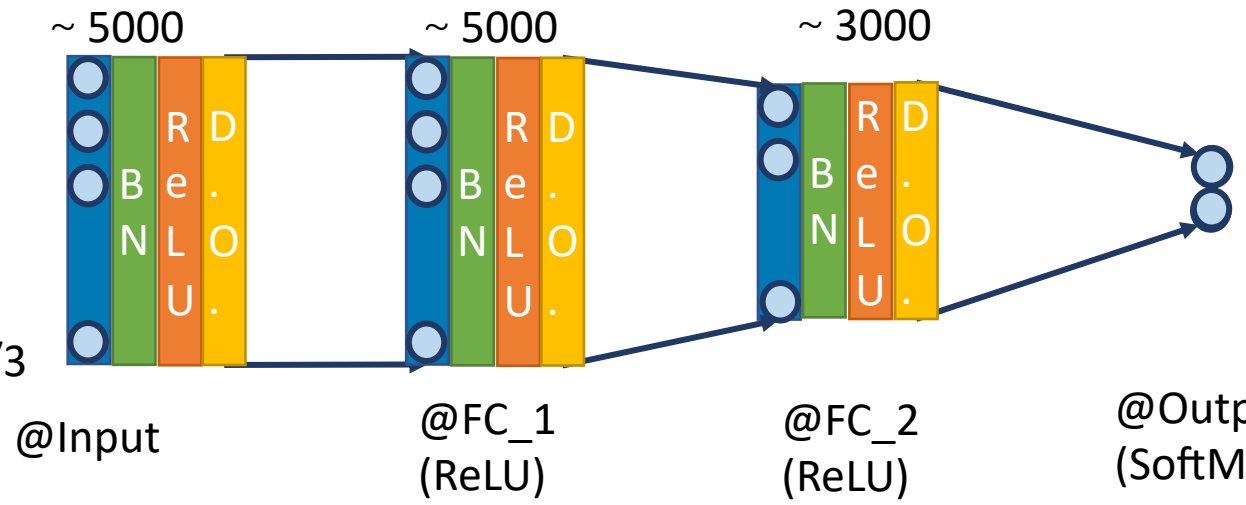
Opt: 1e-1 Decay LR SGD
Reg: L2, 1e-4
Act: ReLU



Auto Tuned – Best NN with Random Search

Model	Training Steps	Training Time	Validation Time	AUC
Logistic Reg	120,232	1 hour 45 mins	20 mins	66.3%
Manual Tuned	120,232	2 hours 05 mins	30 mins	66.8%
Random_Bst_NN	120,232	2 hours 35 mins	28 mins	67.2%

Opt: 1e-1 Decay LR SGD
Reg: L2, 1e-4
Act: ReLU
Dropout: 0.1
BatchNorm
Layer_2 / # Layer_1 = 2/3



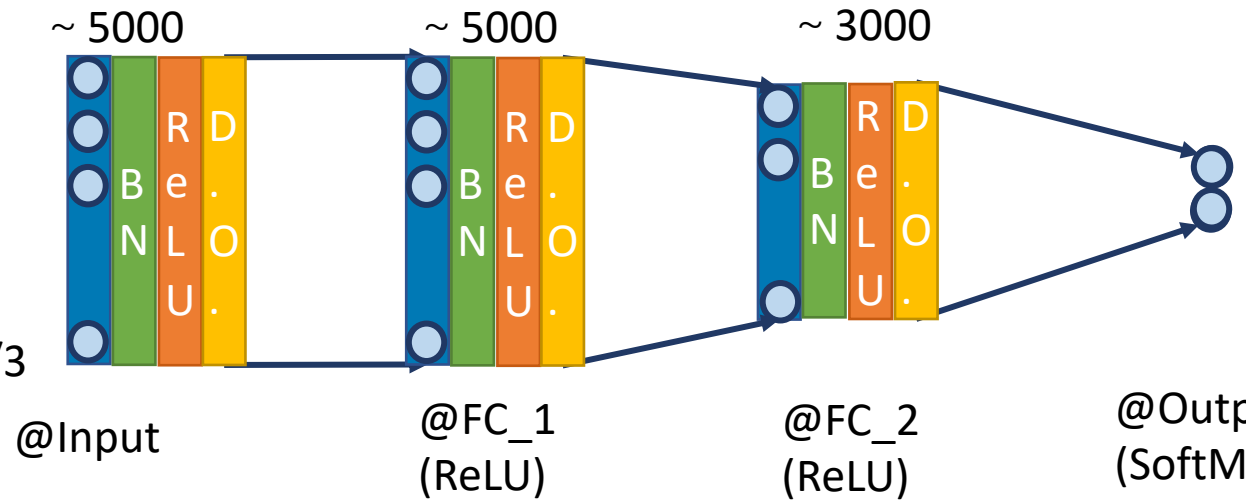
Auto Tuned – Best NN with Random Search

Model	Training Steps	Training Time	Validation Time	AUC
Logistic Reg	120,232	1 hour 45 mins	20 mins	66.3%
Manual Tuned	120,232	2 hours 05 mins	30 mins	66.8%
Random_Bst_NN	120,232	2 hours 35 mins	35 mins	67.2%

Grid Search: Explored 12 sets of parameters, each one trained with down-sampled 3000 steps.

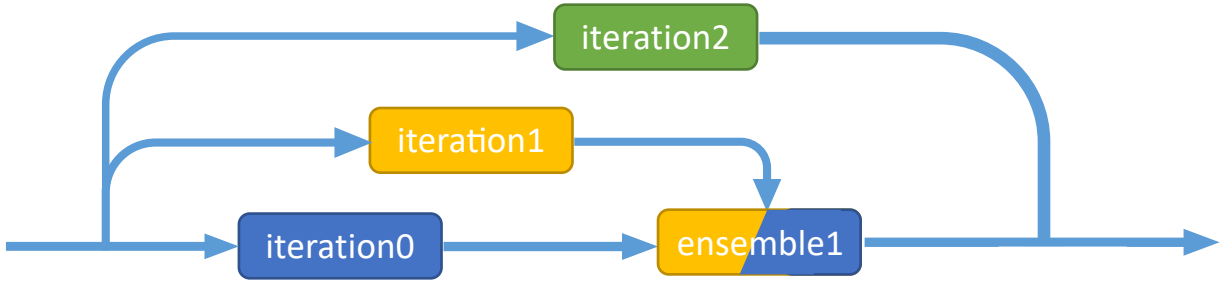
Avg per model	45 mins	Total 12 sets	10 hours 25 mins

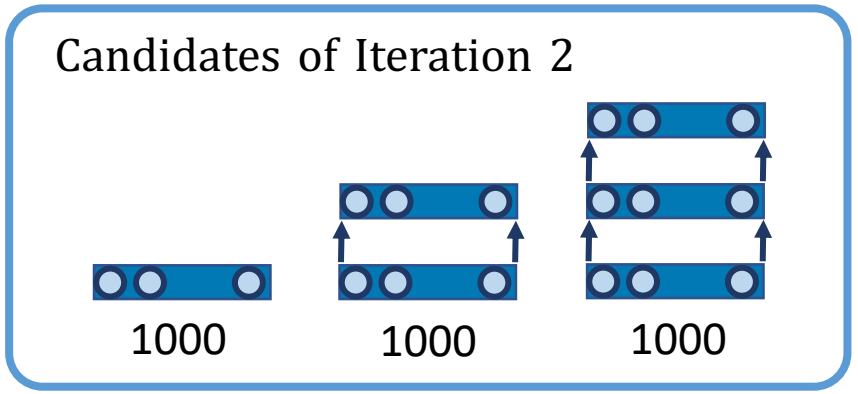
Opt: 1e-1 Decay LR SGD
Reg: L2, 1e-4
Act: ReLU
Dropout: 0.1
BatchNorm
Layer_2 / # Layer_1 = 2/3



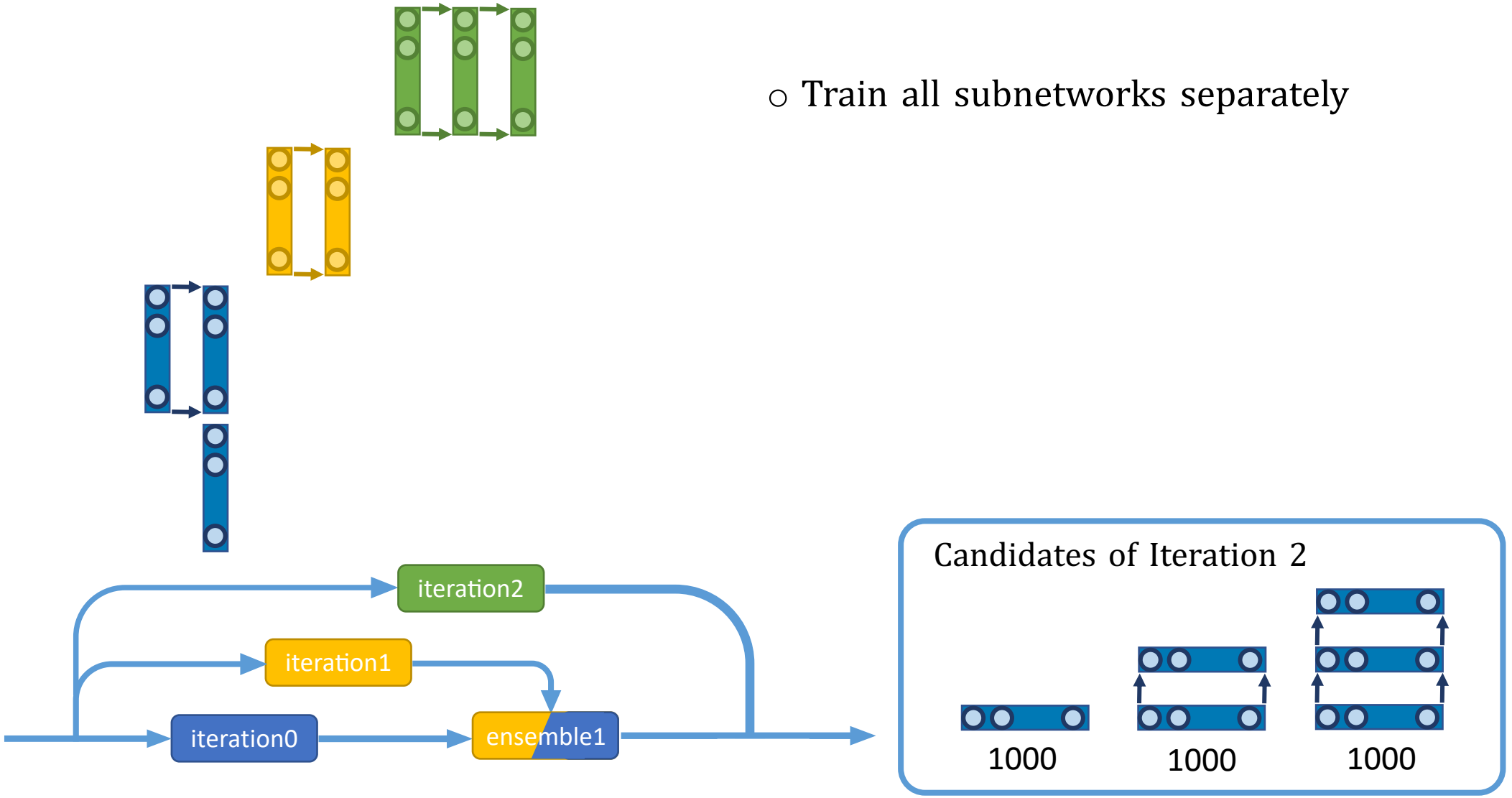
Auto Tuned – AdaNet

Model	Training Steps	Training Time	Validation Time	AUC
Logistic Reg	120,232	1 hour 45 mins	20 mins	66.3%
Manual Tuned	120,232	2 hours 05 mins	30 mins	66.8%
Random_Bst_NN	120,232	2 hours 35 mins	35 mins	67.2%
AdaNet	2 iterations for structure tuning, down-sampled 2000 steps per iteration	36 mins	31 mins	67.6%



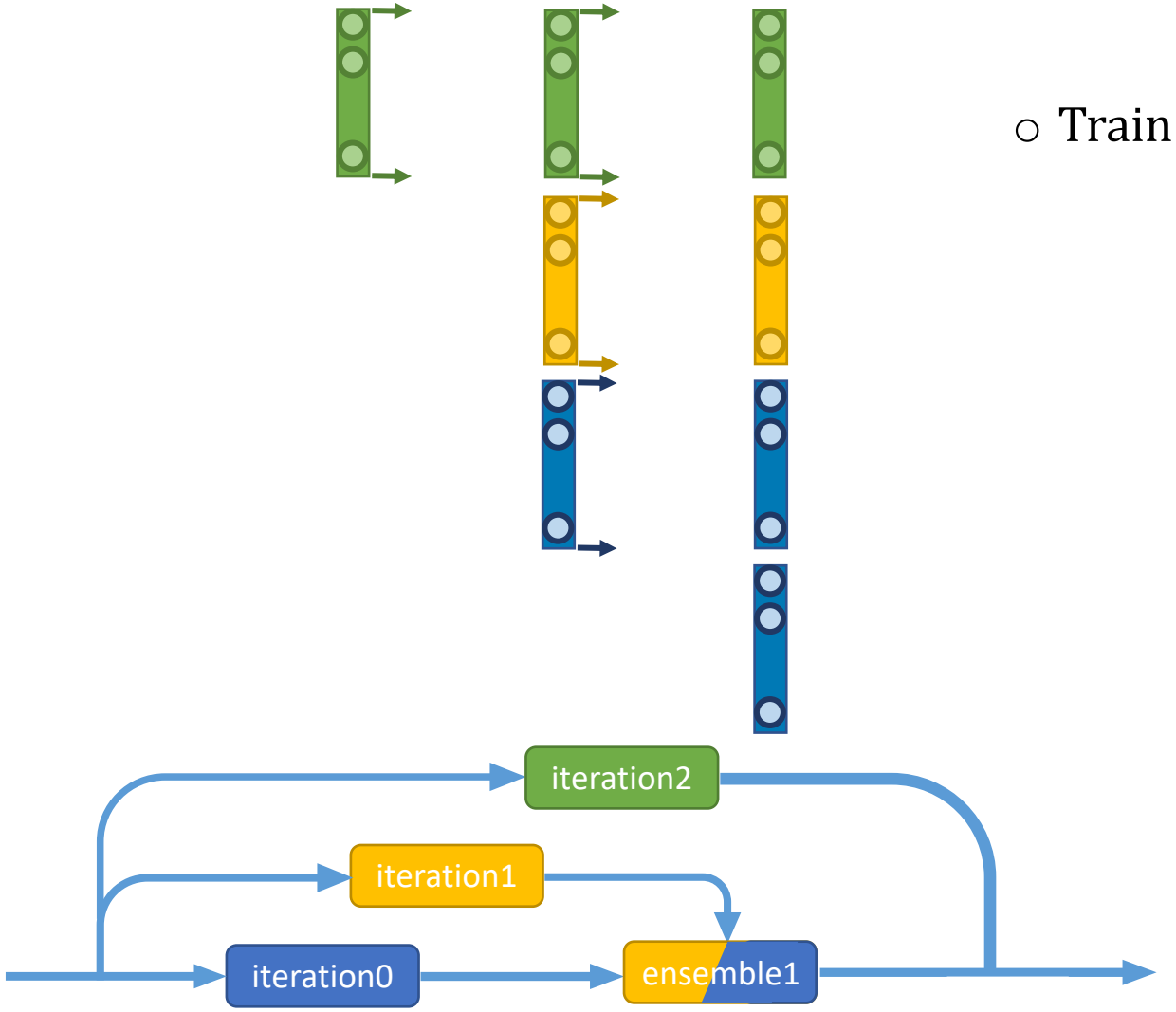


A Closer Look into AdaNet Structure

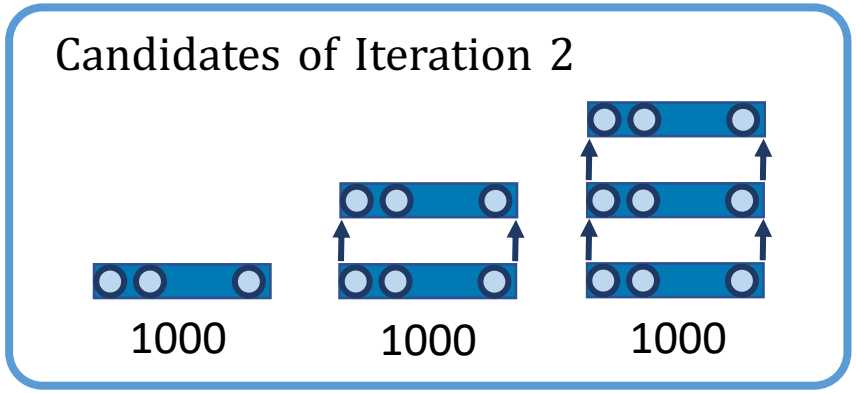


- Train all subnetworks separately

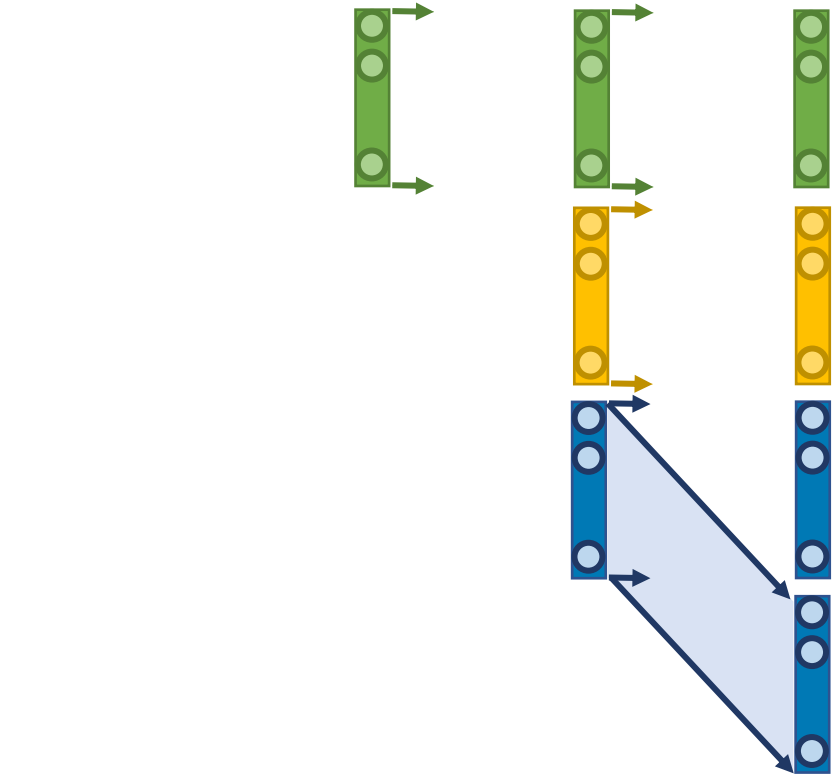
A Closer Look into AdaNet Structure



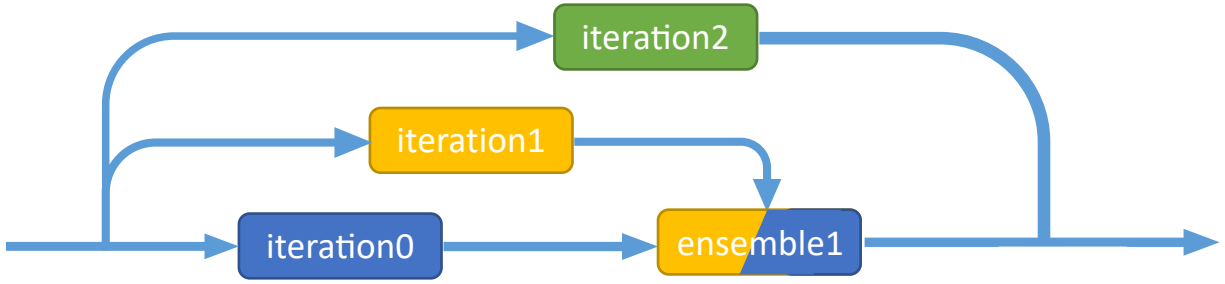
- Train all subnetworks separately

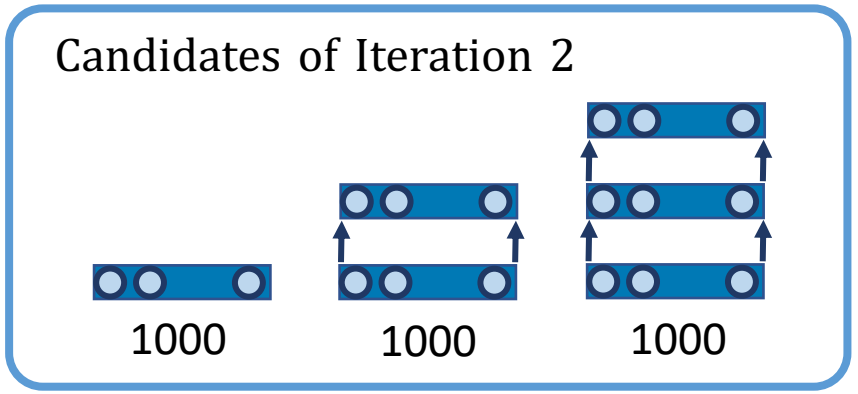


A Closer Look into AdaNet Structure

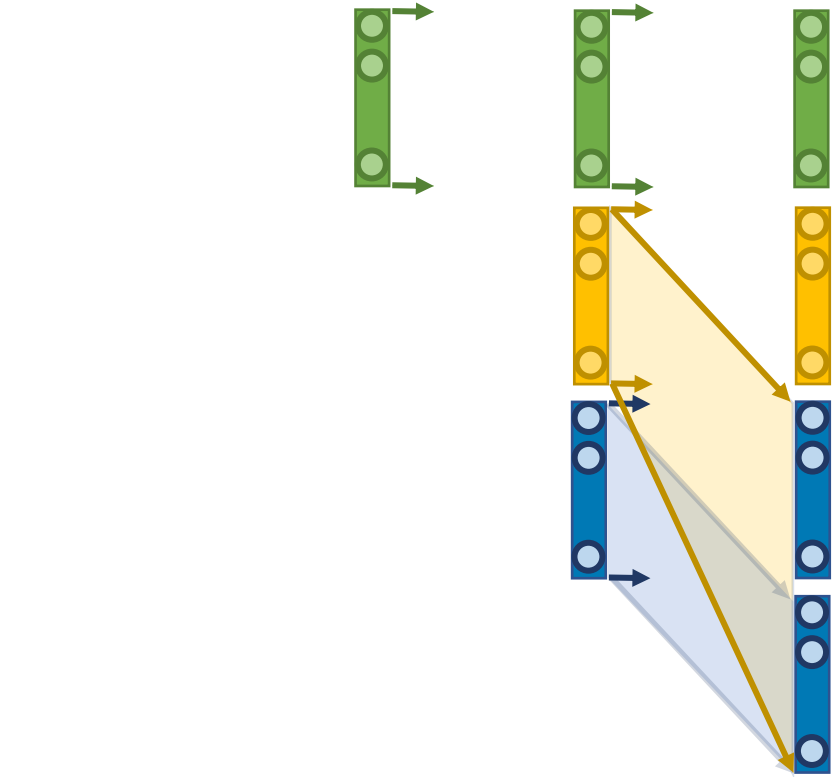


- Iteration 0 ensemble

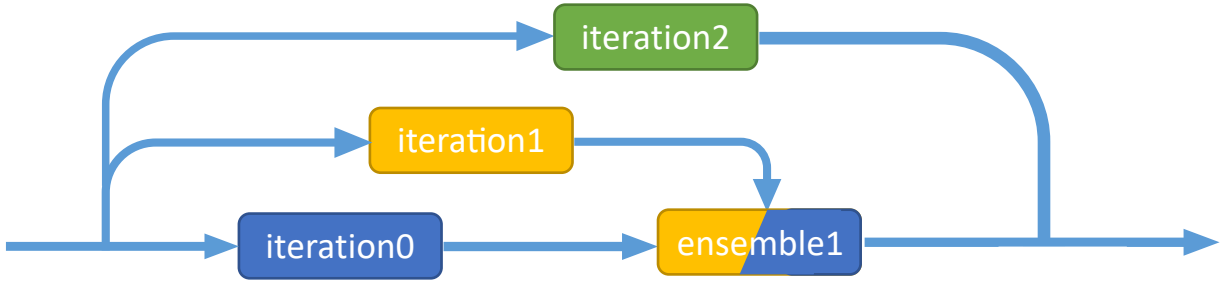


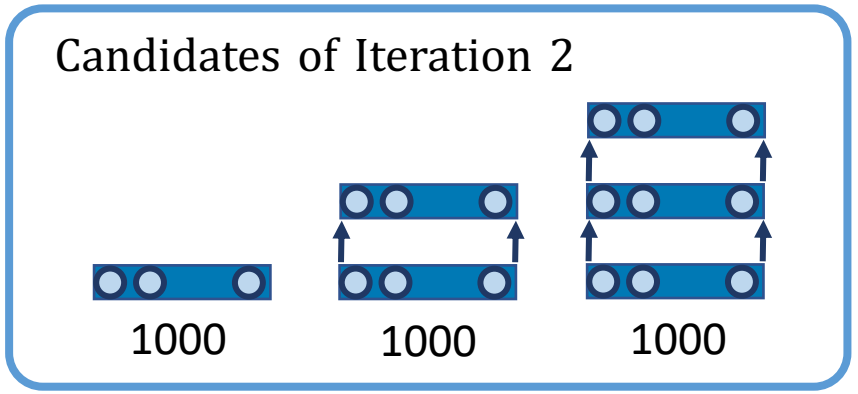


A Closer Look into AdaNet Structure

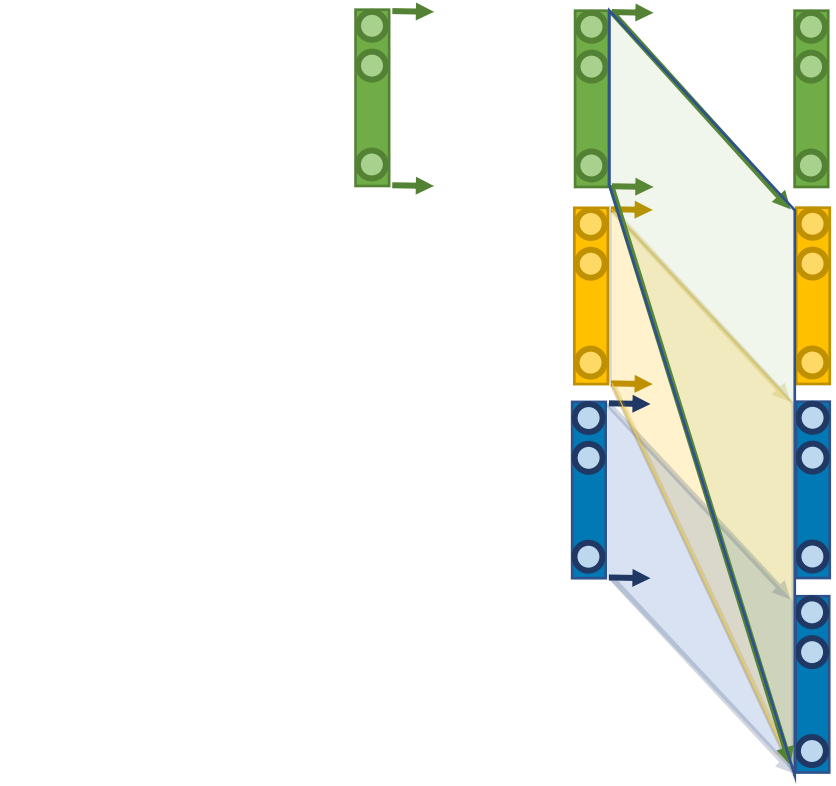


- Iteration 1 ensemble

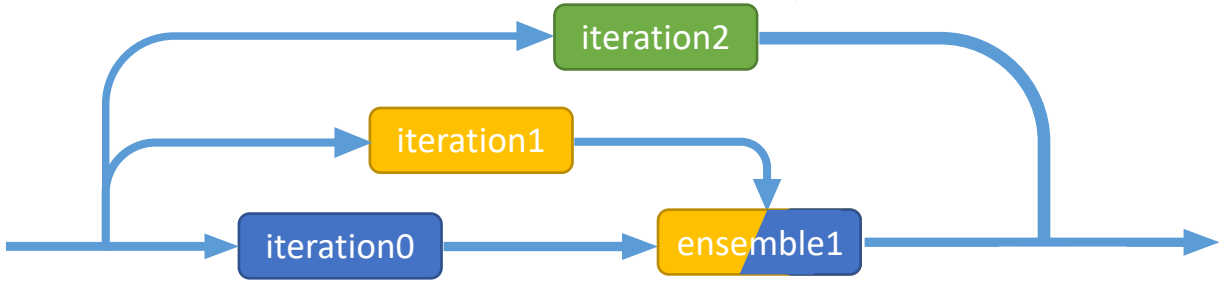


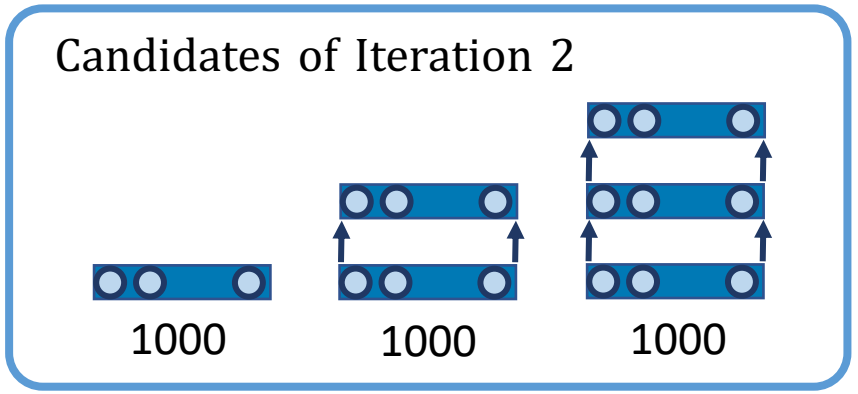


A Closer Look into AdaNet Structure



- Iteration 2 ensemble

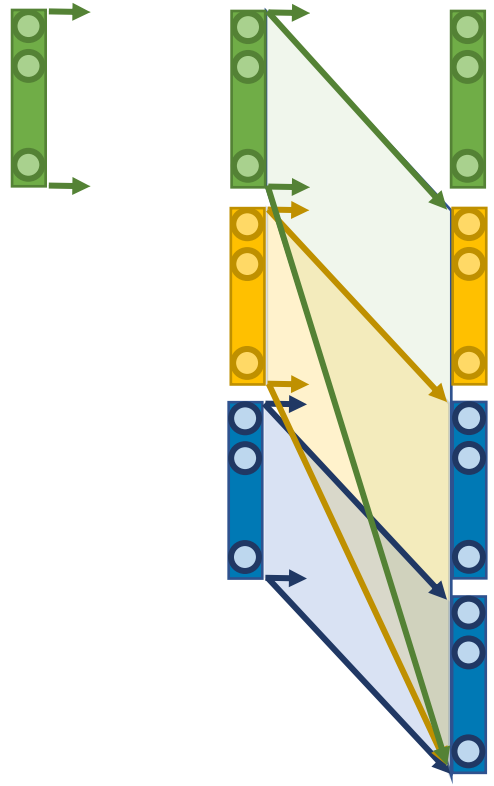




A Closer Look into AdaNet Structure

- One might ask: what is the difference between this structure and a 3-layer fully connected network?

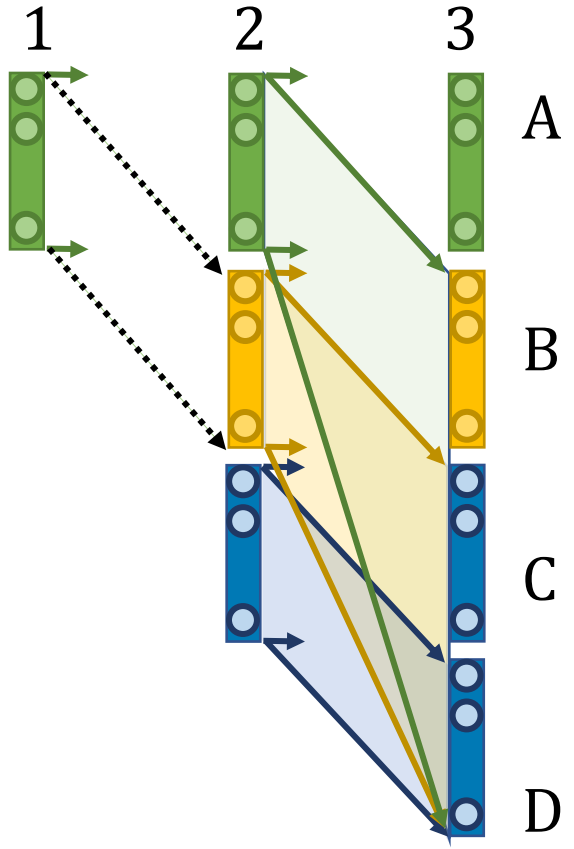
3-layer fully connected network



A Closer Look into AdaNet Structure

- One might ask: what is the difference between this structure and a 3-layer fully connected network?
  - Firstly, not all layers are densely connected
    - **<1A, 2B>**, <1A, 2C>, <1A, 2D>
    - <2B, 3A>, <2C, 3A>, <2C, 3B>

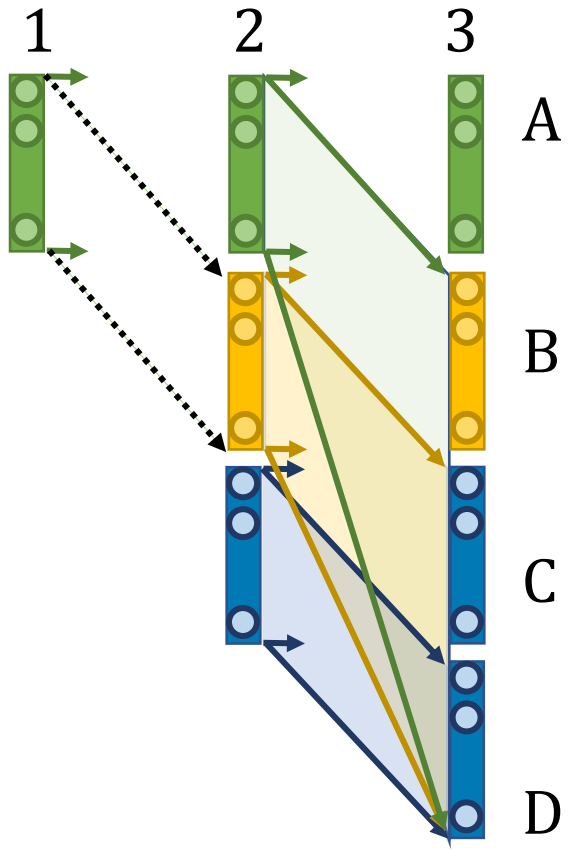
3-layer fully connected network



A Closer Look into AdaNet Structure

- One might ask: what is the difference between this structure and a 3-layer fully connected network?
  - Firstly, not all layers are densely connected
    - **<1A, 2B>**, <1A, 2C>, <1A, 2D>
    - <2B, 3A>, <2C, 3A>, <2C, 3B>
  - Secondly, layers are trained in different order
    - Adding new layers step by step changes loss surface

3-layer fully connected network



Sum Up!

Model	Training Steps	Training Time	Validation Time	AUC
Logistic Reg	120,232	1 hour 45 mins	**20 mins**	66.3%
Manual Tuned	120,232	2 hours 05 mins	30 mins	66.8%
Random_Bst_NN	120,232	2 hours 35 mins	35 mins	67.2%
AdaNet	2 iterations for structure tuning, down-sampled 2000 steps per iteration	**36 mins**	31 mins	**67.6%**

Compared with baseline Logistic Regression model:
We achieved **a 3 × speed-up** and **a lift of AUC of 1.3%.**

Current:

Solution =

Expertise + Data + N × Hours

But can we turn this into:

Solution =

Data + N × Hours (+ Sacrifice of Performance)

Current:

Solution =

Expertise + Data + N × Hours

Yes we can

But ~~can we~~ turn this into:

Solution =

Data + **N/3 × Hours + Lift of Performance**

With Scalable Automatic Tuning!

Acknowledgement

Thank you soooooo much !