Population Based Training
Neural Networks Final Project

Taught by Prof. Aurelio Uncini and Simone Scardapane, PhD.

Ahmed El Sheikh (1873337)
Michal Ostyk-Narbutt (1854051)
Ismagil Uzdenov (1873718)

July 17th, 2019

SAPIENZA
UNIVERSITÀ DI ROMA

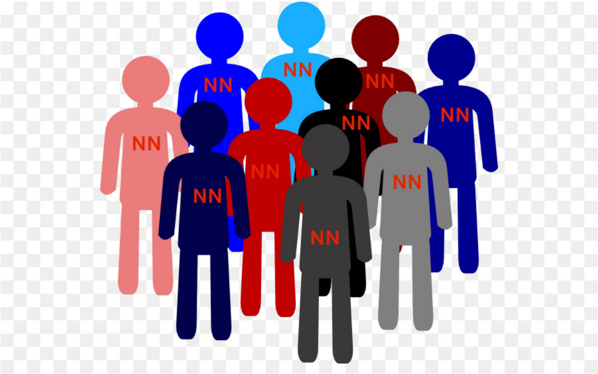
Presentation outline

Outline

Introduction

Deep Learning flourishes, and building a network is quite easy, however, hyperparameter fine tuning is still a challenge.

→ This is where DeepMind introduced a new optimization technique known as **Population-Based Training** of neural networks.



Outline

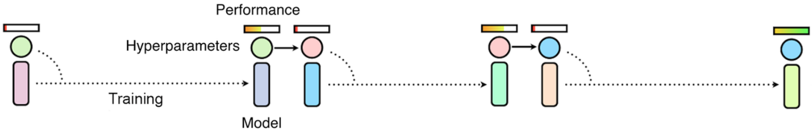
Optimization techniques

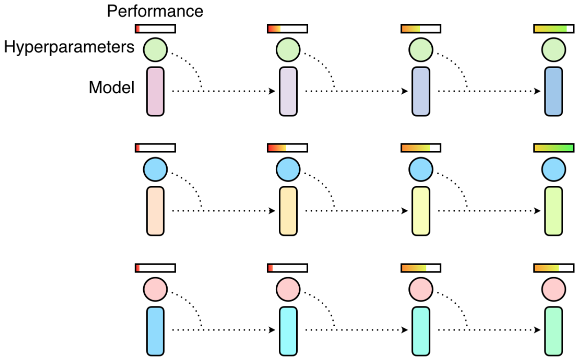
Optimization techniques have been there for a long time, and they are categorized into the following categories:

1. Parallel Search: Training multiple neural networks with different set of hyperparameters and choosing the best out of them (e.g. Grid Search and Random Search.)
2. Sequential Optimization: same paradigm as of parallel search, and use its output to gradually increase the NN performance gradually (e.g. Manual Tuning and Bayesian Optimization.)

Optimization techniques

Parralel(bottom) and Sequential (top) Optimization





Outline

Introduction

Population Based Training – PBT

Motivation: We want hyperparameter tuning to be:

- faster (in terms of wall clock) (Parralel - yes, Sequential - no)
- cheaper (in terms of resources consumed) (Parralel - no, Sequential - yes)
- easier (requiring less domain knowledge and human input) (Parralel - yes, Sequential - no)

- Optimization technique using the best out of both worlds parallel searching as well as sequential optimization.
- It trains a population of Neural Networks asynchronously, it favors information sharing across the population, allowing online transfer of hyperparameters across the population

Problem Formulation

Population Based Training – PBT

Aim in Neural Networks

- optimise parameters of a network θ of a model f to maximise a given objection function $\hat{Q}$
- θ (trainable parameter) is updated via SGD
- actual performance metric Q is often different to $\hat{Q}$

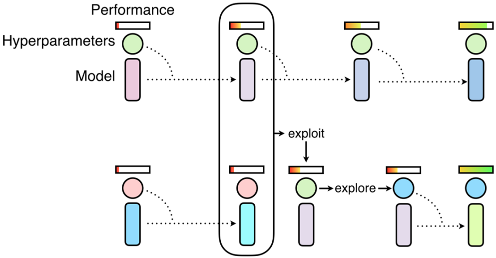
Aim in PBT

- Provide a way to jointly optimise both the parameters θ and hyperparameters h
- Finding optimal model weights: $\theta^* = \text{argmax}_{\theta \in \Theta} \texttt{eval}(\theta)$
- However, iterative approach is computationally expensive and inefficient.

Methodology

Population Based Training – PBT

1. Exploit: which selects to abandon the current member and focus on more promising members, given the performance of the whole population
2. Explore: which given the current solution and hyperparameters proposes new ones to possibly improve the solution space.



Algorithm

Population Based Training

5 "functions" needed to implement PBT

step $\rightarrow$ eval $\rightarrow$ ready $\rightarrow$ exploit $\rightarrow$ explore

Algorithm 1 Population Based Training (PBT)

procedure TRAIN($\mathcal{P}$) $\triangleright$ initial population $\mathcal{P}$
  for $(\theta, h, p, t) \in \mathcal{P}$ (asynchronously in parallel) **do**
    while not end of training **do**
      $\theta \leftarrow \texttt{step}(\theta|h)$ $\triangleright$ one step of optimisation using hyperparameters h
      $p \leftarrow \texttt{eval}(\theta)$ $\triangleright$ current model evaluation
      if $\texttt{ready}(p, t, \mathcal{P})$ **then**
        $h', \theta' \leftarrow \texttt{exploit}(h, \theta, p, \mathcal{P})$ $\triangleright$ use the rest of population to find better solution
        if $\theta \neq \theta'$ **then**
          $h, \theta \leftarrow \texttt{explore}(h', \theta', \mathcal{P})$ $\triangleright$ produce new hyperparameters h
          $p \leftarrow \texttt{eval}(\theta)$ $\triangleright$ new model evaluation
        end if
      end if
      update $\mathcal{P}$ with new $(\theta, h, p, t+1)$ $\triangleright$ update population
    end while
  end for
  return θ with the highest p in $\mathcal{P}$
end procedure

Outline

Experiments

Based on the Empirical world we are living in, experiments are there to prove how good is PBT on different tasks

1. Image Classification
2. Generative Adversial Networks (GANs)
3. Neural Machine Translation (NMT)
4. Deep Reinforcement Learning (DRL)

Tools

Datasets used were open source. Both Tensorflow and Pytorch were used.

Quick introduction

Image Classification

Experimental setup

- Dataset: CIFAR-10
- Neural Network: small Fully-conneted NN with L1-regularization.

Application of PBT

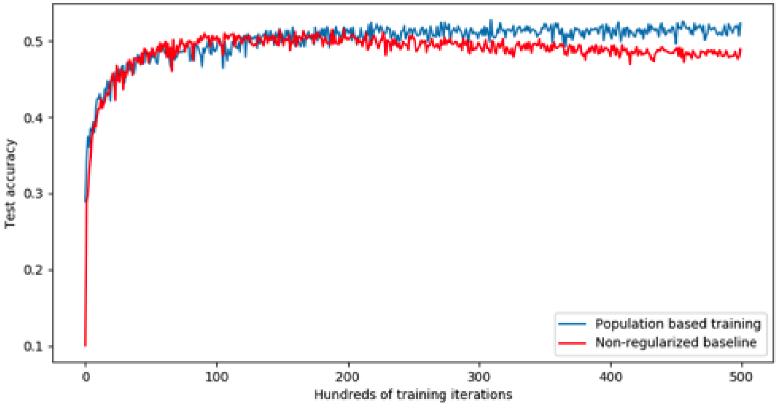
- 50K iteration training
- Population of 10 Neural Networks
- Exploit the best 3 models, by replacing the rest with the best hyperparameters and weights.
- Explore by perturbing (adding noise to our regularizer value) to the worst 3 models. Noise(normal distribution) with $\mu = 0$ and $\sigma = 0$.

Results using PBT

Image Classification

Evaluation

Performed based on the accuracy of prediction.

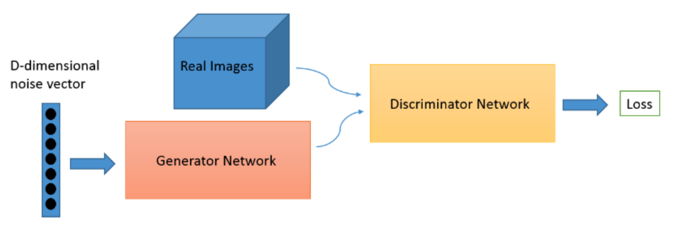


Quick introduction

Generative Adverserial Networks — GANs

Problem statement

- GANs: networks that have the ability to generate new content from the inputs probability distribution



Optimization of GANs

one of the most fragile optimization problems, unlucky random initialization might cause the Network to diverge

Application of PBT

Generative Adverserial Networks

Experimental setup

- Aim: optimize the learning rates of both generative and discriminative models
- population of 20 workers (NN)
- evaluate model using inception score which measures the quality of samples produced, and their diversity
- Generator architecure: de-convolution
- Discriminator architecure: convolution

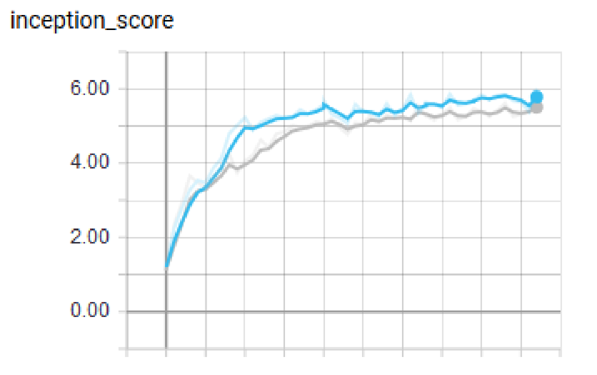
Flow of architecture

2D Conv » LeakyReLu » 2D Conv » BatchNorm » 2D Conv » Batch Norm followed by output layer

Results using PBT

Generative Adverserial Networks

- Dataset: CIFAR-10
- batch-size: 64
- Each GAN trains for 200 epochs
- evaluation interval: $N_{epochs} \times N_{batches}$

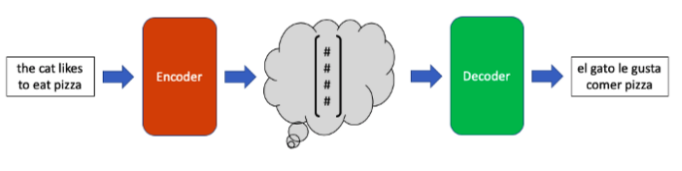


Quick introduction

Neural Machine Translation

Problem statement

- Translate text from one language to another using Neural Networks



NMT model

- output/input – sequence of tokenised words
- 2 RNN (recurrent neural networks)
  1. Encoder RNN – accepts a sequence, produces a context vector
  2. Decoder RNN – uses the context vector, as well as, the target sequence to predict the output word by word

Application of PBT

Neural Machine Translation

PBT for NMT

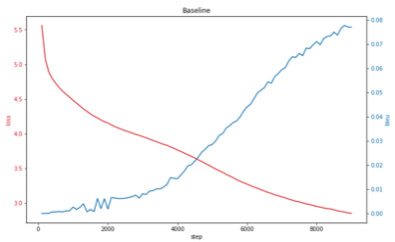
- hyperparmeters to optimise:
  - learning rate
  - attention, layer, ReLU dropout rates
- Model Evaluation: BLEU(bilingual evaluation understanding) score metric, where 1.0 is perfect, and 0.0 is the worst.

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, 30)	0	
decoder_inputs (InputLayer)	(None, 29)	0	
encoder_embed (Embedding)	(None, 30, 100)	2497600	encoder_inputs[0][0]
decoder_embed (Embedding)	multiple	5014100	decoder_inputs[0][0]
encoder_rnn (GRU)	[(None, 30, 96), (No	56736	encoder_embed[0][0]
decoder_rnn (GRU)	multiple	56736	decoder_embed[0][0] encoder_rnn[0][1]
attention1 (Dot)	(None, 29, 30)	0	decoder_rnn[0][0] encoder_rnn[0][0]
attention2 (Activation)	(None, 29, 30)	0	attention1[0][0]
attention3 (Dot)	(None, 29, 96)	0	attention2[0][0] encoder_rnn[0][0]
decoder_attention (Concatenate)	(None, 29, 192)	0	attention3[0][0] decoder_rnn[0][0]
dense (Dense)	multiple	18528	decoder_attention[0][0]
dropout (DropoutHP)	multiple	0	dense[0][0]
prediction (Dense)	multiple	4863677	dropout[0][0]

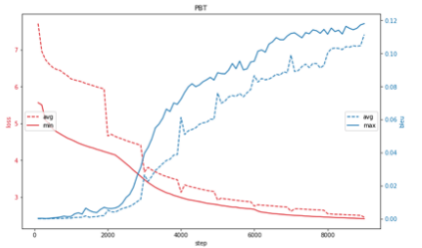
Results using PBT

Neural Machine Translation

- Dataset: 40% of europarl v7 $\approx$ 576000 sentences (English to German)
- Evaluation every 100 steps against a val set of 6k sentences.
- population of 8 models



Baseline showing BLEU score of 0.07951 after 8000 steps



As we can see the effect of PBT reaching better results in 8000 steps

Quick introduction

Deep Reinforcement Learning

Problem statement

- Goal: find an optimal policy that maximizes the expected reward collected by the agent

Deep Double Sarsa (double learning variation of the Sarsa algorithm)

- TD learning algorithm used for control problems. Instead of directly optimizing policy, it estimates state-action values. $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$
- update: $Q^A(S_t, A_t) \Leftarrow Q^A(S_t, A_t) + \alpha[R_{t+1} + \gamma Q^B(S_{t+1}, A_{t+1}) - Q^A(S_t, A_t)]$
- loss function: $Y^A = r + \gamma Q(s', a'; \theta^B) - Q(s, a; \theta^A)$

Application of PBT

Deep Reinforcement Learning

PBT for DRL

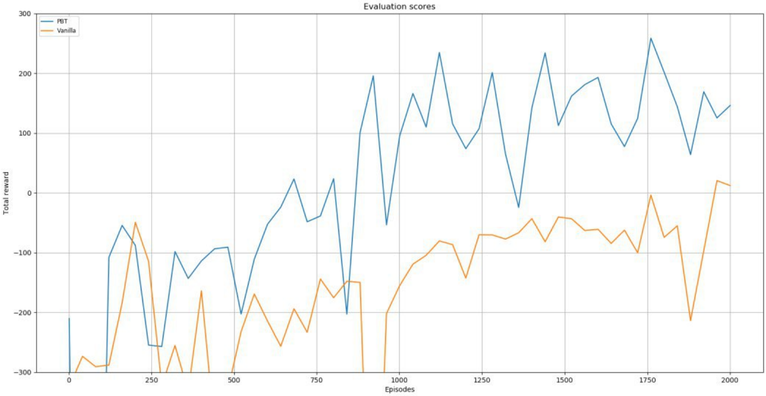
- hyperparmeters to optimise:
  - learning rate
- Training:
  - 100 episodes of training $\rightarrow$ 20 episodes of evaluation
  - repeated 20 times
  - Model Evaluation: average return of the 20 evaluation episodes.
  - 4 NNs in the population get replaced by one of the 4 best NNs (random selection of the best).

Experimental setup

- OpenAI Gym's LunarLander-v2 environment
- 2000 episodes of training time, with ϵ going from 1.0 to 0.1.

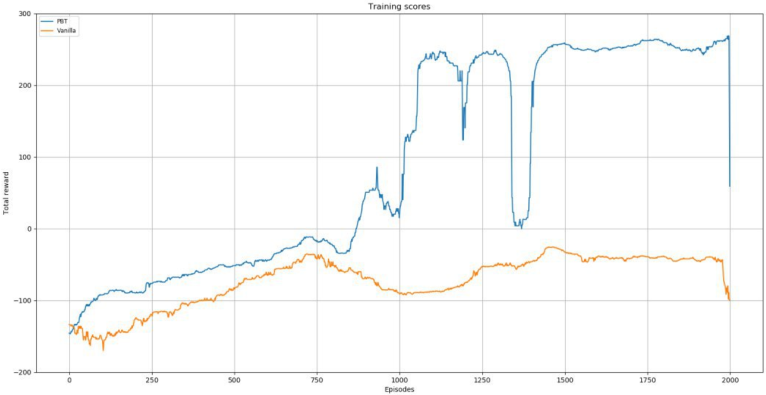
Results using PBT

Deep Reinforcement Learning



Results using PBT

Deep Reinforcement Learning



Outline

Conclusions

concluding points

1. Implemented a DeepMind paper for Population Based training for the purpose of Image classification, GANs, NMT, and DRL.
2. PBT affects the models' performance in the same amount of steps showing better results
3. PBT has yielded consistent improvements in optimizing model weights and hyperparameters
4. fixed set of hyperparameters is not as good as adaptive schedule of hyperparameters tuning

Conclusions

Future work and improvements

1. Image classification: Use a more complex dataset i.e. Venice boats and CNN
2. GAN: Use different GAN implementation i.e. DCGAN
3. NMT: using entire dataset, automatic padding
4. DRL: Use a more stable algorithm