Tensorforce:
building an applied reinforcement learning framework using TensorFlow

Alexander Kuhnle

28th January 2020
Content

- Motivation
- Key features
- TensorFlow as implementation platform
- User feedback
- Applications
About the framework

- Originally developed by Michael Schaarschmidt, Kai Fricke and myself
- Introduction blog post: 11/07/2017
- Since mid-2018 developed by myself
- GitHub: https://github.com/tensorforce/tensorforce
- ~200 pull requests by ~50 contributors
Why build yet another reinforcement learning library?
Existing frameworks

Example: OpenAI Baselines

Largely independent agent implementations
Research frameworks vs practical requirements

- “Standardized” state/action space: single float-array state, int/float action
- States/action space with multiple components, various types and shapes
Research frameworks vs practical requirements

- "Standardized" state/action space:
  single float-array state, int/float action

- States/action space with multiple components, various types and shapes
Research frameworks vs practical requirements

- “Standardized” state/action space: single float-array state, int/float action
- Interaction in training episodes with terminal/goal state
- States/action space with multiple components, various types and shapes
- Continuous interaction, no “natural” termination of interaction
Research frameworks vs practical requirements

- “Standardized” state/action space:
  single float-array state, int/float action

- Interaction in training episodes with terminal/goal state

- States/action space with multiple components, various types and shapes

- Continuous interaction, no “natural” termination of interaction
Research frameworks vs practical requirements

- “Standardized” state/action space: single float-array state, int/float action
- Interaction in training episodes with terminal/goal state
- Agent reference implementations, may include environment-specific details

- States/action space with multiple components, various types and shapes
- Continuous interaction, no “natural” termination of interaction
- (Re-)combination of techniques to suit characteristics of application
Research frameworks vs practical requirements

- “Standardized” state/action space: single float-array state, int/float action
- Interaction in training episodes with terminal/goal state
- Agent reference implementations, may include environment-specific details
- Mix of Python and TensorFlow(/PyTorch)

- States/action space with multiple components, various types and shapes
- Continuous interaction, no “natural” termination of interaction
- (Re-)combination of techniques to suit characteristics of application
- Single implementation platform
Tensorforce: “TF Estimators for RL”

Modular component-based library design
Tensorforce: “TF Estimators for RL”

Modular component-based library design

No fundamental differences internally, all a matter of modular configuration!
Tensorforce: “TF Estimators for RL”

Usage example: DQN agent (configured manually, for illustration)

```python
agent = Agent.create(
    policy=dict(network='auto', temperature=0.0),
    memory=dict(type='replay', capacity=100000),
    update=dict(unit='timesteps', batch_size=64, frequency=8),
    optimizer=dict(type='adam', learning_rate=3e-4),
    objective=dict(type='value', value='action', huber_loss=0.0),
    reward_estimation=dict(horizon=0, discount=0.99, estimate_horizon='late'),
    baseline_policy=dict(network='auto', temperature=0.0),
    baseline_optimizer=dict(type='synchronization', update_weight=0.2)
)

states = environment.reset()
actions = agent.act(states=states)
states, terminal, reward = environment.execute(actions=actions)
agent.observe(terminal=terminal, reward=reward)
```
Key features of Tensorforce
Reinforcement learning architecture
Reinforcement learning architecture

- Environment
- State
- Action
- Agent
- Policy
- Act()
Reinforcement learning architecture

- Environment
  - State
  - Action
  - Reward

- Agent
  - Observe
  - Act
  - Policy
Reinforcement learning architecture

- **Environment**: state → action
- **Agent**:
  - **Policy**: act() → update
  - **Optimization**: sample batch → store
  - **“Short-term” buffer**: observe() → store
  - **“Long-term” memory**: store

- **State**
- **Action**
- **Reward**
Reinforcement learning architecture

- Environment
  - State
  - Action
  - Reward

- Agent
  - Policy
  - Reward Estimation
  - Optimization
  - “Long-term” Memory

- “Short-term” Buffer
  - Store
  - Sample Batch
  - Update

- Act() 
- Observe()
Reinforcement learning architecture

Environment:
- state
- action
- reward

Agent:
- act()
- observe()
- update

Policy:
- “short-term” buffer
- “long-term” memory
- reward estimation
- critic/target
- auxiliary losses
- curiosity, etc

Optimization:
- sample batch
- store
Reinforcement learning architecture

- **Environment**
  - Input: State, Action, Reward
  - Output: Action

- **Agent**
  - **Act()**: Policy
  - **Observe()**: Short-term buffer
  - **Update()**: Optimization, Long-term memory

- **Policy**
  - Update

- **Optimization**
  - Sample batch
  - Update

- **Reward Estimation**
  - Critic/Target
  - Auxiliary losses
  - Curiosity, etc.

- **Short-term buffer**
  - Store

- **Long-term memory**
  - Update

- **Diagram Components**
  - Reinforcement learning architecture overview
  - Flow of data: State → Policy → Observation → Reward → Update → Memory
RL timestep and dependencies
RL timestep and dependencies

... \[ t-1 \quad t \quad t+1 \quad ... \]

- state
- reward
- action
RL timestep and dependencies
RL timestep and dependencies
RL timestep and dependencies

Policy gradient

state, action, rewards
RL timestep and dependencies
RL timestep and dependencies

- Internal state:
  - States
  - Future horizon
  - Past horizon

- Action

- Rewards
  - V(s)
  - Q(s,a)
RL timestep and dependencies

- Internal state
- States
- Action
- Rewards
- Future horizon
- Past horizon
- V(s)
- Q(s,a)
RL timestep and dependencies

- Internal state
- States
- Action
- Rewards
- Future horizon
- Past horizon
- Internal state (terminal, aborted?)
- V(s)
- Q(s, a)
RL timestep and dependencies

Actual batch instance consists of:
- internals[t-p]
- states[t-p+1:t]
- actions[t]
- internals[t+f-p-1]
- states[t+f-p:t+f]
- rewards[t:t+f]
- (actions[t+f])
Other framework features

Optimizers as graph assemblers:
- TensorFlow 1.X: based on loss-tensor
- Keras/PyTorch: based on loss-function
Other framework features

Optimizers as graph assemblers:

- TensorFlow/Keras/PyTorch: based on loss-tensor/-function
- Tensorforce
  - generic “updaters” with a range of potential inputs: loss, KL-divergence, source-vars, etc
  - update modifiers: multi-step, update clipping, batch subsampling
Other framework features

Optimizers as graph assemblers:
- TensorFlow/Keras/PyTorch: based on loss-tensor/-function
- Tensorforce
  - generic “updaters” with a range of potential inputs: loss, KL-divergence, source-vars, etc
  - update modifiers: multi-step, update clipping, batch subsampling

Static vs dynamic hyperparameters:
- TensorFlow/PyTorch: seemingly only learning-rate

  tf.keras.optimizers.schedules.LearningRateSchedule
  torch.optim.lr_scheduler.*
Other framework features

Optimizers as graph assemblers:

- TensorFlow/Keras/PyTorch: based on loss-tensor/-function
- Tensorforce
  - generic “updaters” with a range of potential inputs: loss, KL-divergence, source-vars, etc
  - update modifiers: multi-step, update clipping, batch subsampling

Static vs dynamic hyperparameters:

- TensorFlow/PyTorch: seemingly only learning-rate
- Tensorforce:
  - All dynamic parameters are of type Parameter: constant, decaying, piecewise, etc
  - Parameters scheduled based on timestep/episode/update,... (loss?)
  - Placeholder-with-default for straightforward experimentation
TensorFlow as an implementation platform
TensorFlow as an implementation platform

- Static graph compilation great for verification and TF/Python separation
TensorFlow as an implementation platform

- Static graph compilation great for verification and TF/Python separation

- However, problems have persisted with respect to:
  - Nesting while and cond in combination with gradients and TensorBoard summaries
  - Recently, almost every TF upgrade breaks one thing and/or fixes another
TensorFlow as an implementation platform

- Static graph compilation great for verification and TF/Python separation

- However, problems have persisted with respect to:
  - Nesting while and cond in combination with gradients and TensorBoard summaries
  - Recently, almost every TF upgrade breaks one thing and/or fixes another

- TensorFlow 2.0: Exceptions seem harder to interpret
TensorFlow as an implementation platform

- Static graph compilation great for verification and TF/Python separation

- However, problems have persisted with respect to:
  - Nesting while and cond in combination with gradients and TensorBoard summaries
  - Recently, almost every TF upgrade breaks one thing and/or fixes another

- TensorFlow 2.0: Exceptions are harder to interpret

- TensorFlow 2.1: Version upgrades still change/break basic things
User feedback
Reasons for choosing Tensorforce

- No code digging: easy to get started and obtain results
- Modular structure: clean API and extensive configurability
- Full-on TensorFlow: computation graph can be extracted
- Focus on “core RL” performance, in particular reward estimation
Reasons for choosing Tensorforce

- No code digging: easy to get started and obtain results
- Modular structure: clean API and extensive configurability
- Full-on TensorFlow: computation graph can be extracted
- Focus on “core RL” performance, in particular reward estimation

Limitations and areas for development

- No code digging: hard to modify/extend beyond what’s supported
- Modular structure: no single script, no SOTA reference implementations
- Full-on TensorFlow: incomprehensible exceptions
- No focus on sophisticated hardware management and distributed execution
Applications
DeepCrawl: DRL-controlled game AI
(Alessandro Sestini, Università degli Studi di Firenze)
DeepCrawl: DRL-controlled game AI
(Alessandro Sestini, Università degli Studi di Firenze)

RL / Tensorforce takeaways:
- State space with multiple components
  - Global and ego-centric views of map
  - Categorical and continuous game state values
- Handling of discrete values
  - Main motivation for auto-network
- Exploration also for imperfect behavior
- Deployment to C#

GitHub: https://github.com/SestoAle/DeepCrawl
Flow Control of the 2D Kármán Vortex Street
(Jean Rabault et al., University of Oslo)
Flow Control of the 2D Kármán Vortex Street
(Jean Rabault et al., University of Oslo)

RL / Tensorforce takeaways:

- Costly simulations using FEniCS
  - Simple parallelized environment execution
  - Speedup almost linear <= 60

- Importance of choosing the right characteristic timescales
  - Agent vs simulation timestep rate
  - Horizons and terminal

GitHub: https://github.com/jerabaul29/Cylinder2DFlowControlDRL
Direct shape optimization through DRL

(Jonathan Viquerat et al., MINES ParisTech)

(a) Best shape with 4 points, 1 free point (3 d.o.f.s)
(b) Best shape with 4 points, 3 free points (9 d.o.f.s)
(c) Best shape with 4 points, 4 free points (12 d.o.f.s)
(d) Computed $v_x$ velocity field at $Re \sim 600$ around shape 5c (the domain is cropped).
Direct shape optimization through DRL  
(Jonathan Viquerat et al., MINES ParisTech)

RL / Tensorforce takeaways:

- Importance of state/action parametrization
  - Unambiguous, normalized

- “Degenerate” 1-step RL
  - Non-differentiable optimization

- Potential of reward shaping:
  - Constraints via additional terms

Autonomous order dispatching in the semiconductor industry
(KIT Institute of Production Science, Infineon)
Autonomous order dispatching in the semiconductor industry
(KIT Institute of Production Science, Infineon)

RL / Tensorforce takeaways:
- Agent embedded in simulation framework
- Multiple workers controlled by the same RL agent interacting simultaneously
  - Different type of parallelized execution
- Masking of invalid actions

Paper: https://publikationen.bibliothek.kit.edu/1000091435
And more...

- Drones, autonomous driving
- Recommender systems
- (Bitcoin) trading
- Games
And more...

- Drones, autonomous driving
- Recommender systems
- (Bitcoin) trading
- Games
Summary

Tensorforce: “TF Estimators for reinforcement learning”

- Easy-to-use framework for applied DRL
- Fully modular RL library design with extensive configurability
- TensorFlow as only implementation platform
- Vision: enable (non-ML) practitioners to apply DRL in any application
Thanks for your attention!

Questions?