Data Science Training

November 2017

Reinforcement Learning

Xavier Bresson

Data Science and AI Research Centre
NTU, Singapore

http://data-science-optum17.tk

Note: Based on A. Karpathy's blog on reinforcement learning

Outline

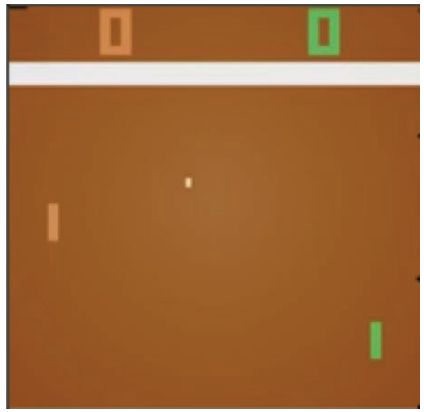
- Introduction
- Policy Network
- Policy Gradient
- Learning to Play Pong
- Non-differentiable Neural Networks
- Conclusion

Outline

- **Introduction**
- Policy Network
- Policy Gradient
- Learning to Play Pong
- Non-differentiable Neural Networks
- Conclusion

Learning Games with RL

- Atari Pong:
  - Sequence of images 210x160x3 = 100K values
  - Move Up or Down
  - Reward:
    - +1 if the ball makes it past the opponent (winning game)
    - -1 if we missed the ball (lost game)
    - 0 otherwise.
- Goal: Move paddle to maximize reward
- Prototype of high-dimensional control problem (e.g. robots)



Outline

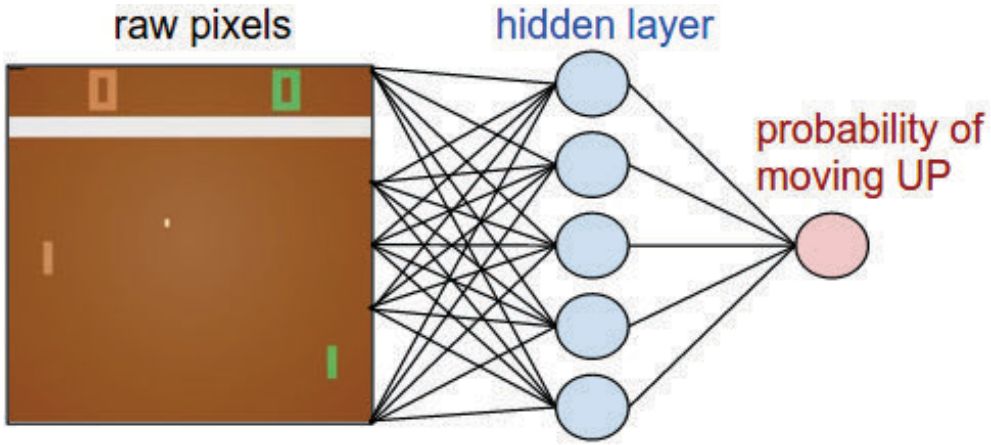
- Introduction
- **Policy Network**
- Policy Gradient
- Learning to Play Pong
- Non-differentiable Neural Networks
- Conclusion

Policy network

- Implement player/agent's actions by a neural network (NN)
- Ex: Single hidden layer NN (#Param=1M)

$$
\begin{aligned}
s &= W_2\sigma(W_1 x) \\
p(x, W) &= \text{sigmoid}(s) \\
x &= I_{t+1} - I_t
\end{aligned}
$$

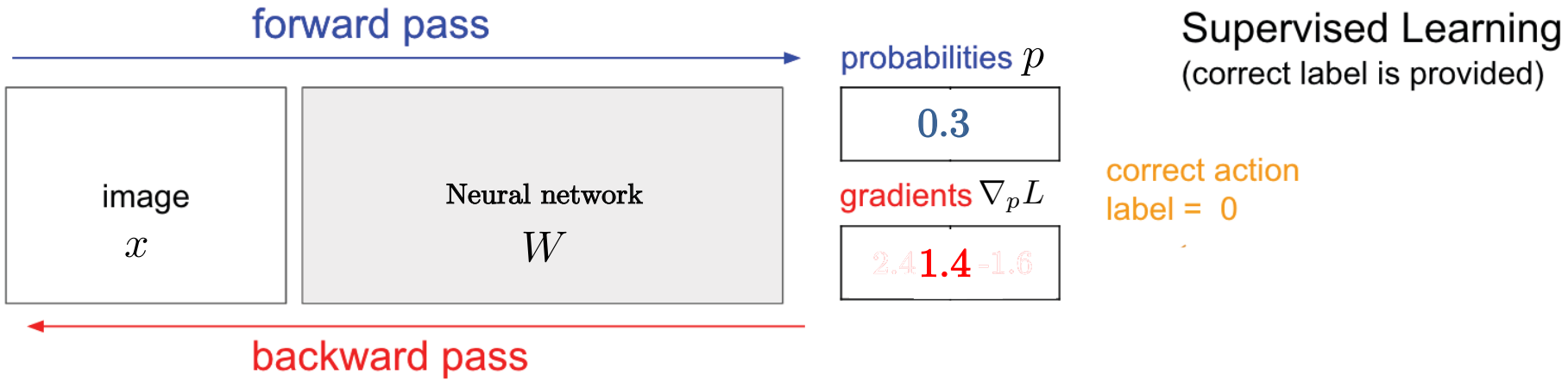
- Challenge:
  - Credit assignment problem (NP-hard)
  - Translate ≈90 frames x 100M values into 1 value (+1 reward winning)
  - Find a path (a sequence of actions/rewards) in high-dim search space that lead to a positive outcome.



Supervised learning

- Standard SL: Feed x to NN and get Pr(Up/Down)
- We know the correct thing to do (label says go Up) and we can backpropagate the gradient of the loss to change the weights W of the NN to do the right thing:

$$
\begin{aligned}
L &= |p(x,W) - p_{\text{label}}|^2/2 \\
W^{iter+1} &= W^{iter} - \tau \,.\, \nabla_W \, L \\
\nabla_W \, L &= \nabla_p \, L \,.\, \nabla_W \, p \\
W^{iter+1} &= W^{iter} - \tau \,.\, (p - p_{\text{label}}) \,.\, \nabla_W \, p
\end{aligned}
$$



Outline

- Introduction
- Policy Network
- **Policy Gradient**
- Learning to Play Pong
- Non-differentiable Neural Networks
- Conclusion

Policy gradient 1/2

- What if we have no label? Solution is policy gradient:

  (1) Use policy network/NN to estimate pr of an action:

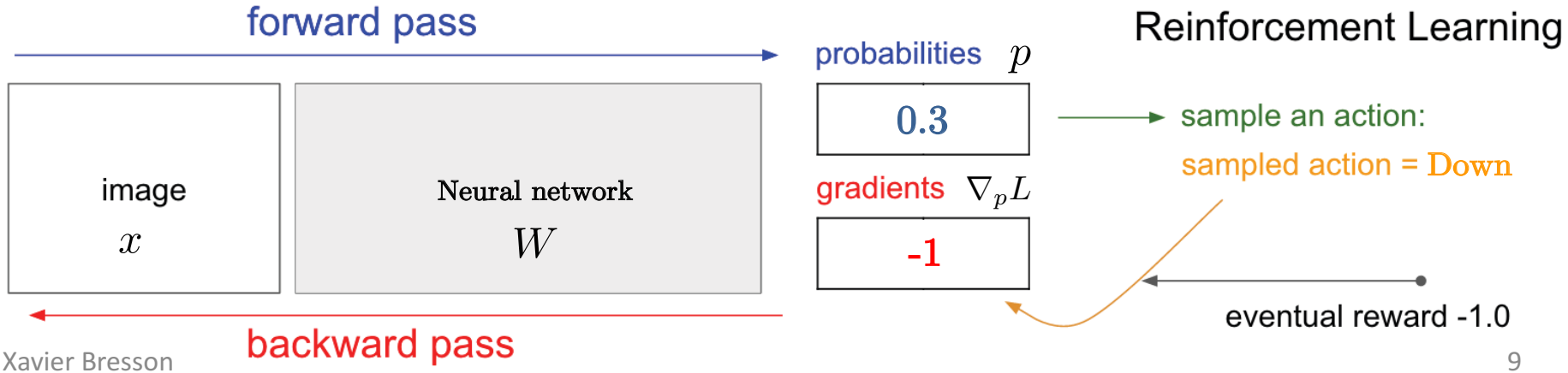
  - 30% pr of going Up given x
  - 70% pr of going Down

  (2) Sample an action from this policy ("flip a coin") ex: action is sampled as going Down and then execute game.

  (3) Repeat until the end of the game and collect reward (+1 or -1)

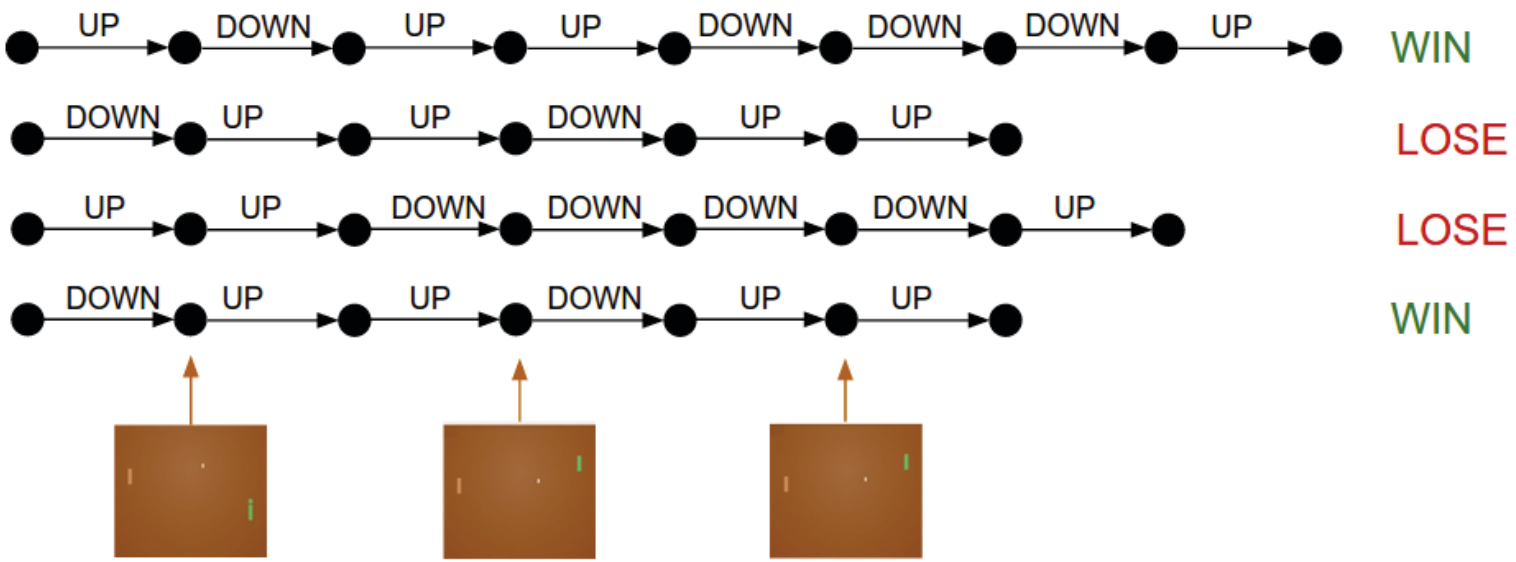
- If we end up losing the game then we discourage NN to take the Down action $\Rightarrow$ $\nabla_p L = -1$ (Note: fake derivative, L not differentiable)

$$W^{iter+1} = W^{iter} - \tau \nabla_p L \; . \; \nabla_W p = W^{iter} - \tau(-1) \; . \; \nabla_W p$$



Policy gradient 2/2

- Definition of PG: A stochastic policy that samples actions, then actions lead to good or bad outcomes, which are then encouraged or discouraged in the NN (by changing the weights).
- Example: Here, we won 2 games and lost 2 games. With PG, we would take the two games we won and slightly encourage every single action we made in that episode. Conversely, we would also take the two games we lost and slightly discourage every single action we made in that episode.



A little formalization

- **SL:** $\min_W - \sum_{i \in \text{labels}} p_W(y_i|x_i), \quad y_i = \text{label}$
- **RL:** $\min_W - \sum_{i \in \text{actions}} r_i . \, p_W(y_i|x_i), \quad y_i = \text{action}$

$$\text{with advantage function } r_i = \begin{cases} +1 \text{ win} \\ -1 \text{ loss} \end{cases}$$

- **For SL, we maximize the pr of good classification with labels.**
- **For RL, we maximize the pr of actions to good outcomes. And inversely, we minimize the pr of actions to bad outcomes.**

Discounted reward

- General case: We receive a reward r_t at every time t, and this "eventual reward" has an influence exponentially less important in time:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad \gamma = 0.99 \text{ (discount factor)}$$

- Normalization: R_t with zero-mean and unit-standard deviation $\Rightarrow$ Better control of the variance of policy gradient estimator (next slide).

Policy gradient formalization

- **Policy reduces to compute expectation of scalar function *f(x)* (reward) under some probability distribution *p(x, W)* parameterized by some *W*:**

$$\mathbb{E}_{x \sim p(x,W)} \left[\, f(x) \, \right]$$

- **Policy gradient:**

$$\begin{aligned}
\nabla_W \mathbb{E}_x \big[f(x) \big] &= \nabla_W \Big(\sum_x p(x, W) f(x) \Big) && \text{definition of expectation} \\
&= \sum_x \nabla_W p(x, W) f(x) && \text{swap linear operators} \\
&= \sum_x p(x, W) \frac{\nabla_W p(x, W)}{p(x, W)} f(x) \\
&= \sum_x p(x, W) \nabla_W \log p(x, W) f(x) && \text{use the fact that } \nabla_\theta \log z = \frac{\nabla_\theta z}{z} \\
&= \mathbb{E}_x \big[f(x) \nabla_W \log p(x, W) \big] && \text{definition of expectation}
\end{aligned}$$

- **Untractable for all x $\Rightarrow$ Sample a few x to estimate it (potential variance issue).**

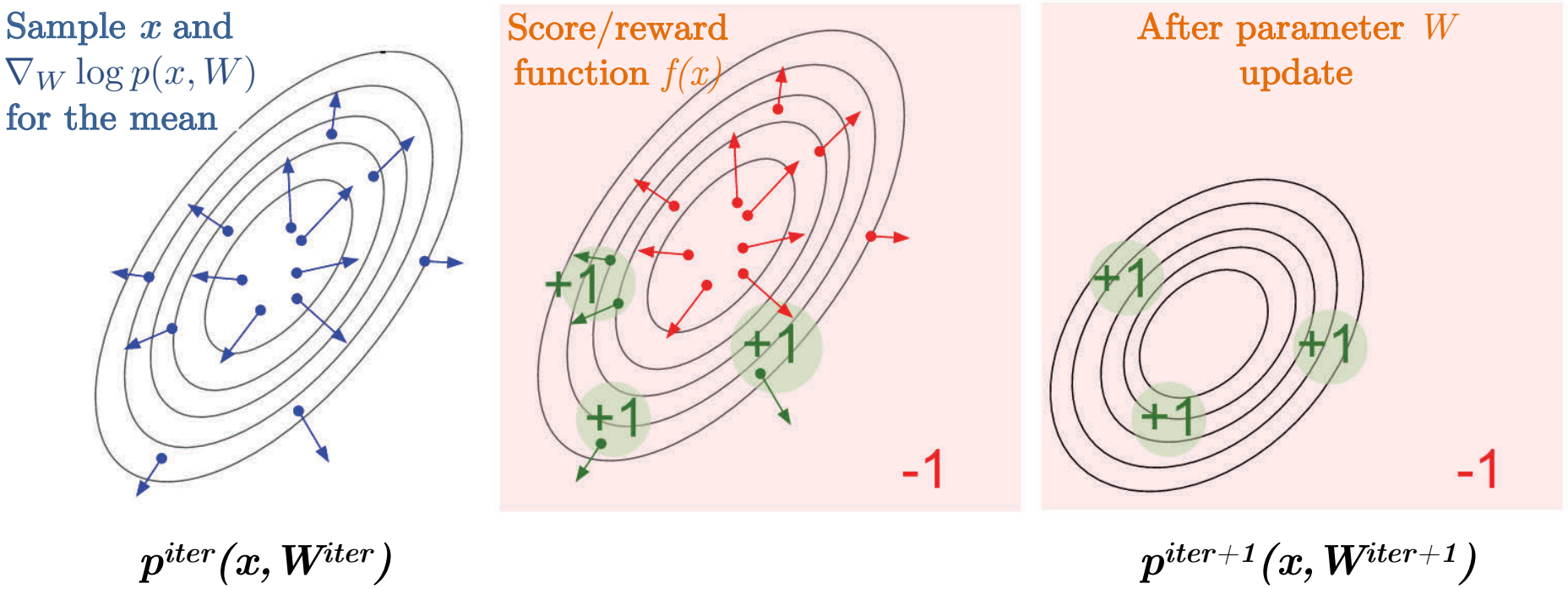
PG for Gaussian distribution

(1) Sample a few points x and compute gradient of sample actions: $\nabla_W \log p(x, W)$

(2) Compute the reward function: $f(x)=+1$ for a win or $f(x)=-1$ for a loss

(3) Add all points x and update W:

$$\begin{aligned} W^{iter+1} &= W^{iter} - \tau \, \nabla_W \mathbb{E}_x(f(x)) \\ &= W^{iter} - \tau \, \mathbb{E}_x(f(x) \nabla_W \log p(x, W)) \\ &\approx W^{iter} - \tau \sum_x f(x) \nabla_W \log p(x, W) \end{aligned}$$



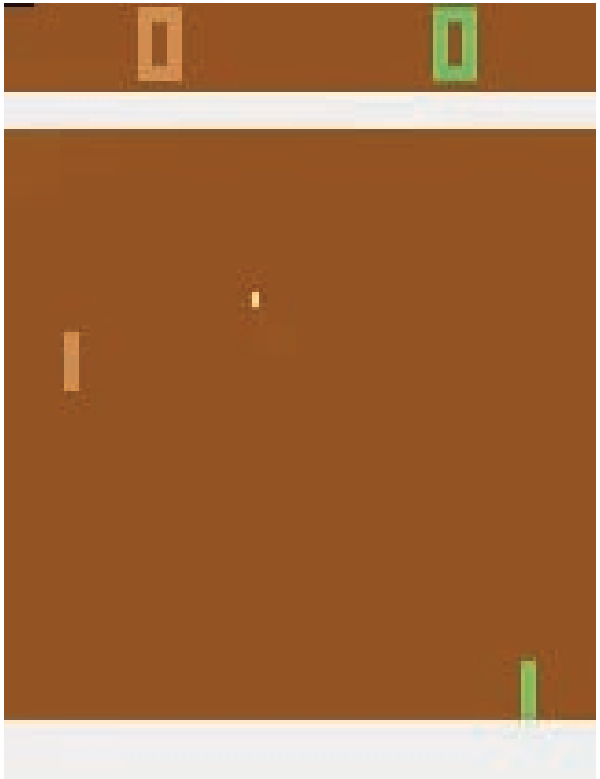
$p^{iter}(x, W^{iter})$ $p^{iter+1}(x, W^{iter+1})$

Outline

- Introduction
- Policy Network
- Policy Gradient
- **Learning to Play Pong**
- Non-differentiable Neural Networks
- Conclusion

Experiments

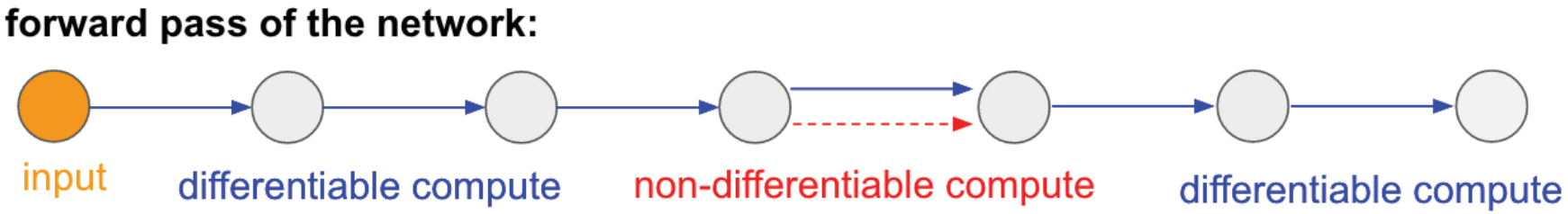
- Pong: 2 layers, 200 neurons
- Train batch: 10 episodes, each episode is ≈ 40 games (21 to win)
- After 200K games:



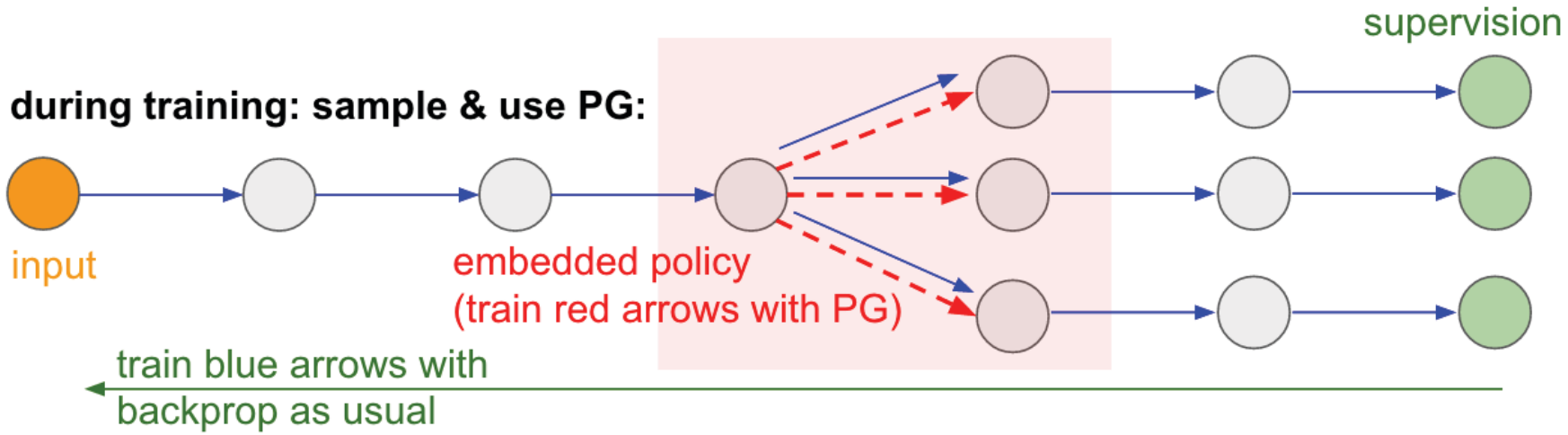
Outline

- Introduction
- Policy Network
- Policy Gradient
- Learning to Play Pong
- **Non-differentiable Neural Networks**
- Conclusion

Non-differentiable NNs

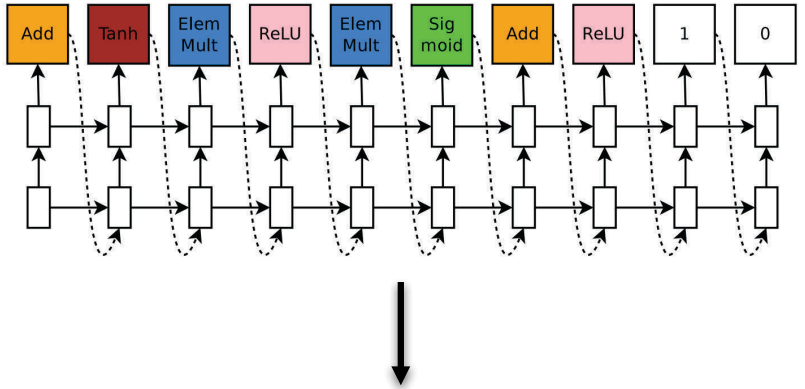


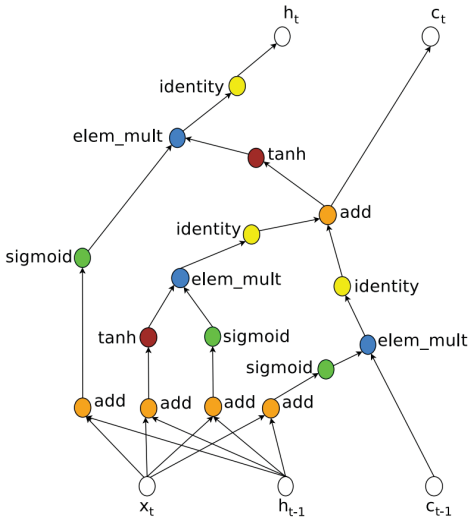
- Use PG:
  - Sample from stochastic policy
  - Keep samples that give good outcomes



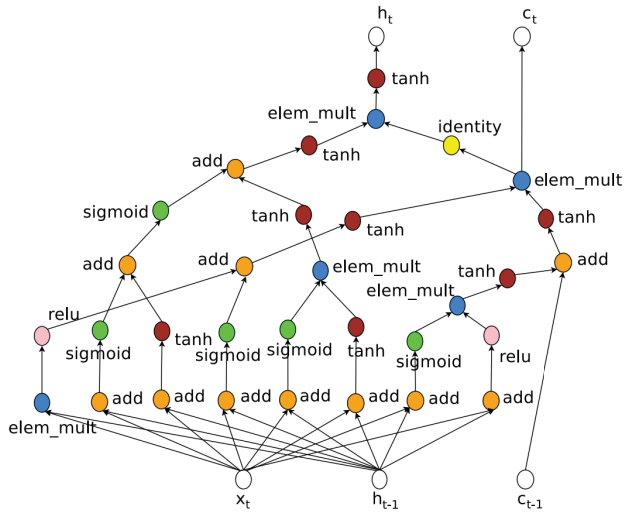
Learn-to-Learn [Zoph-V.Le'17]

- Learn the weights of the architecture and the architecture as well:





Standard LSTM



Learned architecture

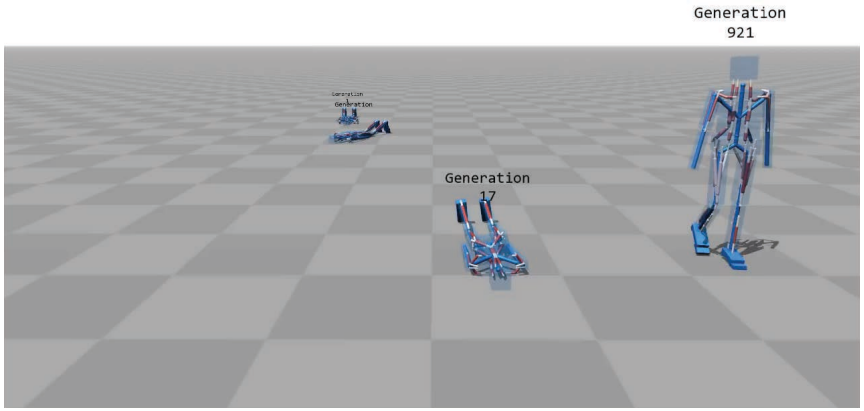
- But 800 GPUs are needed for reinforcement learning.

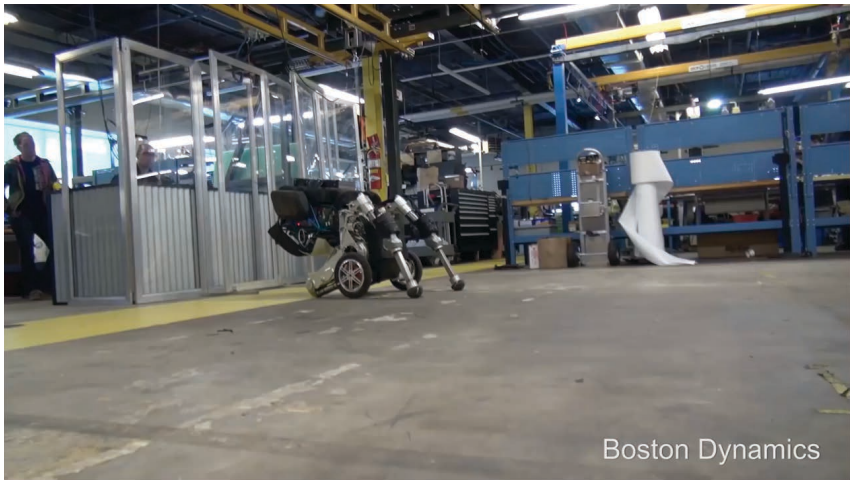
Outline

- Introduction
- Policy Network
- Policy Gradient
- Learning to Play Pong
- Non-differentiable Neural Networks
- **Conclusion**

Conclusion

- Reinforcement learning is hot topic
- AlphaGo
- RL is a lot like SL with a fake gradient that depends on the outcome/award.
- Amazing it works so well: Policy gradient is a brute force solution that discovers by random actions on environment the path to positive rewards.





Questions?