Advanced Topics in Reinforcement Learning

Lecture 16: Deep Reinforcement Learning I

Josiah Hanna
University of Wisconsin — Madison

Announcements

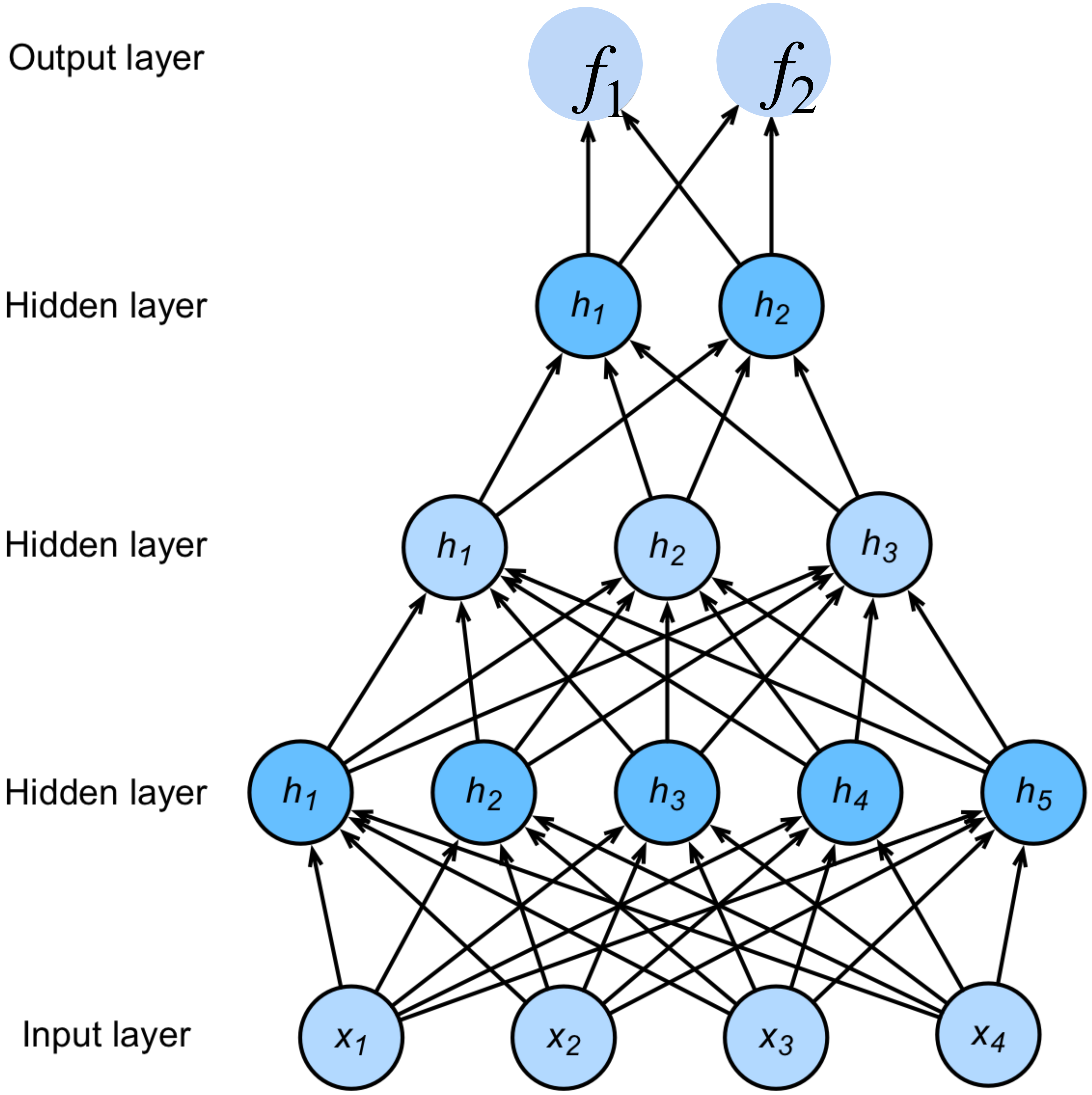
- Literature review due Thursday at 11:59PM Central.
    - Any questions?
- Homework 4 due November 17 (two weeks from Thursday).
- This week: deep RL.
- Next week: policy gradient RL (a new approach to RL!)

Linear Function Approximation Review

- Assume value estimate is a linear function of state-action features.
  - $\hat{q}(s, a, \mathbf{w}) = \mathbf{w}^\top x(s, a) = \sum_{i=1}^{d} w_i x_i(s, a)$
- The features, $x_i(s, a)$, can be non-linear functions of state variables and actions.
  - Expressive choices for $\mathbf{x}(s, a)$ make linear methods more powerful than they first appear.
  - What if we instead learn $\mathbf{x}$ while learning $\hat{q}$? —> Deep RL!!!
- Semi-Gradient Q-learning:
  - $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)) \nabla \hat{q}(S_t, A_t \mathbf{w}_t)$

Neural Network Function approximations

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$



Dyah’s Presentation

- Slides

Hierarchical Representations

- Each layer of a neural network transforms the output of the layer before it (the first layer transforms the input).

- We can say that each layer is producing a new representation of the values at the previous layer.

- Chaining layers together allows the network to learn progressively more complex representations of the data.

    - Pixels —> Edge detectors —> Shape detectors —> object detectors

Neural Network Training in RL

- Semi-Gradient Q-learning:
  - $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)) \nabla \hat{q}(S_t, A_t \mathbf{w}_t)$
- The parameter, $\mathbf{w}_t$, is all weights and biases of the neural network.
- Backpropagation algorithm: use chain rule of calculus to derive gradient of network outputs with respect to each weights or bias of the network.
- Adjust each weight in proportion to gradient of output times TD-error.

Problems with Backpropagation

- With many layers, gradients for parameters in the initial layers are either very large or very small. Why?
  - Chain rule means the gradient is a product of many factors. Gradient magnitude can grow or decay exponentially in network depth.
- Alternative #1: Find a different learning rule.
  - Evolution; Reinforcement Learning.
- Alternative #2: Make our networks more backprop-friendly.

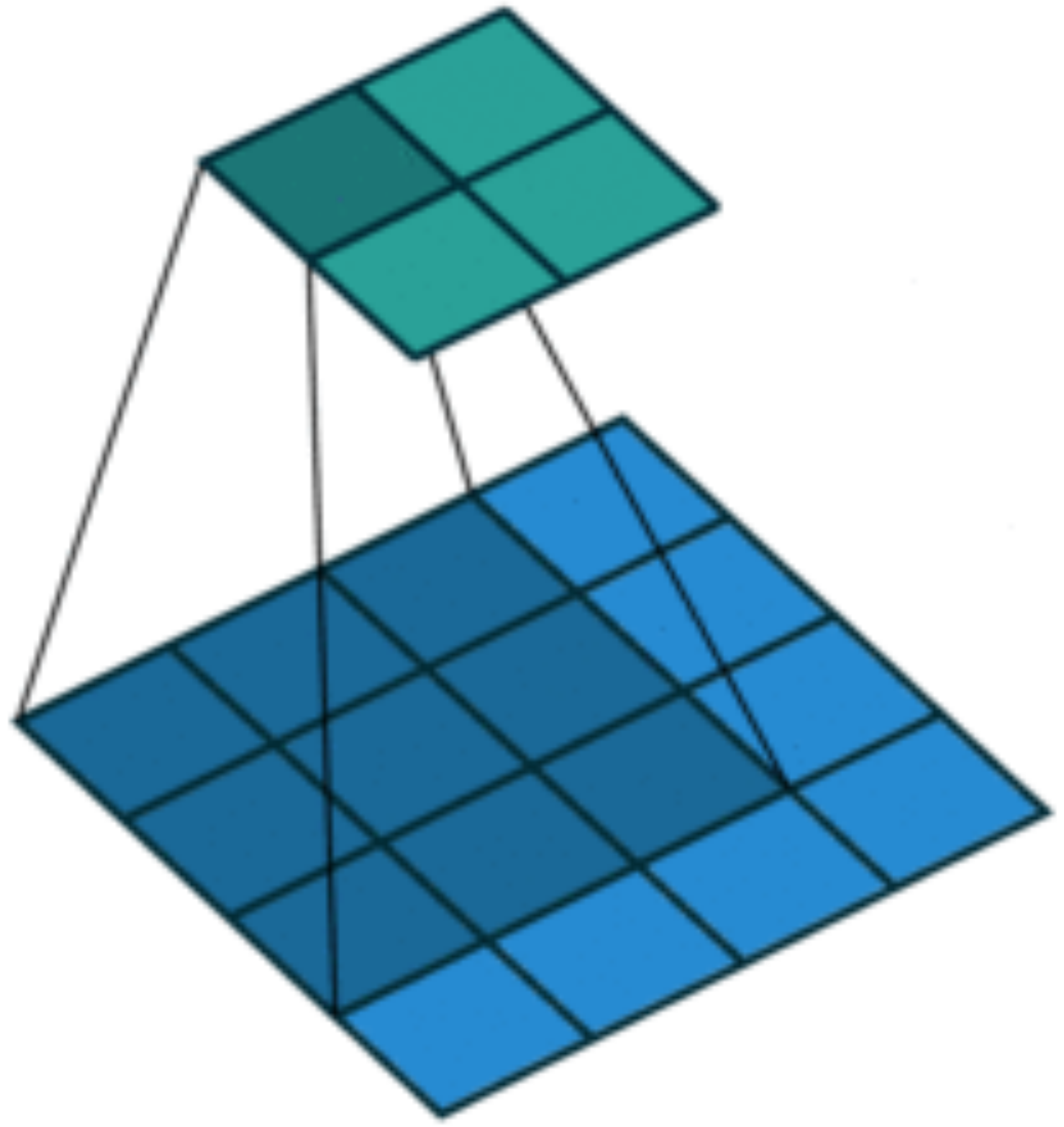
Training Improvements

- Regularization: methods that encourage learning simple models over more complex ones.
    - Add $||\mathbf{w}||_2^2$ as an additional objective; drop-out
- Better initializations: start gradient descent at a good location in weight space.
    - Pre-training; special initialization rules (e.g., Glorot/Xavier initializations)
- Architecture improvements
    - Residual connections, normalize inputs and layer activations.
- Not all of these ideas work well for reinforcement learning!

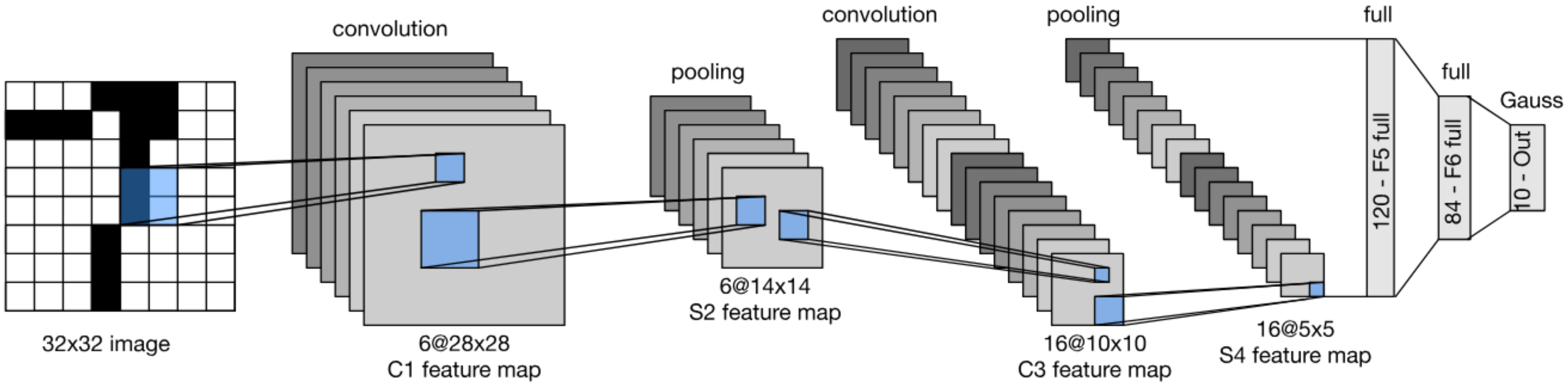
http://joschu.net/docs/nuts-and-bolts.pdf

Convolutional Neural Networks

- Special architecture primarily for processing visual inputs.
- Local connections between inputs and outputs at next layer.
- Learn “filters” that have the same weights no matter where applied on an image.

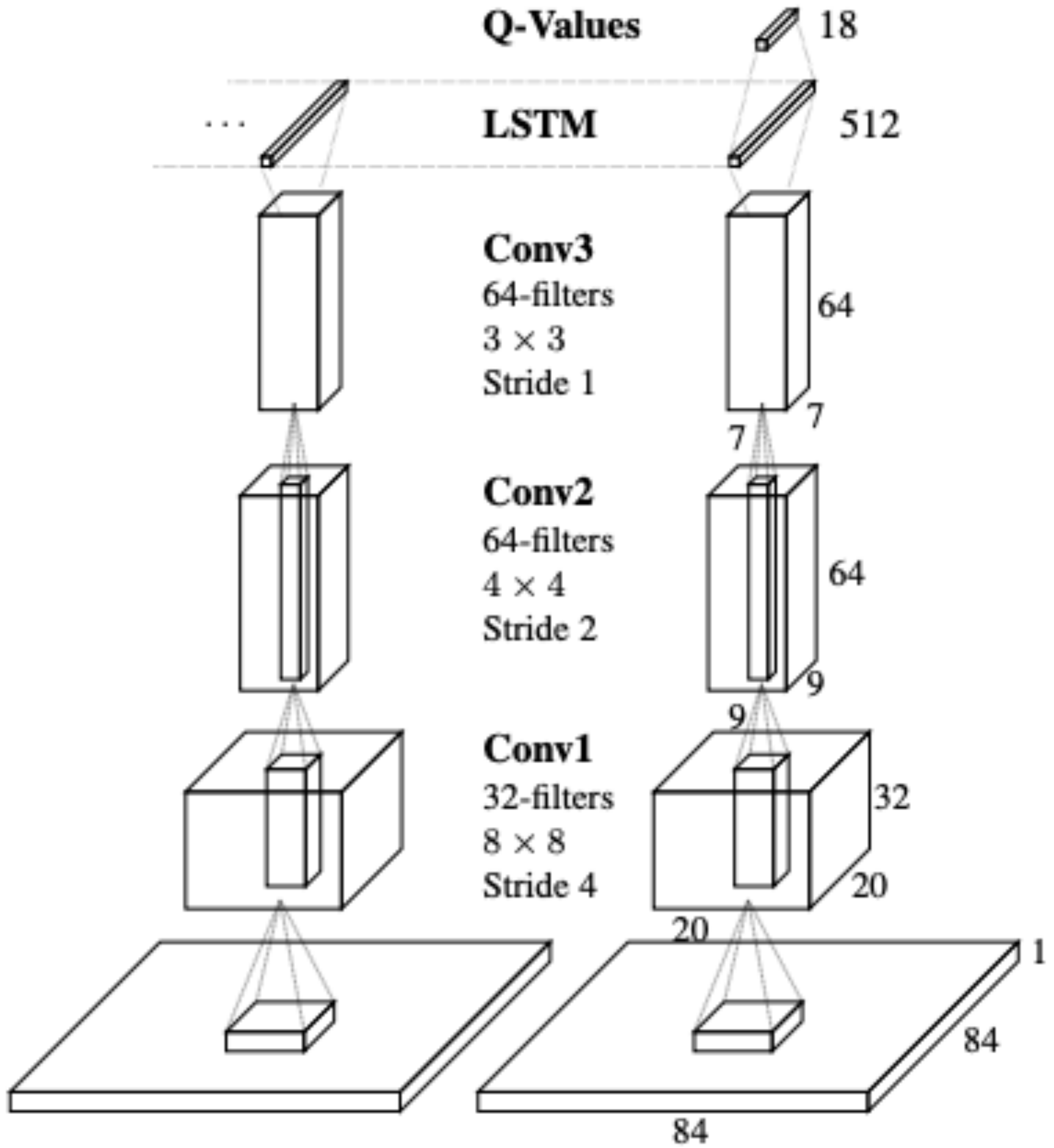


(vdumoulin@ Github)



Recurrent Neural Networks

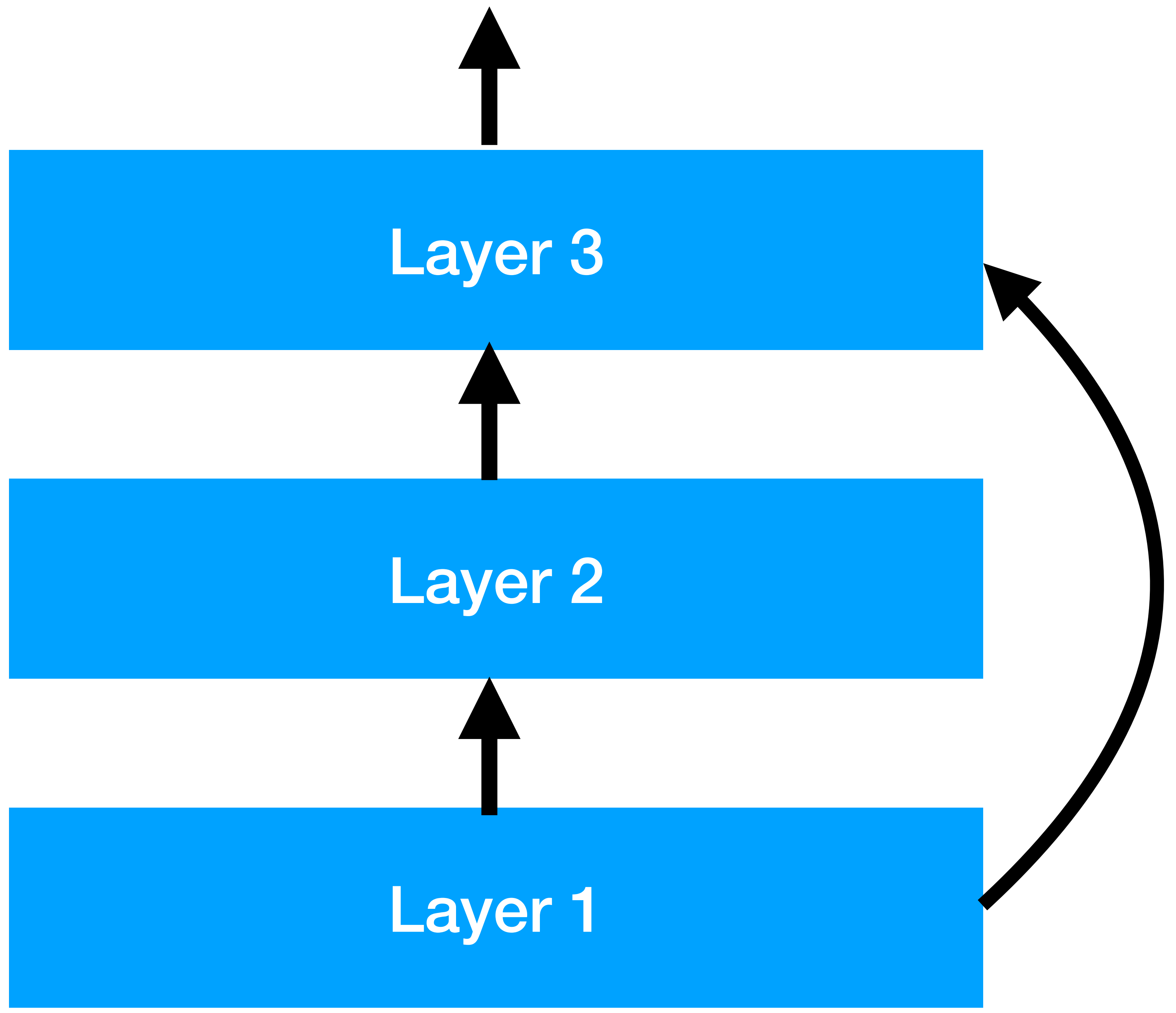
- Recurrent neural networks update a hidden state that is an input to computations at the next time-step.
- Allows networks to remember previously seen inputs when computing future outputs.
- In RL, provides a method to learn a Markov state.
- Alternative to frame-stacking.



Deep Recurrent Q-Learning for Partially Observable MDPs. Hausknecht and Stone. 2015.

Residual Connections

- Standard neural network:
    - $h_l = \sigma(W_{l-1}^\top(W_{l-2}^\top h_{l-2} + b_{l-2}) + b_{l-1})$
- Gradient of h_l with respect to h_{l-2} may be vanishingly small.
- Residual networks allow h_{l-2} to "skip" ahead to contribute to h_l:
    - $h_l = W_r^\top h_{l-2} + \sigma(W_{l-1}^\top(W_{l-2}^\top h_{l-2} + b_{l-2}) + b_{l-1})$



Yeping's Presentation

- Slides

Summary

- Neural networks are differentiable, non-linear function approximations that can be easily (in principle) used with semi-gradient reinforcement learning.

- Instead of learning a linear function of fixed, non-linear features, neural networks also learn the non-linear features.

- Removes the need for feature engineering though may require careful architecture and training considerations.

Action Items

- Literature review due **this** week.
- Begin reading chapter 13 (policy gradients)
- Homework 4 has been released.