**Artificial Intelligence in Games**

**Session 11**

1. Convolutional neural networks: overview
   1. Convolutional neural network (CNN):
      1. Parameterized function
      2. Parameters may be adapted to minimize a cost function using gradient descent
      3. Suitable for image tasks: explores the spatial relationships between pixels
      4. Three important types of layers: convolutional layers, max-pooling layers, and fully connected layers.
2. Convolutional neural networks: notation
   1. Image: a function f : Z^2 → R^c
      1. a in Z^2 is a pixel
      2. f(a) is the value of pixel a
      3. If f(a) = (f1(a), . . . , fc (a)), then fi is channel i
      4. Window W within Z^2 is a finite set W = [s1, S1] × [s2, S2] that corresponds to a rectangle in the image domain
      5. If the domain Z of an image f is a window, it is possible to flatten f into a vector x in R^{c|Z|}
   2. Consider an iid dataset D = (x1, y1), . . . ,(xN, yN), such that xi in R^D and yi in R. Each vector xi corresponds to a distinct image Z^2 → R^c , and all images are defined on the same window Z, such that D = c|Z|
3. Convolutional layer
   1. A neuron in a convolutional layer is not necessarily connected to the activations of all neurons in the previous layer, but only to the activations in a particular w \* h window W
   2. A neuron in a convolutional layer is replicated through parameter sharing for all windows of size w \* h in the domain Z whose centers are offset by pre-defined steps (strides).
   3. Receives an input image f and outputs an image o
   4. Each artificial neuron h in a convolutional layer l receives as input the values in a window W = [s1, S1] × [s2, S2] in Z of size w \* h, where Z is the domain of f. The weighted input z^(l)\_h of that neuron is given by
      1. z^(l)\_h = b^(l)\_h + triple sum of (w\_h,I,j,k^{I} \* a\_i,j,k^{I-1})  
           
         where a^(l−1)\_i,j,k = f\_i(j, k) is the value of pixel (j, k) in channel i of the input image f
   5. Activation function is typically rectified linear: a^(l)\_h = max(0, z^(l)\_h)
   6. An output image o : Z^2 → R^n is obtained by replicating n neurons over the whole domain of the input image
   7. The activations corresponding to a neuron replicated in this way correspond to the values in a single channel of the output image o (appropriately arranged in Z^2)
   8. The total number of free parameters in a convolutional layer is only n(cwh + 1).
   9. If the parameters in a convolutional layer were not shared by replicated neurons, the number of parameters would be mn(cwh + 1), where m is the number of windows of size w \* h that fit into f (for the given strides)
   10. A convolutional layer is fully specified by the size of the filters (window size), the number of filters (number of channels in the output image), horizontal and vertical strides (which are usually 1)
4. Max-pooling layer:
   1. Goal: achieving similar results to using comparatively larger convolutional filters in the next layers with less parameters
   2. Receives an input image f : Z^2 → R^c and outputs an image o : Z^2 → R^c
   3. Reduces the size of the window domain Z of f by an operation that acts independently on each channel   
        
      oi(j, k) = max of (a in W\_j,k fi(a))   
        
      where i in {1, . . . , c}, (j, k) in Z^2 , Z is the window domain of f, and W\_j,k within Z is the input window corresponding to output pixel (j, k).
   4. A max-pooling layer is fully specified by the size of a pooling window and vertical/horizontal strides
5. Fully connected layer
   1. Receives a vector (or flattened image) and outputs a vector
   2. Analogous to a layer in a feedforward neural network
   3. Typically only followed by other fully connected layers
   4. In a regression task, the output layer is typically fully connected with one neuron
6. Deep Q-Networks
   1. Q : S \* A \* R^m → R is represented by a neural network
7. Deep Q-Networks: preprocessing
   1. A sequence of images obtained from the emulator is preprocessed before being presented to the network
   2. Individually for each color channel, an elementwise maximum operation is employed between two consecutive images to reduce rendering artifacts
   3. Such 210 \* 160 \* 3 preprocessed image is converted to grayscale, cropped, and rescaled into an 84 × 84 image xk
   4. A sequence of images xk−12, xk−8, xk−4, xk obtained in this way is stacked into an 84 \* 84 \* 4 image s.
8. Deep Q-Networks: architecture
   1. The image st is input to a neural network architecture given by:
      1. Convolutional layer with 32 rectified linear filters (8 \* 8, stride 4)
      2. Convolutional layer with 64 rectified linear filters (4 \* 4, stride 2)
      3. Convolutional layer with 64 rectified linear filters (3 \* 3, stride 1)
      4. Fully-connected layer with 512 rectified linear units
      5. Fully-connected layer with |A| linear units
   2. Each output unit represents Q(st , a; theta) for a different action a in A
9. Deep Q-networks: algorithm
   1. Algorithm 1 Deep Q-learning with experience replay
   2. Input: replay buffer size M, number of episodes N, maximum episode length T, probability of random action epsilon, frame skip K, batch size B, learning rate alpha, number of episodes between target network updates C.
   3. Output: estimate Q(·; theta) of the optimal action value function Q^\*
   4. Initialize replay buffer D, which stores at most M tuples
   5. Initialize network parameters theta randomly
   6. Theta’ ← theta
   7. for each i in {1, . . . , N} do
      1. s0 ← initial state for episode i
      2. for each t in {0, . . . , T − 1} do
         1. if random() < 1 − epsilon then at ← maximise argument of (Q(st, a; theta)) else at ← random action
         2. Obtain the next state st+1 and reward rt+1 by repeating action at during K frames
         3. if the episode ends at step t + 1 then omega\_t+1 ← 1 else omega\_t+1 ← 0
            1. Store the tuple (st, at, rt+1, st+1, omega\_t+1) in the replay buffer D
            2. Sample a subset D’ in D composed of B tuples
            3. Let L(theta) = P (s,a,r,s 0,Ω0)∈D0 (y − Q(s, a; theta))^2
            4. In the equation above, let y = r + gamma max of(Q(s’ , a’ ; theta’) if theta’ = 0, and y = r if theta’ = 1
            5. theta ← theta – alpha\*Nabla\_theta\*L(theta), noting that theta’ is considered a constant with respect to theta
      3. end for
      4. if i mod C = 0 then
         1. theta’ ← theta
      5. end if
   8. end for
10. Policy gradient methods :
    1. Consider an agent that interacts with its environment in a sequence of episodes, each of which lasts for exactly T time steps
    2. Let tau = s0, a0, r1, s1, a1, r2, . . . , sT−1, aT−1,rT ,sT denote a trajectory in a particular episode
    3. Under the Markov assumption, the probability p(tau given theta) of trajectory tau given the policy parameters theta is given by:  
         
       o(tau given theta) = p(s0) \* product sum of ((s\_t+1, r\_t+1 given st, at) \* (p(at given st, theta))  
         
       where p(at given st , theta) is the probability of action at given state st and policy parameters theta.
    4. The expected return J(theta) of a policy parameterized by theta is given by:  
         
       J(theta) = Expectation of [sum of(Rt given theta)] = sum of (Expectation of (Rt given theta))
    5. Goal: finding a parameter vector theta^\* such that J(theta^\*) = max of ( J(theta))
    6. The gradient of the expected return is a sum of expected values of random vectors that correspond to each time step
    7. In gradient ascent, the expected value for time step t weights a direction that locally increases the probability of each possible decision by its expected (positive or negative) outcome.
    8. Positive expected outcomes contribute towards making the probability of a decision higher.
    9. Negative expected outcomes contribute towards making the probability of a decision lower.
    10. Consider a sequence tau1, . . . , tauN of N trajectories obtained by following the policy parameterized by theta, and let  
          
        taui = si,0, ai,0,ri,1,si,1, ai,1,ri,2, . . . ,si,T−1, ai,T−1,ri,T ,si,T
    11. A Monte Carlo estimate hat(g(theta)) to Nabla\_theta\*J(theta) is given by  
          
        hat(g(theta)) = (1/N) \* double sum of (Nabla\_theta \* log (p(a\_i,t given s\_i,t, theta))) \* sum of (r\_i,t’)  
        = Nabla\_theta \* [(1/N) \* double sum of (log (p(a\_i,t given s\_i,t, theta))) \* sum of (r\_i,t’)]

and may be used for gradient ascent on J

* 1. Theorem:
     1. The gradient Nabla\_theta\*J(theta) of the expected return J(theta) is given by:  
          
        Nabla\_theta\*J(theta) = Expectations (sum of( Nabla\_theta\*log(p(Atgiven St, theta)) ) \* sum of( Rt’ given theta))
     2. Using the law of unconscious statistician:  
          
        J(theta) = sum of (tau given theta) \* sum of (rt) = double sum of (rt \* p(tau given theta))
     3. Assuming J(theta) is differentiable with respect to theta, the partial derivative J(theta) of J with respect to thetaj at theta is given by:  
        del J(theta) / del theta\_j = double sum of (rt\* del p(tau given theta) / del theta\_j)
     4. Suppose that p(tau given theta) is positive for any tau and theta. The so-called likelihood ratio trick uses the fact that:  
          
        del p(tau given theta) / del theta\_j = p(tau given theta) \* (1/p(tau given theta) \* del p(tau given theta) / del theta\_j  
          
        = p(tau given theta) \* (del p(log p(tau given theta)) / del theta\_j)
     5. By using the previous expression:  
          
        del J(theta) / del theta\_j = double sum of (p(tau given theta)) \* rt \* [sum of ((del p(log p(at’ given st’, theta)) / del theta\_j))]
     6. It will be useful to split the innermost summation in the expression above into before and after t, leading to:  
          
        del J(theta) / del theta\_j = double sum of (p(tau given theta)) \* [sum of (sum of (del p(log p(at’ given st’, theta)) / del theta\_j) + rt \* sum of (del p(log p(at’ given st’, theta)) / del theta\_j)]
     7. Alternatively, the expression above can be written as:  
          
        del J(theta) / del theta\_j = double sum of (Expectation of [Rt \* del p(log p(At’ given St’, theta) given theta / del theta\_j) + double sum of Expectation of ( [Rt \* del p(log p(At’ given St’, theta) given theta] / del theta\_j))]
     8. We will now show that the rightmost nested summations in the expression above can be dismissed.
     9. By representing the random variables involved in a trajectory using a Bayesian network, it can be seen that At’ congruent to Rt given St’, theta for t’ >= t. The analogous statement is not generally true for t’ < t.   
          
        For t’ >= t, this independence leads to  
          
        Expectation ([Rt \* del p(log p(At’ given St’, theta) given theta] / del theta\_j))] = triple sum of (p(at’ given st’, theta) \* p(rt, st’ given theta) \* rt \* (del p(log p(at’ given st’, theta)) / del theta\_j)
     10. By reversing the likelihood-ratio trick,  
           
         Expectation ([Rt \* del p(log p(At’ given St’, theta) given theta] / del theta\_j))] = triple sum of (p(at’ given st’, theta) \* rt \* (del p(log p(at’ given st’, theta)) / del theta\_j)
     11. By changing the order of summations and pushing constants outside the innermost summation,  
           
         Expectation ([Rt \* del p(log p(At’ given St’, theta) given theta] / del theta\_j))] = double sum of (p(rt, st’ given theta) \* rt \* sum of(del p(log p(at’ given st’, theta)) / del theta\_j)
     12. Finally, using the fact that the derivative of a constant = 0,  
           
         Expectation ([Rt \* del p(log p(At’ given St’, theta) given theta] / del theta\_j))] = double sum of (p(rt, st’ given theta) \* rt \* sum of(del p(log p(at’ given st’, theta)) / del theta\_j) = 0
     13. We may now remove the rightmost nested summations in the previous expression for,  
           
         del J(theta) / del theta\_j = expectation of (sum of(Rt) \* sum of(del p(log p(At’ given St’, theta)) / del theta\_j)) given theta)
     14. By reordering the summations, the expression above can be conveniently rewritten as,  
           
         del J(theta) / del theta\_j = expectation of (sum of(del p(log p(At’ given St’, theta)) / del theta\_j)) \* sum of(Rt) given theta)

1. Deep learning
   1. Artificial neural networks
      1. Initially inspired by the brain
      2. Mostly studied for their applications
   2. Any artificial neural network with more than one hidden layer is considered deep
2. Recurrent neural networks: overview
   1. Recurrent neural network (RNN):
      1. Parameterized function
      2. Parameters may be adapted to minimize a cost function using gradient descent
      3. Suitable for receiving a sequence of vectors and producing a sequence of vectors
   2. A recurrent neural network summarizes a sequence of vectors into an activation vector
   3. This summary is combined with the input for the current timestep to produce the output and the summary for the next timestep
   4. Parameters are shared across time
3. Long short-term memory networks: overview
   1. Long short-term memory network (LSTM):
      1. Parameterized function
      2. Parameters may be adapted to minimize a cost function using gradient descent
      3. Suitable for receiving a sequence of vectors and producing a sequence of vectors
      4. Mitigates the vanishing gradients problem
      5. Better than simple recurrent neural networks at learning dependencies between input and target vectors that manifest after many time steps
4. Residual layers
   1. Idea: information should be able to flow across layers unaltered
   2. Traditional layer: a^(l) = f(W^(l) \* a^(l−1) + b^(l))
   3. Residual layer: a^(l) = a^(l−1) + f(W^(l) \* a^(l−1) + b^(l))
5. Sequence to sequence model
   1. Idea: using an encoding phase followed by a decoding phase to map between sequences of arbitrary lengths [Cho et al., 2014, Sutskever et al., 2014] Image from [Sutskever et al., 2014]
   2. The recurrent networks that perform encoding and decoding are not necessarily the same
6. Transformers
   1. Idea: using attention mechanisms that allow focusing on specific parts of inputs
7. Differentiable neural computer
   1. Idea: a neural network can learn to read and write from a memory matrix using gating mechanisms
8. PixelRNN
   1. Idea: using a recurrent neural network trained to predict each pixel given the previous pixels as a probabilistic model  
        
      p(x given theta) = product sum of (p(xj given xi,…,x\_j-1, theta))
9. Generative adversarial network
   1. Idea: training a (discriminator) network to discriminate between real and synthetic observations and training another (generator) network to generate synthetic observations from noise that fool the discriminator
10. Variational autoencoder
    1. Idea: training a model with (easy to sample) hidden variables by maximizing a particular lower bound on the log-likelihood  
         
       integral of (p(x given z, theta) \* (p(z given theta))) dz = integral of (N(x given f(z, theta), sigma^2l) \* N(z given 0, 1) dz)
11. MuZero
    1. Idea: combining tree-based search with a learned implicit environment model
    2. Achieves excellent results on ATARI games, Chess, and Go
12. Representation learning
    1. Neural networks made significant progress in learning representations of high-dimensional states
    2. However, credit assignment remains daunting in partially observable environments
    3. Promising: development of inductive biases for specific stimulus modalities that enable long-term information storage and retrieval
13. Efficient exploration
    1. The trade-off between exploration and exploitation is one of the earliest challenges recognized in reinforcement learning
    2. However, scalable exploration methods are often unsound
    3. Promising: scaling up posterior sampling to complex environments
14. Efficient and reliable planning
    1. Planning is crucial for sample efficient reinforcement learning
    2. However, planning across a large number of time steps can be extremely expensive
    3. Promising: planning in a latent space that abstracts irrelevant aspects of the environment
    4. Furthermore, compounding model errors make long-term planning unreliable
    5. Promising: representing and considering uncertainty when planning