Transfer learning in Reinforcement learning tasks - learning the task correspondence

SDZ

Marko Ruman

supervisor: Ing. Tatiana Valentine Guy, Ph.D.

Outline

Definition of fully connected NN

A fully connected neural network is a tuple

$$(\mathbf{G}, \mathbf{\Phi}, \Theta_0, \mathcal{L})$$

- $\mathbf{G}$ is a directed graph defining the architecture of the network, $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, where
  - $\mathbf{V} = \left\{ \left\{ n_i^k \right\}_{i=1}^{r_k}, k \in \{1, 2, ..., m\} \right\}$ is a set of neurons, the set $\left\{ n_i^k \right\}_{i=1}^{r_k}$ is *k-th layer*,
  - m is the number of layers,
  - r_k is the number of neurons in the k-th layer,
  - $\mathbf{E}$ is the set of edges, $\mathbf{E} = \left\{ \left(n_i^k, n_j^{k+1} \right), i \in \{1, 2, ..., r_k\}, j \in \{1, 2, ..., r_{k+1}\}, k \in \{1, 2, ..., m\} \right\}$ and
    - to each edge of the network $\left(n_i^k, n_j^{k+1} \right)$, there is a corresponding weight $\theta_{ij}^k \in \mathbb{R}$
    - the weights can be organized into matrices Θ^k, where θ_{ij}^k is its (i,j)-th entry, $\Theta = \left\{ \Theta^k \right\}_{k=1}^m$

Definition of fully connected NN

A fully connected neural network is a tuple

$$(\mathbf{G}, \mathbf{\Phi}, \Theta_0, \mathcal{L})$$

- $\mathbf{\Phi}$ is a set of activation functions, $\mathbf{\Phi} = \left\{\phi^k\right\}_{k=1}^m$, where $\phi^k : \mathbb{R} \mapsto \mathbb{R}$ is activation function at layer k
- Θ_0 is an initial value of weights, $\Theta_0 = \left\{\Theta_0^k\right\}_{k=1}^m$

Definition of fully connected NN

A fully connected neural network is a tuple

$$(\mathbf{G}, \mathbf{\Phi}, \Theta_0, \mathcal{L})$$

- $\mathcal{L} = \mathcal{L}(\mathbf{X}, \Theta)$ is the loss function, that has the following form

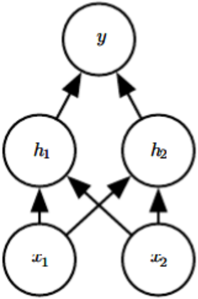
$$\mathcal{L}(\mathbf{X}, \Theta) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_p\left(x_i, y_i, \Theta\right) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_i, \tag{1}$$

where:
  - $\mathcal{L}_p$ is chosen usually as squared L2 norm or L1 norm,
  - $\mathbf{X} = \{(x_i, y_i)\}_{i=1}^{N}$ is *dataset*, i.e. set of input-output pairs of the size $N \in \mathbb{N}$,

The functional form that NN constructs

$$f(x|\Theta) = \phi^m\left(\Theta^m \phi^{m-1}\left(\Theta^{m-1} \ldots \phi^1\left(\Theta^1 x\right)\ldots\right)\right)$$

NN example



- $m = 2, r_1 = 2, r_2 = 1$
- $\phi^1(x) = ReLu(x) = \max(0, x)$
- $\phi^2(x) = x$

$$f(x|\Theta) = \Theta_2 \mathrm{max}(0, \Theta_1 x)$$

Backpropagation algorithm

Gradient descent

$$\Theta_{t+1} = \Theta_t - \alpha \frac{\partial \mathcal{L}(\mathbf{X},\Theta_t)}{\partial \Theta}, \qquad \text{i.e.} \qquad \theta^k_{t+1;ij} = \theta^k_{t;ij} - \alpha \frac{1}{N} \sum_{l=1}^{N} \frac{\partial \mathcal{L}_l}{\partial \theta^k_{ij}}$$

Backpropagation formula

$$\frac{\partial \mathcal{L}_i}{\partial \theta^k_{ij}} = \delta^k_i o^{k-1}_j = \frac{d}{de^k_i} \phi^k(e^k_i) o^{k-1}_j \sum_{l=1}^{r_k} \delta^{k+1}_l \theta^k_{li}$$

- $e^k = \Theta^{k-1} o^{k-1}$ *activation* of k-th layer
- $o^k = \phi^k(e^k)$ *output* of k-th layer
- $\delta^k_i = \frac{\partial \mathcal{L}_i}{\partial e^k_i}$

Backpropagation algorithm

Backpropagation algorithm

1. Calculate the forward phase for each input-output pair (x_i, y_i) from a mini-batch and store the results $f(x_i)$ (**??**), e_j^k and o_j^k by proceeding from layer $k = 0$, the input layer, to layer $k = m$, the output layer.
2. Calculate the backward phase for each input-output pair (x_i, y_i) from a mini-batch and store the results $\frac{\partial \mathcal{L}_i}{\partial \theta_{ij}^k}$.
    - Evaluate the loss term for the final layer $\delta_j^m = \frac{\partial \mathcal{L}_i}{\partial e_j^m}$ for each node.
    - Backpropagate the loss terms for the other layers δ_j^k, working backwards from the layer $k = m - 1$.
    - Evaluate the partial derivatives of the individual loss $\frac{\partial \mathcal{L}_i}{\partial \theta_{ij}^k}$.
3. Update the weights by gradient descent equation.

Convolutional NN

$$[\Theta * x]_{i_1,i_2,\ldots,i_n} = \sum_{j_1} \sum_{j_2} \ldots \sum_{j_n} x_{i_1-j_1,i_2-j_2,\ldots,i_n-j_n} \Theta_{j_1,j_2,\ldots,j_n}$$

Convolution layer

A 2-D convolutional layer *input channels* C_{in}, *output channels* C_{out} and *kernel size* k contains:

- C_{out} parameter matrices $\Theta_1, \ldots \Theta_{C_{out}}$. Each $h_i \in \mathbb{R}^{k,k,C_{in}}$.
- activation function ϕ.

It takes $x \in \mathbb{R}^{C_{in},n,n}$ as an input and gives the following output o:

- $e = [\Theta_1 * x, \Theta_2 * x, \ldots, \Theta_{C_{out}} * x] \in \mathbb{R}^{C_{out},n_{out},n_{out}}$ is the *activation*,
- $o = \phi(e)$ is *output* of the layer.