Lecture 10: Neural Nets and SGD

Course: Reinforcement Learning Theory
Instructor: Lei Ying
Department of EECS
University of Michigan, Ann Arbor

Nonlinear function approximation using neural networks

- Define

$$G(\theta) = \frac{1}{2}(\hat{Q}(x_k, u_k; \theta) - r_k - \alpha \max_v \hat{Q}(x_{k+1}, v; \theta))^2$$

Then

$$\frac{\partial G(\theta)}{\partial \theta} \approx (\hat{Q}(x_k, u_k, \theta) - r_k - \alpha \max_v \hat{Q}(x_{k+1}, v; \theta))\nabla \hat{Q}(x_k, u_k, \theta)$$

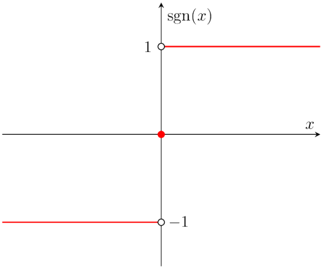
- r_k: from samples
- $\max_v \hat{Q}(x_{k+1}, v; \theta_k), \hat{Q}(x_k, u_k; \theta_k)$: from neural networks
- Question: how to compute $\nabla_\theta \hat{Q}(x, u; \theta)$?

Preliminaries:

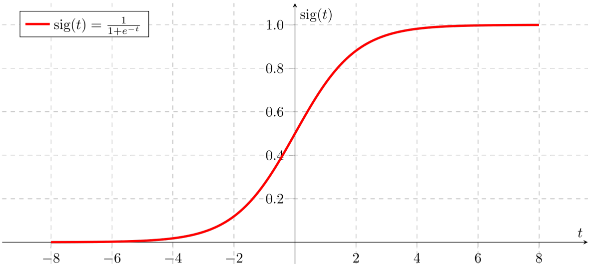
- A neuron is a nonlinear function which takes $x \in \mathbb{R}$ as input and produce $\sigma(x) \in \mathbb{R}$.

$$x \to \sigma(x)$$

Examples:

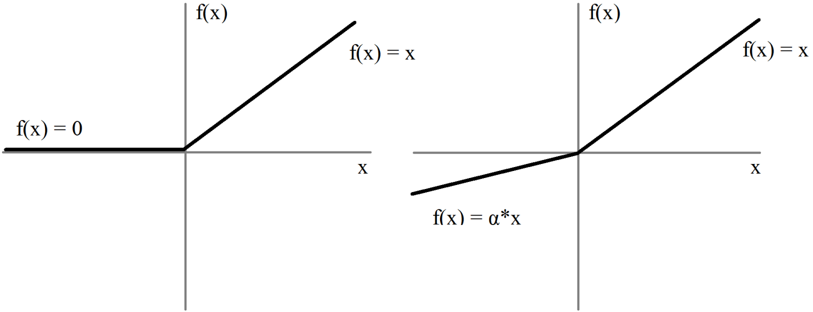


Sign function



Sigmoid function

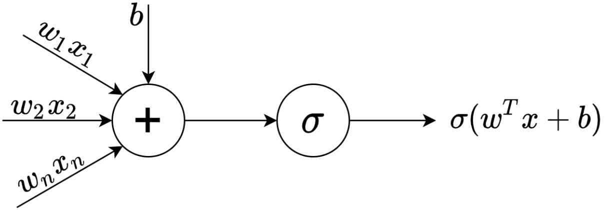
Nonlinear function approximation using neural networks



ReLU (Rectifier Linear Unit) and Leaky ReLU

Nonlinear function approximation using neural networks

- Vector input: $\sigma(w^T x + b)$



  where b is the bias.

- $w^T x + b = 0$ is a hyperplane, which divides the input space into two parts.

Nonlinear function approximation using neural networks

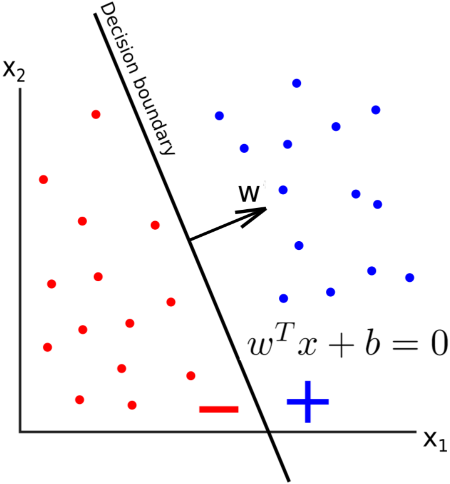
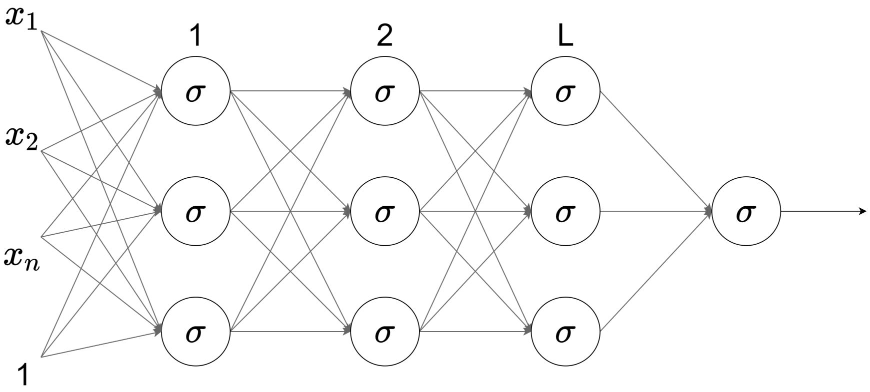


Illustration in 2D

Nonlinear function approximation using neural networks

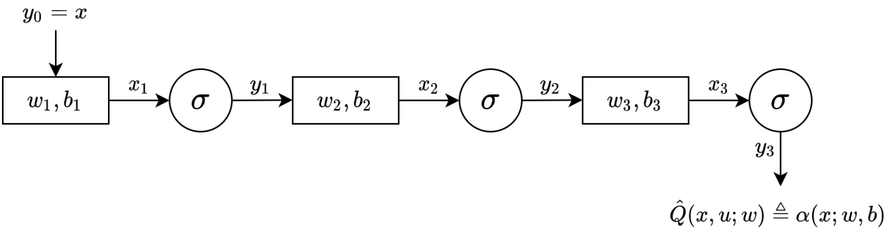
Multilayer NN's



- The number of neurons at each layer can be different.

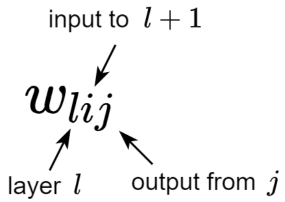
Nonlinear function approximation using neural networks

- Look at a single path:

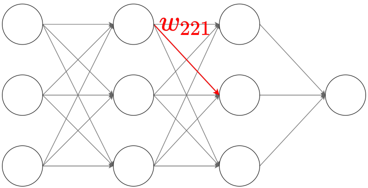


Nonlinear function approximation using neural networks

Goal: compute $\frac{\partial L(x;w,b)}{\partial w_{lij}}$ and $\frac{\partial L(x;w,b)}{\partial b_{li}}$, where

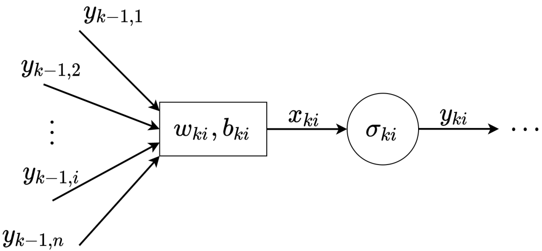


For example:



Nonlinear function approximation using neural networks

$$
\begin{aligned}
\frac{\partial L}{\partial w_{kij}} &= \frac{\partial L}{\partial y_{ki}} \frac{\partial y_{ki}}{\partial w_{kij}} \\
&= \frac{\partial L}{\partial y_{ki}} \underbrace{\frac{\partial y_{ki}}{\color{red}{\partial x_{ki}}}}_{\sigma'(x_{ki})} \underbrace{\frac{\color{red}{\partial x_{ki}}}{\partial w_{kij}}}_{y_{k-1,j}}
\end{aligned}
$$



Nonlinear function approximation using neural networks

$$\frac{\partial L}{\partial b_{ki}} = \frac{\partial L}{\partial y_{ki}} \frac{\partial y_{ki}}{\partial x_{ki}} \underbrace{\frac{\partial x_{ki}}{\partial b_{ki}}}_{=1} = \frac{\partial L}{\partial y_{ki}} \sigma'(x_{ki})$$

$$\begin{aligned}
\frac{\partial L}{\partial y_{li}} &= \sum_k \frac{\partial L}{\partial x_{l+1,k}} \frac{\partial x_{l+1,k}}{\partial y_{li}} \\
&= \sum_k \frac{\partial L}{\partial y_{l+1,k}} \frac{\partial y_{l+1,k}}{\partial x_{l+1,k}} \frac{\partial x_{l+1,k}}{\partial y_{li}} \\
&= \sum_k \underbrace{\frac{\partial L}{\partial y_{l+1,k}}}_{\text{from layer } l+1} \sigma'(x_{l+1,k}) w_{l+1,k,i}
\end{aligned}$$

Similarly,

$$\frac{\partial L}{\partial b_{li}} = \frac{\partial L}{\partial y_{li}} \frac{\partial y_{li}}{\partial x_{li}} \frac{\partial x_{li}}{\partial b_{li}} = \frac{\partial L}{\partial y_{li}} \sigma'(x_{li})$$

Nonlinear function approximation using neural networks

Summary:

- Forward propagation:
  - Given data $\tilde{x}, y_0 = \tilde{x}$, use NN to compute $y_{li} \quad \forall l, i$
- Backward propagation:
  1. Compute $\frac{\partial L}{\partial y_{ki}}$ (assume k is the output layer)
  2. For $1 \leq l \leq k$,

$$\frac{\partial L}{\partial w_{lij}} = \frac{\partial L}{\partial y_{li}} \sigma'(x_{li}) y_{l-1,j}$$

$$\frac{\partial L}{\partial b_{li}} = \frac{\partial L}{\partial y_{li}} \sigma'(x_{li})$$

$$\frac{\partial L}{\partial y_{l-1,i}} = \sum_k \frac{\partial L}{\partial y_{lk}} \sigma'(x_{lk}) w_{lki}$$

Nonlinear function approximation using neural networks

PyTorch implementation

- Sample a mini-batch. For example, (state, action, target-Q-value)=($\tilde{x}_m$, $\tilde{u}_m$, $\tilde{Q}_m$)
- Define a loss function

$$L(w) = \sum_m \left(\hat{Q}(\tilde{x}_m, \tilde{u}_m; w) - \tilde{Q}_m\right)^2 .$$

- Update $w \leftarrow w - \beta \nabla_w L(w)$ by calling
  opt.zero grad()
  opt.backward()
  opt.step()

Reference

- This lecture is based on R. Srikant's lecture notes on *Back-Propagation Algorithm* available at https://sites.google.com/illinois.edu/mdps-and-rl/lectures?authuser=1

Acknowledgements: I would like to thank Alex Zhao for helping prepare the slides, and Honghao Wei and Zixian Yang for correcting typos/mistakes.