Discussions on Deep Neural Networks and Reinforcement Learning

May 25, 2020

Outline of Today's Discussion

- Deep neural networks (DNNs) √
- Convolutional neural networks (CNNs) √
- Recurrent neural networks (RNNs) √
- Reinforcement Learning (RL)

Artificial Neural Network

Input layer

Neuron $\Sigma \mid \sigma$

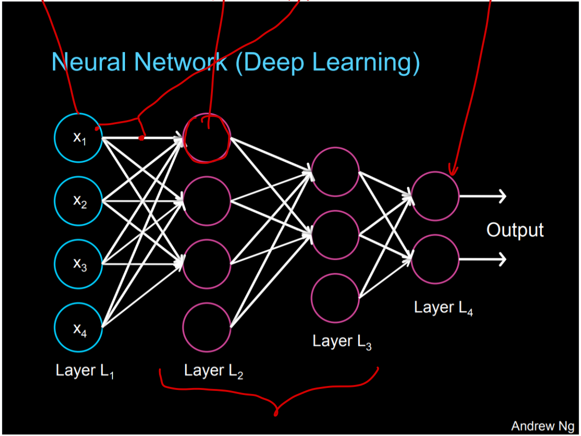
Function!

Output layer

Sigmoid

ReLu

$$\tanh x = \frac{e^{2x}-1}{e^{2x}+1}.$$



Hidden Layer

- Network Layers
- Activation function
- Weights
- Cost function Metric
- Learning algorithm

A live demo

SGD

mini-GD

Artifical Neural Network

· ANN: learns very useful non-linear representation,

Artificial Neural Networks to learn f: X → Y

- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

- Represent f by *network* of logistic units
- Each unit is a logistic function

$$unit\ output = \frac{1}{1 + exp(w_0 + \sum_i w_i x_i)}$$

- MLE: train weights of all units to (minimize sum of squared errors of predicted network outputs)
- MAP: train to minimize sum of squared errors plus weight magnitudes $P(w)$

$\|W\|_2 \leftarrow$

$\|W\|_1 \leftarrow$

· regularization

control the model complexity

Fully Connected Layers (Multi-Layer Perceptron)

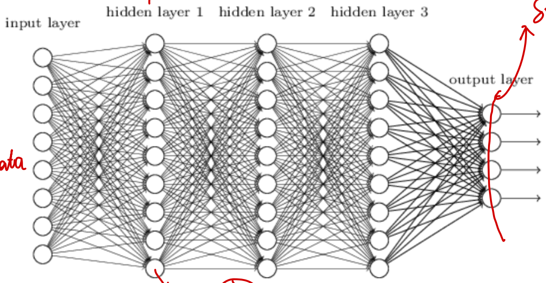
FC networks are a stepping stone on the path to other state-of-the-art network models.

Matrix- Vector multiplication

CNN

- Feature extraction

: Capture better representation of data



Softmax

- FC classifiction.

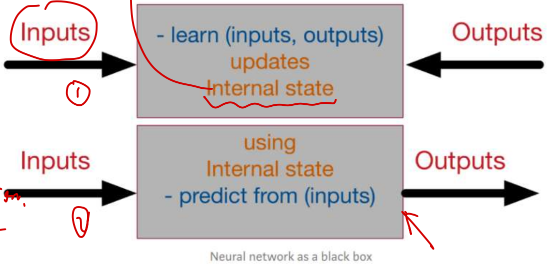
$\Sigma | \sigma$

- **Feedforward**. Information flows through the function being evaluated from $\boldsymbol{x}$, through the intermediate computations used to define f, and finally to the output $\boldsymbol{y}$.

Learning and Prediction

Input → Update [Internal State] "W" → { The Calculate output
Desired output ≶ The desired output

{W ; b}



1
2

Training ✓

Inference.

Speech Recognization.
↓
mobile device.

Figure: A black box with 2 methods.

Gradient-based Learning

$\langle W^{0^T}, X \rangle \to \boxed{} \to$ Output

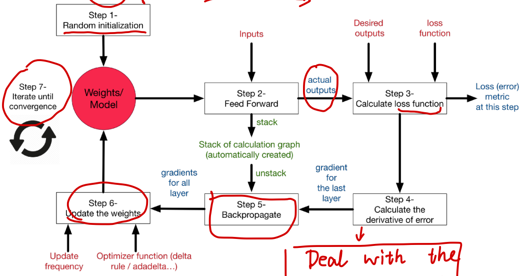
$W^0 \to W'$

$w_{ij}^0 \to w_{ij}^1$

- Model initialization
- Forward propagate ✓
- Loss function
- Differentiation ✓
- **Back-propagation**
- Weight update

Actual output $\min \mathcal{L} = \frac{1}{2} \| O_d - t_d \|_2^2$

Desired output



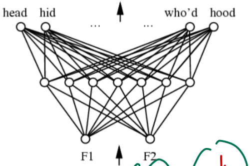
Deal with the derivative of $\mathcal{L}$.

Figure: Overall picture

 : Repeatedly ~ Weights adjustment.

Chain Rule: Gradient-Based learning strategy.

Backpropagation Algorithm (MLE)



Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_i$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial o_d}\,\frac{\partial o_d}{\partial z_d}\,\frac{\partial z_d}{\partial w_i}$$

x_d = input

t_d = target output

o_d = observed unit output

w_{ij} = wt from i to j

CNN targets the processing of 2-D features (e.g., images) instead of the 1-D ones in pure FC models.

A set of filter:

- **Convolutional Layer.** — Core building block.
- **Pooling Layer.** — in-between successive Conv layer. Reduce the spatial size of the representation, of Parameter and Computation.
- **Fully-Connected Layer.**

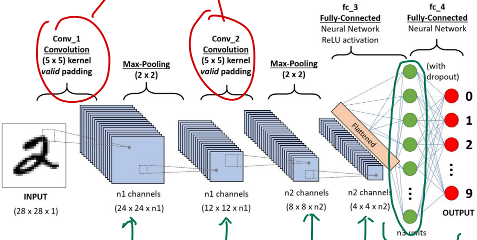


Figure: Source

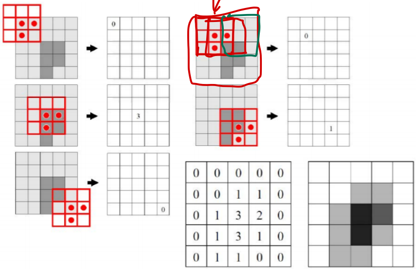
Convolutional Neural Networks

Sliding + element-wise multiplying

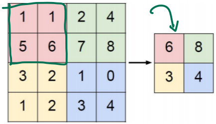
stride

kernel matrix.

Convolution



Max Pooling



average-pooling

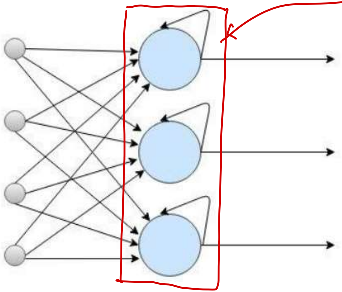
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Recurrent Neural Networks

Motivation: FC modelas a generic function approximator and convolutional neural networks for efficiently extracting local information from data. RNN is designed with intralayer recurrent connections, as shown below, which is different from the sole feedforward structure in FC and CNN.

Speech Recognization

NLP

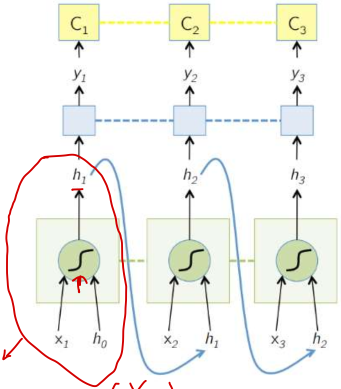


Use hidden layer in network to capture history.

Recurrent Neural Network

Figure: Source

The Vanilla RNN Forward



$$h_1 = \tanh W \begin{pmatrix} x_1 \\ h_0 \end{pmatrix}$$

$$\begin{cases} h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \\ y_t = \mathrm{F}(h_t) \\ C_t = \mathrm{Loss}(y_t, \mathrm{GT}_t) \end{cases}$$

------ indicates shared weights

Figure: Source

The Vanilla RNN Forward

$$\begin{cases} S_t = U^T x_t + W^T \phi(S_{t-1}) \\ O_t = \phi(V^T S_t) \end{cases}$$

t

- Vanishing gradient issue
- LSTM

Input Gate

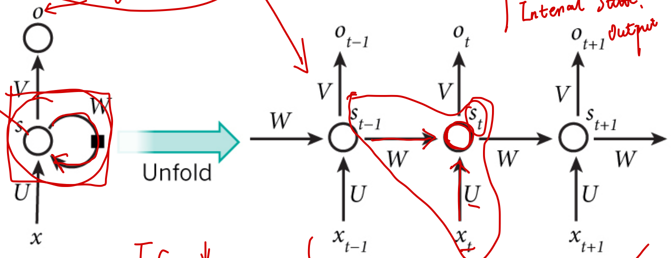
$\boxed{\text{Forget Gate}}$

Internal state.

Output gate.

σ: sigmoid.

hidden state at t



MNIST

CIFAR 10.

FC ↓

CNN

Conv

Max-pooling

Figure: Source

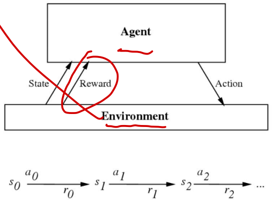
Acc ↑

$$f_t = \sigma\left(W_f \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b_f\right),$$

Code

Reinforcement Learning Problem

E: The object interacts with agent.



Goal: Learn to choose actions that maximize

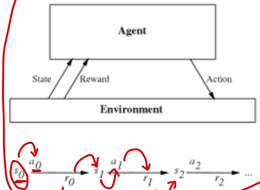
$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots, \text{ where } 0 \leq \gamma < 1.$$

Discount Factor

- **Agent** is an entity (learner or decision maker) that is equipped with
  - sensors, in order to sense the environment
  - end-effectors to act in the environment
  - goals that it wants to achieve
- **Action**
  - Used by the agent to interact with the environment
- A **Reward** R_t is a scalar feedback signal +
  - Indicates how well agent is doing at step t. −
  - The agent's job is to maximize cumulative reward.

Markov Decision Process = Reinforcement Learning Setting

· "The future is independent of the past given the present".



Finite.

- Set of states S $S=\{s_1,\cdots,s_n\}$
- Set of actions A $A=\{a_1 \cdots a_m\}$
- At each time, agent observes state $s_t \in S$, then chooses action $a_t \in A$
- Then receives reward r_t, and state changes to s_{t+1}
- Markov assumption: $P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t)$
- Also assume reward Markov: $P(r_t \mid s_t, a_t, s_{t-1}, a_{t-1},\ldots) = P(r_t \mid s_t, a_t)$ } MDP.

E.g., if tell robot to move forward one meter, maybe it ends up moving forward 1.5 meters by mistake, so where the robot is at time t+1 can be a probabilistic function of where it was at time t and the action taken, but shouldn't depend on how we got to that state.

- The task: learn a policy π: S → A for choosing actions that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \quad 0 < \gamma \leq 1$$

for every possible starting state s_0

Agent

Given a policy $\pi : S \rightarrow A$, define

a distribution over actions given state $\{$

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1}$$

- Assuming action sequence chosen according to π, starting at state s.
- Expected discounted reward we will get starting from state s if we $\boxed{\text{follow policy } \pi.}$

For any MDP

Goal: find the optimal policy π^* where

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \ \forall s. \tag{2}$$

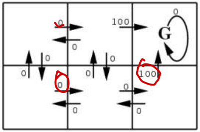
exists.

- $\boxed{V^*(s) \text{ and } P(s_{t+1} \mid s_t, a_t)} \rightarrow \pi^*(s)$

- Policy whose value function is the maximum out of all policies simultaneously for all states.

Value Function

$$V^{\pi^*}(s) = \sum_{t=0}^{\infty} \gamma^t r_t = (0.9)^0 \cdot 0 + (0.9)^1 \cdot 0 + (0.9)^2 \cdot 100 = 81$$

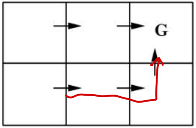


$r(s,a)$ (immediate reward) values

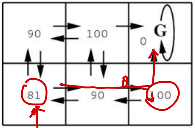
Immediate rewards r(s,a)

State values V*(s)

$\gamma = 0.9$



One optimal policy



$V^*(s)$ values

Recursive Definition for $V^*(S)$

$$r(s, \pi^*(s)) \qquad \Rightarrow \sum_{s' \in S} P(s' | s, \pi^*(s)) \; V^*(s')$$

$$V^*(s) = \overbrace{E\left[r(s, \pi^*(s))\right]}^{r(s,\pi^*(s))} + \underbrace{\gamma E_{s'|s,\pi^*(s)}\left[V^*(s')\right]}_{} .$$

- Optimal value of any state s is the expected reward of performing $\pi^*(s)$ from s plus γ times the expected value, over states s' reached via performing that action from state s, of the optimal value of s'.

Value Iteration for Learning $V^*(\cdot)$ assumes $P(S_{t+1} \mid S_t, A)$ known

Value Iteration for learning V* : assumes $P(S_{t+1}|S_t, A)$ known

$t=0$

Initialize V(s) to 0 [optimal value can get in zero steps]

For t=1, 2, … [Loop until policy good enough]

Loop for s in S : Traverse all states.

Loop for a in A S_1

- $Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$

$\boxed{V(s) \leftarrow \max_a Q(s,a)}$

End loop

End loop

- Round t=0 we have V(s)=0 for all s.
- After round t=1, a top-row of 0, 100, 0 and a bottom-row of 0, 0, 100.
- After the next round (t=2), a top row of 90, 100, 0 and a bottom row of 0, 90, 100.
- After the next round (t=3) we will have a top-row of 90, 100, 0 and a bottom row of 81, 90, 100, and it will then stay there forever

ML

$r(s_1, a_1)$ 100

$+ \gamma \sum_{s \in S} P(s'|s,a) V(s')$

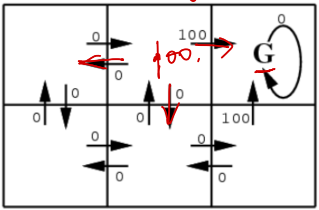
0

$Q(s, a_1)$

$Q(s_1, a_2)$

(s_1, a_3)

e.g.:



Q Learning

Define new function, closely related to V*

$$V^*(s) = E[r(s, \pi^*(s))] + \gamma E_{s'|\pi^*(s)}[V^*(s')]$$

V*(s) is the expected discounted reward of following the optimal policy from time 0 onward.

$$Q(s, a) = E[r(s, a)] + \gamma E_{s'|a}[V^*(s')]$$

Q(s,a) is the expected discounted reward of first doing action a and then following the optimal policy from the next step onward.

If agent knows Q(s,a), it can choose optimal action without knowing $P(s_{t+1}|s_t,a)$!

$$\pi^*(s) = \arg\max_a Q(s, a) \qquad V^*(s) = \max_a Q(s, a)$$

Just chose the action that maximizes the Q value

$$\pi^*(s) = \arg\max_\pi V^\pi(s).$$

And, it can learn Q without knowing $P(s_{t+1}|s_t,a)$

using something very much like the dynamic programming algorithm we used to compute V*.

ML

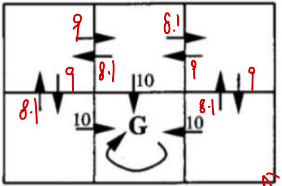
Q learning

(1). $\pi^*(s) = \arg\max_{\pi} V^{\pi}(s) \quad \forall s$.

$V^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r_t]$

$Q(s,a) = E[r(s,a)] + \gamma E_{s'|a}[V^*(s')]$

$V^*(s)$



$S_t \times A_t \to S_{t+1}$.

\gamma=0.9

Unknown.

$S_t \times A_t \to S_{t+1}$.

(1) Give an optimal policy for the above problem;
(2) Calculate the V*(s) values;
(3) Calculate the Q(s,a) values.