PYTORCH

A Quick Tour

Michael Guerzhoy
Li Ka Shing Centre for Healthcare Analytics Research and Training,
St. Michael's Hospital
Dept. of Computer Science, University of Toronto
guerzhoy@cs.toronto.edu

Outline

- What is PyTorch?
- Evaluating derivatives with PyTorch
- Maximum likelihood using Gradient Descent
- Feedforward Neural Networks for Face Recognition
- REINFORCE with PyTorch

- Assumptions
  - You haven't used PyTorch before
  - You know basic machine learning
  - You want to know more about writing ML algorithms from scratch and understanding PyTorch/TF code better
- https://github.com/guerzh/pytorch_tutorial
  - git clone https://github.com/guerzh/pytorch_tutorial

What is PyTorch

- Python library
- Tensor objects
  - Tensors are similar to NumPy n-d array
  - Matrix multiplication is sped up with GPUs
- Variable objects
  - Variables wrap Tensor objects
  - Variables can be operated on similarly to Tensors
  - We can *automatically differentiate* Variables
- A torch.nn module enable quick and concise definition of neural network architectures
- Nice built-in support for vision applications (e.g., pretrained VGG-16)

Evaluating derivatives with PyTorch

- See `The Basics of PyTorch.ipynb`

Maximum Likelihood

- A biased coin has a probability of θ of coming up Heads (1) and a probability of $(1-\theta)$ of coming up Tails (0)
- Likelihood: $P(data|\theta) = \prod_i \theta^{t_i}(1-\theta)^{(1-t_i)}$
  - t_i is 1 if the i-th toss comes up Heads and 0 otherwise
- Maximize the Likelihood (equivalently, the log-Likelihood) to find the θ for which the sample data is as plausible as possible
- See `Maximum Likleihood for Bernoulli with PyTorch.ipynb`

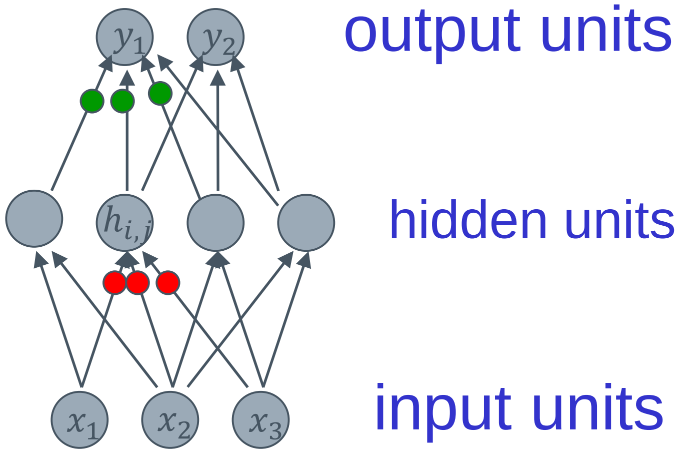
Feedforward Neural Networks

$$h_{i,j} = \mathrm{g}(\mathrm{W}_{i,j}x)$$
$$= g\left(\sum_k W_{i,j,k}x_k + b_{i,j}\right)$$

g is the **activation function**

We'll use g = ReLU

$$ReLU(x) = \begin{cases} x, x > 0 \\ 0, otherwise \end{cases}$$



Feedforward Neural Networks

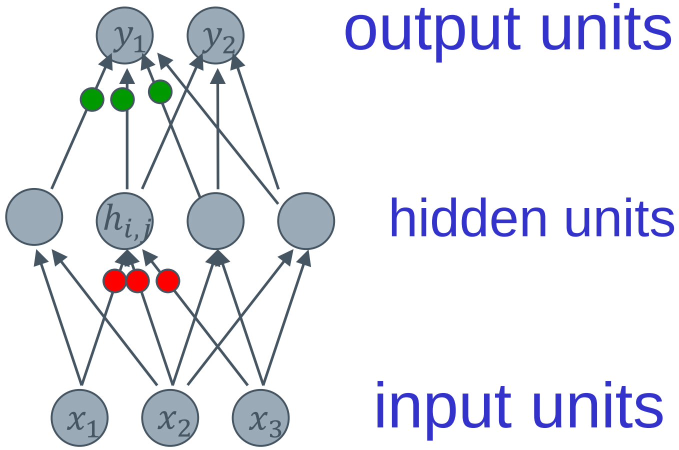
$$\hat{z}_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

Possible cost function:

$$-\sum_i z_i \log(\hat{z}_i)$$

$$z_i = \begin{cases} 1, if\ i\ is\ the\ correct\ answer \\ 0, otherwise \end{cases}$$

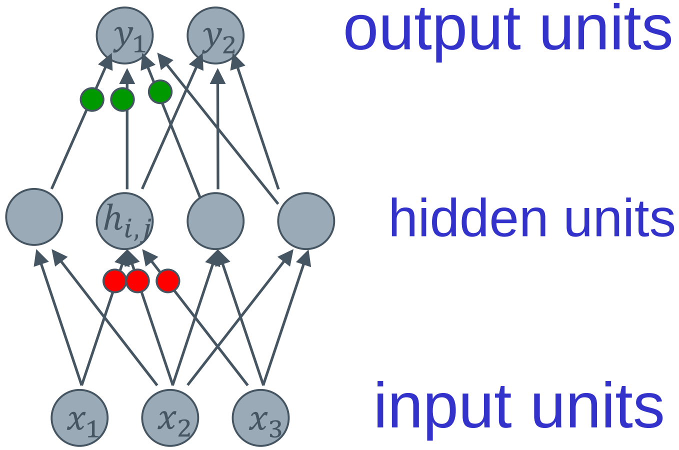
For a single case, we are computing the negative log probability of the *correct answer*



Feedforward Neural Networks

To learn a network:

- Minimize the sum of the cost functions for every training case

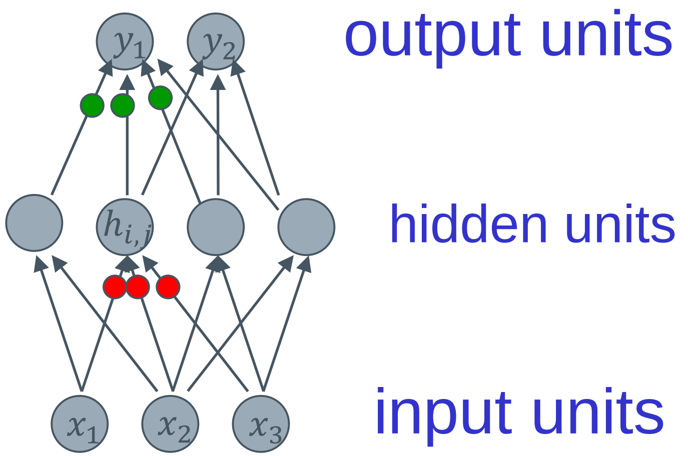


Feedforward Neural Networks

See

`Face Recognition with PyTorch.ipynb`

`Lower-level Programming with PyTorch.ipynb`



Reinforcement Learning with Policy Gradients

- An *agent* interacts with the *environment*
- In an episode, an agent interacts with the environment for n steps
- At time-step *t,* the observed state is S_t, and an agent takes action A_t with probability $P_\theta(A_t|S_t)$
- The parameter vector θ determines the policy that the agent follows
- We want to learn the policy θ

The Iterated Prisoner's Dilemma

- Two prisoners are considering whether to give evidence to the police
- Prisoners cooperate with each other: each is penalized by -1
- One prisoners cooperates (stays mum) and one defects (talks): the defector is penalized by 0, the cooperator is penalized by -3
- Both defect: each is penalized by -2

REINFORCE

Repeat

Generate an episode

For each step of the episode

$G_t \leftarrow$return from step t (i.e., total rewards)

$\theta \leftarrow \theta + \alpha\gamma^t \nabla_\theta \log \pi_\theta(A_t|S_t)$

- This is gradient descent on the *parameters of the policy*
  - Pretty remarkable!

REINFORCE

See

REINFORCE with PyTorch.ipynb

Thank you!

Questions?