Practical Deep Neural Networks

GPU computing perspective

Feedforwad Neural Networks

Yuhuang Hu   Chu Kiong Loo

Advanced Robotic Lab
Department of Artificial Intelligence
Faculty of Computer Science & IT
University of Malaya

Outline

Outline

Assumed prerequisites

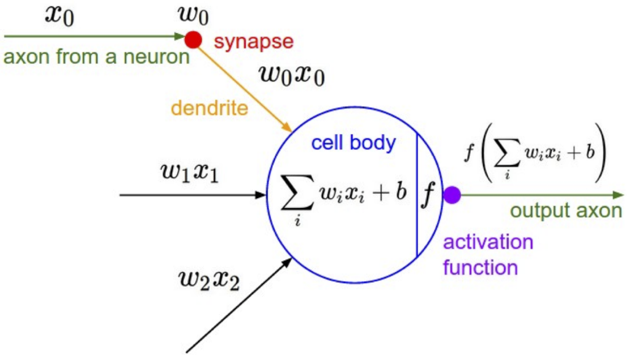
- Numerical Computation [DL book chapter 4]
- Machine Learning Basics [DL book chapter 5]

Suggest Readings

- ✍ UFLDL Tutorial: Multi-Layer Neural Network
- ✍ Deep Learning book: Feedforward Deep Networks
- ✍ CS231n: Neural Networks Part 1, Part 2, Part 3.
- ✍ Pattern Recognition and Machine Learning: Chapter 5

Outline

Neuron


x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
output axon
activation function
$w_2 x_2$

Activation function: Sigmoid

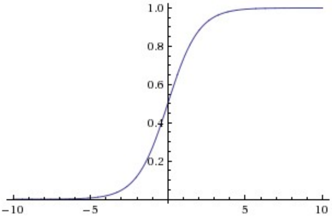


Figure: Sigmoid

- $f(x) = \frac{1}{1+\exp(-x)}$
- ✓ Rescale numbers to $[0, 1]$
- ✓ Historically, it's very popular since it's nice to interpret "firing rate".
- ✗ Saturated neurons "kill" the gradients
- ✗ Sigmoid outputs are not zero-centered

Activation function: $\tanh$

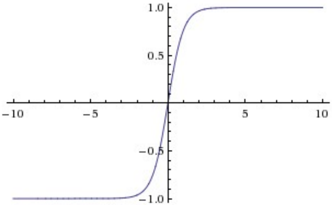


Figure: $\tanh$

- $f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$

✓ Rescale numbers to $[-1, 1]$

✓ Output is zero-centered

✗ Still "kill" gradients saturated

Activation function: ReLU

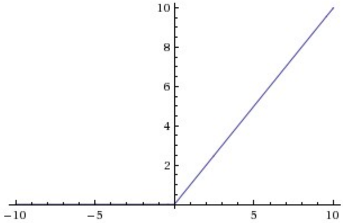


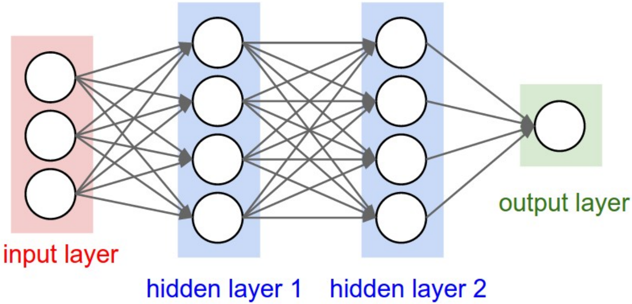
Figure: ReLU

- $f(x) = \max(0, x)$
- ✓ Does not saturate
- ✓ Very computationally efficient
- ✓ Converge much faster than sigmoid/$\tanh$ in practice

Activation function: in practice

- Use ReLU. Be careful with your learning rates
- Try out $\tanh$ but don't expect much
- Never use sigmoid

MLP Network



MLP Network

- Layer

$$\mathbf{h} = f_l(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- MLP Network Feedforward pass:
  For $l = 1 \ldots, n$:

$$\mathbf{h}^l = f_l(\mathbf{W}^l \mathbf{h}^{l-1} + \mathbf{b}^l)$$

where $\mathbf{h}^0 = \mathbf{x}$.

$$\mathbf{y}^{\text{out}} = \mathbf{h}^n$$

Cost Function

- Let's take an example of regression:

$$L(X, \mathbf{y}|\mathbf{W}, \mathbf{b}) = \frac{1}{2N}\sum_{\mathbf{x}_i \in X}\left(\|y_i^{\text{out}} - y_i\|^2\right)$$

- $\mathcal{L}^2$ regularization:

$$L(X, \mathbf{y}|\mathbf{W}, \mathbf{b}) = \frac{1}{2N}\sum_{\mathbf{x}_i \in X}\left(\|y_i^{\text{out}} - y_i\|^2\right) + \frac{\lambda}{2}\sum_{l=1}^{n}\|\mathbf{W}^l\|^2$$

Backpropagation Algorithm

- Target: choose optimal parameter

$$\mathbf{W}^{\star}, \mathbf{b}^{\star} = \arg\min_{\mathbf{W}, \mathbf{b}} L(X, \mathbf{y})$$

- Update by SGD!

$$\mathbf{W}^{*} = \mathbf{W} - \frac{\partial}{\partial \mathbf{W}} L$$
$$\mathbf{b}^{*} = \mathbf{W} - \frac{\partial}{\partial \mathbf{b}} L$$

Outline

Auto-Encoder

- Learn hidden representation $\mathbf{h}$:

$$\mathbf{h} = \sigma_{\text{encode}}(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\hat{\mathbf{x}} = \sigma_{\text{decode}}(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

- Minimize cost (Cross-entropy cost):

$$L(X, \hat{X}) = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^i \log \hat{\mathbf{x}}^i + (1 - \mathbf{x}^i)\log(1 - \hat{\mathbf{x}}^i)$$

Denosing Auto-Encoder

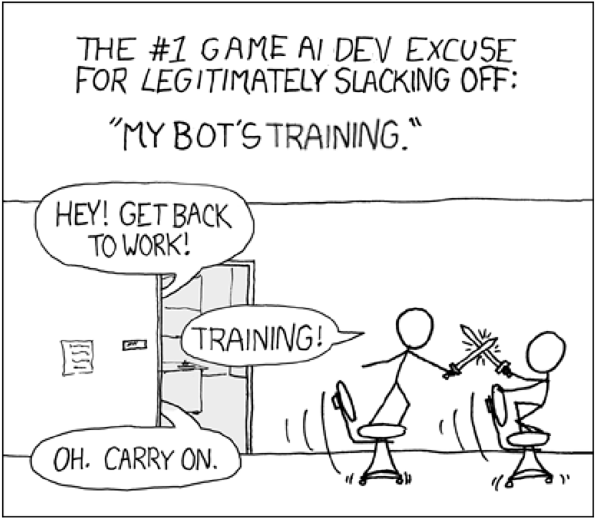
- Idea: reconstruct input from corrupted version!
- $\tilde{X} = X + \eta$, popular choices of the noise η is binomial noise and Gaussian noise.
- Learn hidden representation $\mathbf{h}$:

$$\mathbf{h} = \sigma_{\text{encode}}(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$$
$$\hat{\mathbf{x}} = \sigma_{\text{decode}}(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

- Minimize cost (As same as previously!):

$$L(X, \hat{X}) = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^i \log \hat{\mathbf{x}}^i + (1 - \mathbf{x}^i)\log(1 - \hat{\mathbf{x}}^i)$$

Q&A


THE #1 GAME AI DEV EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY BOT'S TRAINING."
HEY! GET BACK TO WORK!
TRAINING!
OH. CARRY ON.