CS-570 Computer Vision

Nazar Khan

Department of Computer Science
University of the Punjab

10. Deep Learning

Why Deep Learning?

- Most CV problems are increasingly being solved via Deep Learning (DL).
- DL mimics learning in biological brains.
- DL can sometimes beat human performance.

What is Deep Learning?

- Imagine that you have a dataset of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ that you want to map to corresponding targets $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_N$.
- Assume you have function $y = f_1(\mathbf{x}; \mathbf{w}_1)$ that maps inputs $\mathbf{x}$ to outputs $\mathbf{y}$ using parameters $\mathbf{w}_1$.
- You would want parameters $\mathbf{w}_1$ to be such that f_1 maps inputs to targets.
- Any parameters $\mathbf{w}_1$ can be evaluated via an *error function* over the dataset

$$E(\mathbf{w}_1) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2 = \frac{1}{2}\sum_{n=1}^{N}(f_1(\mathbf{x}_n; \mathbf{w}_1) - t_n)^2$$

- Optimal $\mathbf{w}_1^*$ can be found as

$$\mathbf{w}_1^* = \arg\min_{\mathbf{w}_1} E(\mathbf{w}_1)$$

- Such automatic learning of parameters $\mathbf{w}_1^*$ is called *machine learning*.

What is Deep Learning?

- We can use output of f_1 as input to another function f_2 with parameters $\mathbf{w}_2$.

$$y = f_2(f_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2)$$

- Composition of both functions yields a more powerful function.
- Parameters $\mathbf{w}_1, \mathbf{w}_2$ can be evaluated as before

$$E(\mathbf{w}_1, \mathbf{w}_2) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2 = \frac{1}{2}\sum_{n=1}^{N}(f_2(f_1(\mathbf{x}_n; \mathbf{w}_1); \mathbf{w}_2) - t_n)^2$$

- Parameters can be learned as before

$$\mathbf{w}_1^*, \mathbf{w}_2^* = \arg\min_{\mathbf{w}_1, \mathbf{w}_2} E(\mathbf{w}_1, \mathbf{w}_2)$$

- Learning a sequence of such function $f_1, f_2, \ldots, f_L$ with parameters $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_L$ is called *deep learning*.

Minimization

- Minima of a function $E(\theta)$ are characterized by the condition

$$\nabla_\theta E = \mathbf{0}$$

- To reach a (local) minimum, *gradient descent* can be used

$$\theta^{\tau+1} = \theta^{\tau} - \eta \, \nabla_\theta E|_{\theta^\tau}$$

- Modern deep learning frameworks provide
  - more sophisticated methods of reaching local minima (Adam, AdaGrad, etc.), and
  - automatic computation of gradient $\nabla_\theta E$.

  Therefore, we will assume that gradient computation and error minimization is always available.
- *Just need to implement the error function for your problem*.

The Artificial Neuron

- An artificial neuron is a very simple *non-linear* function

$$f(\mathbf{x};\mathbf{w}) = h(\mathbf{w}^T\mathbf{x} + b)$$

where h is usually the ReLU function

$$h(a) = ReLU(a) = \begin{cases} a & a \geq 0 \\ 0 & a < 0 \end{cases}$$

- A neuron can be viewed as a detector of its own weights.
- When $\mathbf{w}^T\mathbf{x}$ is high, neuron is more likely to *fire*.

Neural Networks

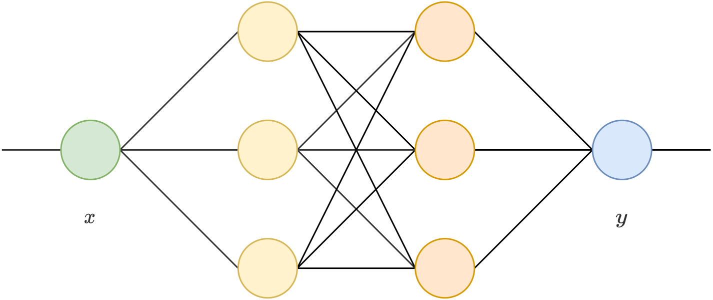


Figure: A simple 3 layer neural network mapping scalar input x to scalar output y. Author: N. Khan (2021)

Neural Networks

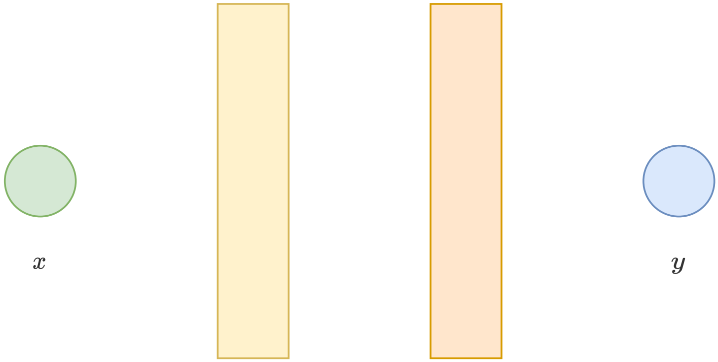


Figure: A simple 3 layer neural network with *hidden* neurons folded in space (viewed as vectors). Author: N. Khan (2021)

Neural Networks

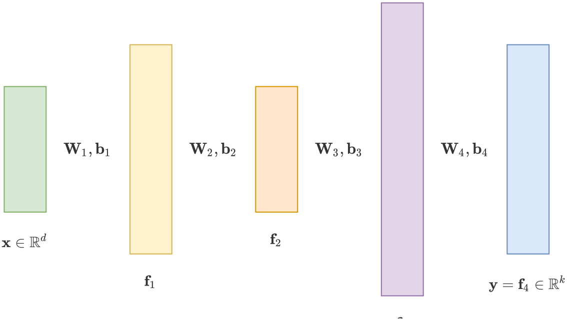


Figure: A general 3 layer neural network with vector inputs, vector hidden layers and vector outputs. Author: N. Khan (2021)

Loss Functions for Machine Learning

Notation:

- Let $x \in \mathbb{R}$ denote a *univariate* input.
- Let $\mathbf{x} \in \mathbb{R}^D$ denote a *multivariate* input.
- Same for targets $t \in \mathbb{R}$ and $\mathbf{t} \in \mathbb{R}^K$.
- Same for outputs $y \in \mathbb{R}$ and $\mathbf{y} \in \mathbb{R}^K$.
- Let θ denote the set of *all* learnable parameters of a machine learning model.

Loss Functions for Machine Learning
Regression

- Univariate

$$L(\theta) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2$$

- Multivariate

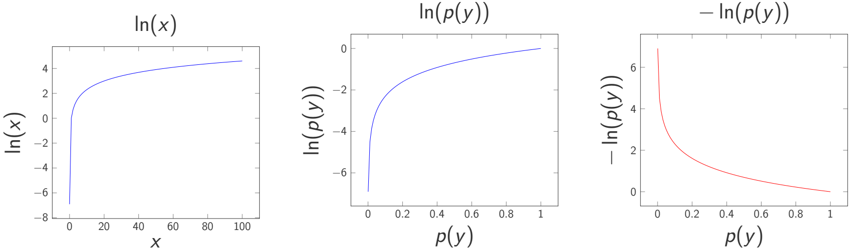
$$L(\theta) = \frac{1}{2}\sum_{n=1}^{N}\|\mathbf{y}_n - \mathbf{t}_n\|^2$$

- Known as half-sum-squared-error (SSE) or ℓ_2-loss.
- *Verify that both losses are 0 when outputs match targets for all n. Otherwise, both losses are greater than 0.*

Background

Probability and Negative of Natural Logarithm

- Logarithm is a monotonically increasing function.
- Probability lies between 0 and 1.
- Between 0 and 1, logarithm is negative.
- So $-\ln(p(x))$ approaches ∞ for $p(x) = 0$ and 0 for $p(x) = 1$.
- Can be used as a loss function.



Loss Functions for Machine Learning
Binary Classification

- For *two-class classification*, targets can be binary.
  - $t_n = 0$ if $\mathbf{x}_n$ belongs to class $\mathcal{C}_0$.
  - $t_n = 1$ if $\mathbf{x}_n$ belongs to class $\mathcal{C}_1$.
- If output y_n can be restricted to lie between 0 and 1, we can *treat* it as probability of $\mathbf{x}_n$ belonging to class $\mathcal{C}_1$. That is, $y_n = P(\mathcal{C}_1|\mathbf{x}_n)$.
- Then $1 - y_n = P(\mathcal{C}_0|\mathbf{x}_n)$.
- Ideally,
  - y_n should be 1 if $\mathbf{x}_n \in \mathcal{C}_1$, and
  - $1 - y_n$ should be 1 if $\mathbf{x}_n \in \mathcal{C}_0$.
- Equivalently,
  - $-\ln y_n$ should be 0 if $\mathbf{x}_n \in \mathcal{C}_1$, and
  - $-\ln(1 - y_n)$ should be 0 if $\mathbf{x}_n \in \mathcal{C}_0$.

Loss Functions for Machine Learning

Binary Classification

- So depending upon t_n, either $-\ln y_n$ or $-\ln(1-y_n)$ should be considered as loss.
- Using t_n to *pick* the relevant loss, we can write total loss as

$$L(\theta) = -\sum_{n=1}^{N} t_n \ln y_n + (1-t_n)\ln(1-y_n)$$

- Known as *binary cross-entropy (BCE) loss*.
- *Verify that BCE loss is 0 when outputs match targets for all n. Otherwise, loss is greater than 0.*

Loss Functions for Machine Learning
Multiclass Classification

- For *multiclass classification*, targets can be represented using *1-of-K coding*. Also known as *1-hot vectors*.
  - 1-hot vector: only one component is 1. All the rest are 0.
  - If $t_{n3} = 1$, then $\mathbf{x}_n$ belongs to class 3.
- If outputs of K neurons can be restricted to
  1. $0 \leq y_{nk} \leq 1$, and
  2. $\sum_{k=1}^{K} y_{nk} = 1$,

  then we can *treat* outputs as probabilities.
- Later, we shall see activation functions that produce per-class probability values.

$$\mathbf{t}_n = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{y}_n = \begin{bmatrix} P(\mathcal{C}_1|\mathbf{x}_n) \\ P(\mathcal{C}_2|\mathbf{x}_n) \\ P(\mathcal{C}_3|\mathbf{x}_n) \\ P(\mathcal{C}_4|\mathbf{x}_n) \\ P(\mathcal{C}_5|\mathbf{x}_n) \end{bmatrix}$$

Loss Functions for Machine Learning
Multiclass Classification

- Similar to BCE loss, we can use t_{nk} to *pick* the relevant negative log loss and write overall loss as

$$L(\theta) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

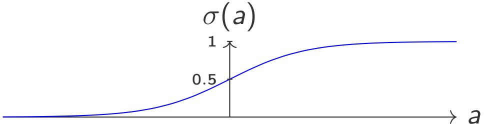
- Known as *multiclass cross-entropy (MCE) loss*.
- *Verify that MCE loss is 0 when outputs match targets for all n. Otherwise, loss is greater than 0.*

Activation Functions

- Recall that a perceptron has a non-differentiable activation function, i.e., step function.
    - Zero-derivative everywhere except at 0 where it is non-differentiable.
- Prevents gradient descent.
- Can we use a smooth activation function that behaves similar to a step function?
- Perceptron with a smooth activation function is called a *neuron*.
- Neural networks are also called multilayer perceptrons (MLP) even though they do not contain any perceptron.

Logistic Sigmoid Function

- For $a \in \mathbb{R}$, the *logistic sigmoid* function is given by $\sigma(a) = \frac{1}{1+e^{-a}}$
- *Sigmoid* means S-shaped.
- Maps $-\infty \leq a \leq \infty$ to the range $0 \leq \sigma \leq 1$. Also called *squashing* function.
- **Can be treated as a probability value**.



Activation Functions

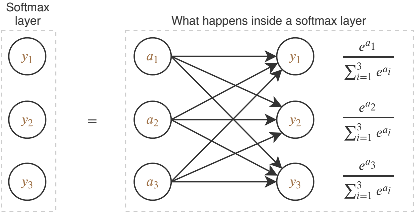
Regression

- Univariate: use 1 output neuron with identity activation function $y(a) = a$.
- Multivariate: use K output neurons with identity activation functions $y(a_k) = a_k$.

Classification

- Binary: use 1 output neuron with logistic sigmoid $y(a) = \sigma(a)$.
- Multiclass: use K output neurons with *softmax* activation function.

Softmax Activation Function



- For real numbers $a_1, \ldots, a_K$, the *softmax* function is given by

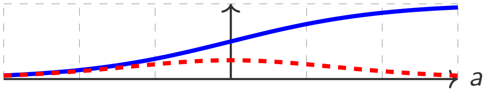
$$y(a_k; a_1, a_2, \ldots, a_K) = \frac{e^{a_k}}{\sum_{i=1}^{K} e^{a_i}}$$

- Output of k-th neuron depends on activations of *all neurons in the same layer*.

Softmax Activation Function

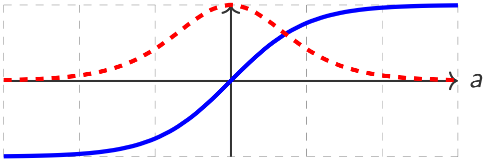
- Softmax is ≈ 1 when $a_k >> a_j \; \forall j \neq k$ and ≈ 0 otherwise.
- Provides a smooth (differentiable) approximation to finding the *index of the maximum element*.
  - Compute softmax for $1, 10, 100$.
  - Does not work everytime.
    - Compute softmax for $1, 2, 3$. Solution: multiply by 100.
    - Compute softmax for $1, 10, 1000$. Solution: subtract maximum before computing softmax.
- Also called the *normalized exponential* function.
- Since $0 \leq y_k \leq 1$ and $\sum_{k=1}^{K} y_k = 1$, *softmax outputs can be treated as probability values.*

Logistic Sigmoid



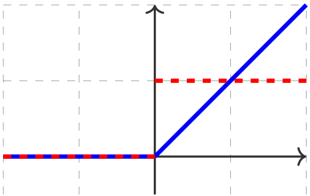
Activation function	$y(a) = \frac{1}{1+e^{-a}}$
Derivative	$y'(a) = y(a)(1 - y(a))$
Maximum magnitude of derivative	$\frac{1}{4}$
Problem	Cause vanishing gradients

Hyperbolic Tangent



Activation function	$y(a) = \tanh(a)$
Derivative	$y'(a) = 1 - y^2(a)$
Maximum magnitude of derivative	1
Problem	Cause vanishing gradients

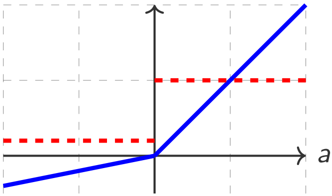
Rectified Linear Unit (ReLU)



Activation function	$y(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{if } a \leq 0 \end{cases}$
Derivative	$y'(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{if } a \leq 0 \end{cases}$
Advantage	Avoids vanishing gradients
Problem	Dead neurons[1]

[1] This can be an advantage as well since death implies fewer neurons.

Leaky ReLU



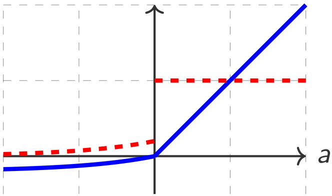
Activation function $y(a) = \begin{cases} a & \text{if } a > 0 \\ ka & \text{if } a \leq 0 \end{cases}$

where $0 \leq k \leq 1$

Derivative $y'(a) = \begin{cases} 1 & \text{if } a > 0 \\ k & \text{if } a \leq 0 \end{cases}$

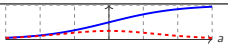
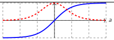
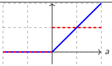
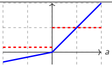
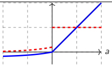
Advantage Neuron is always learning

Exponential Linear Unit (ELU)



Activation function	$y(a) = \begin{cases} a & \text{if } a > 0 \\ k(e^a - 1) & \text{if } a \leq 0 \end{cases}$ where $k > 0$
Derivative	$y'(a) = \begin{cases} 1 & \text{if } a > 0 \\ y(a) + k & \text{if } a \leq 0 \end{cases}$
Maximum magnitude of derivative	1
Advantage	Neuron is mostly learning

Activation Functions
Summary

Name	$y(a)$	Plot	$y'(a)$	Comments
Logistic sigmoid	$\frac{1}{1+e^{-a}}$		$y(a)(1-y(a))$	Vanishing gradients
Hyperbolic tangent	$\tanh(a)$		$1-y^2(a)$	Vanishing gradients
Rectified Linear Unit (ReLU)	$\begin{cases} a & \text{if } a > 0 \\ 0 & \text{if } a \leq 0 \end{cases}$		$\begin{cases} 1 \\ 0 \end{cases}$	Dead neurons. Sparsity.
Leaky ReLU	$\begin{cases} a & \text{if } a > 0 \\ ka & \text{if } a \leq 0 \end{cases}$		$\begin{cases} 1 \\ k \end{cases}$	$0 < k < 1$
Exponential Linear Unit (ELU)	$\begin{cases} a & \text{if } a > 0 \\ k(e^a - 1) & \text{if } a \leq 0 \end{cases}$		$\begin{cases} 1 \\ y(a) + k \end{cases}$	$k > 0$.

- Saturated sigmoidal neurons stop learning. Piecewise-linear units keep learning by avoiding saturation.
- ELU has been shown to lead to better accuracy and faster training.
- *Take home message*: For hidden neurons, use a member of the LU family. They avoid *i*) saturation and *ii*) the vanishing gradient problem.

Regularization in Neural Networks

- A model that performs well on training data but poorly on test data is said to be *over-fitting*.
- Over-fitting can be lessened via *regularization* which can be understood as restricting the power of the model.
  1. Penalise magnitudes of weights: $\tilde{L}(\mathbf{w}) = L(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$.
  2. *Dropout*: *During training*, a randomly selected subset of activations are set to zero within each layer.
  3. *Early stopping* by checking $E(\mathbf{w})$ on a validation set. Stop when error on validation set starts increasing.
  4. Training with *augmented*/transformed data.
  5. Batch Normalization.

Summary

- Deep learning can no longer be avoided by a CV practitioner.
- Very brief introduction to deep learning.
- Enough to get you started.
- Proper understanding can be obtained through a complete deep learning course.
- Overall idea: transform input $\mathbf{x}$ into another representation $\mathbf{f}(\mathbf{x})$ which is more useful for making decisions about $\mathbf{x}$.