

Mathematics of Deep Learning

Authors: Amir Hajian

July 2019

This Slide Deck is part of the workshop "Mathematics of Deep Learning" run by Aggregate Intellect Inc. (<https://ai.science>), and is released under 'Creative Commons Attribution-NonCommercial-ShareAlike CC BY-NC-SA' license. This material can be altered and distributed for non-commercial use with reference to Aggregate Intellect Inc. as the original owner, and any material generated from it must be released under similar terms (<https://creativecommons.org/licenses/by-nc-sa/4.0/>).

Outline

- What we will learn in these 9 hours?
- What are the hands-on sessions?
- How can I get the most out of it?
- How to do deep learning in 2019? (we will come back to this at the end of the last day)



What to aim for

You should be able to follow this work by the end of the workshop

Dynamic Deep Networks for Retinal Vessel Segmentation

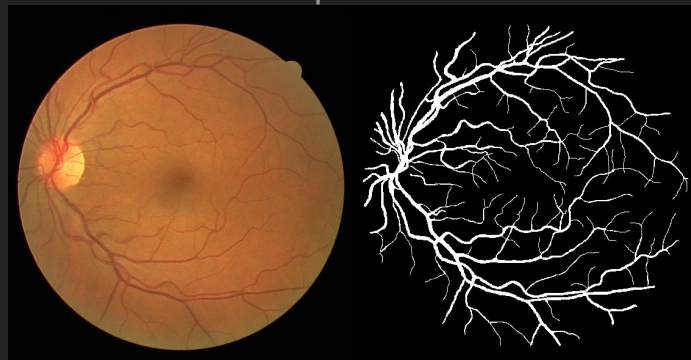
AASHIS KHANAL^{1,2} AND ROLANDO ESTRADA^{1,3}

¹Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA

²akhanal1@student.gsu.edu

³restrada1@gsu.edu

Abstract: Segmenting the retinal vasculature entails a trade-off between how much of the overall vascular structure we identify vs. how precisely we segment individual vessels. In particular, state-of-the-art methods tend to under-segment faint vessels, as well as pixels that lie on the edges of thicker vessels. Thus, they underestimate the width of individual vessels, as well as the ratio of large to small vessels. More generally, many crucial bio-markers—including the



Search or jump to...



Pull requests

Issues

Marketplace



sraashis / ature

<> Code

Issues 1

Pull requests 0

Projects 0

Wiki

Security

pytorch based framework for Convolutional Neural Network

A decorative graphic on the left side of the slide, consisting of two overlapping, semi-transparent green trapezoidal shapes that create a 3D effect.

Recap from last session

ORIGINAL CONTRIBUTION

Multilayer Feedforward Networks are Universal Approximators

KURJ HORNIK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBEK WHITE

University of California, San Diego

(Received 16 September 1988; revised and accepted 9 March 1989)

Abstract—This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

Keywords—Feedforward networks, Universal approximation, Mapping networks, Network representation

[Hornik et al \(1989\)](#)

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

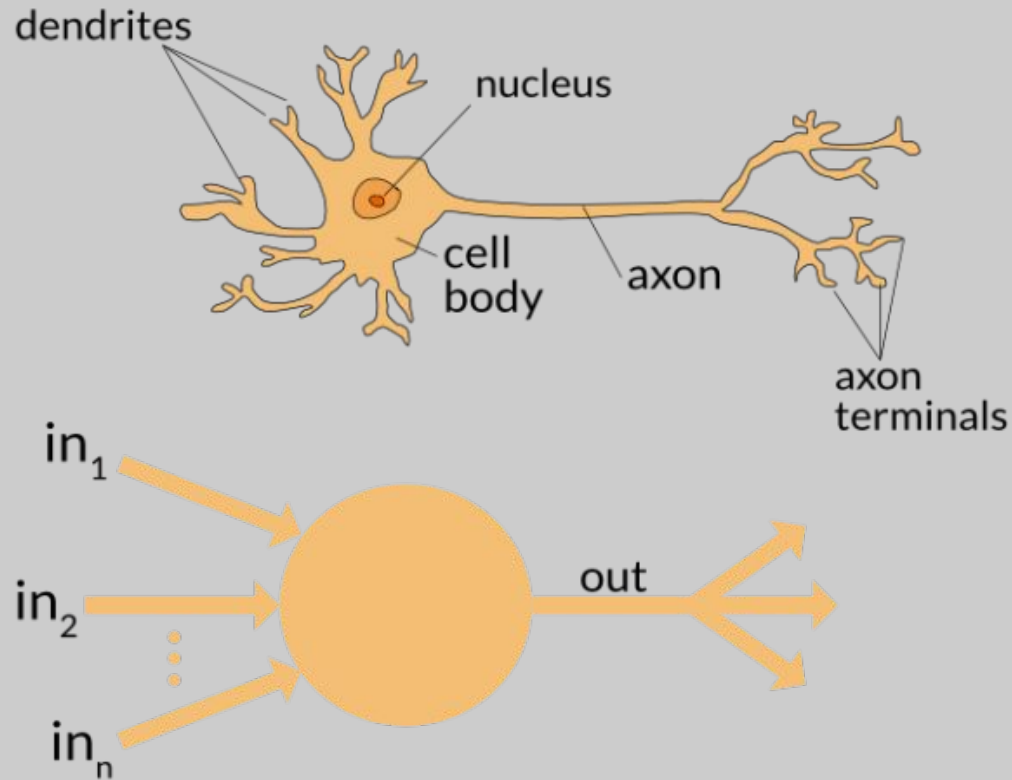
[Cybenko \(1989\)](#)

Why deep learning works?

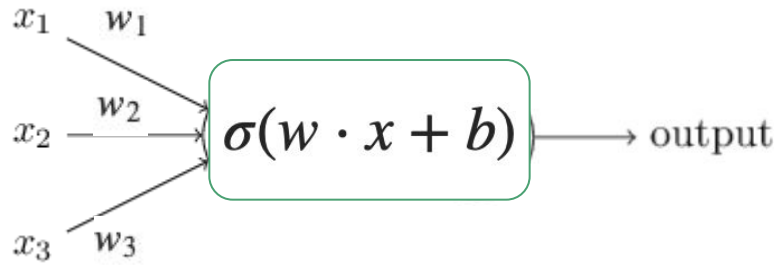
Any continuous function can be approximated by multilayer feedforward networks with a hidden layer and arbitrary bounded and non-constant activation functions.

Universality
Theorem

An artificial neuron



Simplifying the notation



$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = w \cdot x$$

It's all about **matrices**, **vectors** and **exploring the parameter space** to find the right parameters!



Linear Algebra: Tensors and Scalars

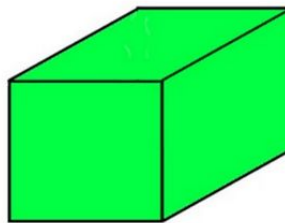
Tensors

1D TENSOR/
VECTOR

5
7
4 5
1 2
- 6
3
2 2
1
6
3
- 9

2D TENSOR /
MATRIX

- 9	4	2	5	7
3	0	1 2	8	6 1
1	2 3	- 6	4 5	2
2 2	3	- 1	7 2	6



3D TENSOR/
CUBE

- 9	4	2	5	7
3	0	1 2	8	6 1
1	2 3	- 6	4 5	2
2 2	3	- 1	7 2	6

Pytorch Tensors

Ways to create a Tensor in PyTorch:

- with pre-existing data, use [torch.tensor\(\)](#).
- with specific size, use torch.* tensor [Creation Ops](#).
- with the same size (and similar types) as another tensor, use torch.*_like tensor creation ops.
- with similar type but different size as another tensor, use tensor.new_* creation ops.

```
import torch

x = torch.tensor([-1, 7])

x = torch.rand(5, 3)

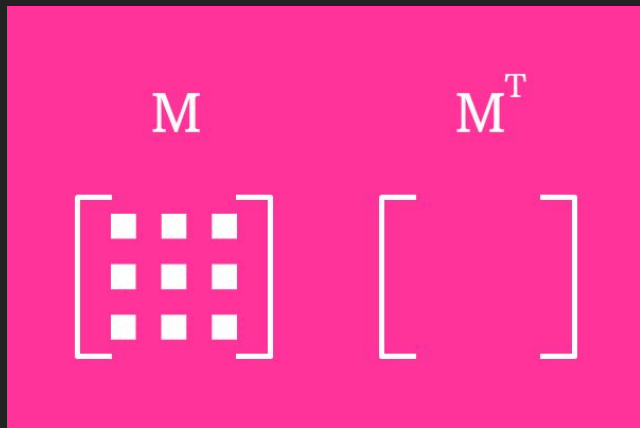
y = torch.randn_like(x,
dtype=torch.float)

y = x.new_ones(5, 3)
```

Hands on exercise

- Getting started with PyTorch:
 - What is PyTorch
 - How to import it
- Exercises with Tensors and Scalars
 - Define a tensor in PyTorch
 - Fill a PyTorch tensor with a certain scalar
 - Fill a PyTorch tensor with random numbers
 - Find minimum value of a PyTorch tensor
 - Convert a Py list to a PyTorch tensor and vice versa

Linear Algebra: Matrix Transpose



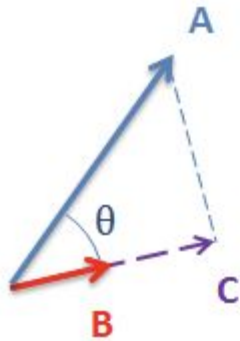
- `torch.t(M)`

Properties:

$$(AB)^T = B^T A^T$$
$$(A + B)^T = A^T + B^T$$

Linear Algebra: Dot Product

*



$$A \cdot B = |A||B| \cos(\theta)$$

if the magnitude of B is 1, then...

$$C = A \cdot B = |A| \cos(\theta)$$

Linear Algebra: Dot Product

*

$$\mathbf{a} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$$

$$\mathbf{a} \cdot \mathbf{b} = x_1 x_2 + y_1 y_2 + z_1 z_2$$

$$\mathbf{a} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 3 \\ 5 \\ 7 \end{pmatrix}$$

$$\mathbf{a} \cdot \mathbf{b} = 2(3) + 4(5) + 6(7)$$

$$= 68$$

Linear Algebra: Dot Product

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b}$$

$$= (a_1 \ a_2 \ \cdots \ a_n) \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$= a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$= \sum_{i=1}^n a_i b_i,$$

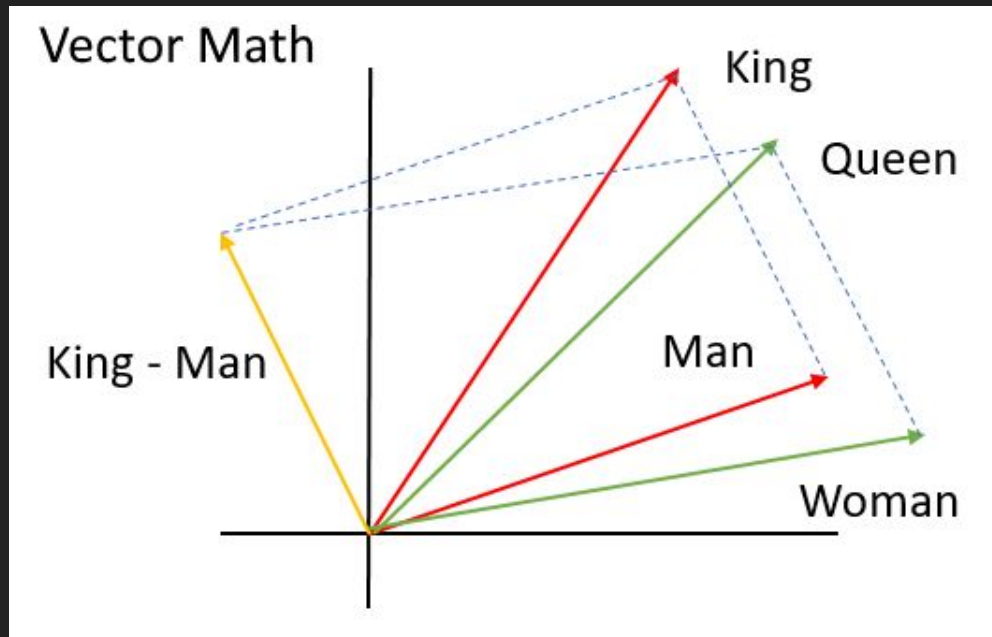
- `torch.matmul(a, b)`

Linear Algebra: Dot Product

- **Application:**
 - Word Embeddings

Assignment:

- Experiment with word embeddings



Linear Algebra: Matrix Multiplication



```
torch.matmul(M1, M2)
```

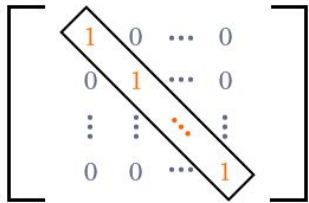
Linear Algebra: Special Matrices

$$\begin{bmatrix} -3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

$$A = \text{diag}(v)$$

- Diagonal Matrix

```
torch.diag(torch.tensor([-3, 4, 7]))
```


$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

- Identity Matrix

```
torch.eye(5)
```

If A is a $m \times n$ matrix, then

$$I_m A = A \text{ and } A I_n = A$$

If A is a $n \times n$ matrix, then

$$A I_n = I_n A = A$$

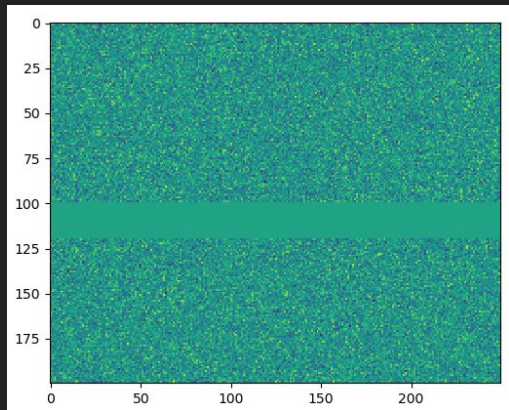
- Null Matrix

```
torch.zeros(5, 5)
```

Exercise: Transpose images

Simulated Data:

- Create a random 2D matrix with dimensions 200x250, set columns 100:120 to zero, display it, transpose the matrix, display it again.

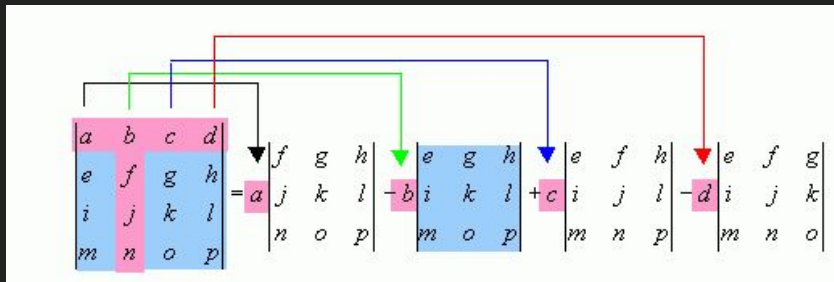


Real image data:

- Read the image provided to you, display it, transpose it, and display it again.



Linear Algebra: Matrix Determinant



- **Properties:**

- `torch.det(M)`

$$\det(A) = \det(A^T)$$

$$\det(A) = \frac{1}{\det(A^{-1})}$$

$$\det(cA) = c^n \det(A) \quad \text{for } n \times n \text{ matrix}$$

$$\det(A^n) = \det(A)^n$$

$$\det(AB) = \det(A) \det(B)$$

$$\det(A) = \lambda_1 \lambda_2 \dots \lambda_n \quad \det(A) = \text{product of all eigenvalues}$$

Linear Algebra: Matrix Inversion

$$AA^{-1} = A^{-1}A = I$$

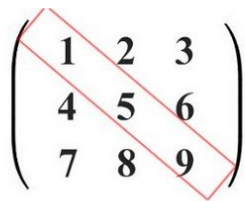
- `torch.inverse(M)`

- Special case:

$$\begin{pmatrix} \mathbf{A}_1 & 0 & \dots & 0 \\ 0 & \mathbf{A}_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{A}_n \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}_1^{-1} & 0 & \dots & 0 \\ 0 & \mathbf{A}_2^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{A}_n^{-1} \end{pmatrix}.$$

Linear Algebra: Trace

$$\text{Tr}(\mathbf{A}) = \sum_i A_{ii}$$


$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- **Properties:**

- `torch.trace(M)`

$$\text{Tr}(\mathbf{A}) = \sum_i \lambda_i, \quad \lambda_i = \text{eig}(\mathbf{A})$$

$$\text{Tr}(\mathbf{A}) = \text{Tr}(\mathbf{A}^T)$$

$$\text{Tr}(\mathbf{AB}) = \text{Tr}(\mathbf{BA})$$

$$\text{Tr}(\mathbf{A} + \mathbf{B}) = \text{Tr}(\mathbf{A}) + \text{Tr}(\mathbf{B})$$

$$\text{Tr}(\mathbf{ABC}) = \text{Tr}(\mathbf{BCA}) = \text{Tr}(\mathbf{CAB})$$

$$\mathbf{a}^T \mathbf{a} = \text{Tr}(\mathbf{a}\mathbf{a}^T)$$

[Read more](#)

Linear Algebra: Eigendecomposition

$$Av = \lambda v.$$

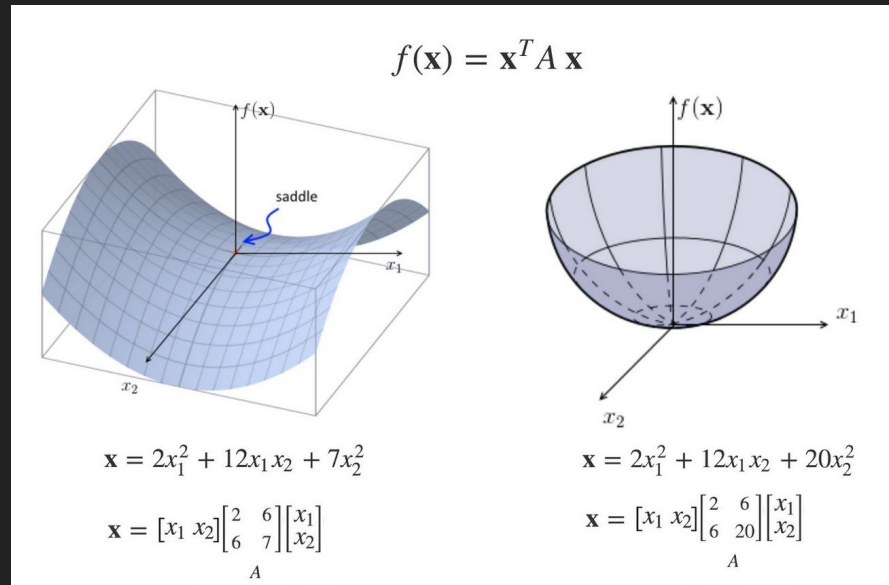
- `torch.eig(M)`

- If all eigenvalues > 0 called **positive definite**
- If all eigenvalues ≥ 0 called **positive semidefinite**

- [Calculate Eigenvalues](#)
- Read more: [Eigenvalues in ML](#)

Linear Algebra: Positive Definite Matrices

- **positive definite** matrices lead to convex functions that are desirable in optimization
- We'll discuss this more in future sessions



Linear Algebra: Working with Matrices

In the literature you might run into expressions like this

$$A = V \Lambda V^{-1}$$

What is it and what are they used for?

[Special Matrices](#)

Linear Algebra: Working with Matrices

Orthogonal Matrices

$$Q^T Q = Q Q^T = I$$

Nice property:

$$Q^{-1} = Q^T$$

[Special Matrices](#)

Linear Algebra: Working with Matrices

Symmetric Matrices

$$\mathbf{A} = \mathbf{A}^T$$

every real symmetric matrix can be decomposed into an expression like this using only real-valued eigenvectors and eigenvalues

orthogonal matrix preserves norm $\|Qx\| = \|x\|$

only factor that impact the norm of S

$$S = Q\Lambda Q^T$$

symmetric

eigenvalues of S

Spectral theorem: S can be diagonalized (factorized) with Q formed by the orthonormal eigenvectors v_i of S and Λ is a diagonal matrix holding all the eigenvalues.

$$S = \underbrace{\begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}}_Q \underbrace{\begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_n \end{pmatrix}}_{\Lambda} \underbrace{\begin{pmatrix} \leftarrow \mathbf{v}_1^T \rightarrow \\ \leftarrow \mathbf{v}_2^T \rightarrow \\ \vdots \\ \leftarrow \mathbf{v}_n^T \rightarrow \end{pmatrix}}_{Q^T}$$

Special Matrices

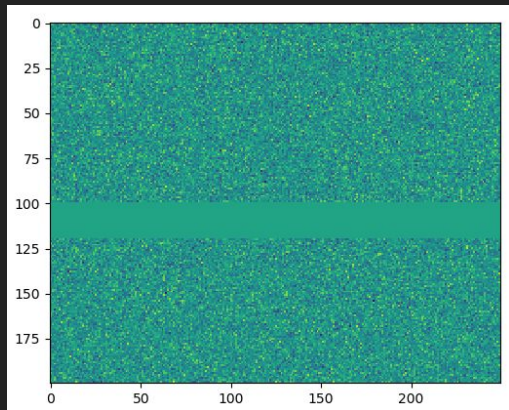
Experiments: Hands-on time

- Define a tensor in PyTorch
- Fill a PyTorch tensor with a certain scalar
- Fill a PyTorch tensor with random numbers
- Find minimum value of a PyTorch tensor
- Reshape a tensor
- Flatten a tensor
- Convert a Py list to a PyTorch tensor and vice versa
- Multiply tensors with a scalar
- Dot product two tensors
- Transpose a matrix in PyTorch
- Matrix Multiplications in PyTorch

Exercise: Transpose images

Simulated Data:

- Create a random 2D matrix with dimensions 200x250, set columns 100:120 to zero, display it, transpose the matrix, display it again.



Real image data:

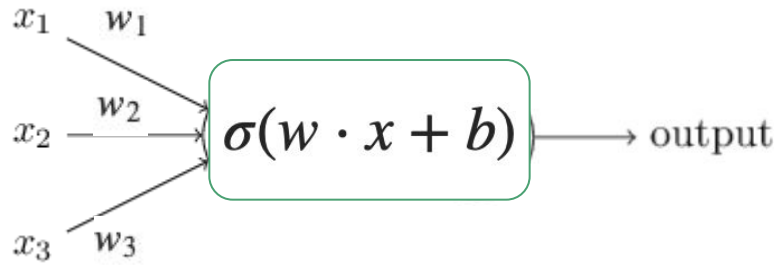
- Read the image provided to you, display it, transpose it, and display it again.



A decorative graphic on the left side of the slide, consisting of two overlapping, semi-transparent green trapezoidal shapes that create a 3D effect.

Non-linearities

Simplifying the notation



$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = w \cdot x$$

It's all about **matrices**, **vectors** and **exploring the parameter space** to find the right parameters!

Non-linearities, activation functions

- Affine Maps:

$$f(x)=Ax+b$$

- PyTorch way:

```
lin = nn.Linear(5, 3)
data = torch.randn(2, 5)
lin(data)
```

Non-linearities:

- Superposition of two affine maps is an affine map:

$$f(x) = Ax + b$$

$$g(x) = Cx + d$$

$$\begin{aligned} f(g(x)) &= A(Cx + d) + b \\ &= ACx + (Ad + b) \\ &= D x + E \end{aligned}$$

- What to use after an affine map?
 - Non-linear activation function

Non-linearities:

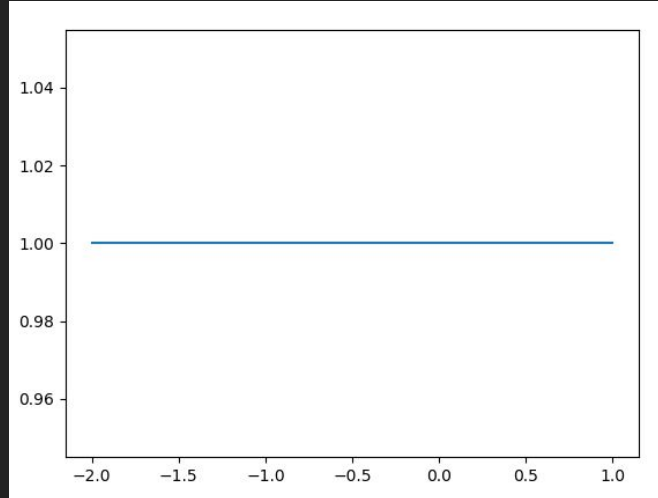
- Activation functions:

$$Y = \text{Activation}(\Sigma(\textit{weight} * \textit{input}) + \textit{bias})$$

- What is a good activation function?
 - A smooth function with a "nice" derivative.

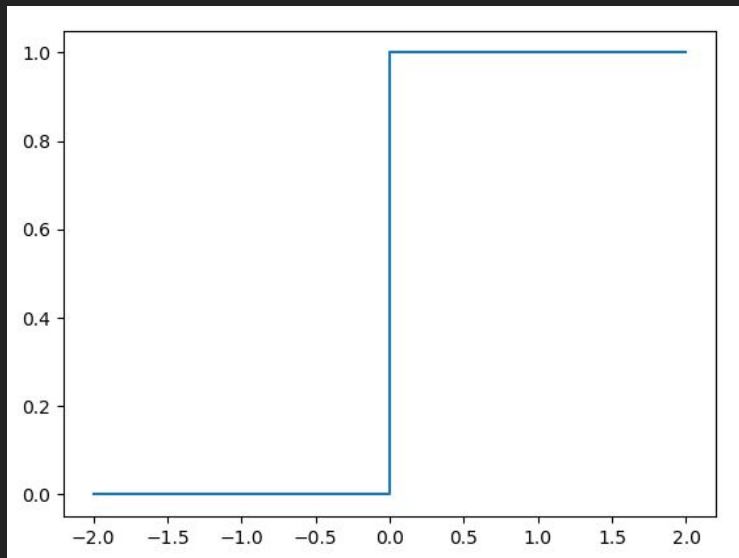
Non-linearities:

- Flat function:
 - Gradient is zero \rightarrow no learning.



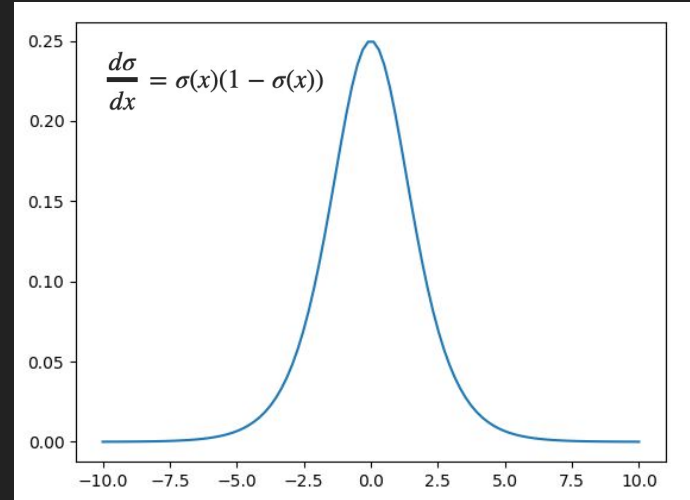
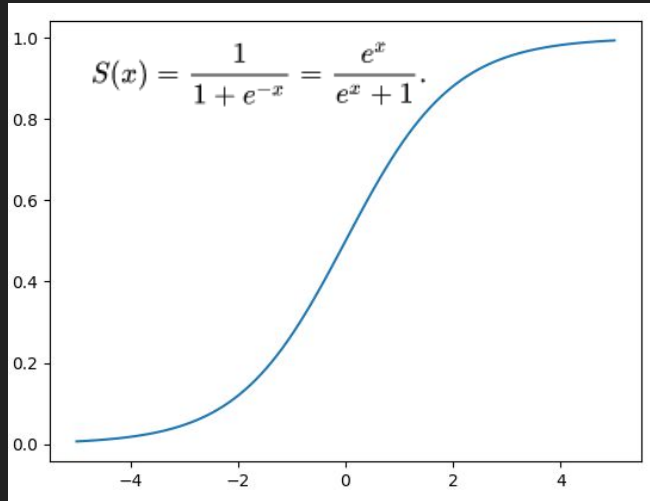
Non-linearities:

- Step function:
 - Gradient is almost always zero \rightarrow no efficient learning.



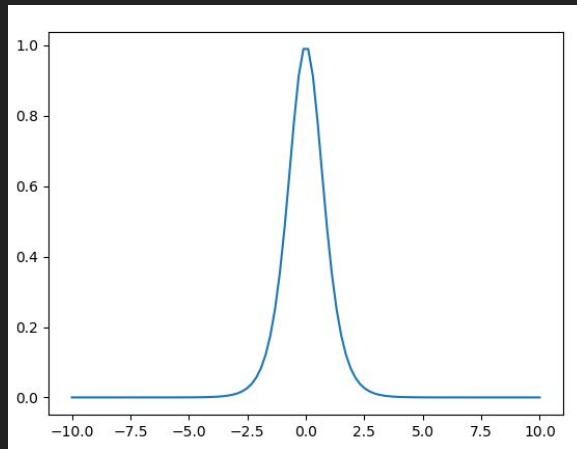
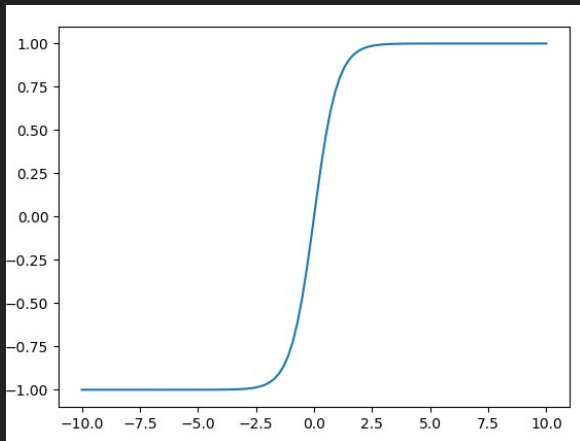
Non-linearities:

- Sigmoid



Non-linearities:

- Tanh

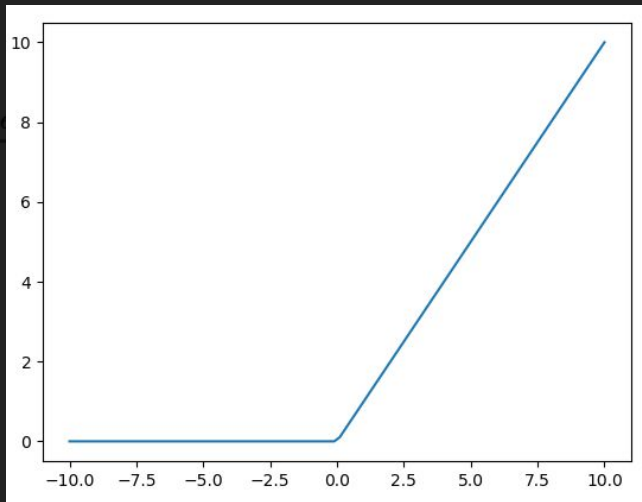


`torch.Tanh(x)`

Non-linearities:

- ReLU

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$



`torch.relu(x)`

Non-linearities:

- Softmax: last operation done in a network

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$

$$\frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

`torch.softmax(x, dim)`

This is actually a probability distribution: each component is positive, and the sum of all components is 1.

Hands on:

- Experiment with non-linear activation functions in PyTorch
 - Types of activation functions and what we use in PyTorch
- Define a ReLU layer in PyTorch
- Work with non-linearities:
 - Plot a relu function
 - Plot a tanh function
 - Plot a sigmoid function
 - Apply ReLU to the image we uploaded earlier
 - Experiment with softmax and verify it is a probability distribution

Loss functions

Loss functions in PyTorch

`torch.nn.MSELoss`

$$\text{loss}(x, y) = (x - y)^2$$

`torch.nn.CrossEntropyLoss`

$$\text{loss}(x, y) = - \sum x \log y$$

[A visual explanation](#)

Assignments

Reading:

- Chapter 2 of [Deep Learning Book](#)
- Word Embeddings ([+](#) & [+](#))
- [How we put words into computers](#)
- *Intermediate*: section 2.12 in chapter 2 of [Deep Learning Book](#)
- *Advanced*: [Towards understanding word embeddings](#)

Programming

- Learn to work with word embeddings. Use what you have learned so far to find the distance between two word vectors, find nearest neighbors, visualize words, get creative.
 - Getting started with word2vec ([blog](#), [notebook](#))
 - Another [Tutorial with Gensim](#)
 - *FastText* [Tutorial](#) on word embeddings

Assignments

- Implement linear regression in PyTorch and compare it to the implementation in numpy - notice a change in the speed when you run it on GPU?

Hint: you can look up the solution [here](#). But do it after implementing your own solution.

Discussion Points