HSLU Lucerne University of Applied Sciences and Arts

Preliminaries

NLP
Andreas Marfurt

Information Technology
20.02.2025

Content of this Lecture

- Things you were taught in previous courses
- Things I wish you were taught in previous courses ;-)

Overview

- Linear algebra
  - Tensor operations
- Neural networks (ADML)
  - Input & output
  - Architecture & weights
  - Loss computation
  - Backpropagation & optimization

Linear Algebra

Naming

- Scalar: 0-dimensional tensor, e.g.: 5
- Vector: 1-dimensional tensor, e.g.: [1, 2, 3]
- Matrix: 2-dimensional tensor, e.g.: [[1, 0], [0, 1]]

- Tensor: n-dimensional tensor, with $n \geq 0$

Indexing Matrix Elements

- mxn ("m by n") matrix: m rows, n columns
- Elements: a_{ij}
  - i-th row
  - j-th column

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \dots & & \\ & & & \\ a_{m1} & \dots & & a_{mn} \end{pmatrix}$$

HSLU

Vector as Matrix

- A vector is a mx1 or an 1xn matrix

$$\begin{pmatrix} a_1 \\ a_2 \\ \dots \\ \\ a_m \end{pmatrix} \qquad \begin{pmatrix} a_1 & a_2 & \dots & & a_n \end{pmatrix}$$

HSLU

Vector Operations

- Vector addition
  - Must have same dimensionality
  - Add elements at the same position (“element-wise” operation)
- Dot-product

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

- Outer product

$$a \otimes b = \begin{bmatrix} a_1 b_1 & \cdots & a_1 b_n \\ \vdots & \ddots & \vdots \\ a_n b_1 & \cdots & a_n b_n \end{bmatrix}$$

Vector Operations

- Norm: Different ways to compute
  - Most common: Euclidean norm

$$\|a\|_2 = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

- Cosine similarity

$$\cos(a, b) = \frac{a \cdot b}{\|a\|_2 \|b\|_2}$$

Matrix Operations

- Matrix addition: Same dimension, element-wise
- Matrix multiplication
  - Inner dimension must match
    - lxm x mxn
  - Result has dimension lxn

$$c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk}$$

- Compare to vector dot-product: 1xn x nx1
  - Same with matrix-vector multiplication

Definition (Matrixmultiplikation)

$\mathbf{A} \in \mathbb{R}^{l \times m}$ ist eine Matrix mit l Zeilen, m Spalten und den Elementen a_{ij}.
$\mathbf{B} \in \mathbb{R}^{m \times n}$ ist eine Matrix mit m Zeilen, n Spalten und den Elementen b_{jk}.
Das Produkt $\mathbf{AB} = \mathbf{C} \in \mathbb{R}^{l \times n}$ hat l Zeilen, n Spalten und die Elemente

$$c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk}\,, \quad i = 1, 2, \ldots, l\,, \quad k = 1, 2, \ldots, n\,.$$

Beispiel

$$\begin{pmatrix} 4 & -2 & -2 & 0 \\ -2 & -7 & 3 & 8 \\ 0 & 1 & -2 & -1 \end{pmatrix} \cdot \begin{pmatrix} 4 & -5 \\ 3 & -1 \\ 6 & 4 \\ 0 & -3 \end{pmatrix} = \begin{pmatrix} -2 & -26 \\ -11 & 5 \\ -9 & -6 \end{pmatrix}$$

$$\begin{aligned} c_{21} &= \sum_{j=1}^{4} a_{2j} b_{j1} = a_{21} b_{11} + a_{22} b_{21} + a_{23} b_{31} + a_{24} b_{41} \\ &= -2 \cdot 4 - 7 \cdot 3 + 3 \cdot 6 + 8 \cdot 0 = -8 - 21 + 18 + 0 = -11 \end{aligned}$$

Neural Networks

Tensor in NLP (Fort.)

tensor example in NLP

sentence	vector representation
hi John	[[1,0,0,0], [0,1,0,0]]
hi James	[[1,0,0,0], [0,0,1,0]]
hi Brian	[[1,0,0,0], [0,0,0,1]]

mini batch input will be

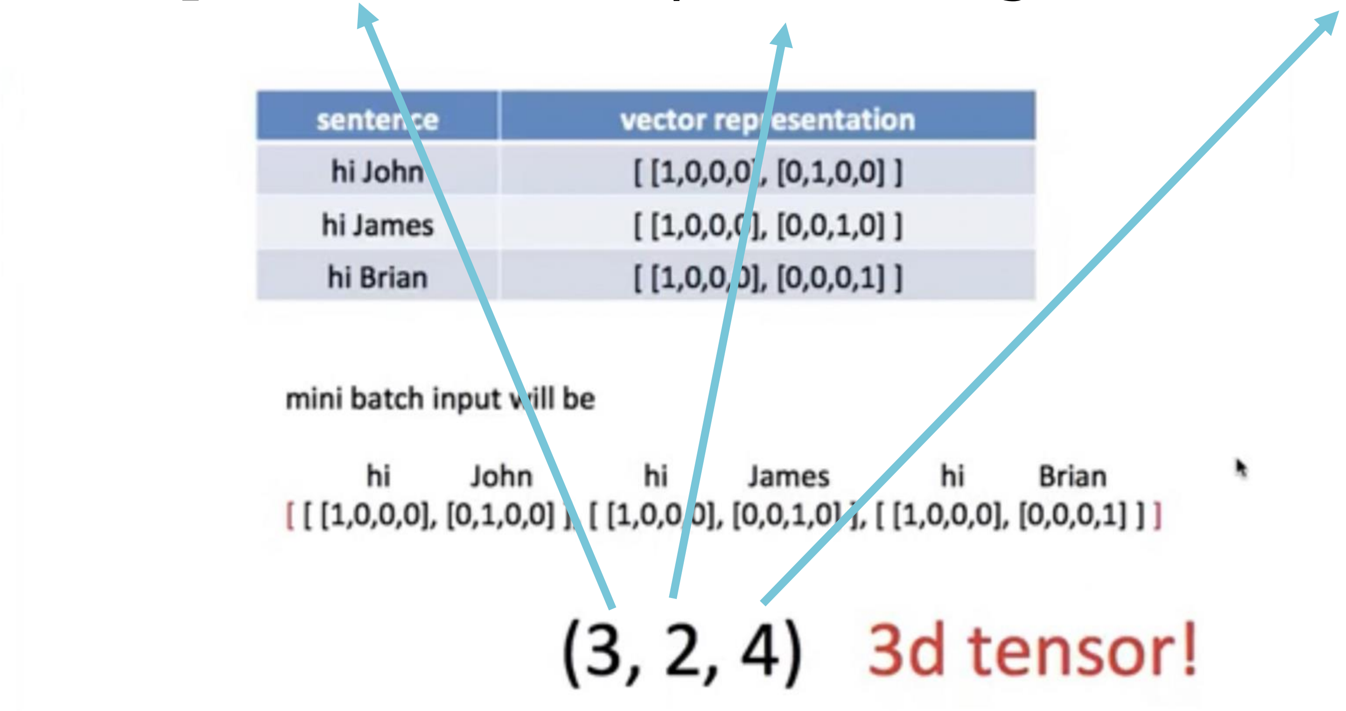
hi John hi James hi Brian

[[[1,0,0,0], [0,1,0,0]], [[1,0,0,0], [0,0,1,0]], [[1,0,0,0], [0,0,0,1]]]

(3, 2, 4) 3d tensor!

Input

Input dimensions: [batch size, sequence length, hidden size/dim]



sentence	vector representation
hi John	[[1,0,0,0], [0,1,0,0]]
hi James	[[1,0,0,0], [0,0,1,0]]
hi Brian	[[1,0,0,0], [0,0,0,1]]

mini batch input will be

hi John hi James hi Brian
[[[1,0,0,0], [0,1,0,0]], [[1,0,0,0], [0,0,1,0]], [[1,0,0,0], [0,0,0,1]]]

(3, 2, 4) 3d tensor!

Input

Sentences:

- "Hi John", "Hi James", "Hi Brian"
  - Batch size?
  - Sequence length?
  - Hidden size?

- "Hi John", "Hi James, how are you?"
  - Batch size?
  - Sequence length?
  - Hidden size?

Input

Sentences:

- "Hi John", "Hi James", "Hi Brian"
  - Batch size? 3
  - Sequence length? 2
  - Hidden size? 4

- "Hi John", "Hi James, how are you?"
  - Batch size?
  - Sequence length?
  - Hidden size?

Input

Sentences:

- "Hi John", "Hi James", "Hi Brian"
  - Batch size? 3
  - Sequence length? 2
  - Hidden size? 4

- "Hi John", "Hi James, how are you?"
  - Batch size? 2
  - Sequence length? 5/7 (are “,” and “?” tokenized separately?)
  - Hidden size? 6/8

Input: Encoding words

- One-hot
- Bag-of-words (BoW)
- TF-IDF

dim = vocab size

- Word embeddings (e.g. word2vec)
- Neural networks: Input embedding matrix

dim = hidden size

Neural Network

- Defined by architecture and weights
- Architecture predefined
  - Number of hidden layers
  - Number of “neurons” per layer
  - Type of layer: Connectivity pattern, dropout, normalization, ...
- Weights & biases
  - Randomly initialized
  - Updated by training iterations (“learning”)
  - Loss/objective determines how

Neural Network

- Defined by architecture and weights
- Architecture predefined
  - Number of hidden layers
  - Number of “neurons” per layer
  - Type of layer: Connectivity pattern, dropout, normalization, ...
- Weights & biases
  - Randomly initialized
  - Updated by training iterations (“learning”)
  - Loss/objective determines how

Hyperparameters: Predefined settings (number of layers, hidden size, dropout probability, …)

Parameters: Learnable weights & biases

Neural Network

- Defined by architecture and weights
- Architecture predefined
  - Number of hidden layers
  - Number of "neurons" per layer
  - Type of layer: Connectivity pattern, dropout, normalization, ...
- Weights & biases
  - Randomly initialized
  - Updated by training iterations ("learning")
  - Loss/objective determines how

Hyperparameters: Predefined settings (number of layers, hidden size, dropout probability, …)

Parameters: Learnable weights & biases

This is where the knowledge is stored!

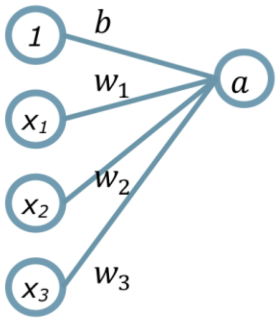
Linear Layer

- Also called “dense layer” or “fully-connected layer”

$$y = Wx + b$$

- x: input
- y: output
- W: weights
- b: bias

A Single Neuron (i.e. a Perceptron)



$a = g(b + x_1 w_1 + x_2 w_2 + x_3 w_3)$

Let's vectorize: $a = g(b + \mathbf{w}^T \mathbf{x})$

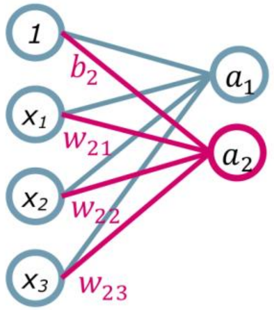
- $\mathbf{w} = [w_1 \quad w_2 \quad w_3]^T$ (weights)
- $\mathbf{x} = [x_1 \quad x_2 \quad x_3]^T$ (input)

Data flows left-to-right:

A **feed-forward network**

Dr. Donnacha Daly
donnacha.daly@hslu.ch

Add a Second Neuron



$$a_1 = g(b_1 + x_1 w_{11} + x_2 w_{12} + x_3 w_{13})$$

$$a_2 = g(b_2 + x_1 w_{21} + x_2 w_{22} + x_3 w_{23})$$

Vectorize again: $\mathbf{a} = \mathbf{g}(\mathbf{b} + \mathbf{W}\mathbf{x})$

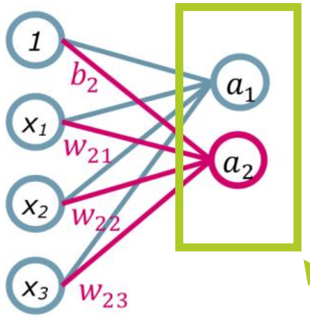
- $\mathbf{a} = [a_1 \quad a_2]^T$ (output)
- $\mathbf{b} = [b_1 \quad b_2]^T$ (bias)
- $\mathbf{W}|_{ij} = w_{ij}$ (weights)
- $\mathbf{g}$ is element-wise function g

Dr. Donnacha Daly
donnacha.daly@hslu.ch

Output

- Output representations (usually vectors)
- Decisions in neural networks: Classifier
  - Map from output representation to decision space
    - Binary decision: 1 or 2 dimensions
    - ImageNet: 1000 classes → 1000 dimensions
    - Words: vocab size = #dimensions
      - This is the inverse mapping from input embeddings
      - Can use the same matrix (but transposed)
  - Simple classifier: Linear layer → softmax (gives probabilities over classes)
  - Common: Linear layer → non-linearity (e.g. ReLU) → linear layer → softmax

Add a Second Neuron



$$a_1 = g(b_1 + x_1 w_{11} + x_2 w_{12} + x_3 w_{13})$$

$$a_2 = g(b_2 + x_1 w_{21} + x_2 w_{22} + x_3 w_{23})$$

Vectorize again: $\mathbf{a} = \mathbf{g}(\mathbf{b} + \mathbf{W}\mathbf{x})$

- $\mathbf{a} = [a_1 \quad a_2]^T$ (output)
- $\mathbf{b} = [b_1 \quad b_2]^T$ (bias)
- $\mathbf{W}|_{ij} = w_{ij}$ (weights)
- $\mathbf{g}$ is element-wise function g

Output representation/vector

Neural Network Losses

- Mean squared error (MSE) loss
- Cross-entropy loss
- Margin loss
- Contrastive loss

Backpropagation

- Important to understand concept
- Store activations in forward pass
- Loss is scalar
- Backward pass computes updates for each NN weight to make a better prediction (smaller loss) next time
- Optimization decides how strongly to update weights

Optimization

- Basic idea: stochastic gradient descent (SGD)
    - Update in the direction of the gradient of the current batch
- Momentum: When the loss is large, increase the update step size
    - Vice versa when the loss is small
    - Optimizers: e.g. SGD with (Nesterov) momentum, Adam
- Adam is the most popular optimization algorithm
    - Several adaptations exist, e.g. AdamW (with weight decay)