

MACHINE LEARNING IN BIOINFORMATICS

ANN ARCHITECTURES

Philipp Benner
philipp.benner@bam.de

VP.1 - eScience
Federal Institute of Materials Research and Testing (BAM)

February 8, 2026

■ Part of this lecture:

- ▶ Embeddings
- ▶ Auto-encoders
- ▶ Convolutions on images and graphs
- ▶ Attention mechanism

■ Other important architectures not covered here:

- ▶ Generative adversarial networks (GANs)
- ▶ Deep tensor factorization
- ▶ Recurrent neural networks (LSTM/GRU)

EMBEDDINGS

ONE-HOT ENCODING

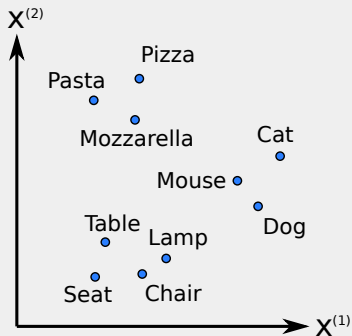
- Assume we want to work with categorical data, e.g.
 - ▶ DNA or protein sequences
 - ▶ Text (vectors of words)
- Traditionally, we would use one-hot encoding, which use a dimension for each category
- For example, a DNA sequence **ACGTTA** could be represented as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

ONE-HOT ENCODING

- One-hot encodings have several problems
- For data with many categories, we obtain very high-dimensional feature vectors, e.g.
 - ▶ Protein sequences would already require 20 dimensions
 - ▶ Text would require one dimension per word type
- One-hot encodings should be used for purely categorical data, where we have no similarity between categories
- However, for most data we have certain similarities, e.g.
 - ▶ Amino acid replacements have different effects, which suggests that some amino acids are more similar in function than others

EMBEDDINGS



- We assign each category k a feature vector $x_k \in \mathbb{R}^p$
- The representations x_k are randomly initialized and optimized during training
- After training we often observe that similar categories cluster together

CONVOLUTIONAL NEURAL NETWORKS FOR IMAGES

IMAGE PATTERN DETECTION

Conway's Game of Life - glider gun:

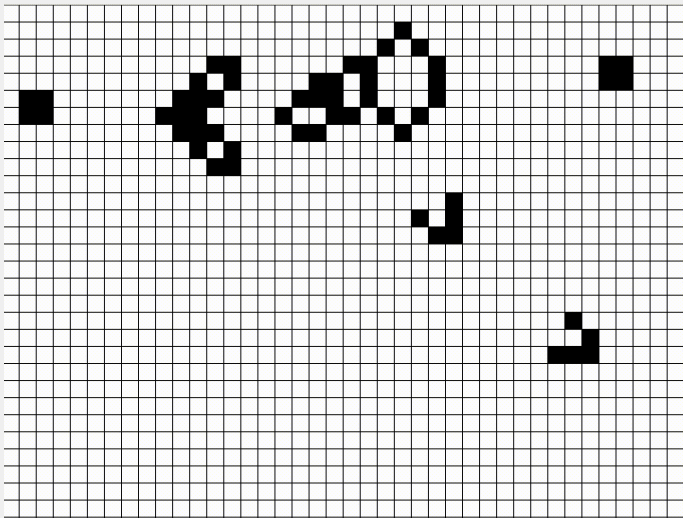
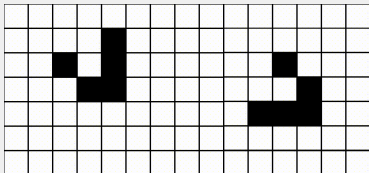


IMAGE PATTERN DETECTION

Glider gun detector:



Glider pattern:

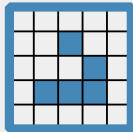


IMAGE PATTERN DETECTION

Glider gun detector:

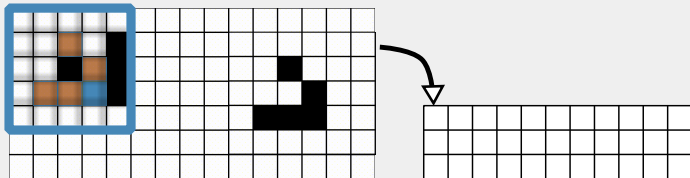


IMAGE PATTERN DETECTION

Glider gun detector:

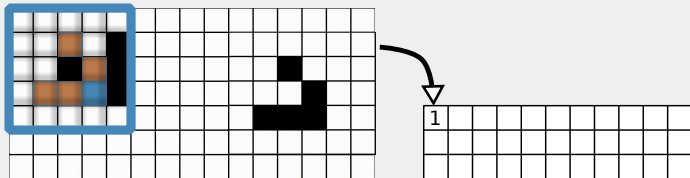


IMAGE PATTERN DETECTION

Glider gun detector:

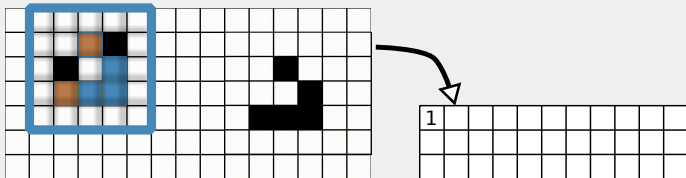


IMAGE PATTERN DETECTION

Glider gun detector:

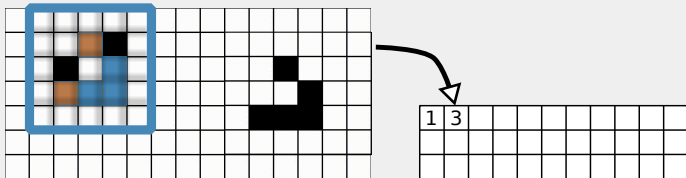


IMAGE PATTERN DETECTION

Glider gun detector:

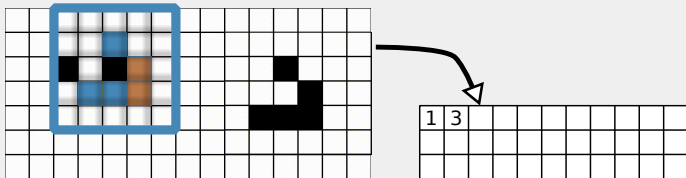


IMAGE PATTERN DETECTION

Glider gun detector:

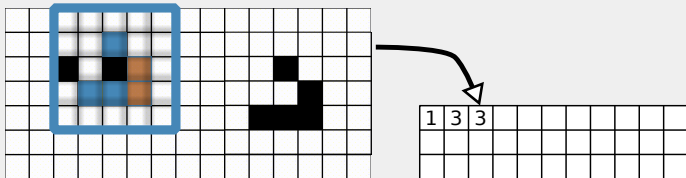


IMAGE PATTERN DETECTION

Glider gun detector:

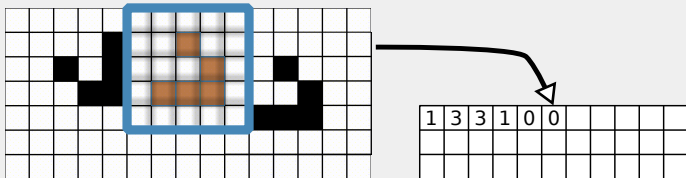


IMAGE PATTERN DETECTION

Glider gun detector:

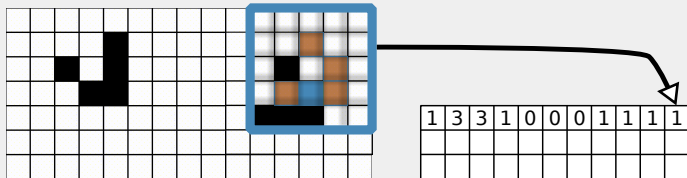


IMAGE PATTERN DETECTION

Glider gun detector:

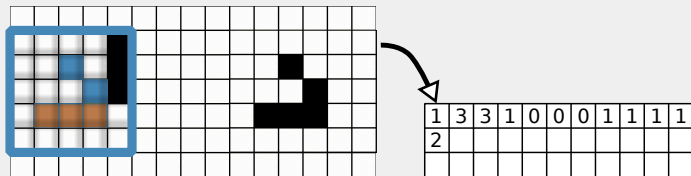


IMAGE PATTERN DETECTION

Glider gun detector:

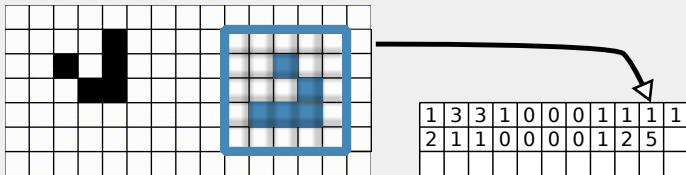


IMAGE PATTERN DETECTION

Glider gun detector:

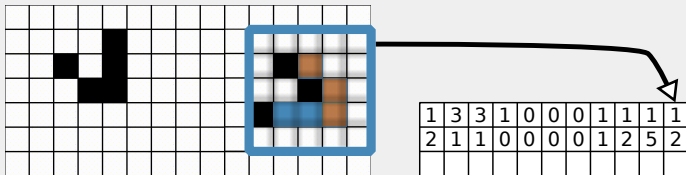


IMAGE PATTERN DETECTION

Glider gun detector:

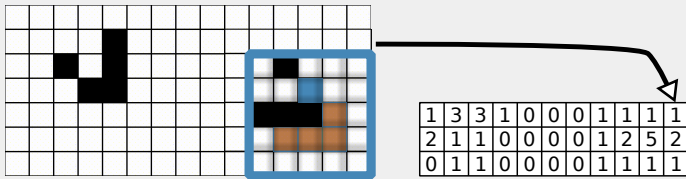


IMAGE PATTERN DETECTION

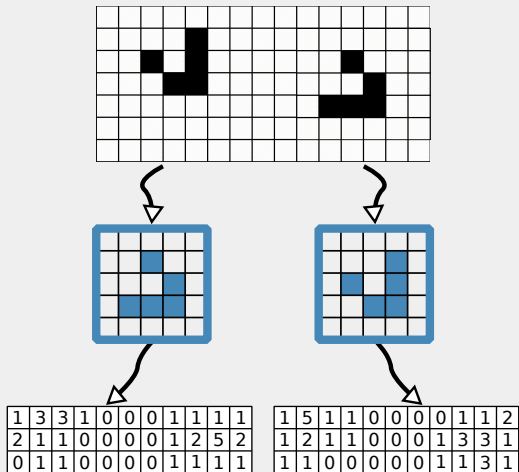
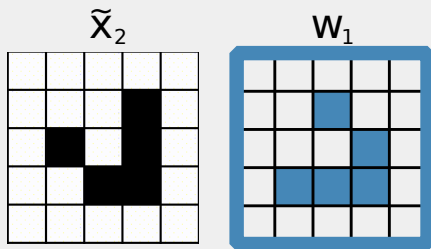


IMAGE PATTERN DETECTION - CONVOLUTION



- Let $\tilde{x}_j \in \mathbb{R}^r$ denote the j -th image patch of image X , e.g.

$$\tilde{x}_2 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, \dots)^\top$$

- Let $w_k \in \mathbb{R}^r$ denote the k -th glider pattern or **kernel**, e.g.

$$w_1 = (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, \dots)^\top$$

- The output y_j at position j is given by

$$y_j = \tilde{x}_j^\top w_k$$

IMAGE PATTERN DETECTION - CONVOLUTION

- Let $\tilde{X} \in \mathbb{R}^{q \times r}$ denote the matrix of q image patches from image X and $W \in \mathbb{R}^{r \times p}$ the matrix of kernels, i.e.

$$\tilde{X} = \begin{bmatrix} \tilde{X}_1^T \\ \tilde{X}_2^T \\ \vdots \\ \tilde{X}_q^T \end{bmatrix}, \quad W = [w_1, w_2, \dots, w_p]$$

- The result $Y \in \mathbb{R}^{q \times p}$ of applying the kernel matrix W to image X is given by

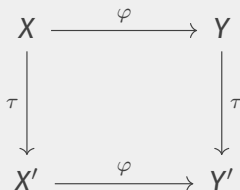
$$Y = \tilde{X}W = X * W$$

where " $*$ " is called *convolution*¹

¹Technically, we are computing a cross-correlation and not a convolution

EQUIVARIANCE

- Let X be an image and W a filter
- $\varphi(X) = X * W$ denotes a convolution with W
- $\tau(X)$ is a translation of an image
- The following diagram shows that φ is *equivariant* with respect to τ

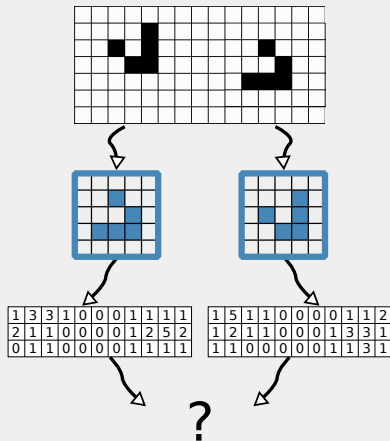


- Exception are the borders of images

WHY EQUIVARIANCE AND NOT INVARIANCE?



IMAGE PATTERN DETECTION



- Stack multiple convolutions
- Case 1: All images have the same dimension
⇒ Feed into neural network
- Case 2: Images have variable dimension
⇒ Compute summary statistics (*global pooling*)
 - ▶ mean
 - ▶ max

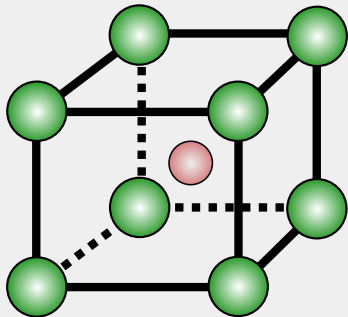
POOLING LAYERS

- Applying kernels leads to *translation-equivariant* features
- Pooling layers add (limited amount of) translation invariance
- Average pooling
- Max pooling

1	3	3	1	1	0	0	1	1
2	1	1	0	0	2	1	1	2
0	1	1	0	3	3	0	1	6
1	5	1	1	0	3	1	0	1
1	2	1	1	2	4	0	1	3
1	1	0	0	3	0	1	1	1

3	3	6
5	4	3

GRAPH CONVOLUTIONAL NEURAL NETWORKS (GCNNs)



GRAPH CONVOLUTIONS

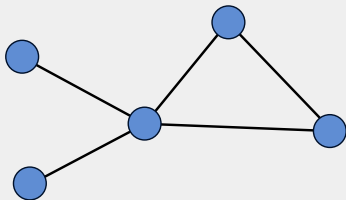
- Convolutions are not only restricted to image and time-series data
- Graph convolutions are used to **model the interaction between nodes**
- Let $G = (N, E)$ denote a graph with nodes N and edges E
- How could we implement a convolution of G with a weight matrix W ?
- The result of a convolution is again a graph², i.e.

$$G' = G * W$$

²Remember that convolution on images also returns an image

GRAPH CONVOLUTIONS

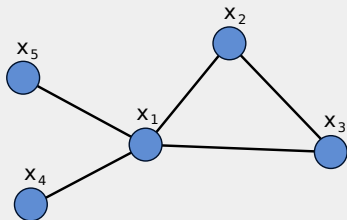
- Graph G with 5 nodes and 5 edges:



- We assign a feature vector $x_i \in \mathbb{R}^p$ to the i -th node
- The feature vector can depend on the type of the node
- Nodes of the same type might share the same feature vector

GRAPH CONVOLUTIONS

- Graph G with 5 nodes and 5 edges:



- We assign a feature vector $x_i \in \mathbb{R}^p$ to the i -th node
- The feature vector can depend on the type of the node
- Nodes of the same type might share the same feature vector

GRAPH CONVOLUTIONS

- Let $A = (a_{ij})_{ij} \in \mathbb{R}^{k \times k}$ denote the adjacency matrix of a graph with k nodes
- The strength of the connection between node i and j is given by a_{ij}
- Self-connections $a_{ii} \neq 0$ allow to incorporate the features of the nodes itself
- The convolution operation updates the feature vector of node i by summing over the contributions of all neighbor nodes, i.e.

$$x'_i = \sigma \left(\sum_{j \neq i} a_{ij} W x_j \right)$$

where $W \in \mathbb{R}^{p \times p}$ and σ is the activation function³

³Graph convolutions are *permutation equivariant*

GRAPH CONVOLUTIONS

- For the full graph we obtain

$$\underbrace{X'}_{k \times p} = \sigma(\underbrace{A}_{k \times k} \underbrace{X}_{k \times p} \underbrace{W^T}_{p \times p})$$

where $X \in \mathbb{R}^{k \times p}$ is the matrix of k feature vectors

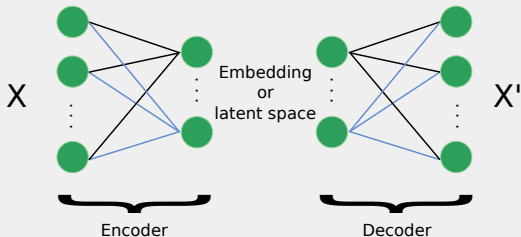
- Note that the weight matrix W does not depend on the size and connectivity of the graph
- W can be applied to multiple graphs and optimized during training of the graph convolutional neural network (GCNN)
- GCNNs typically apply multiple convolutions and afterwards compute summary statistics of the feature vectors, the result can then be used in a conventional neural network

³Many extensions and generalizations exist
[Battaglia et al., 2018, Dwivedi et al., 2020]

AUTO-ENCODERS

AUTO-ENCODERS

- Embeddings implicitly group categories by their similarity
- Auto-encoders [Kramer, 1991] learn hidden representations for non-categorical data:



- During training, the error between X and X' is minimized
- The embedding or latent space should have lower dimension than the input space

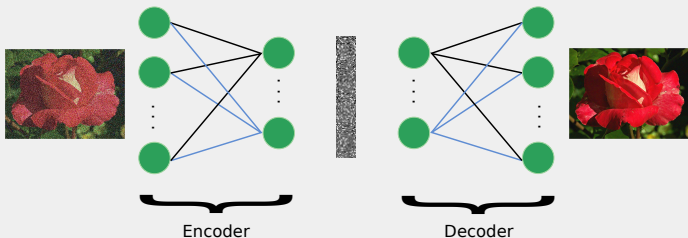
AUTO-ENCODERS - FORMAL DEFINITION

- The encoder $f_W : \mathbb{R}^p \rightarrow \mathbb{R}^q$ is a neural network with weights W that maps a sample $x \in \mathbb{R}^p$ into a q -dimensional feature space
- The decoder $g_V : \mathbb{R}^q \rightarrow \mathbb{R}^p$ takes a point in feature space and maps it back to input space
- Given a set of training points $\{x_i\}_i$ we train the auto-encoder by minimizing the error between the input and output of the network, i.e.

$$W, V = \arg \min_{W, V} \|x_i - (g_V \circ f_W)(x_i)\|_2^2$$

AUTO-ENCODERS - PURPOSE

- Dimensionality reduction and visualization (similar to PCA and t-SNE)
- Compression to most important features (encoder output)
- Denoising and image restoration (decoder output), by adding noise to images before sending it to the encoder

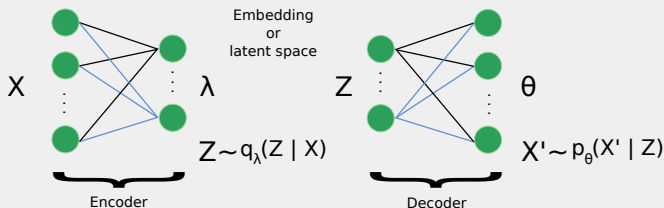


- Clustering and outlier detection on the latent space

VARIATIONAL AUTO-ENCODERS (VAEs)

- Can we use auto-encoders for generating data? I.e. we could sample a point from the latent space and decode the corresponding data point
- Practice has shown that this approach does not work
- The latent space has many *holes* where the decoder generates garbage
- Variational auto-encoders (VAEs) [Kingma and Welling, 2013] are a probabilistic formulation of auto-encoders, that regularize the latent space

VARIATIONAL AUTO-ENCODERS (VAEs)



- Instead of learning latent representations directly, VAEs learn the parameters of given distributions
- The encoder learns the parameters λ of the distribution $q_\lambda(z | x)$
- The decoder learns the parameters θ of the distribution $p_\theta(x | z)$
- Training is more complicated, i.e. minimize the KL-divergence

ATTENTION

SEQUENTIAL DATA

Output: Translated word 1 Translated word 2 Translated word n

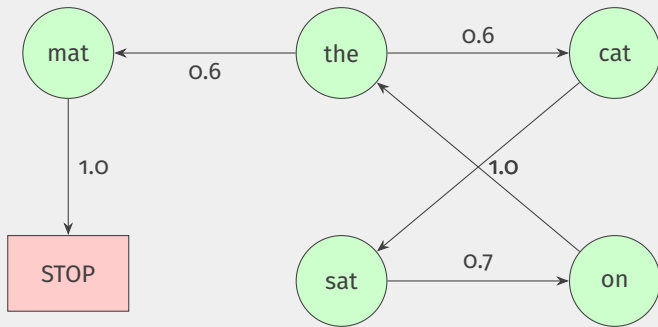
y_1 y_2 y_n

Input / query: x_1 x_2 ... x_n

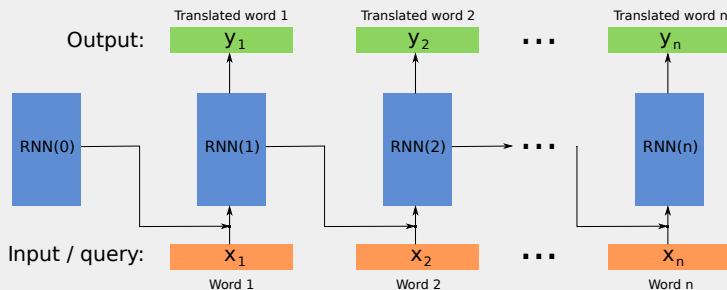
Word 1 Word 2 Word n

- Translations require special architectures that can deal with:
 - ▶ Variable sentence lengths, i.e. variable n
 - ▶ Long-range dependencies

MARKOV MODEL FOR NEXT WORD PREDICTIONS



RECURRENT NEURAL NETWORKS

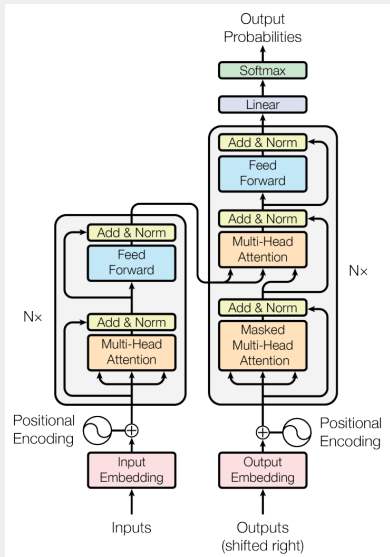


- Recurrent neural networks (RNNs) are sequentially applied to each input x_i
- The architecture and weights are the same for all steps i.e. for $RNN(0)$, $RNN(1)$, \dots , $RNN(n)$
- At each step i , RNNs take the input x_i and the state of the previous step $i - 1$ as input

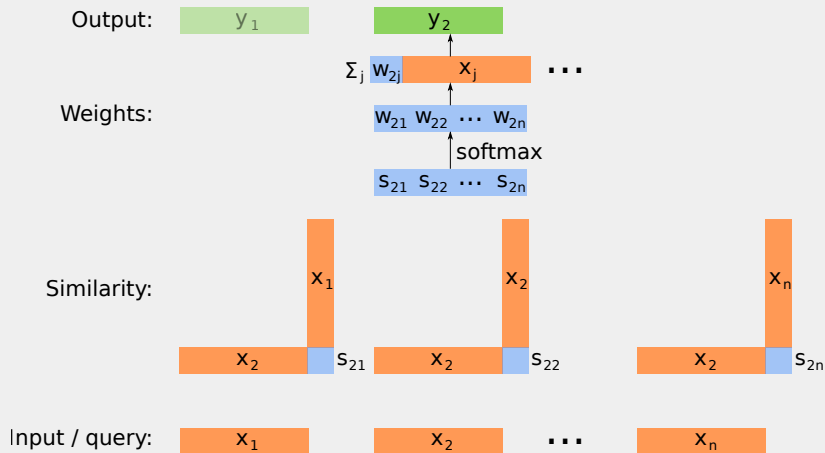
RECURRENT NEURAL NETWORKS

- Recurrent neural networks (RNNs) were traditionally used for sequence data and to model long-range interactions
- Traditional RNNs have extreme vanishing / exploding gradient problem
- Long-short term memory (LSTM)
[Hochreiter and Schmidhuber, 1997] solved this problem, but is still difficult to train
 - ▶ On a large input sequence it corresponds to a very deep neural network
 - ▶ Transfer learning never worked for LSTM
- Transformers with attention layer
[Bahdanau et al., 2014, Vaswani et al., 2017] are an alternative to RNNs and show better performance

TRANSFORMERS



SELF-ATTENTION LAYER



SELF-ATTENTION LAYER

- Let $X = [x_1^\top, x_2^\top, \dots, x_n^\top] \in \mathbb{R}^{n \times p}$ denote the data matrix, i.e. the embeddings of the input sequence
- The self-attention layer computes the i -th output $y_i \in \mathbb{R}^p$ as follows:

$$s_i = x_i^\top X^\top$$

$$w_i = \text{softmax}(s_i) = \left(\frac{e^{s_{ij}}}{\sum_{k=1}^n e^{s_{ik}}} \right)_{j=1,2,\dots,n}$$

$$y_i = w_i X$$

- The self-attention layer computes the entire output $Y \in \mathbb{R}^{n \times p}$ as follows:

$$Y = \text{softmax}(\underbrace{XX^\top}_{\text{kernel}}) X$$

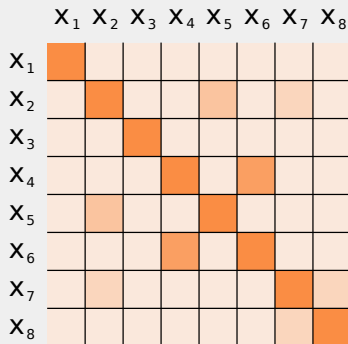
where the softmax is applied independently to each row

SELF-ATTENTION MAPS

- The self-attention map is defined as

$$A = \text{softmax}(XX^T)$$

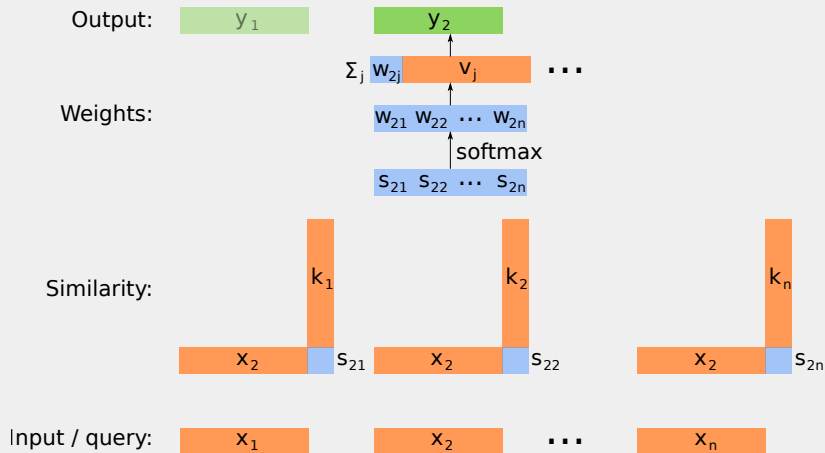
- The matrix A can be visualized to inspect attention



ATTENTION LAYER

- Except for the embeddings $(x_i)_i$, the self-attention layer has no parameters that can be optimized
- For self-attention, the input sequence focuses attention on the input sequence itself and a linear combination of the input sequence x_1, x_2, \dots, x_n is returned
- The attention layer is a generalization of the self-attention layer, where
 - ▶ attention is focused on a set of m keys k_1, \dots, k_m , with $k_j \in \mathbb{R}^p$
 - ▶ a linear combination of m values v_1, \dots, v_m is returned, where $v_j \in \mathbb{R}^p$
- The attention layer implements a differentiable data retrieval method for a database of m keys and values

ATTENTION LAYER



ATTENTION LAYER

- Let $K \in \mathbb{R}^{m \times p}$ and $V \in \mathbb{R}^{m \times p}$ denote a set of m keys and values
- The attention layer computes the entire output $Y \in \mathbb{R}^{n \times p}$ as follows:




$$Y = \text{softmax}(XK^T)V$$

- Remarks:
 - ▶ There exist several variants of the attention layer
 - ▶ Transformers use a both attention and self-attention layers
 - ▶ The sequential order is lost for self-attention and attention layers
 - ▶ Transformers use another encoding for restoring relative word positions
 - ▶ Multiple *attention heads* are commonly used





TRANSFER LEARNING

- Some of the most successful deep learning models:
 - ▶ Protein folding: AlphaFold [Jumper et al., 2021]
 - ▶ Vision: GoogLeNet [Szegedy et al., 2015], Squeeze-and-Excitation Networks (SENet) [Hu et al., 2018]
 - ▶ Translation: BERT [Devlin et al., 2018], Text-to-Text Transfer Transformer (T5) [Raffel et al., 2019]
- Training T5 (11B-parameter variant) costs well above \$1.3 million [Sharir et al., 2020]
- True deep neural networks are not affordable for most academics
- Transfer learning allows to adapt pre-trained models





REFERENCES I

-  BAHDANAU, D., CHO, K., AND BENGIO, Y. (2014).
NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE.
arXiv preprint arXiv:1409.0473.
-  BATTAGLIA, P. W., HAMRICK, J. B., BAPST, V., SANCHEZ-GONZALEZ, A., ZAMBALDI, V., MALINOWSKI, M., TACCHETTI, A., RAPOSO, D., SANTORO, A., FAULKNER, R., ET AL. (2018).
RELATIONAL INDUCTIVE BIASES, DEEP LEARNING, AND GRAPH NETWORKS.
arXiv preprint arXiv:1806.01261.
-  DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. (2018).
BERT: PRE-TRAINING OF DEEP BIDIRECTIONAL TRANSFORMERS FOR LANGUAGE UNDERSTANDING.
arXiv preprint arXiv:1810.04805.



REFERENCES II

-  DWIVEDI, V. P., JOSHI, C. K., LAURENT, T., BENGIO, Y., AND BRESSON, X. (2020).
BENCHMARKING GRAPH NEURAL NETWORKS.
arXiv preprint arXiv:2003.00982.
-  HOCHREITER, S. AND SCHMIDHUBER, J. (1997).
LONG SHORT-TERM MEMORY.
Neural computation, 9(8):1735–1780.
-  HU, J., SHEN, L., AND SUN, G. (2018).
SQUEEZE-AND-EXCITATION NETWORKS.
In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7132–7141.
-  JUMPER, J., EVANS, R., PRITZEL, A., GREEN, T., FIGURNOV, M.,
RONNEBERGER, O., TUNYASUVUNAKOOL, K., BATES, R., ŽÍDEK, A.,
POTAPENKO, A., ET AL. (2021).
HIGHLY ACCURATE PROTEIN STRUCTURE PREDICTION WITH ALPHAFOLD.
Nature, 596(7873):583–589.

REFERENCES III

-  KINGMA, D. P. AND WELLING, M. (2013).
AUTO-ENCODING VARIATIONAL BAYES.
arXiv preprint arXiv:1312.6114.
-  KRAMER, M. A. (1991).
NONLINEAR PRINCIPAL COMPONENT ANALYSIS USING AUTOASSOCIATIVE NEURAL NETWORKS.
AIChE journal, 37(2):233–243.
-  RAFFEL, C., SHAZEER, N., ROBERTS, A., LEE, K., NARANG, S., MATENA, M., ZHOU, Y., LI, W., AND LIU, P. J. (2019).
EXPLORING THE LIMITS OF TRANSFER LEARNING WITH A UNIFIED TEXT-TO-TEXT TRANSFORMER.
arXiv preprint arXiv:1910.10683.
-  SHARIR, O., PELEG, B., AND SHOHAM, Y. (2020).
THE COST OF TRAINING NLP MODELS: A CONCISE OVERVIEW.
arXiv preprint arXiv:2004.08900.

REFERENCES IV

-  SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., AND RABINOVICH, A. (2015).
GOING DEEPER WITH CONVOLUTIONS.
In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9.
-  VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł., AND POLOSUKHIN, I. (2017).
ATTENTION IS ALL YOU NEED.
Advances in neural information processing systems, 30.