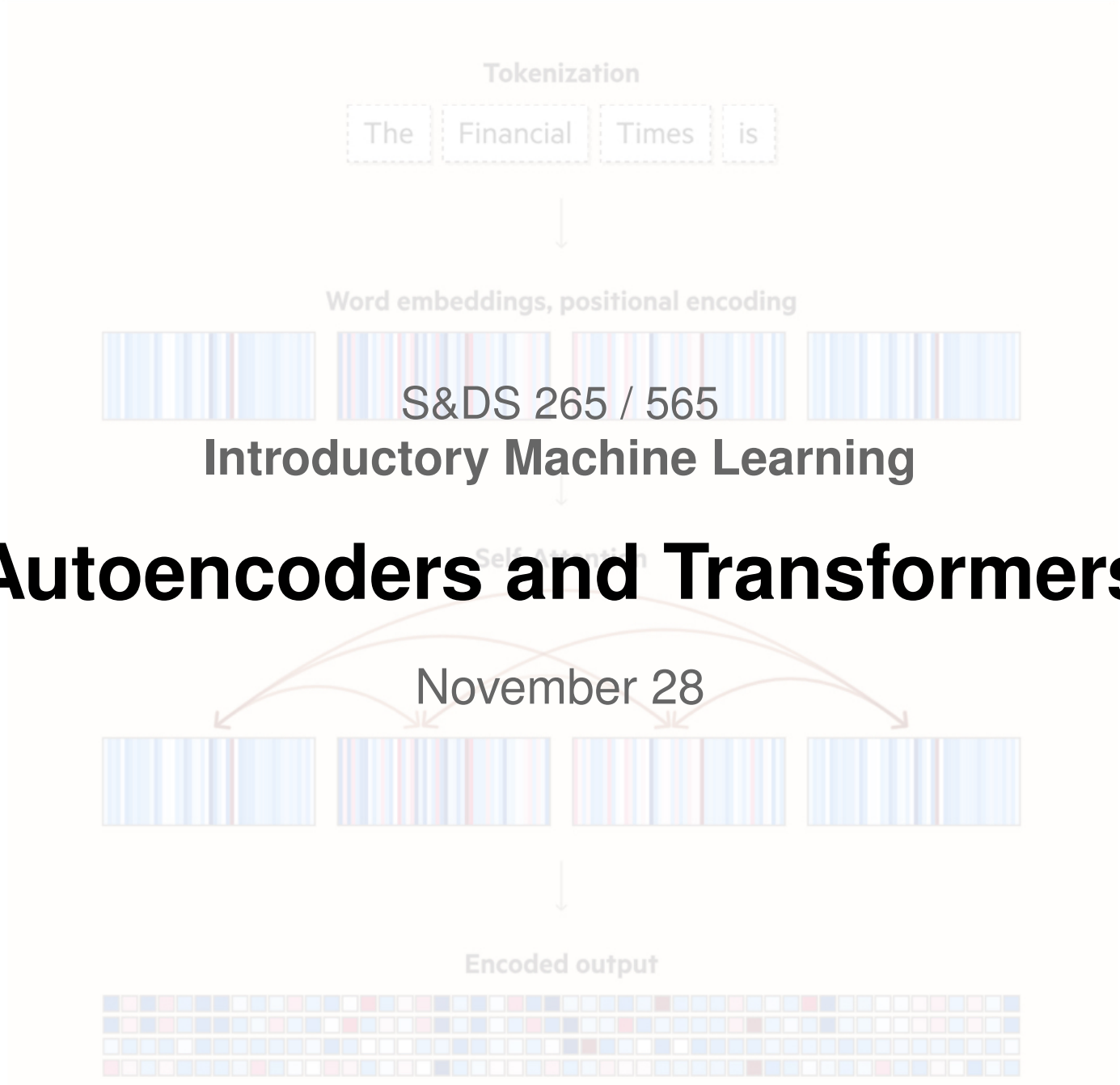


S&DS 265 / 565

Introductory Machine Learning

Autoencoders and Transformers

November 28

Yale

While you were feasting...

TECH

OpenAI researchers warned board of AI breakthrough ahead of CEO ouster

PUBLISHED WED, NOV 22 2023•6:09 PM EST | UPDATED THU, NOV 23 2023•1:53 AM EST

REUTERS

WATCH LIVE

Sam Altman, chief executive officer of OpenAI, during a fireside chat organized by Softbank Ventures Asia in Seoul, South Korea, on Friday, June 9, 2023.

SeongJoon Cho | Bloomberg | Getty Images

While you were feasting...

The maker of ChatGPT had made progress on Q* (pronounced Q-Star), which some internally believe could be a breakthrough in the startup's search for superintelligence, also known as artificial general intelligence (AGI), one of the people told Reuters. OpenAI defines AGI as AI systems that are smarter than humans.

Given vast computing resources, the new model was able to solve certain mathematical problems, the person said on condition of anonymity because they were not authorized to speak on behalf of the company. Though only performing math on the level of grade-school students, acing such tests made researchers very optimistic about Q*'s future success, the source said. Reuters could not independently verify the capabilities of Q* claimed by the researchers.

---

https://www.cnbc.com/2023/11/22/sam-altmans-ouster-at-openai-precipitated-by-letter-to-board-about-ai-breakthrough-sources-tell-reuters.html

Some reminders

- Quiz 5 before break
- Quiz 6 on December 7
- Assn 5 out; due Thursday
- Final exam: Friday Dec 15, 2pm in SSS 114
- Practice exams are posted
- Review sessions TBA

---

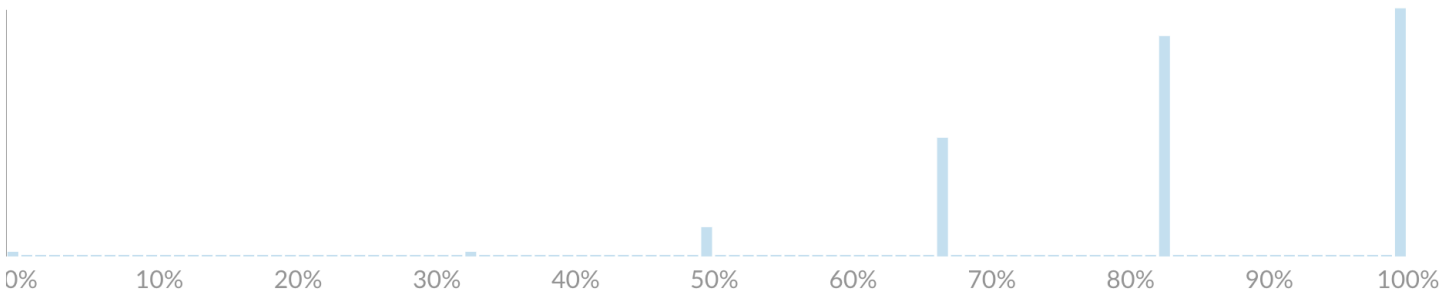
https://registrar.yale.edu/general-information/final-exams

Quiz 5

Quiz Summary

Section Filter ▾ | Student Analysis | Item Analysis

Average Score	High Score	Low Score	Standard Deviation	Average Time
85%	100%	0%	0.97	14:17



Requesting your help

- To make future iML classes run more smoothly, we are experimenting with a “ticketing” system
- Request: *For remainder of the semester, please make any requests (late assignments, regrades, etc.) by emailing* `sds265@yale.edu`
- This will allow us to test the system and process your requests more efficiently—thanks!

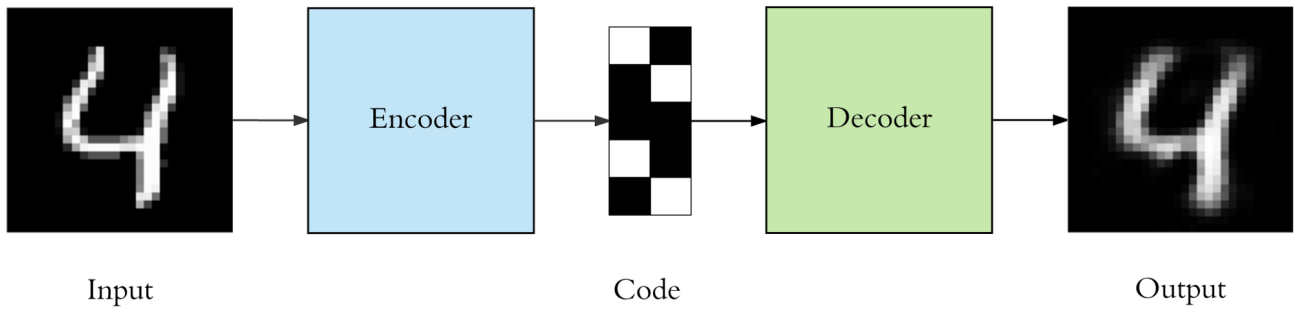
For today

- Autoencoders
- Attention
- Transformers—high level

Autoencoders

- Unsupervised learning methods
- Squeeze high dimensional data through a “bottleneck” of lower dimension
- Train to minimize reconstruction error

Autoencoders

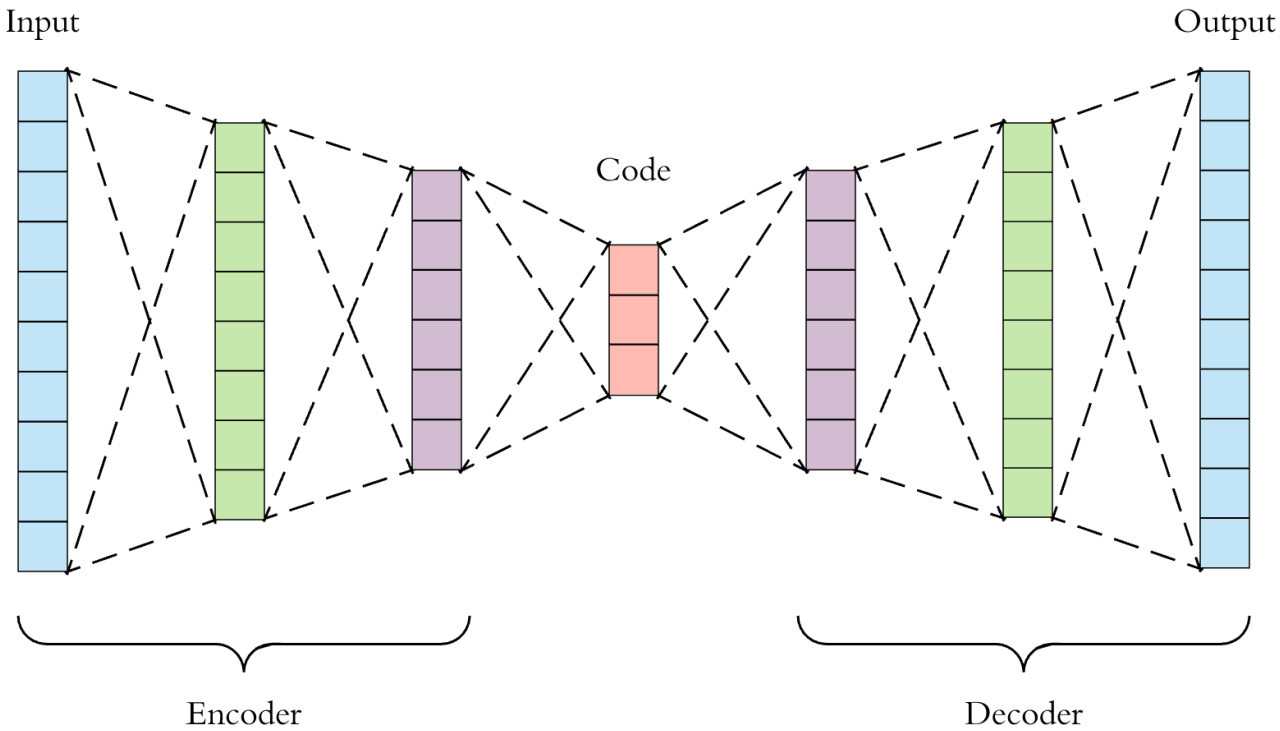


https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

Important aspects

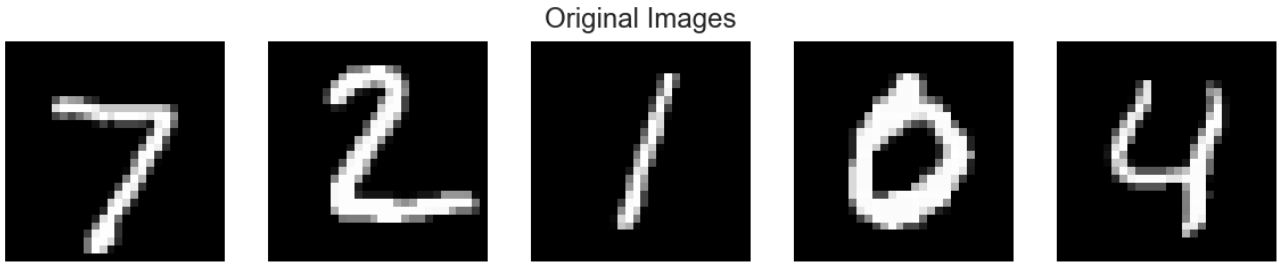
- Unsupervised: No labels used, discovers useful features of input
- Compression: Code reduces dimension of data
- Lossy: Input won't be reconstructed exactly
- Trained: The compression algorithm is learned for specific data

Deep architecture



https://github.com/ardendertat/Applied-Deep-Learning-with-Keras

MNIST data



$$28 \times 28 = 784 = D$$

Training

- The loss function is the squared error between the input and output—typically just summed over all pixels
- Weights in each layer are trained with SGD and backpropagation
- Training is organized in “epochs”—each epoch is a pass through the training data, in random minibatches
- Multiple epochs (passes through the data) are often required until the model parameters have converged
- Details of the SGD can make a big difference

Adam optimizer

- Variant of stochastic gradient descent where separate learning rate (step size) is maintained for each network weight (parameter)
- Each step size adapted as learning progresses based on moments of the derivatives

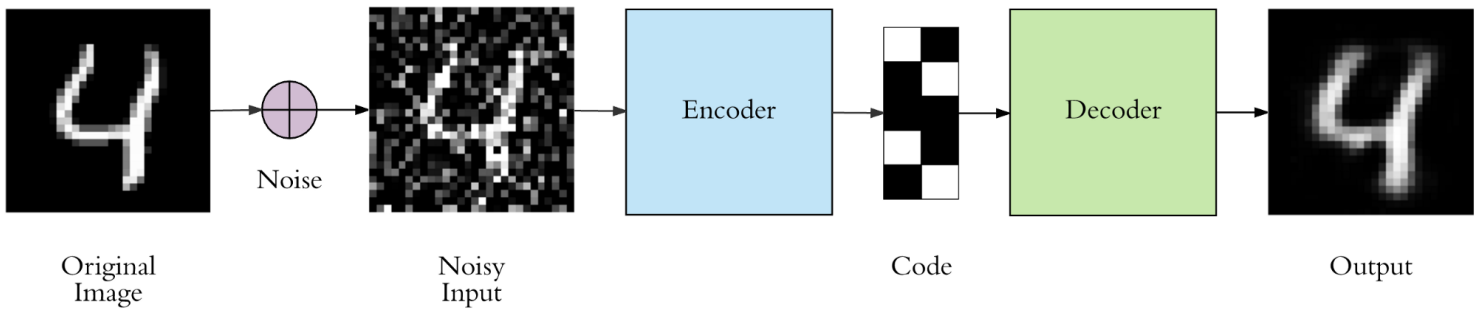
---

"Adam: A method for stochastic optimization," D. Kingma and J. Ba, https://arxiv.org/abs/1412.6980

Implementation using Keras

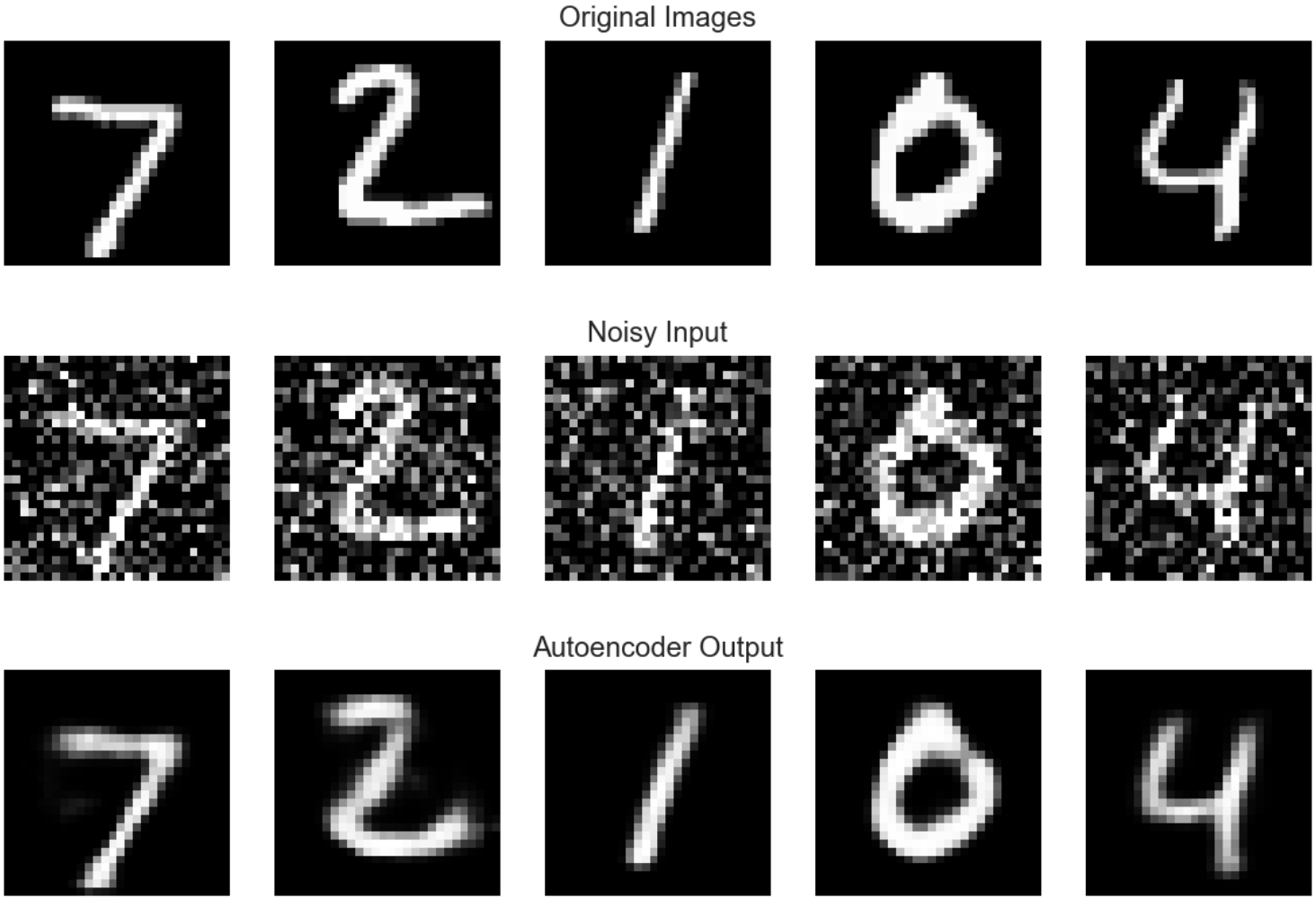
```
input_size = 784
hidden_size = 128
code_size = 32

input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu')(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)

autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=5)
```

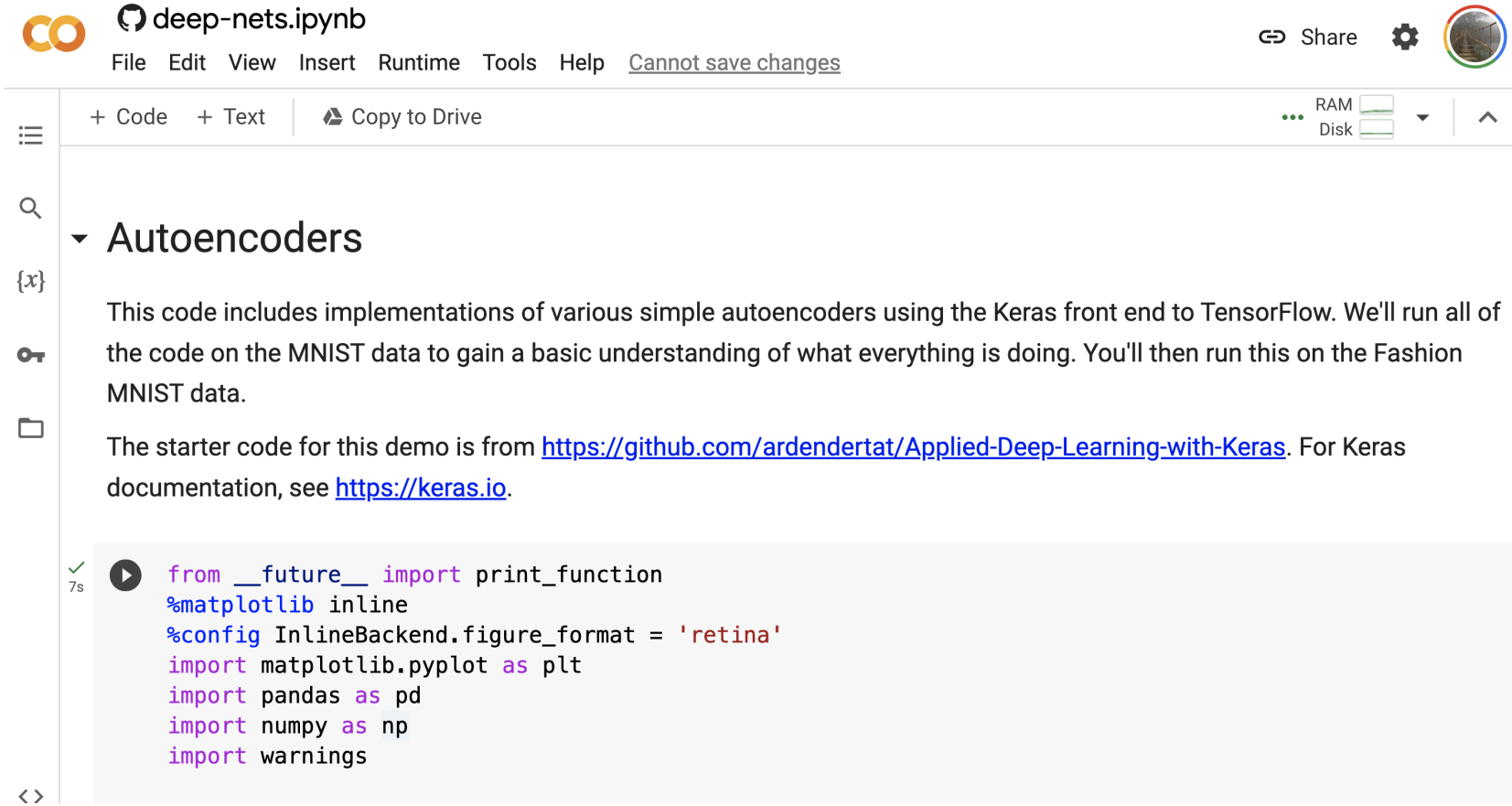
Variant: Denoising autoencoder



- Feed in noisy data
- Train to match to denoised data

Example result on MNIST




deep-nets.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM
Disk
Autoencoders
This code includes implementations of various simple autoencoders using the Keras front end to TensorFlow. We'll run all of the code on the MNIST data to gain a basic understanding of what everything is doing. You'll then run this on the Fashion MNIST data.
The starter code for this demo is from https://github.com/ardendertat/Applied-Deep-Learning-with-Keras. For Keras documentation, see https://keras.io.
7s
from __future__ import print_function
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import warnings

Summary of Autoencoders

- Autoencoders compress the input and then reconstruct it
- Bottleneck forces extraction of useful features
- Will overfit and “memorize” the data
- Overfitting mitigated by denoising autoencoders

Next: High-level overview of transformers

- We'll treat transformers at a high level
- Key notion: Attention
- Main intuition illustrated with word embeddings

Artificial Intelligence

Generative AI exists because of the transformer

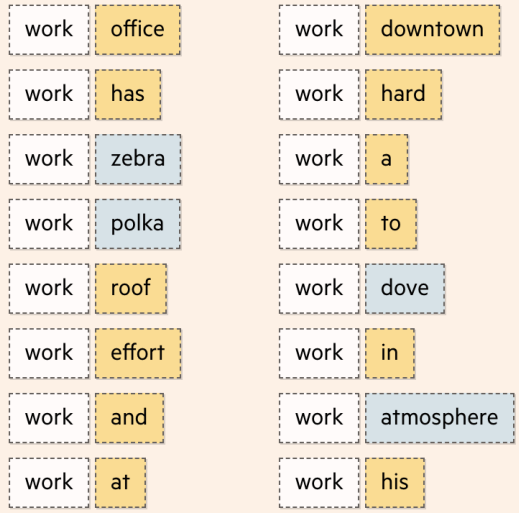
This is how it works

By **Visual Storytelling Team** and **Madhumita Murgia** in London SEPTEMBER 12 2023

https://ig.ft.com/generative-ai/

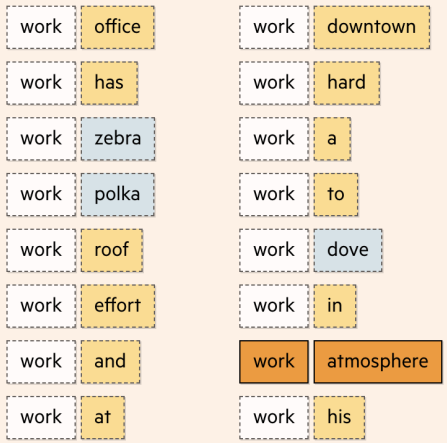
Starting point: Word embeddings

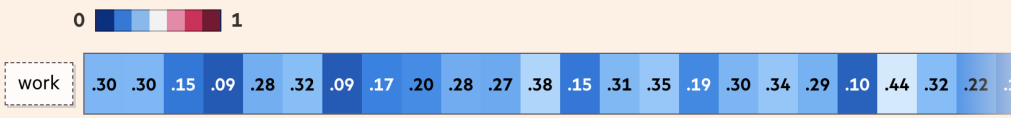
Eventually, we end up with a huge set of the words found **alongside** **work** in the training data, as well as those that **weren't** found near it.



Starting point: Word embeddings

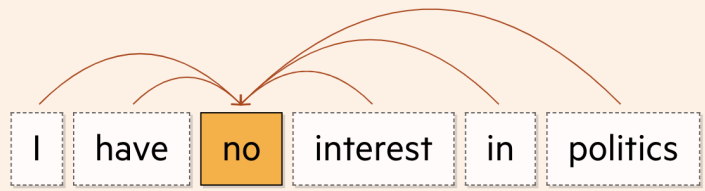
As the model **processes** this set of words, it produces a vector — or list of values — and adjusts it based on each word's proximity to **work** in the training data. This vector is known as a word embedding.





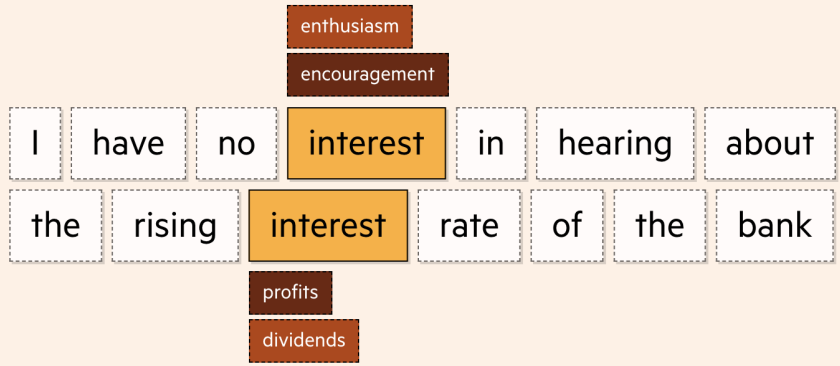
Introducing context: Attention

Self-attention looks at each **token** in a body of text and decides which others are most important to understanding its meaning.



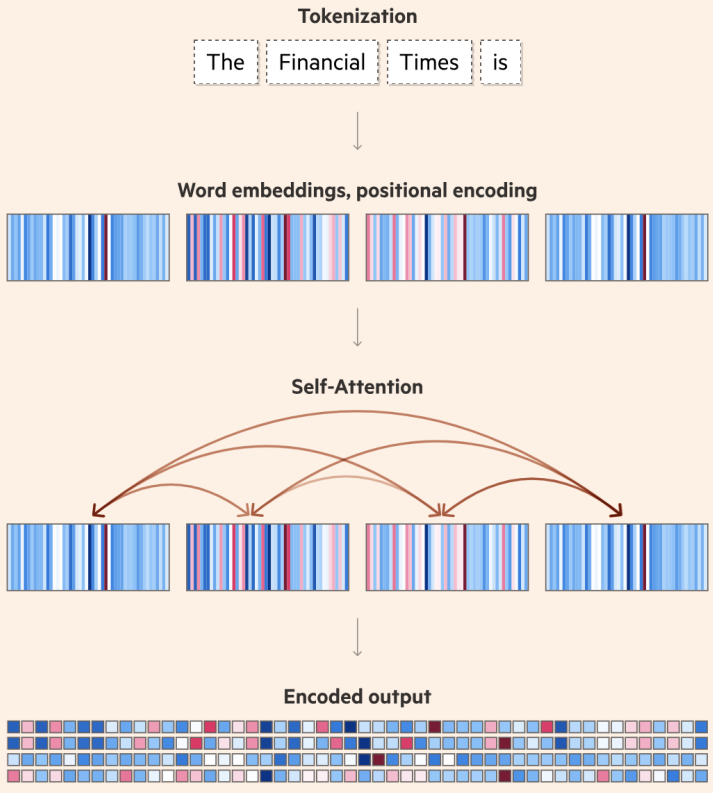
Introducing context: Attention

This functionality is crucial for advanced text generation. Without it, words that can be interchangeable in some contexts but not others can be used incorrectly.



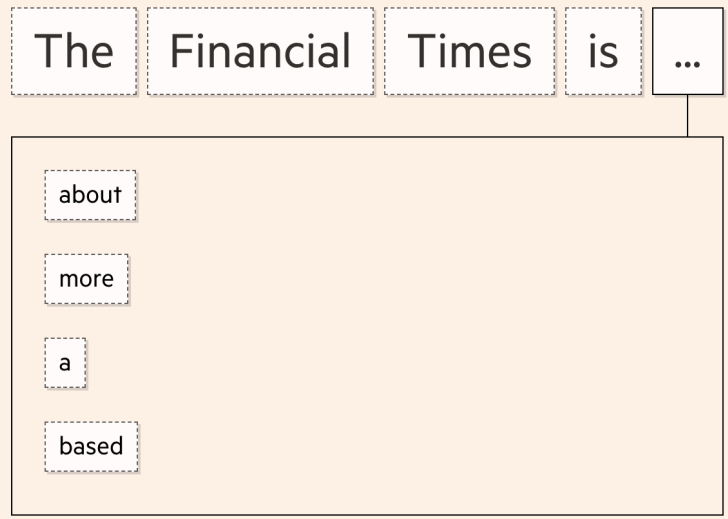
Building up: Encoding Layers

After tokenising and encoding a prompt, we're left with a block of data representing our input as the machine understands it, including meanings, positions and relationships between words.



Building up: Encoding Layers

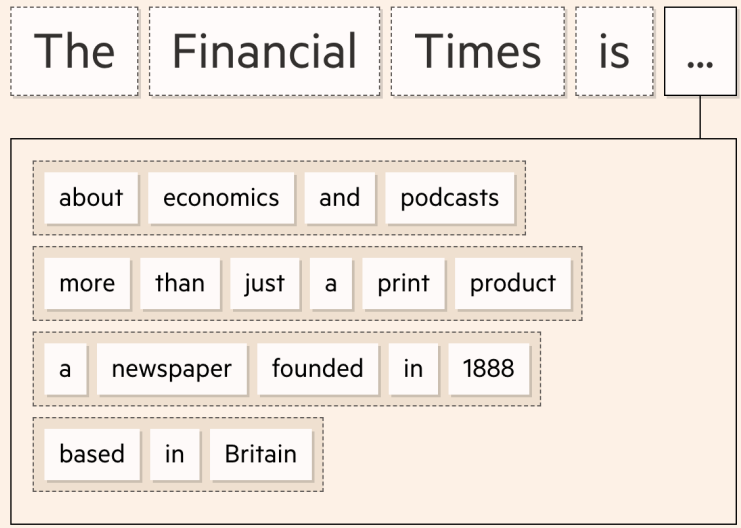
At its simplest, the model's aim is now to predict the next word in a sequence and do this repeatedly until the output is complete.



Making predictions: Beam search

Transformers use a number of approaches to address this problem and enhance the quality of their output. One example is called **beam search**.

Rather than focusing only on the next word in a sequence, it looks at the probability of a larger set of tokens as a whole.



Attention/Alignment

We'll next illustrate attention in a "sequence-to-sequence" task, like translating a sentence from French to English.

- Each source/targe word is associated with an encoder/decoder state—a vector of neurons.
- On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

Attention/Alignment

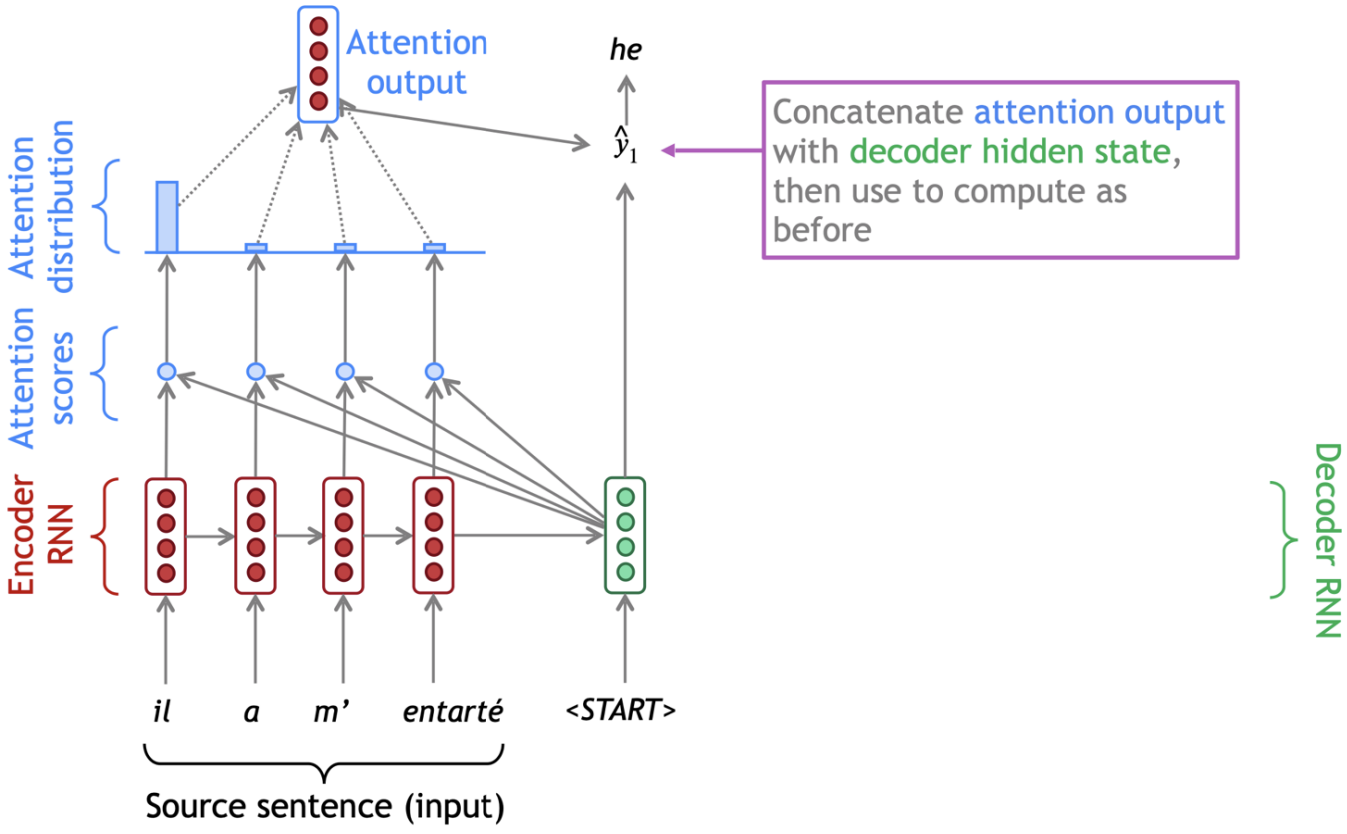


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment

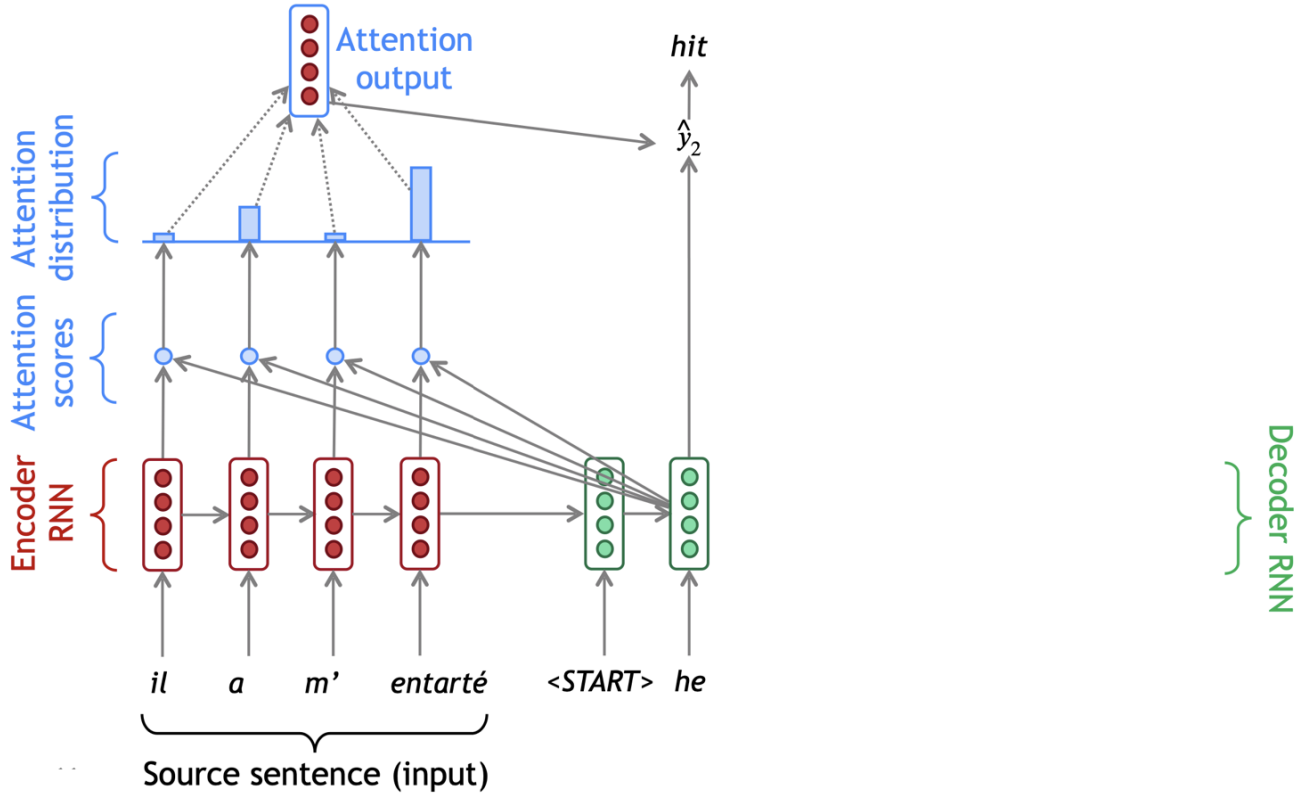


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment

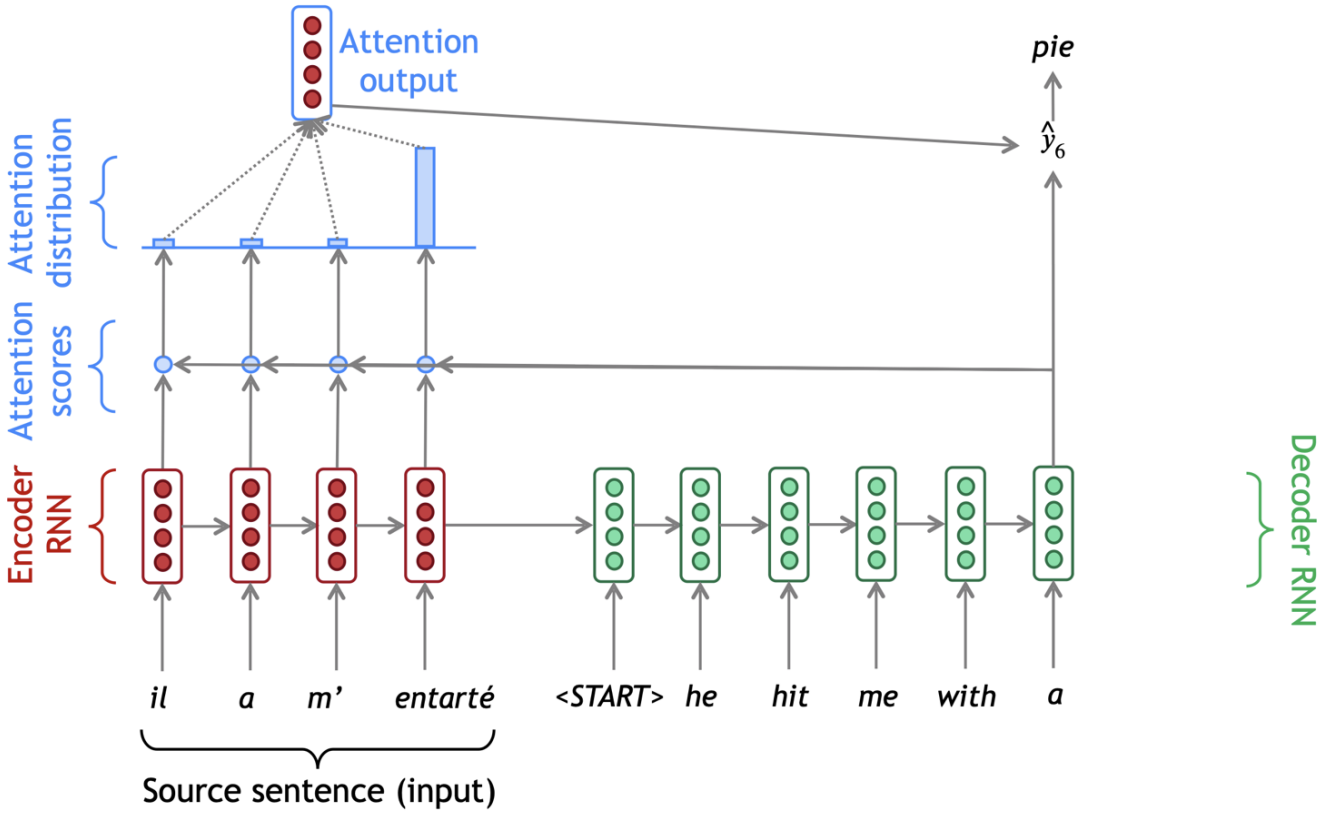
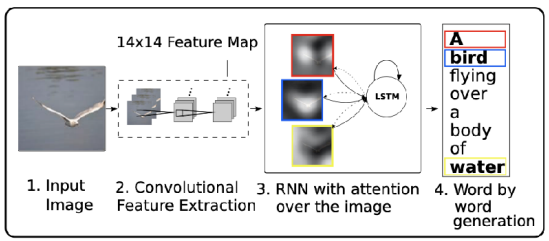
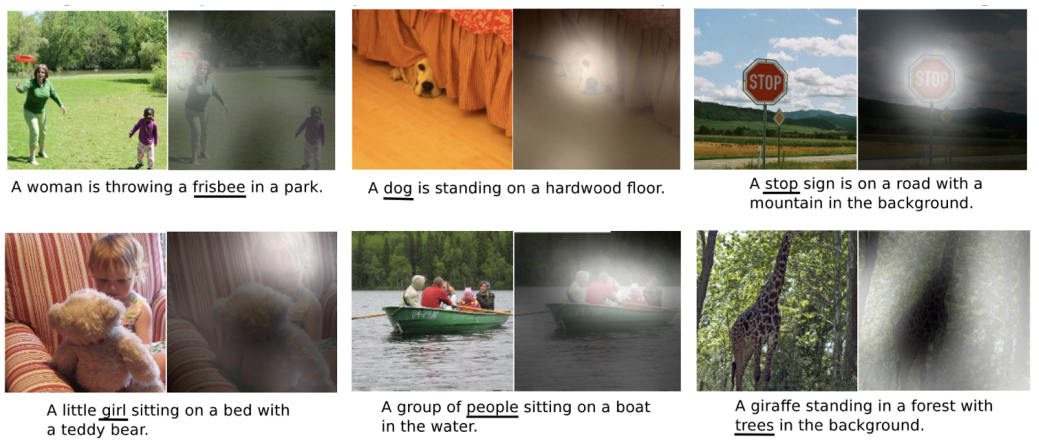


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

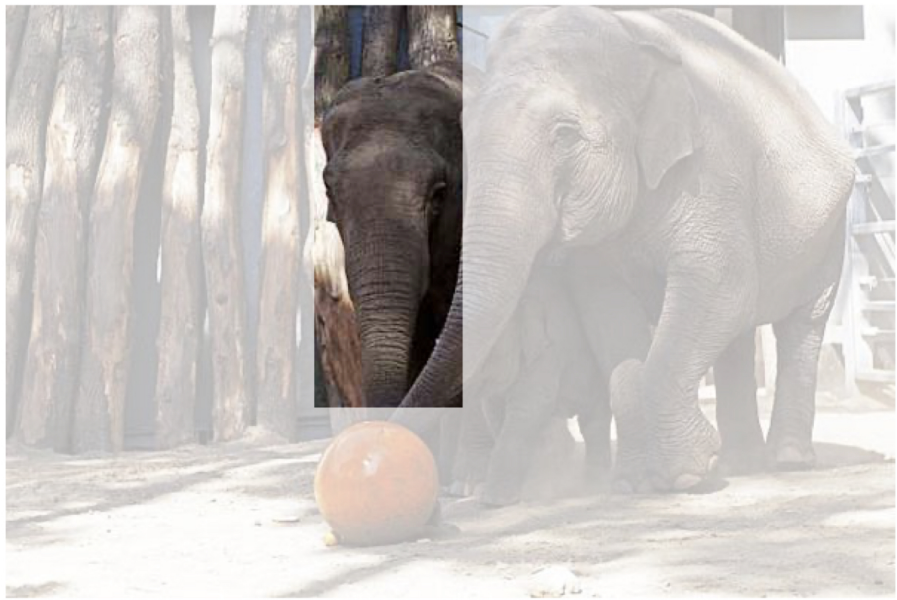
Attention/Alignment

Attention can also be used for other generative models





* "Show, attend and tell: neural image caption generation with visual attention", Xu et al. 2016, https://arxiv.org/pdf/1502.03044.pdf



The two elephants played with an orange ball

The two elephants played with an orange ball

 → orange

 → ball

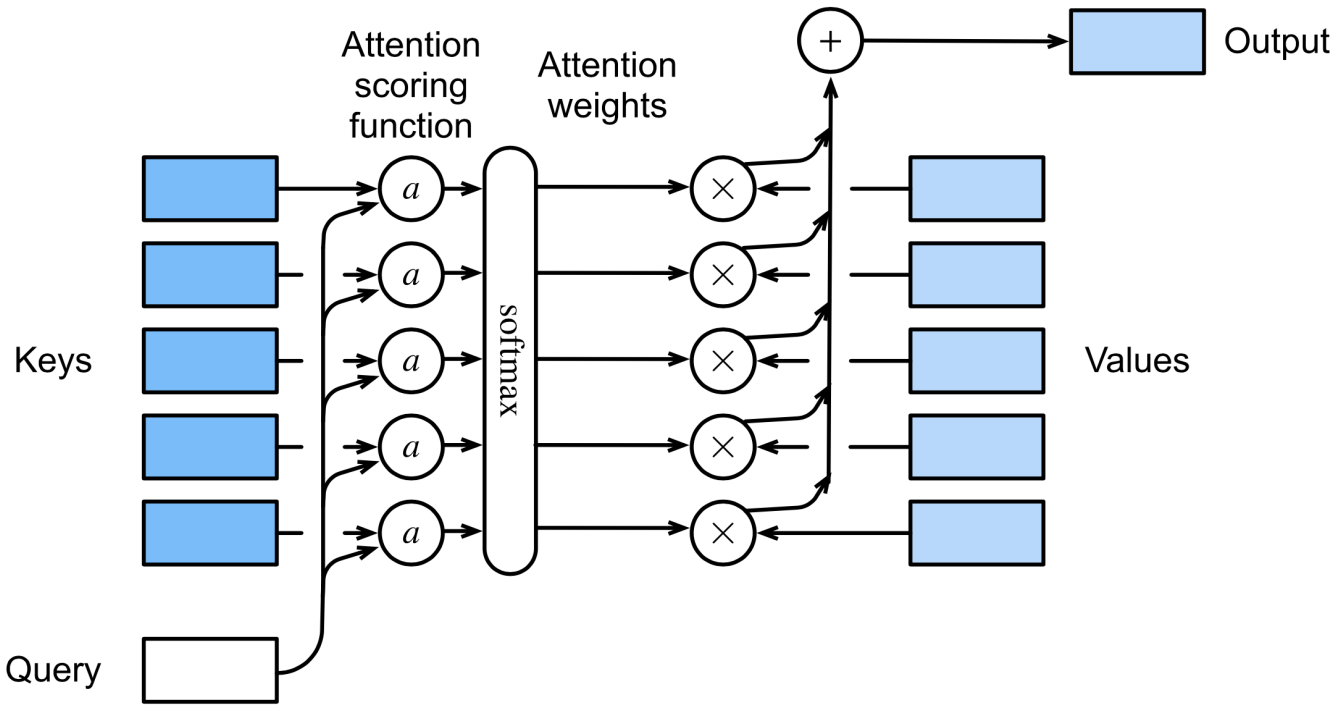
Attention in terms of query/key/values

Basic idea behind attention mechanisms:

- We have a set of m feature vectors or values $V \in \mathbb{R}^{m \times v}$
- The model dynamically chooses which to use
- based on how similar a query vector $q \in \mathbb{R}^{q}$ is to a set of m keys $K \in \mathbb{R}^{m \times k}$.
- If q is most similar to key i, then we use value (feature) v_i.

Attention in terms of query/key/values

Basic idea behind attention mechanisms:



Attention in terms of query/key/values

Basic idea behind attention mechanisms:

$$\text{Attn}(q, \{(k_1, v_1), \ldots (k_m, v_m)\}) = \sum_{i=1}^{m} w_i(q, k_{1:m}) v_i$$

where weights w_i are softmax of attention scores $a(q, k)$:

$$w_i(q, k_{1:m}) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))}$$

Dot product attention

Attention scores are computed as

$$a(q, k_i) = (W_1 q)^T (W_2 k_i)$$

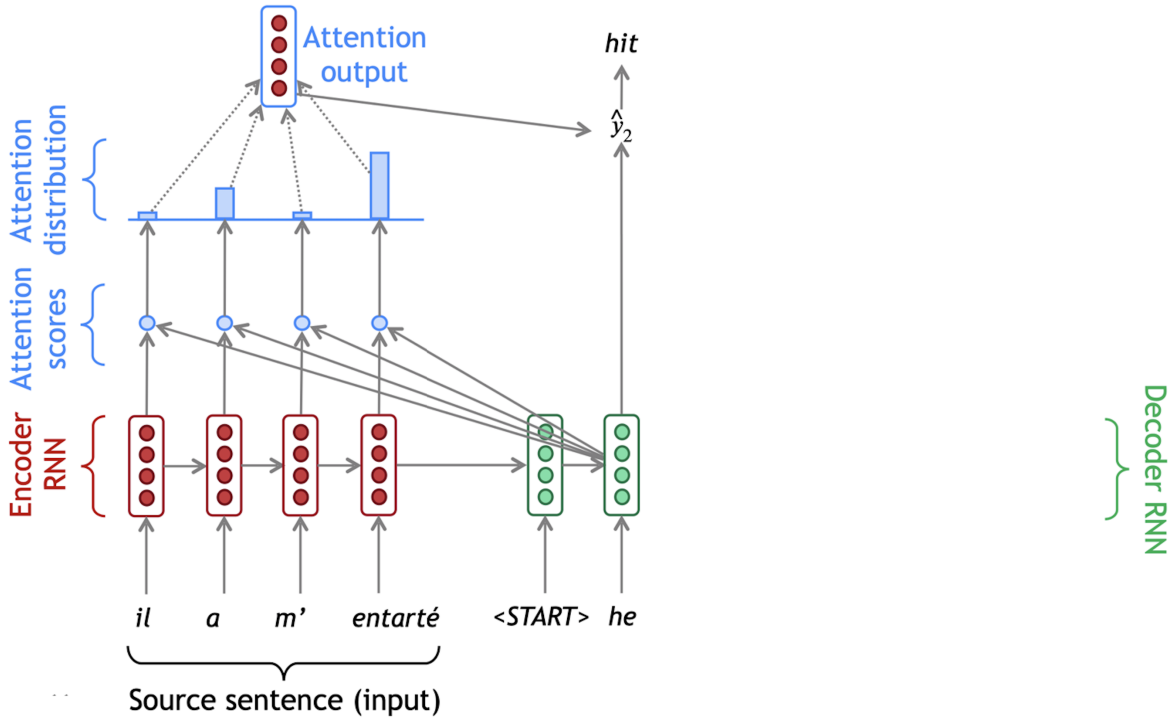
and then the attention weights are the softmax:

$$\text{Attn}(q, k_{1:m}, v_{1:m}) = \text{Softmax}\left(a(q, k)\right) v$$

The matrices W_1 and W_2 allow comparison of query and keys from different "languages"—-French and English or images and words

Retour d'entarté

In the *entarté* example, the keys and values are the red state vectors, the queries are the green state vectors



Transformers

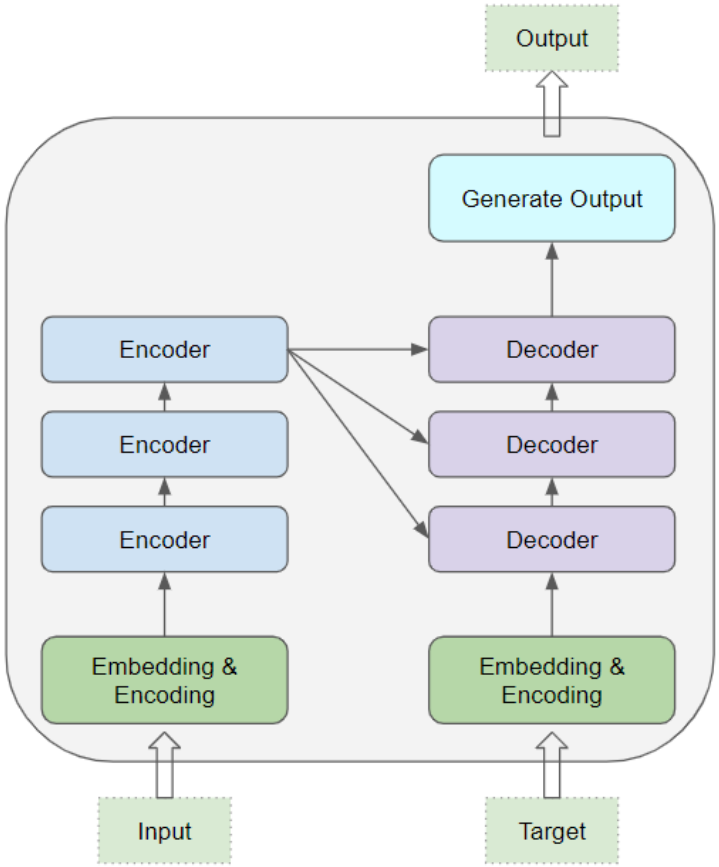
The current state-of-the-art is based on *transfomers*

- Attention is the key ingredient
- Rather than processing sequences word-by-word, transformers handle larger chunks of text at once
- The separation between words matters less

Combining attention vectors

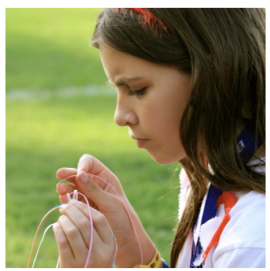
- Different attention vectors are stacked up on top of each other
- Words “hungry” and “sweet” are predicted using all of them

Transformer architecture



Demo

Attention



This demo illustrates some of the concepts behind attention in transformers, using word embeddings. The goal is simply to show how the basic calculations underlying attention work, and how weighting embeddings with attention can change the words that are similar. As implemented in Transformers, attention can be notoriously difficult to understand.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import gensim
```

Summary

- Attention is a type of alignment between related words
- Attention allows dynamic construction of word contexts, useful for predicting the next word
- Attention is computed using inner products, after mapping query and key to a common space
- Transformers process all words together, using layers of encoders and decoders, each with an attention component