NLP With Deep Learning For Everyone

Bruno Gonçalves

www.data4sci.com/newsletter

data4sci.substack.com

https://github.com/DataForScience/AdvancedNLP

Events

data4sci.substack.com

LangChain for Generative AI Pipelines

May 28, 2025 - 10am-2pm (PST)

Machine Learning with PyTorch for Developers

Jun 4, 2025 - 10am-2pm (PST)

Bruno Gonçalves

https://data4sci.com

info@data4sci.com

https://data4sci.com/call

Question

https://github.com/DataForScience/AdvancedNLP

- What's your job title?
    - Data Scientist
    - Statistician
    - Data Engineer
    - Researcher
    - Business Analyst
    - Software Engineer
    - Other

Question

https://github.com/DataForScience/AdvancedNLP

- How experienced are you in Python?
    - Beginner (<1 year)
    - Intermediate (1-5 years)
    - Expert (5+ years)

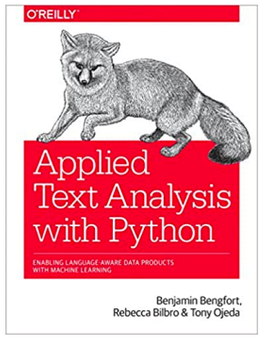
Question

https://github.com/DataForScience/AdvancedNLP

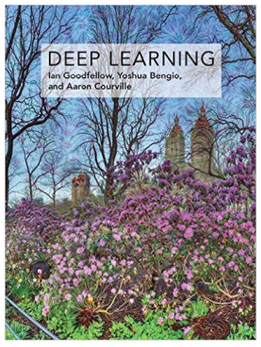
- How did you hear about this webinar?
  - O'Reilly Platform
  - Newsletter
  - data4sci.com Website
  - Previous event
  - Other?

References

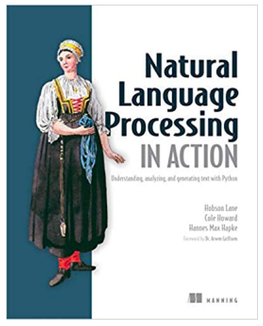
https://github.com/DataForScience/AdvancedNLP



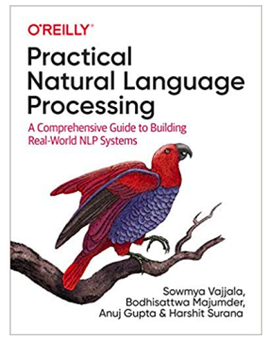
https://amzn.to/3iMqanY



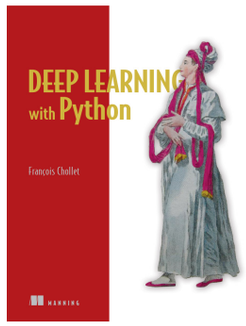
https://amzn.to/2BGr0RL



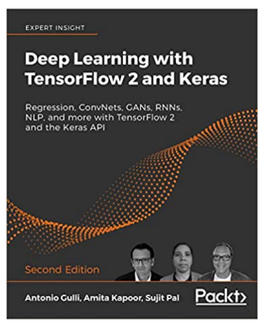
https://amzn.to/3sXAZbm



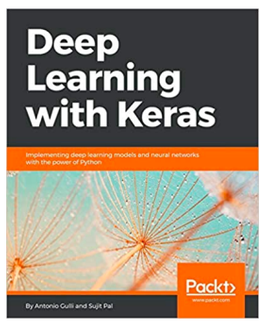
https://amzn.to/3a2fhui



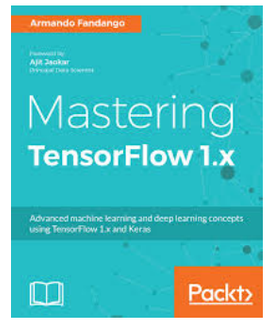
https://amzn.to/30fTJqB



https://amzn.to/30fTMCN



https://amzn.to/3qR3rKh



https://amzn.to/2AavBuT

Table of Contents

Lesson 1:
Foundations of NLP

Lesson 1.1:
One-Hot Encoding

One-Hot Encoding

- The first step in analyzing text is to represent it in a way that can be easily manipulated numerically. Typically this takes the form of representing each term by a vector

- Many approaches have been developed for different purposes

- The most basic one is known as One-Hot Encoding:

    - Each word corresponds to a different dimension in a high-dimensional space

    - All elements of the vector are zero, except the one corresponding to the word

$$v_{fleece} = (0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0, \cdots)^T$$
$$v_{everywhere} = (0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0, \cdots)^T$$

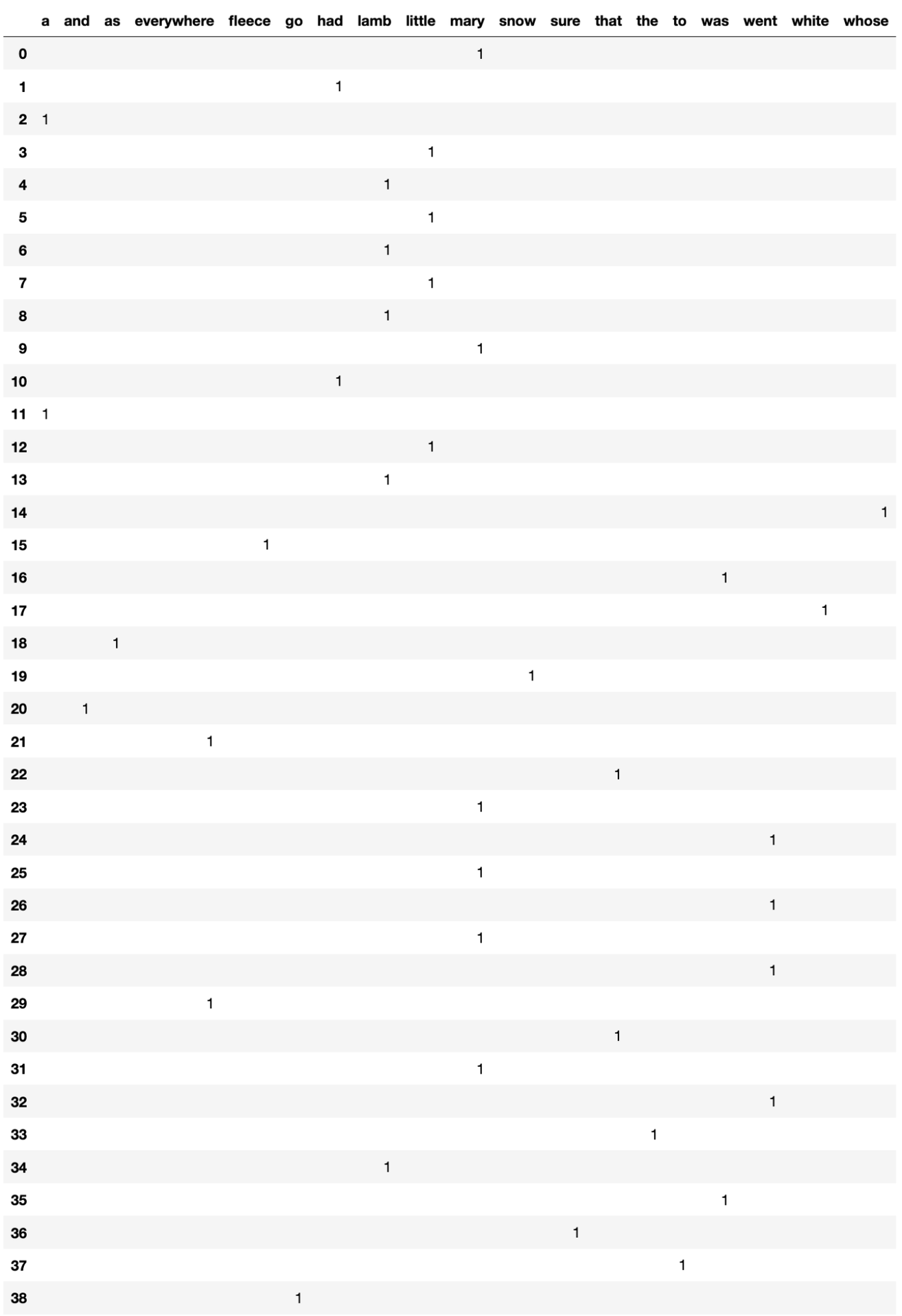
- One-hot encoded vectors are extremely sparse and contain no semantic information

One-Hot Encoding

- So the text for "Mary had a little lamb":

Mary had a little lamb, little lamb,
little lamb, Mary had a little lamb
whose fleece was white as snow.
And everywhere that Mary went
Mary went, Mary went, everywhere
that Mary went
The lamb was sure to go.

- Could be represented using this one-hot encoded matrix (we omit the 0 values for clarity).



	a	and	as	everywhere	fleece	go	had	lamb	little	mary	snow	sure	that	the	to	was	went	white	whose
0										1									
1							1												
2	1																		
3									1										
4								1											
5									1										
6								1											
7									1										
8								1											
9										1									
10							1												
11	1																		
12									1										
13								1											
14																			1
15						1													
16																1			
17																		1	
18			1																
19										1									
20	1																		
21				1															
22									1										
23										1									
24																			1
25										1									
26																			1
27										1									
28																			1
29				1															
30									1										
31										1									
32																			1
33									1										
34								1											
35																1			
36										1									
37									1										
38						1													

Bag-of-Words

- A closely related is that of Bag of Words, where we keep track of how many times a word is used within a piece of text
- For our little nursery rhyme, this could simply be:
- Similar representations could be generated for different documents, allowing us to compare or cluster them easily.

	Count
a	2
and	1
as	1
everywhere	2
fleece	1
go	1
had	2
lamb	5
little	4
mary	6
snow	1
sure	1
that	2
the	1
to	1
was	2
went	4
white	1
whose	1

Lesson 1.2:
Stemming and Lemmatization

Stemming and Lemmatization

- In practical applications, Vocabularies can become extremely large (English is estimated to have over 1 million unique words).

- Several techniques have been developed to help reduce the vocabulary size with minimal loss of information. In particular:

  - Stemming - Use heuristics to identify the root (or stem) of the word.
    The stem doesn't need to be a "real" word as long as the mapping is consistent.

  - Lemmatization - Identify the "dictionary form" (lemma) of the word. This approach requires identifying the Part-of-Speech being used and using hand curated tables to find the correct lemma.

  - Stopwords - Remove the most common words that don't contain any semantic information (the, and, a, etc)

love	
loved	
loves	love
loving	
lovingly	
…	

Stemming

- NLTK contains several different stemmer algorithms, with varying support for different languages

  - Cistem - German
  - ISRIStemmer - Arabic
  - LancasterStemmer - English
  - PorterStemmer - English (the original one)
  - RSLPStemmer - Portuguese
  - RegexpStemmer - English (using Regular Expressions)
  - SnowballStemmer - Arabic, Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish and Swedish

- The SnowballStemmer is a good default choice and tends to perform well across most languages.

	LancasterStemmer	PorterStemmer	RegexpStemmer	SnowballStemmer
playing	play	play	play	play
loved	lov	love	loved	love
ran	ran	ran	ran	ran
river	riv	river	river	river
friendships	friend	friendship	friendship	friendship
misunderstanding	misunderstand	misunderstand	misunderstand	misunderstand
trouble	troubl	troubl	troubl	troubl
troubling	troubl	troubl	troubl	troubl

Lemmatization

- NLTK implements the WordNetLematizer algorithm that uses the WordNet database of concepts.

- WordNetLematizer algorithm is guaranteed to return a “real” word but the results depend on correct Part-Of-Speech identification. The result for a Noun will be different than the result for a Verb, Adverb, etc.

Lemmatization

- Lemmatization tends to be computationally more expensive than Stemming
- Depending on your specific application, you might prefer Stemming or Lemmatization

	LancasterStemmer	PorterStemmer	RegexpStemmer	SnowballStemmer	WordNetLemmatizer Noun	WordNetLemmatizer Verb
playing	play	play	play	play	playing	play
loved	lov	love	loved	love	loved	love
ran	ran	ran	ran	ran	ran	run
river	riv	river	river	river	river	river
friendships	friend	friendship	friendship	friendship	friendship	friendships
misunderstanding	misunderstand	misunderstand	misunderstand	misunderstand	misunderstanding	misunderstand
trouble	troubl	troubl	troubl	troubl	trouble	trouble
troubling	troubl	troubl	troubl	troubl	troubling	trouble

Lesson 1.3: Stopwords

Stopwords

- Stopwords are usually the most common words that don't contain any semantic information (the, and, a, etc), but there is no unique universal list of stop words.
- Different applications might use different sets of stop words of none at all.
- The goal of removing them from your text is to significantly reduce the number of words you must process while losing as little information as possible.
- Naturally, these are language dependent.
- NLTK supports 23 languages out of the box. These are typically stored as plain text files under '~/nltk_data/corpora/stopwords/'
- You can add more by simply adding a text file in the proper directory with one word per line.
- Stopwords can be loaded to NTLK by using the file name

```
'dutch'
'german'
'slovene'
'hungarian'
'romanian'
'kazakh'
'turkish'
'russian'
'italian'
'english'
'greek'
'tajik'
'norwegian'
'portuguese'
'finnish'
'danish'
'french'
'swedish'
'azerbaijani'
'spanish'
'indonesian'
'arabic'
'nepali'
```

Original	Filtered
Mary	Mary
had	
a	
little	little
lamb	lamb
little	little
lamb	lamb
little	little
lamb	lamb
Mary	Mary
had	
a	
little	little
lamb	lamb
whose	whose
fleece	fleece
was	
white	white
as	
snow	snow
And	
everywhere	everywhere
that	
Mary	Mary
went	went
Mary	Mary
went	went
MARY	MARY
went	went
Everywhere	Everywhere
that	
mary	mary
went	went
The	
lamb	lamb
was	
sure	sure
to	
go	go

Lesson 1.4: N-grams

N-grams

- N-grams are co-occurring sequences of N items from a sequence of words of characters
- NLTK provides the nltk.util.ngrams utility function to easily generate N-Grams of specific lengths
- N-grams are important to account for modifiers, Named Entity Recognition, etc.
- But how can we know if a N-Gram is significant?

Collocations

- A closely related concept is that of Collocation - N-Grams that occur more commonly than expected by chance.

- The nltk.collocations submodule provides objects to identify and compute the most significant Bigram, Trigram and Quadgrams:

    - Bigram/Trigram/QuadgramCollocationFinder - support for different ways of finding 2, 3, and 4-grams.

    - Bigram/Trigram/QuadgramAssocMeasures - selection of metrics to quantify the relative importance of each 2, 3, and 4-grams. In particular:

        - chi_sq/jaccard/likelihood_ratio/mi_like/pmi/poisson_stirling/raw_freq/student_t

- Significant collocations can prove useful for entity extraction, topic detection, etc.

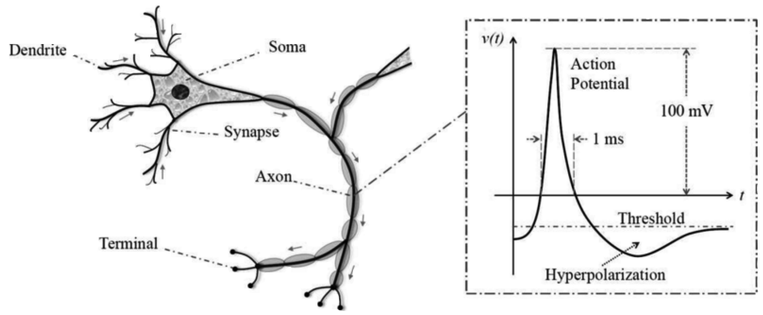
Code - Foundations of NLP

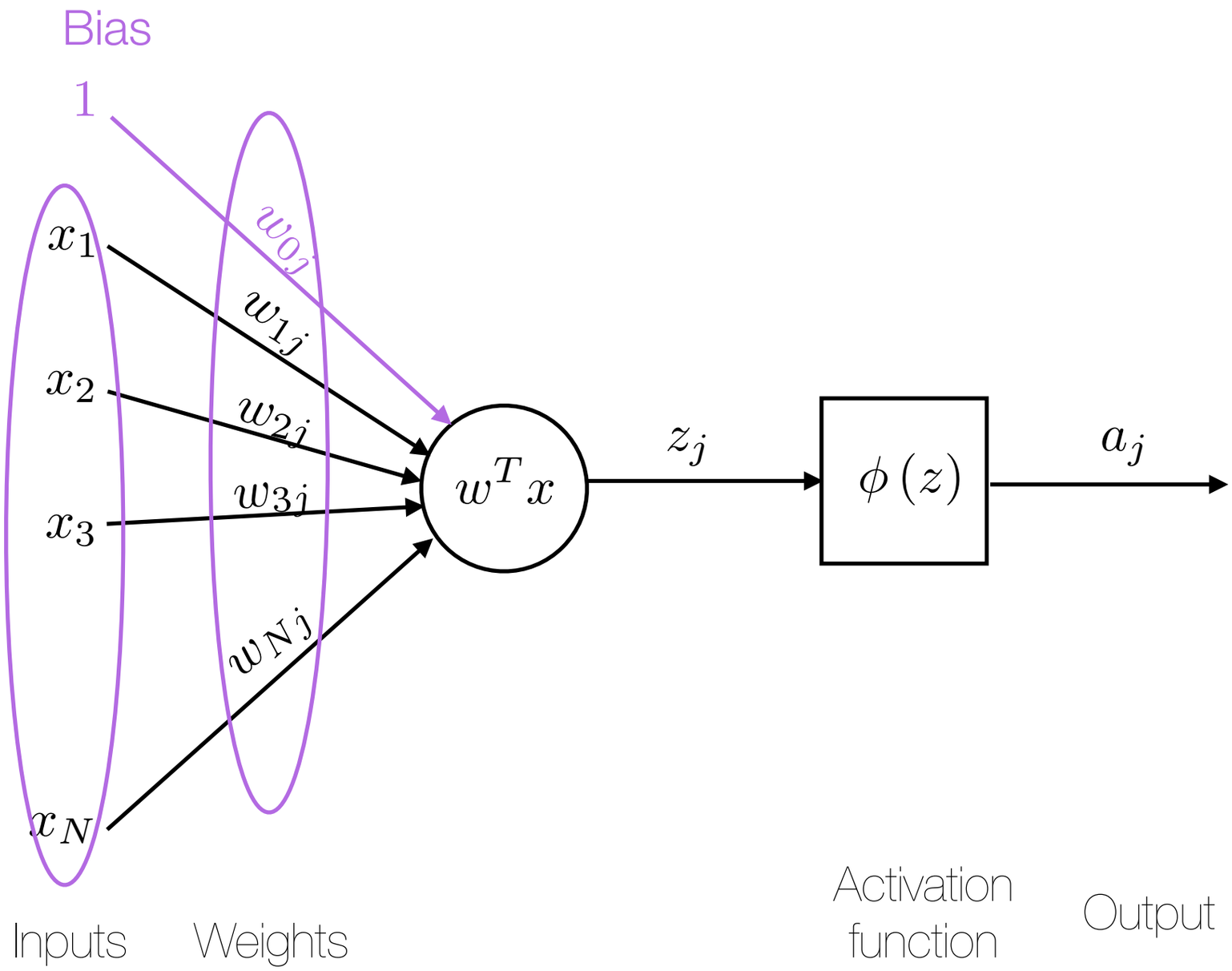
https://github.com/DataForScience/AdvancedNLP

Lesson 2:
Neural Networks with PyTorch

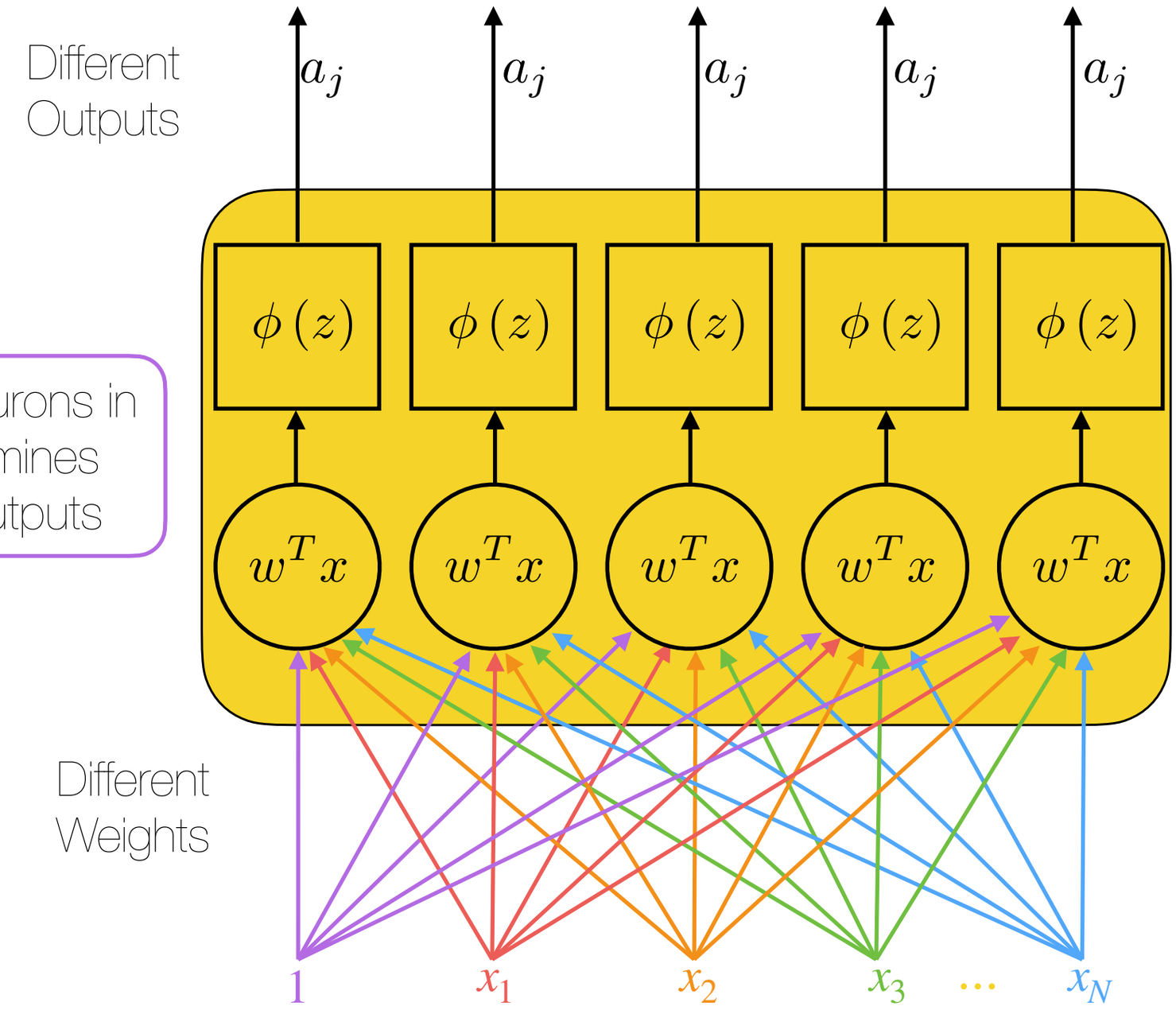
Lesson 2.1: PyTorch Overview

Artificial Neuron





Feed Forward Networks



Outputs from a layer become inputs to the next

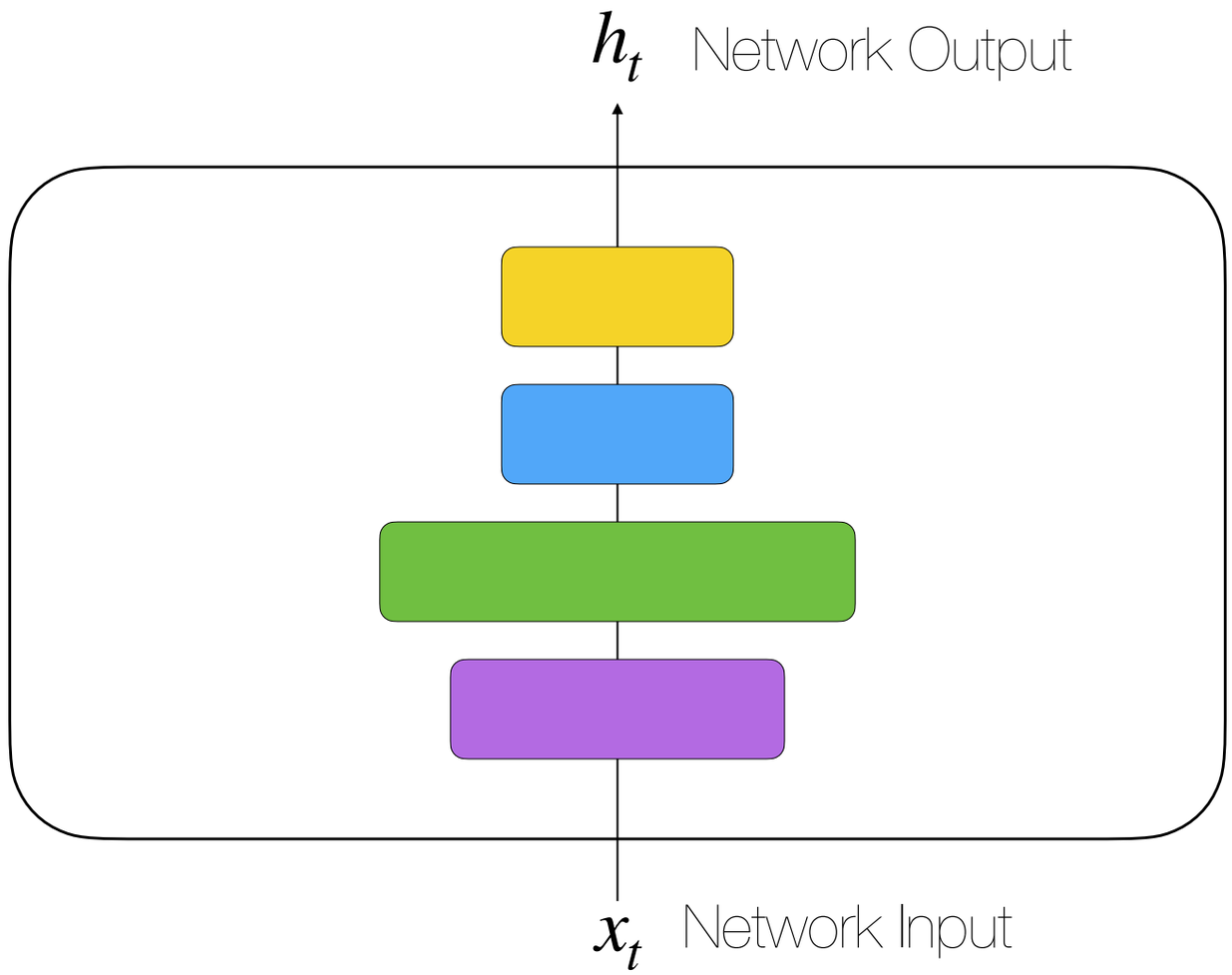
Number of Neurons in a layer determines number of outputs

Every neuron has the same activation function

Same input values to each neuron

$$h_t = f\left(x_t\right)$$

(Deep) Feed Forward Networks



Number of Outputs from a layer much match the number of inputs in the next

Networks can have arbitrary numbers of layers with varying numbers of neurons

$$h_t = f\left(x_t\right)$$

Lego Blocks

pytorch.org

Lego Blocks

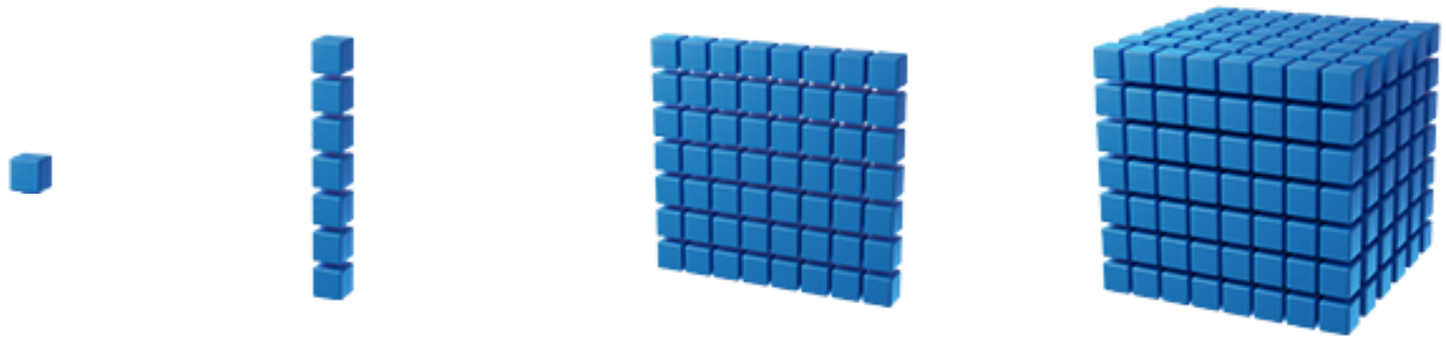
pytorch.org

PyTorch Overview

- Developed by Facebook's AI Research (FAIR) team and initially released in January 2017.
- High level structure allowing for intuitive model definition, especially for complex architectures
- Robust Tensor data structure and automatic differentiation mechanism to compute gradients with minimal overhead
- Seamless support for a wide range of specialized hardware (GPUs, MPS, etc)
- Ready-to-use building blocks that can be chained together to create models ranging from simple linear regression to state of the art deep learning architectures
- Support for distributed and multi-GPU Training
- Large community contributes new models, tutorials, and extensions regularly

Tensors

- Tensors are a mathematical generalization of scalars, vectors, and matrices to arbitrary dimensions:



0D Tensor (scalar) | 1D Tensor (vector) | 2D Tensor (matrix) | 3D Tensor

- 0D tensor (rank-0): a single number like 5
- 1D tensor (rank-1): an array of numbers [1, 2, 3]
- 2D tensor (rank-2): a matrix of numbers [[1, 2], [3, 4]]
- Higher-dimensional tensors continue this pattern with 3D, 4D, and beyond.

Tensors

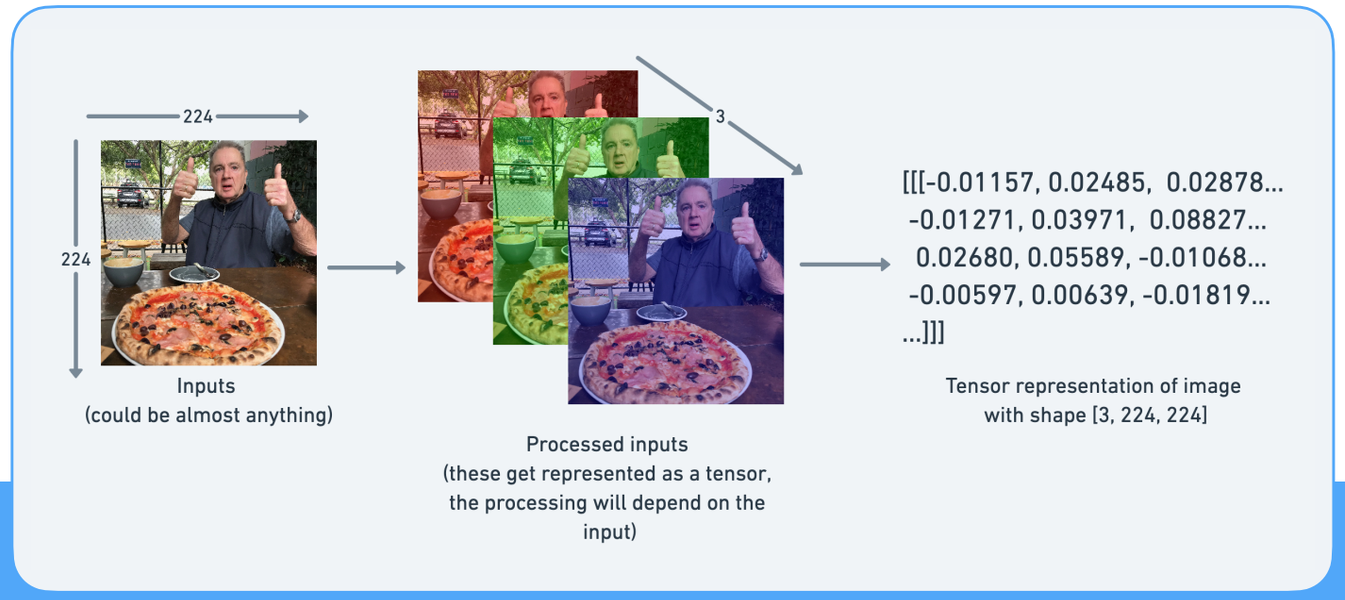
https://github.com/mrdbourke/pytorch-deep-learning/blob/main/00_pytorch_fundamentals.ipynb

- In PyTorch, tensor is the fundamental primary data structure: multidimensional arrays that can be efficiently manipulated on GPUs.
- Tensors are a unified way of describing complicated systems.
- Tensors in practice:
  - Timeseries (1D tensor: timesteps)
  - Tabular data (2D tensor: rows x columns)
  - Images (3D tensors: height × width × color channels)
  - Video (4D tensors: frames × height × width × channels)
  - Textual data (embedding dimensions × sequence length × batch size)

Tensors

https://github.com/mrdbourke/pytorch-deep-learning/blob/main/00_pytorch_fundamentals.ipynb

- In PyTorch, tensor is the fundamental primary data structure: multidimensional arrays that can be efficiently manipulated on GPUs.
- Tensors are a unified way of describing complicated systems.
- Tensors in practice:
  - Timeseries (1D tensor: timesteps)
  - Tabular data (2D tensor: rows x columns)
  - Images (3D tensors: height × width × color channels)
  - Video (4D tensors: frames × height × width × channels)
  - Textual data (embedding dimensions × sequence length × batch size)



Tensors in PyTorch

https://github.com/mrdbourke/pytorch-deep-learning/blob/main/00_pytorch_fundamentals.ipynb

- Tensors are the fundamental data structure, conceptually similar to NumPy arrays
- Features:
    - Automatic Differentiation: allow PyTorch to automatically compute gradients with respect to the operations performed
    - GPU acceleration: Tensors can reside on either CPU or GPU.
        - Wide range of data types: boolean, integer, floating point, and complex in 8, 16, 32, 64, and 128 bit formats
- Extensive support for mathematical and linear algebra operations

Hardware Acceleration

https://pytorch.org/docs/stable/backends.html

- PyTorch is capable of leveraging a wide range of specialized hardware:
  - Nvidia GPUs - **device** = "cuda"
  - AMD GPUs - **device** = "cuda" 🤓
  - Apple Silicon - **device** = "mps"
- Tensors can be moved between devices with a simple **.to**(device) call
- Leverage multi-core CPUs, vectorized instructions (via libraries like MKL or OpenBLAS), and efficient threading for faster CPU-based computations.
- Utilities like DataParallel and DistributedDataParallel allow PyTorch to scale models across multiple GPUs, in or or more machines, improving throughput and training speed.

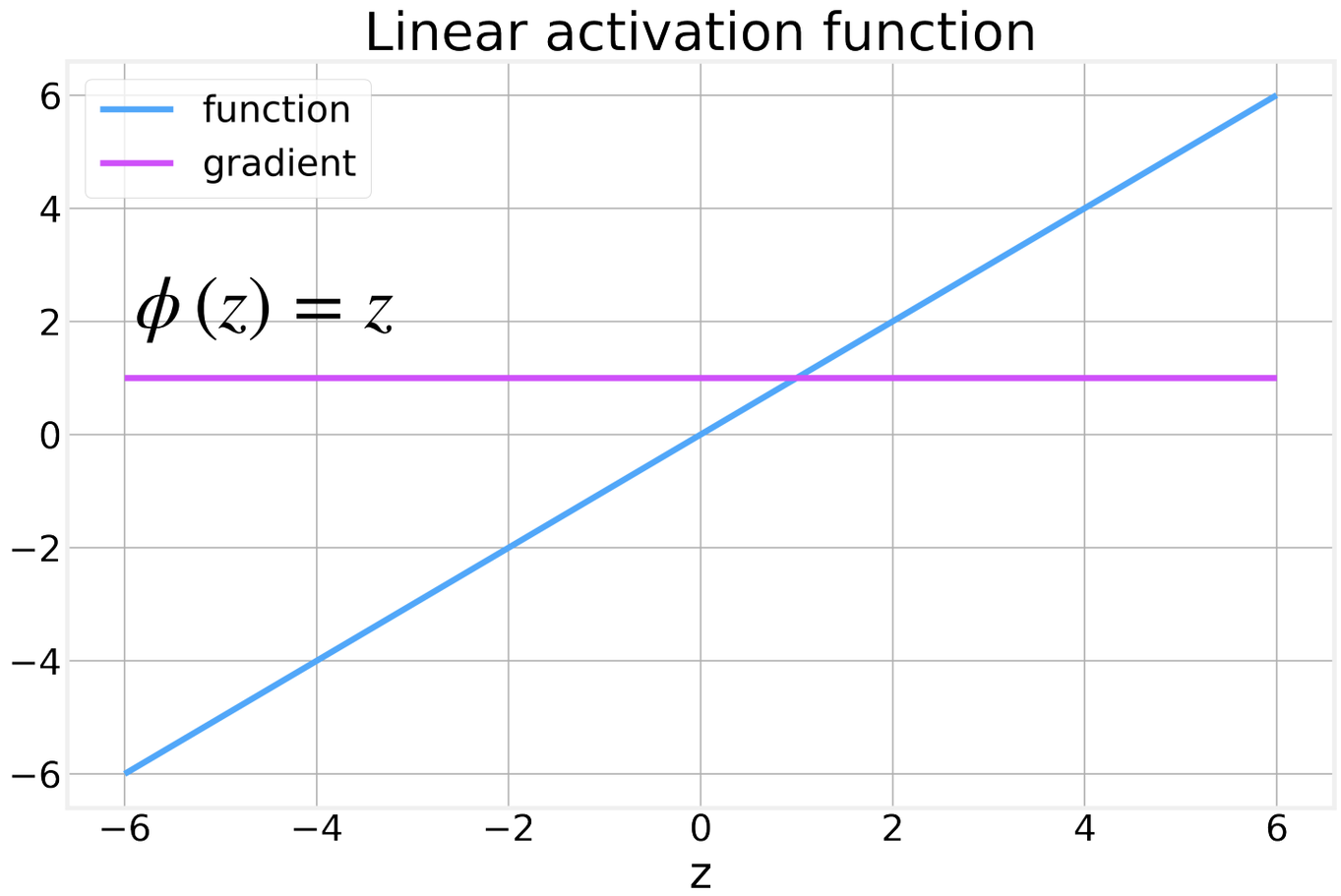
Advanced Features

- PyTorch comes complete with high-level frameworks, to simplify developing complex algorithms:
  - torch.nn module: Provides building blocks (layers, loss functions) for neural network construction.
  - torch.optim: A suite of optimization algorithms (e.g., SGD, Adam) that can be coupled with PyTorch Tensors and autograd for easy training loops.
- Specialized packages have been developed that leverage PyTorch's tensors under the hood:
  - torchvision - popular datasets, model architectures, and common image transformations for computer vision.
  - torchtext - Text utilities, models, transforms, and datasets for PyTorch (deprecated)
  - torchaudio - Building Blocks for Audio and Speech Processing.

Lesson 2.2:
Activation Functions

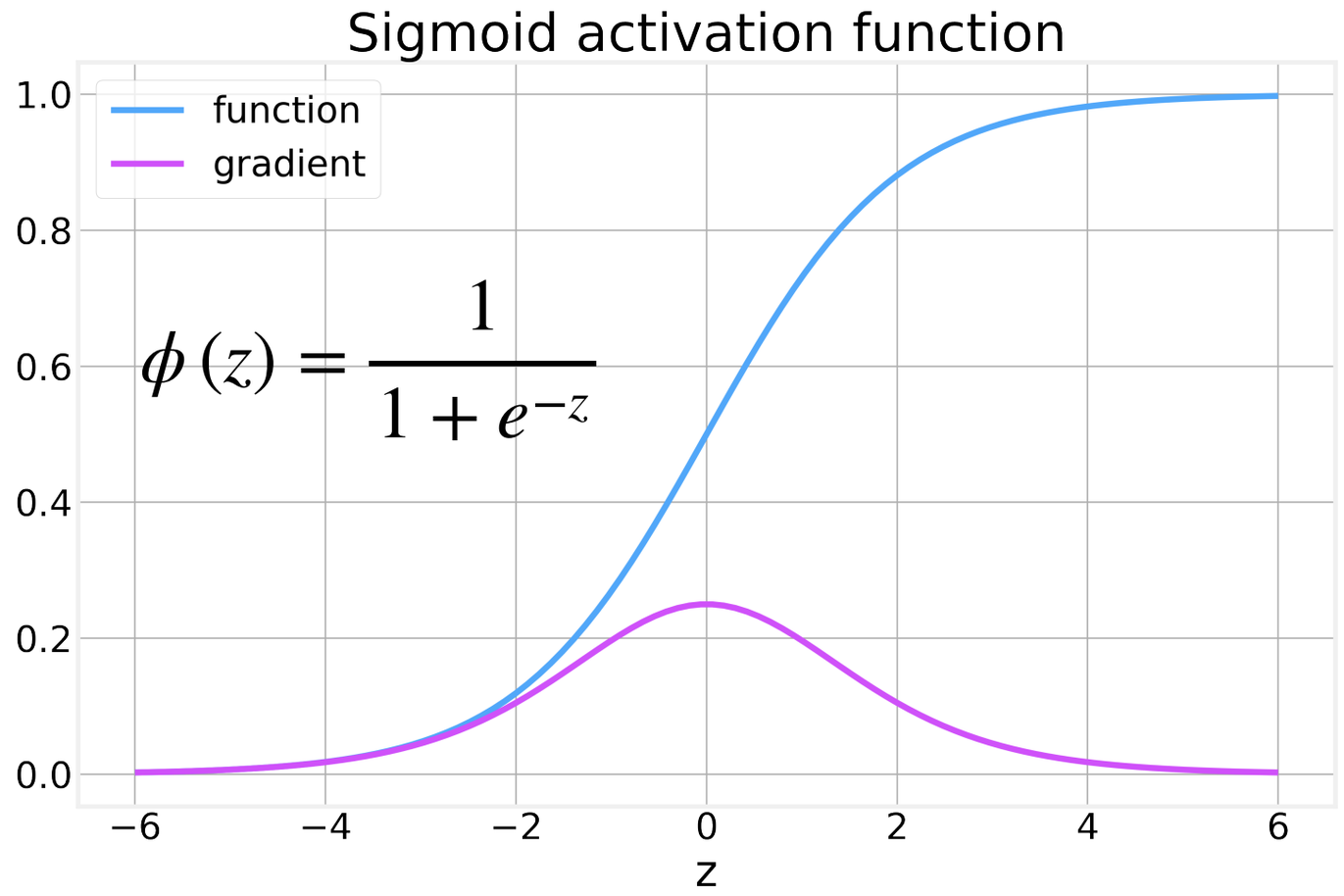
Activation Function - Linear

- Linear function
- Differentiable
- Non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- The simplest



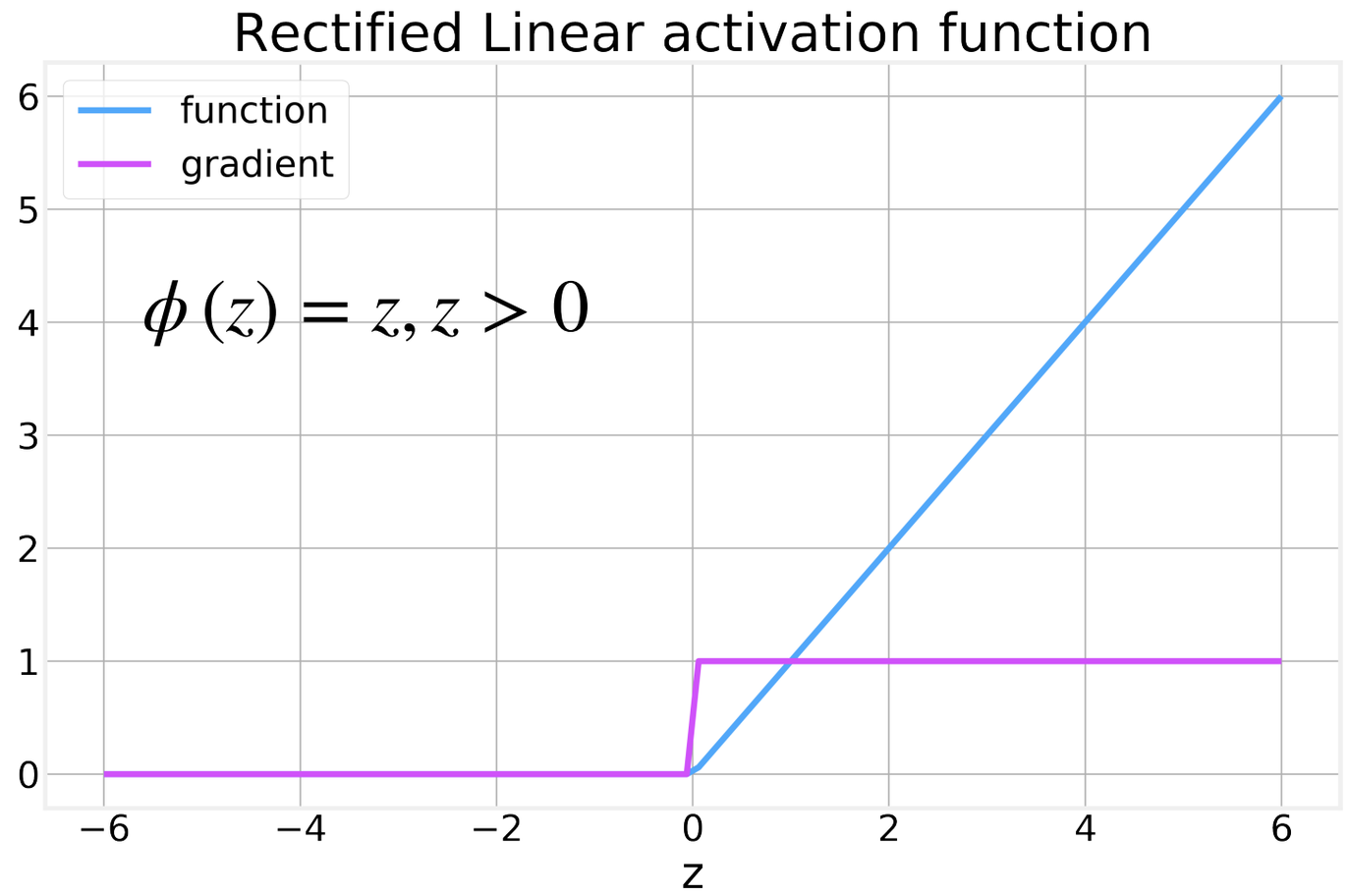
Activation Function - Sigmoid

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Perhaps the most common



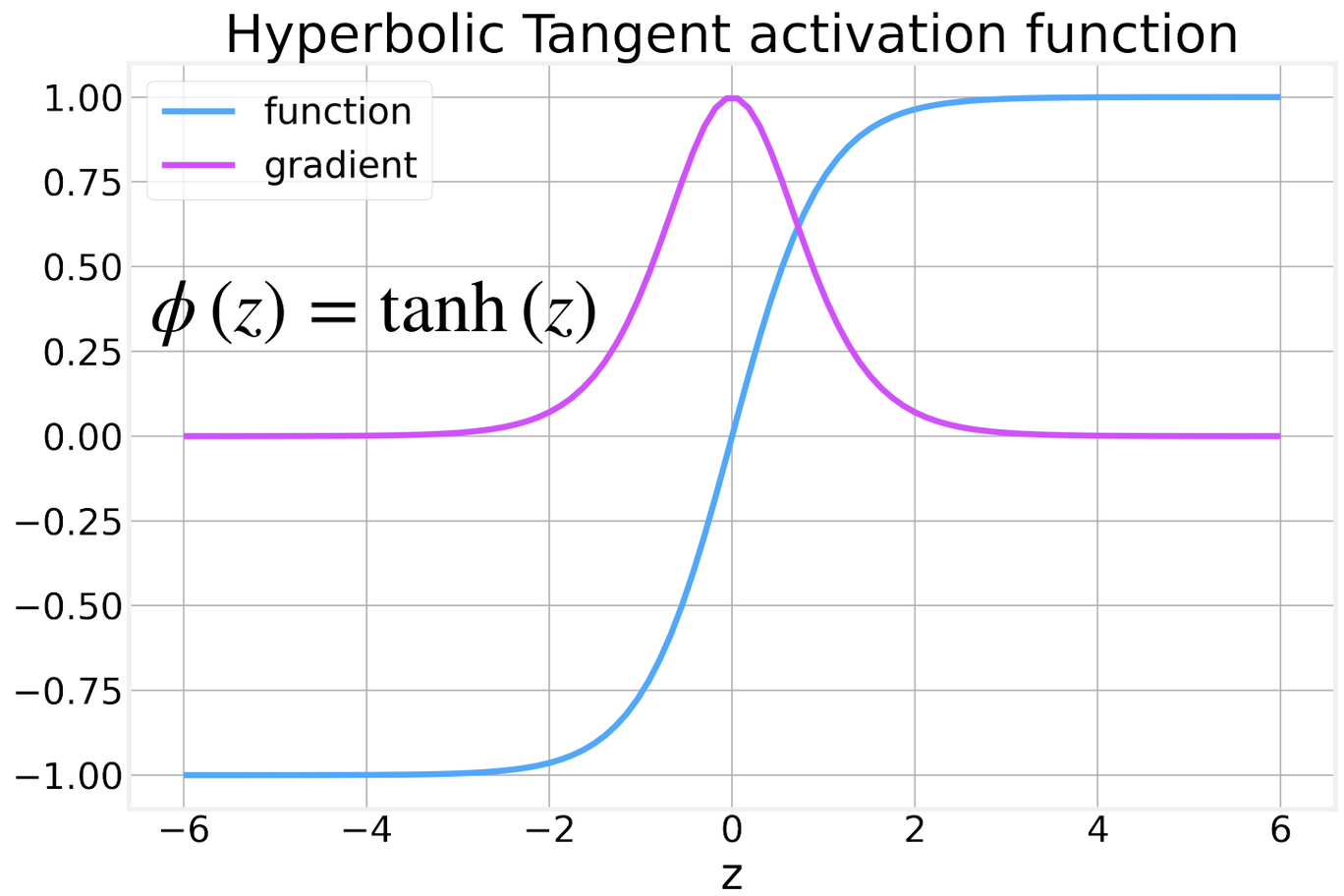
Activation Function - ReLu

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Results in faster learning than with sigmoid



Activation Function - Hyperbolic Tangent

- Non-Linear function
- Differentiable
- non-decreasing
- Compute new sets of features
- Each layer builds up a more abstract representation of the data
- Produces bounded positive and negative values



Lesson 2.3: Loss Functions

Loss-Functions

https://pytorch.org/docs/stable/nn.html#loss-functions

- Loss-Functions quantify the error we are making at each step
- Depend intrinsically on the output of our network (the final layer). Two major types:
  - Probabilistic Losses - Compare two probability distributions (Classification)
    - Cross-Entropy: $J_w\left(X,\vec{y}\right) = -\frac{1}{m}\left[y^T \log\left(h_w\left(X\right)\right) + \left(1-y\right)^T \log\left(1-h_w\left(X\right)\right)\right]$
  - Regression Losses - Compare two arbitrary numbers (Regression)
    - Mean Squared Error: $J_w\left(X,\vec{y}\right) = \frac{1}{2m}\sum_i \left[h_w\left(x^{(i)}\right) - y^{(i)}\right]^2$
- Many other variants:
  - Mean Absolute Error Loss
  - Mean Squared Error Loss
  - Negative Log-Likelihood Loss
  - Cross-Entropy Loss
  - Hinge Embedding Loss
  - Margin Ranking Loss
  - Kullback-Leibler divergence

Lesson 2.4:
Training Procedures

Optimization Problem

- (Machine) Learning can be thought of as an optimization problem.
- Optimization Problems have 3 distinct pieces:
  - The constraints
  - The function to optimize
  - The optimization algorithm.

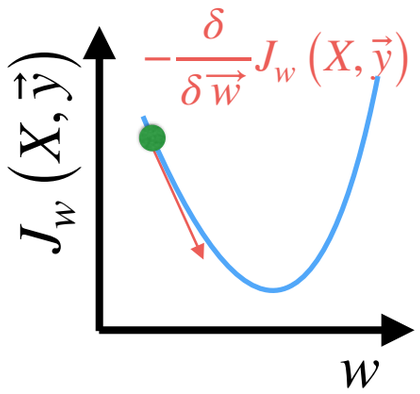
Optimization Problem

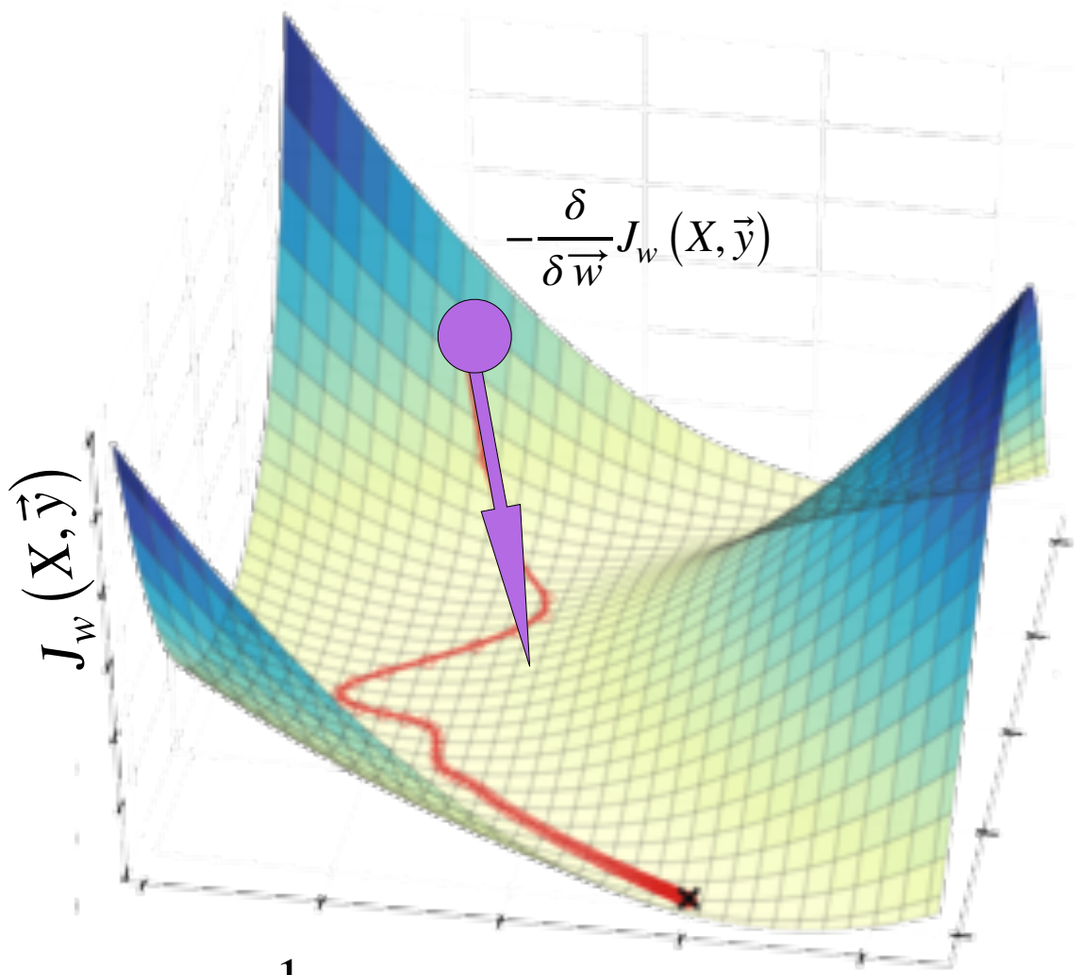
- (Machine) Learning can be thought of as an optimization problem.
- Optimization Problems have 3 distinct pieces:
  - The constraints — Network Structure
  - The function to optimize — Loss Function
  - The optimization algorithm. — Gradient Descent

Gradient Descent

- Goal: Find the minimum of $J_w\left(X,\vec{y}\right)$ by varying the components of $\vec{w}$

- Intuition: Follow the slope of the error function until convergence





- Algorithm:

  - Guess $\vec{w}^{(0)}$ (initial values of the parameters)
  - Update until "convergence":

step size

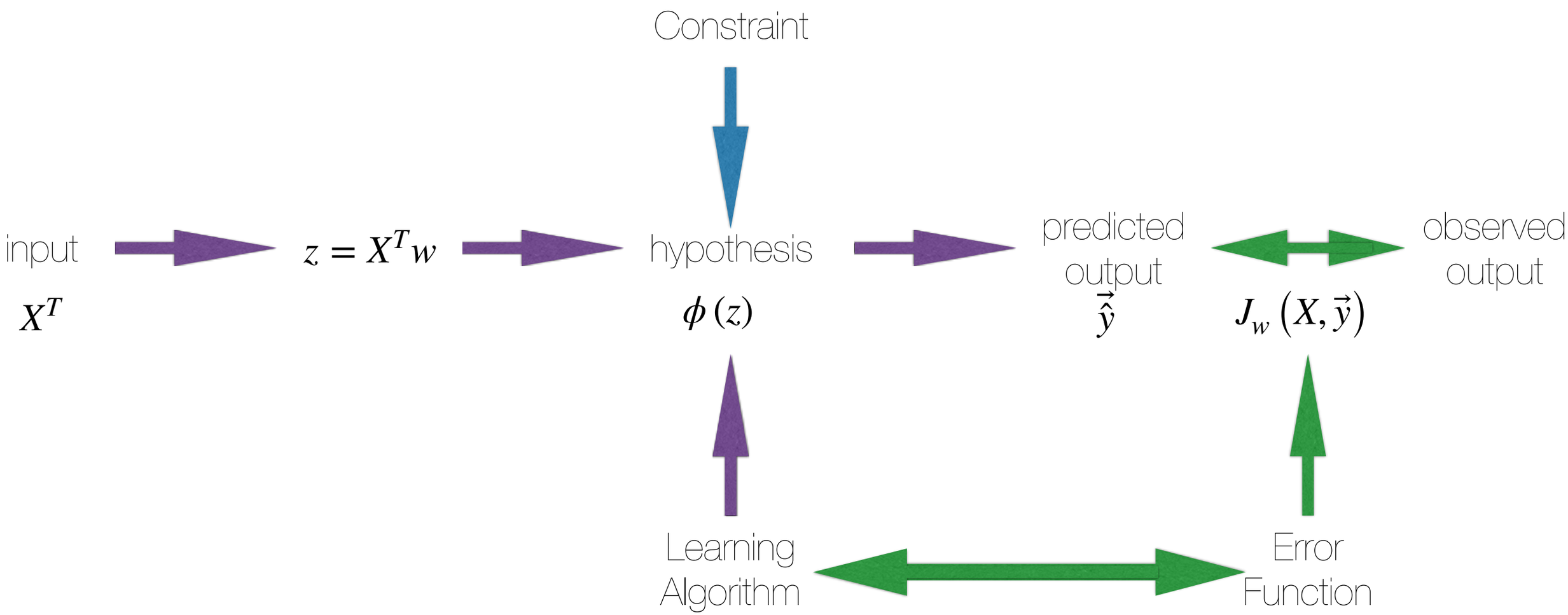
$$w_j = w_j - \alpha \frac{\delta}{\delta w_j} J_w\left(X,\vec{y}\right) \qquad \frac{\delta}{\delta w_j} J_w\left(X,\vec{y}\right) = \frac{1}{m} X^T \cdot \left(h_w(X) - \vec{y}\right)$$

Optimizers

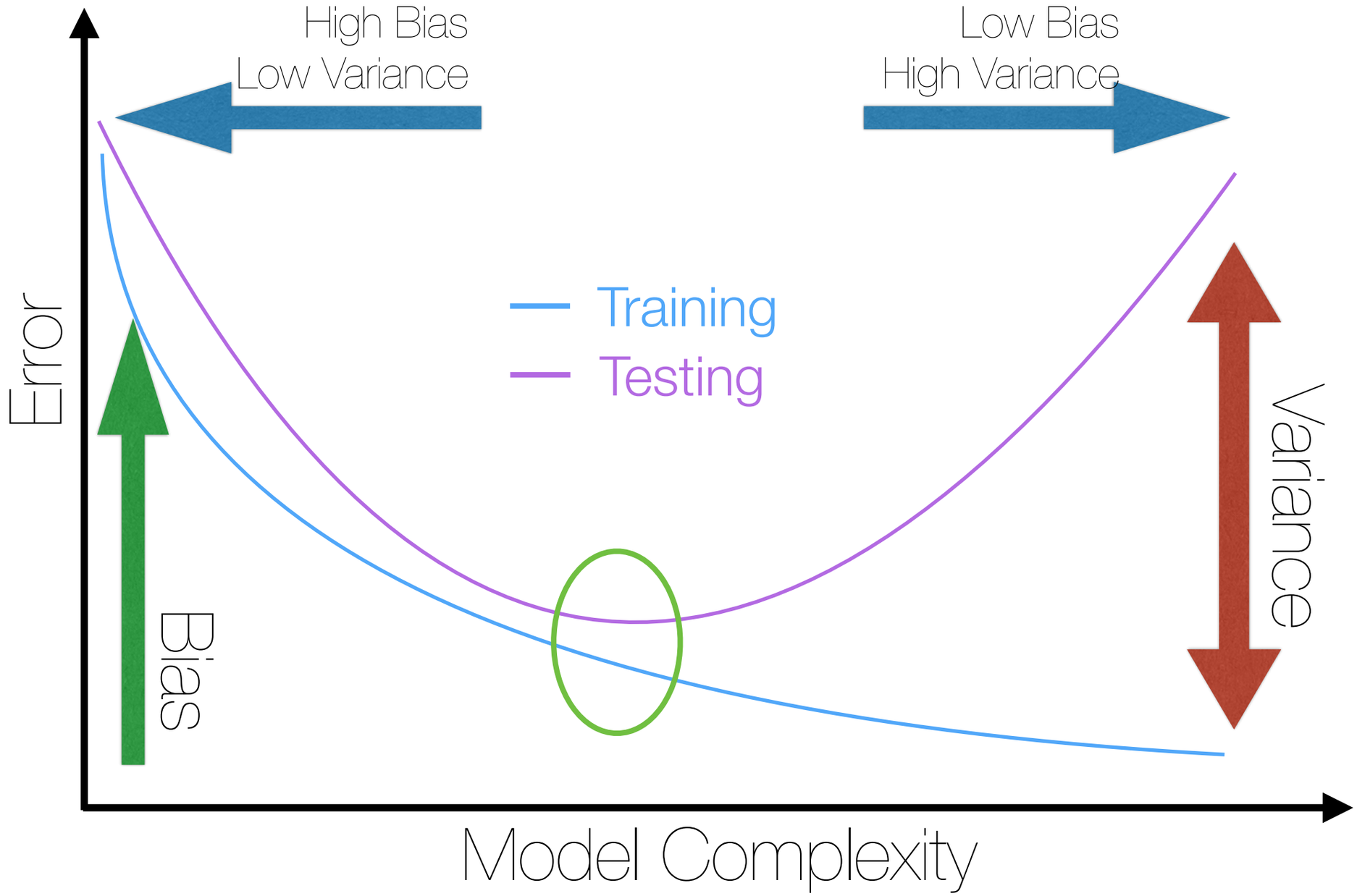
https://pytorch.org/docs/stable/optim.html

- PyTorch has a wide range of Optimizers available:
  - SGD - Stochastic Gradient Descent (with momentum)
  - RMSprop - Divide the gradient by a discounted moving average of previous gradients
  - Adam - SGD using adaptive estimation of higher-order moments.
  - Adadelta -SGD with an adaptive learning rate
  - Adagrad - SGD with parameter-specific learning rates
  - Adamax - Infinity norm Adam
- Each optimizer tries to deal with one or more limitations of the basic SGD algorithm that causes it to fail in specific cases
- Adam is a good general purpose choice

Learning Procedure



Bias-Variance Tradeoff



Code - NN with PyTorch

https://github.com/DataForScience/AdvancedNLP

Lesson 3:
Text Classification

Lesson 3.1:
Text Classification

Text Classification

- Our prototypical example will be Text Classification
- We'll learn how to classify IMDB reviews as Positive or Negative
- Reviews can have arbitrary lengths and vocabulary

"<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of norman and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

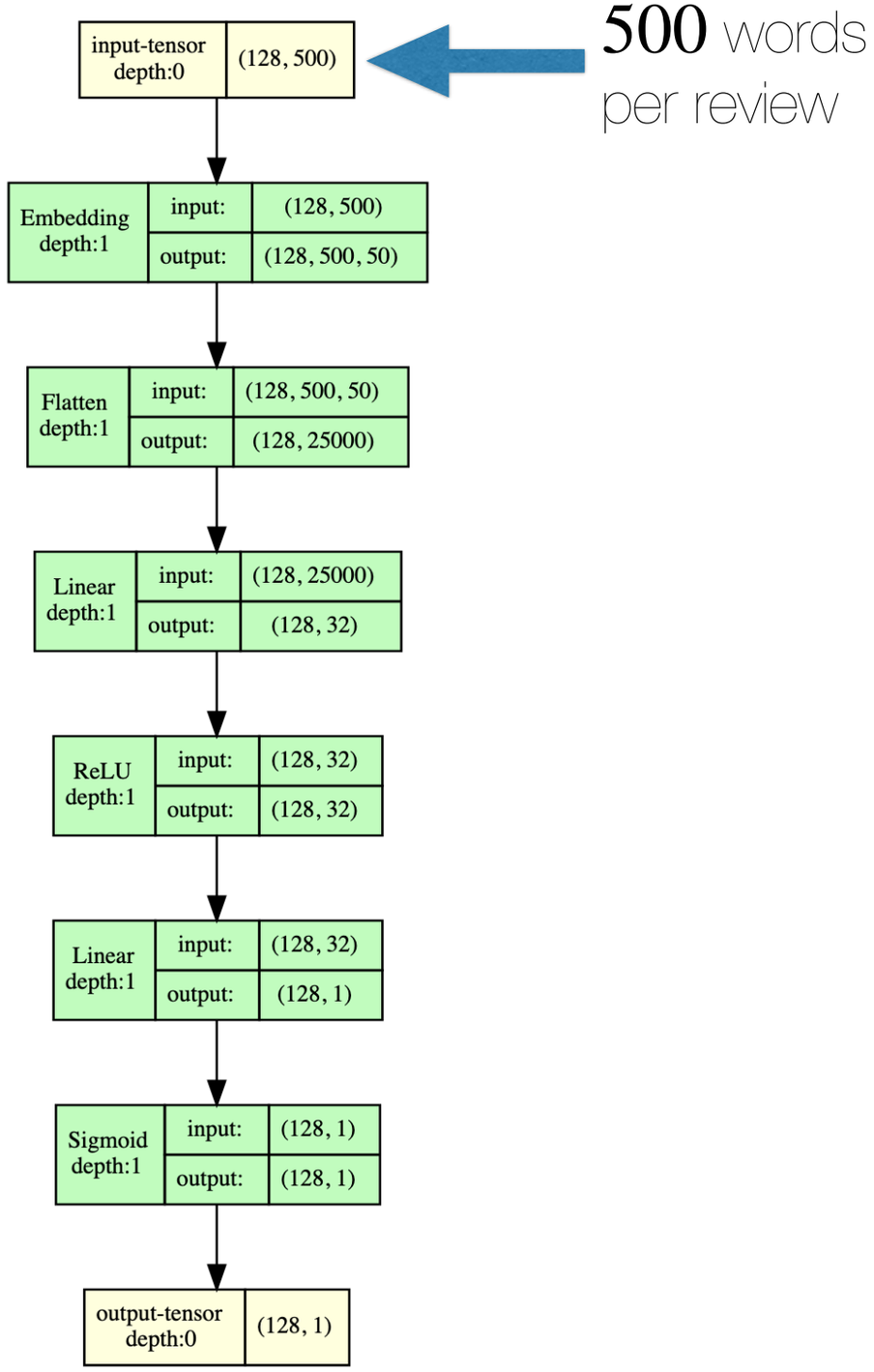
"<START> worst mistake of my life br br i picked this movie up at target for 5 because i figured hey it's sandler i can get some cheap laughs i was wrong completely wrong mid way through the film all three of my friends were asleep and i was still suffering worst plot worst script worst movie i have ever seen i wanted to hit my head up against a wall for an hour then i'd stop and you know why because it felt damn good upon bashing my head in i stuck that damn movie in the <UNK> and watched it burn and that felt better than anything else i've ever done it took american psycho army of darkness and kill bill just to get over that crap i hate you sandler for actually going through with this and ruining a whole day of my life"

- For convenience, we'll consider only the 10,000 most frequent words and truncate the reviews at 500 words.
- Removed words are marked by a special **<UNK>** token

Lesson 3.2:
Feed Forward Networks

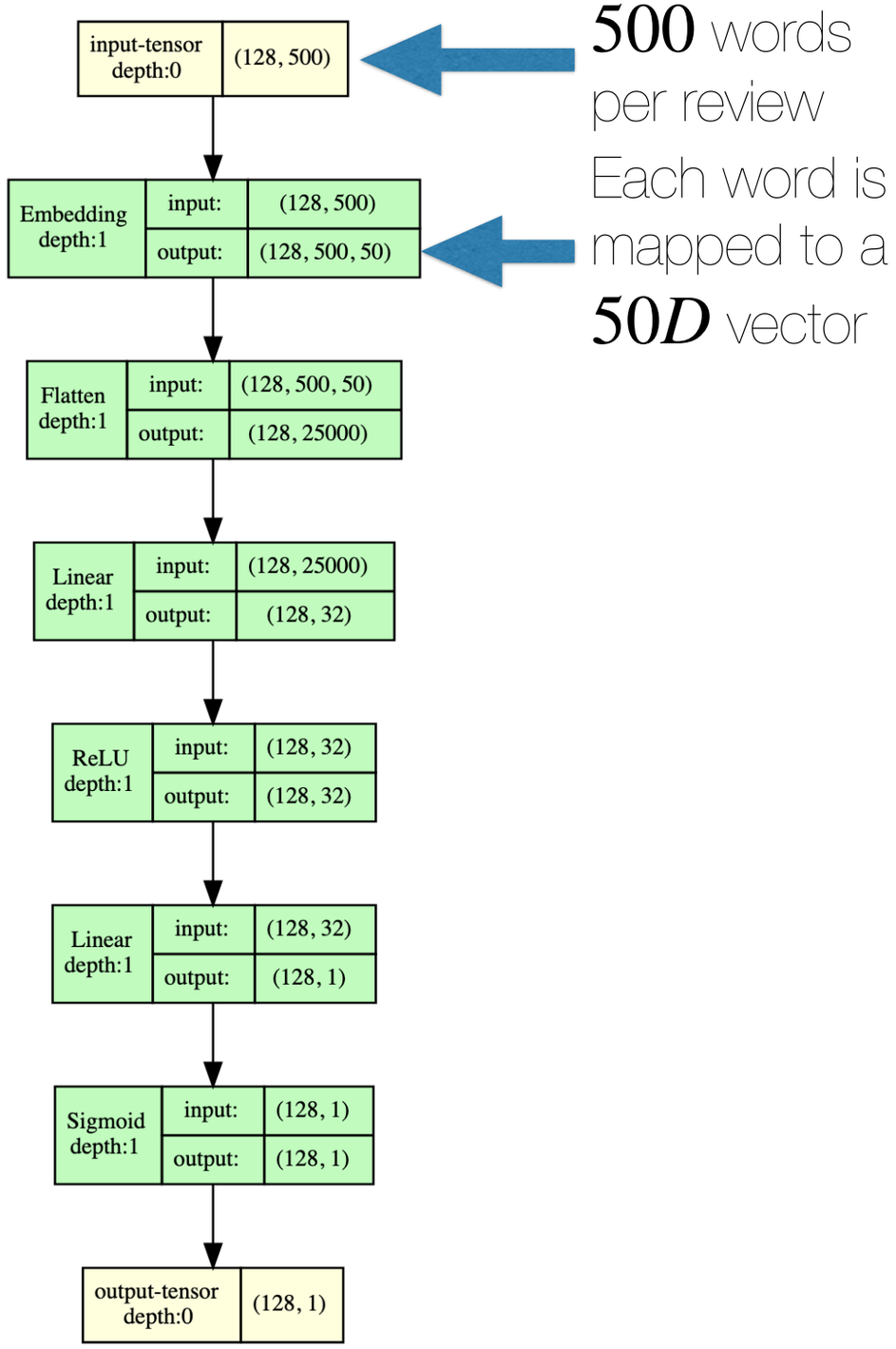
Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.



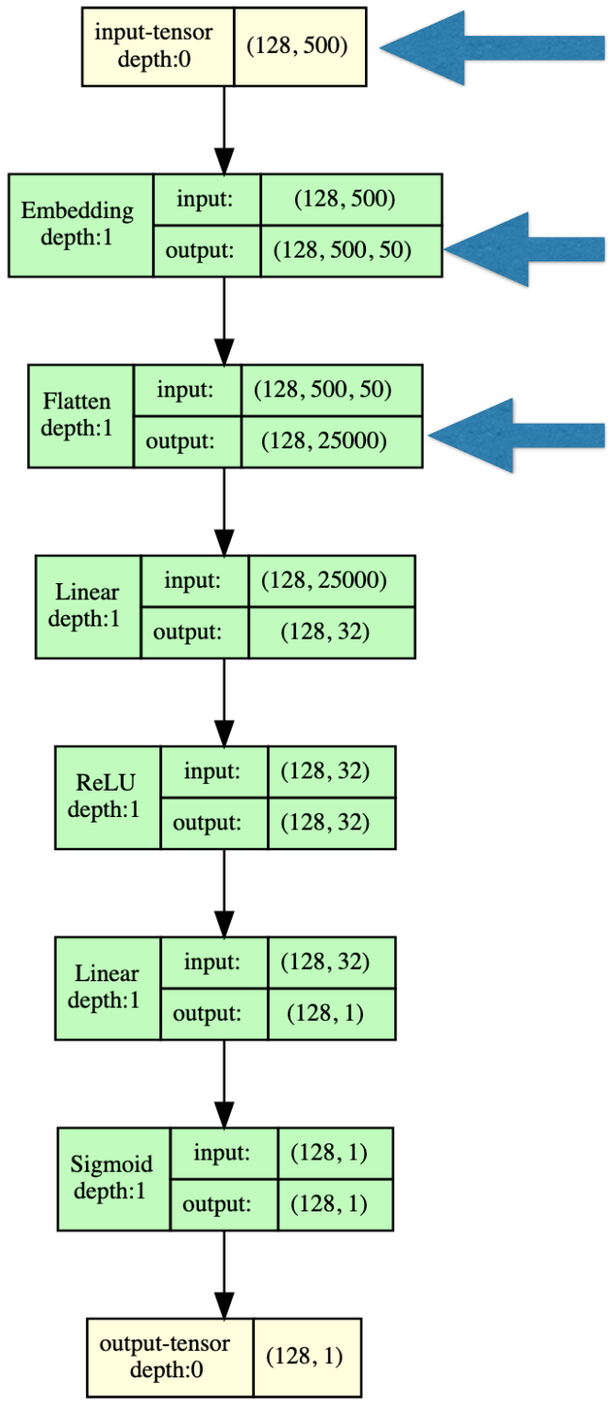
Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.



Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.



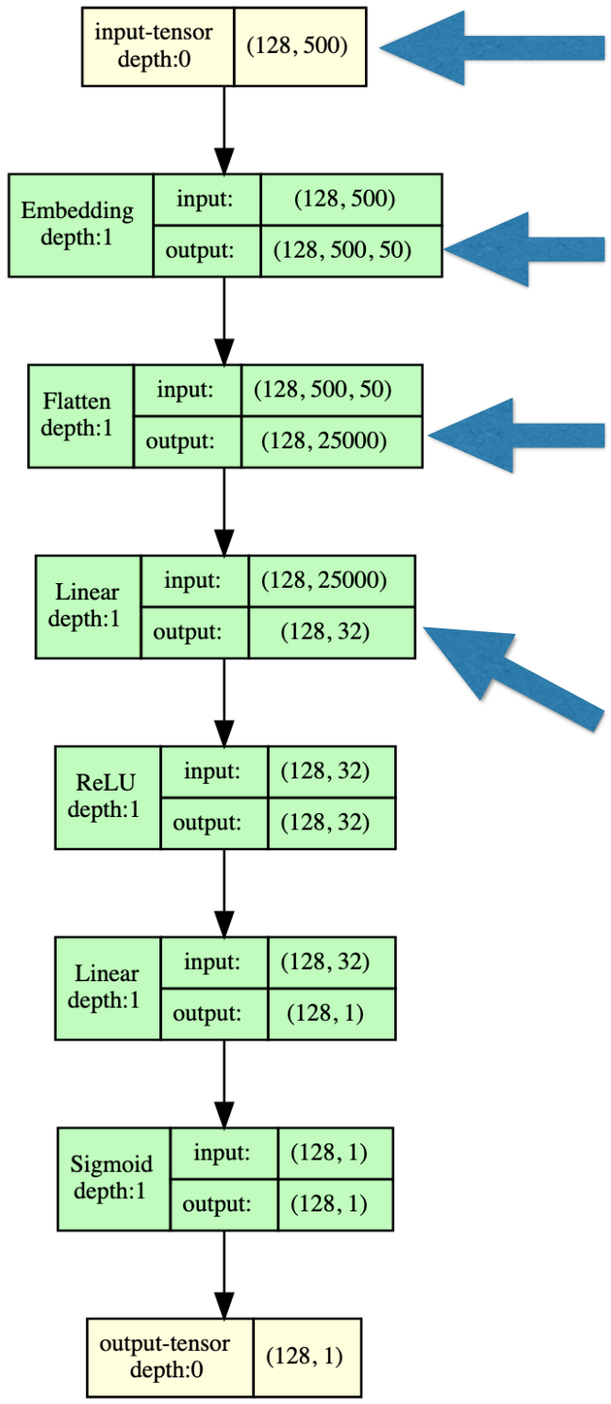
500 words per review

Each word is mapped to a $50D$ vector

Flatten the $3D$ tensor into a $2D$ matrix

Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.



500 words per review

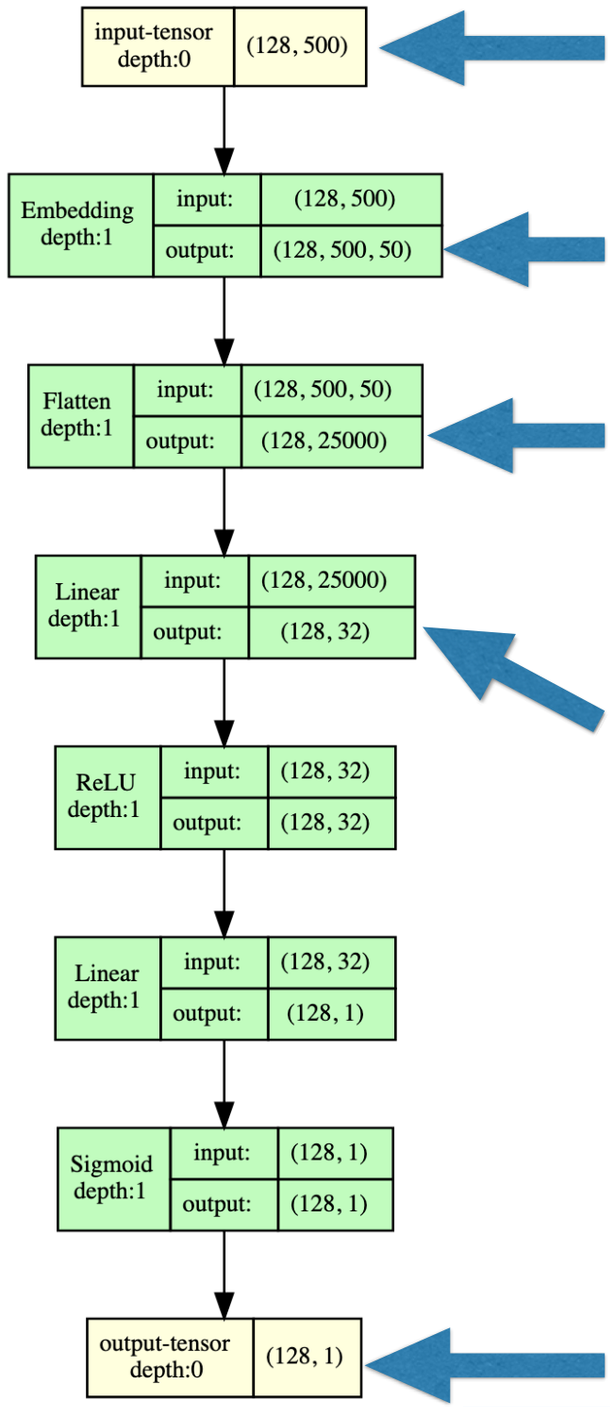
Each word is mapped to a $50D$ vector

Flatten the $3D$ tensor into a $2D$ matrix

The first linear layer reduces each instances from $25{,}000D$ to $32D$

Feed Forward Network

- Words are mapped to individual numerical IDs (in order of frequency), before being fed into the model.
- The first layer of the network is an Embedding layer that maps numerical ids to a dense low dimensional vector.



500 words per review

Each word is mapped to a $50D$ vector

Flatten the $3D$ tensor into a $2D$ matrix

The first linear layer reduces each instances from $25{,}000D$ to $32D$

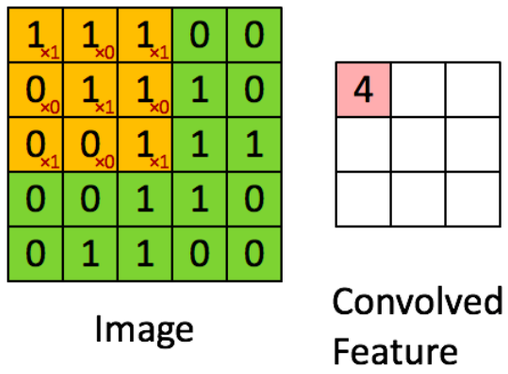
Batch size

Lesson 3.2:
Convolutional Neural Networks

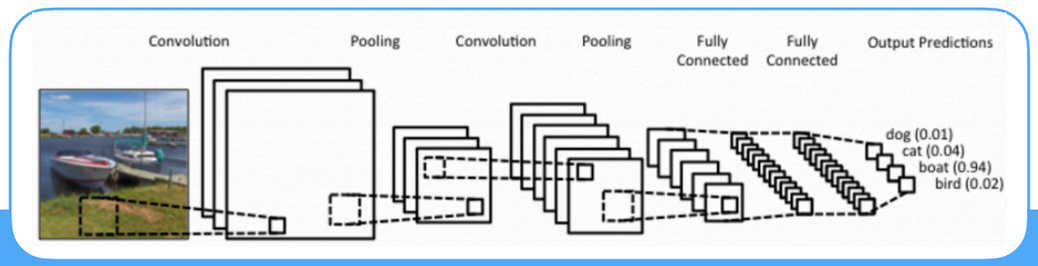
Convolutional Neural Network

http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

- Originally developed for Image Processing
- A Convolution Layer computes a value along a moving window as it slides through the image
- The output of the Convolution is smaller than the original image while still capturing relevant information

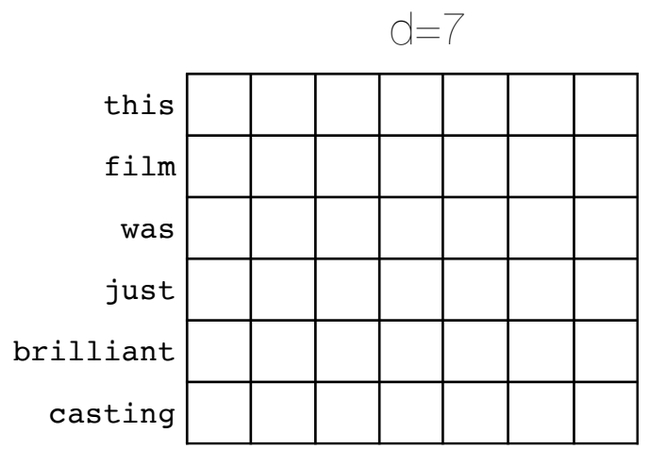


- Different convolution operations produce different effects on the original image:
  - Extract Edges, Blur, Emboss, etc
- Convolution layers are used to extract features from the original image

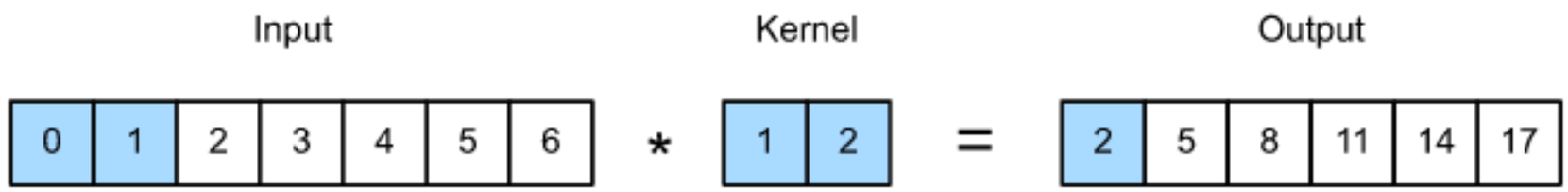


Convolutional Neural Network

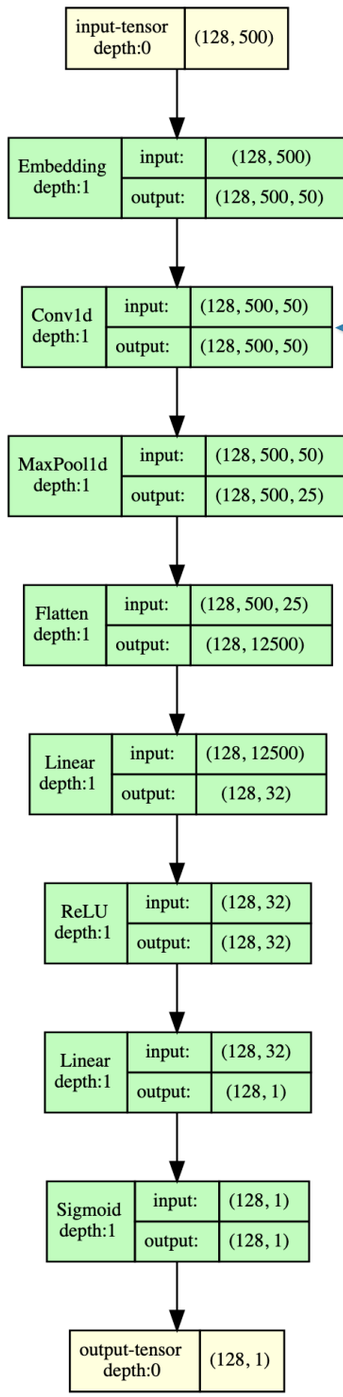
- Images are just arrays of numbers, just like our input matrices of words!



- The Kernel for each Conv1D layer can be learned by the Network itself

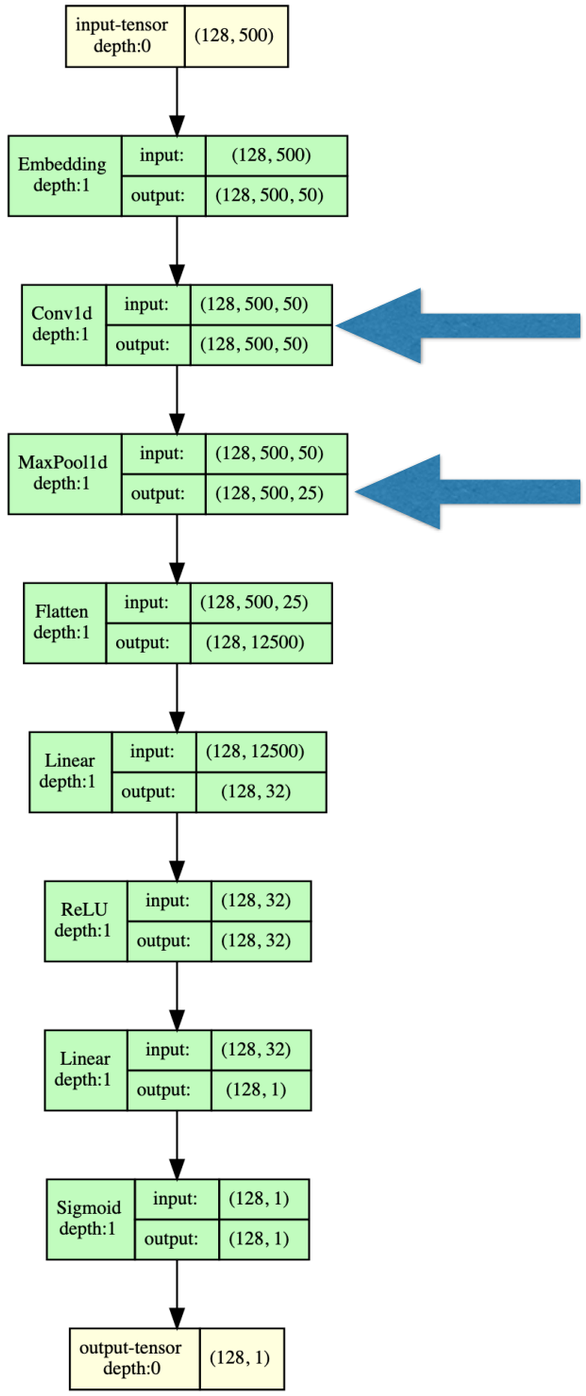


Convolutional Neural Network



The Conv1d layer preserves the dimensions

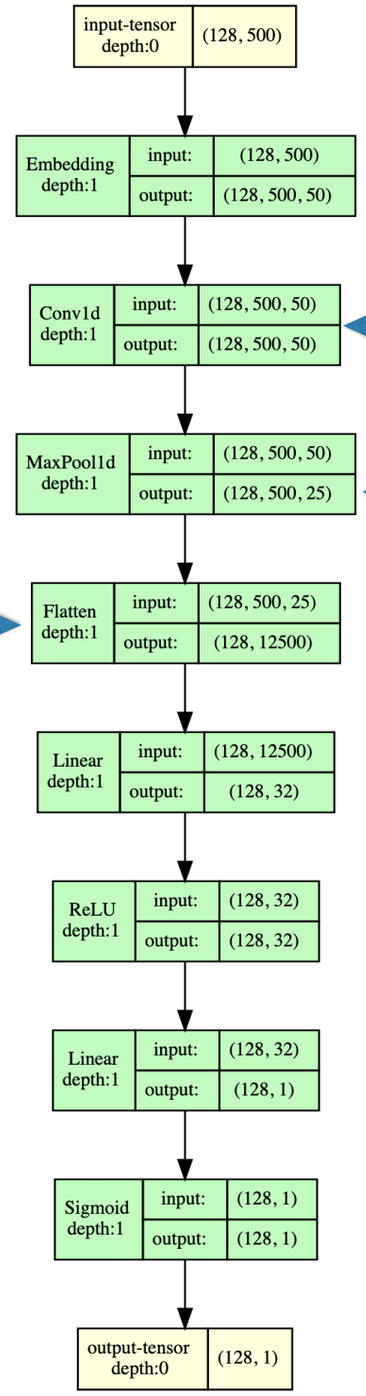
Convolutional Neural Network



The Conv1d layer preserves the dimensions

The MaxPool layers reduces the dimension from each embedding from $50D$ to $25D$

Convolutional Neural Network

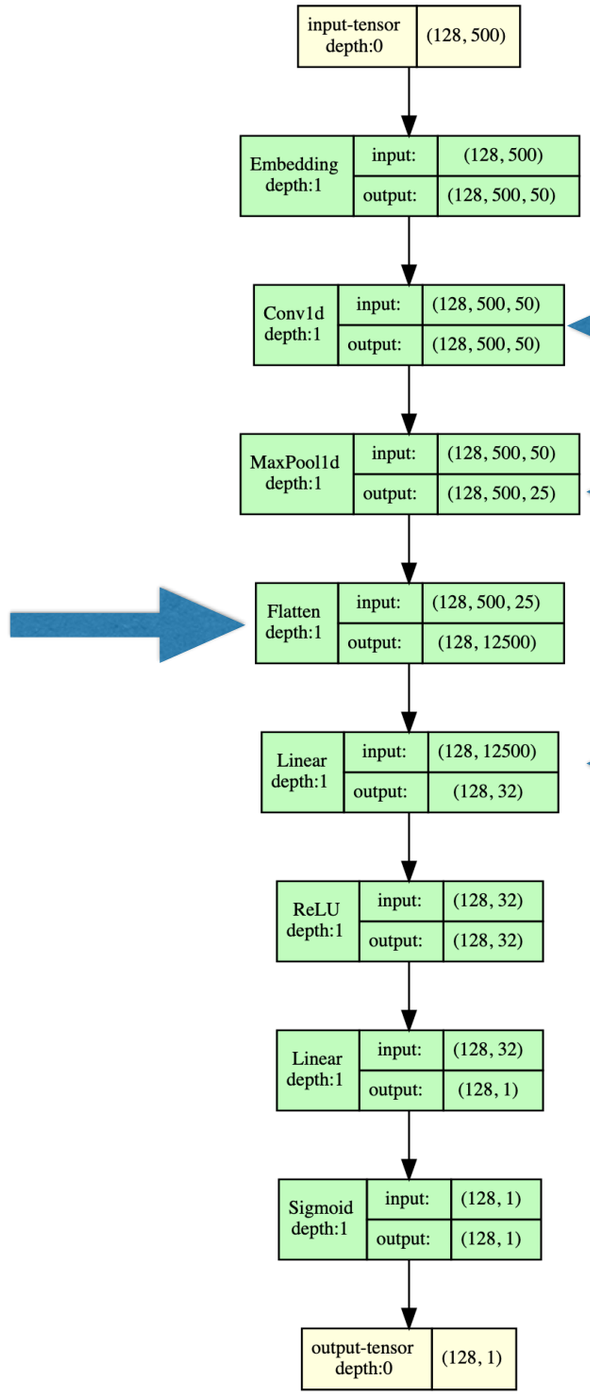


Flatten the input from a $3D$ tensor to a $2D$ matrix

The Conv1d layer preserves the dimensions

The MaxPool layers reduces the dimension from each embedding from $50D$ to $25D$

Convolutional Neural Network



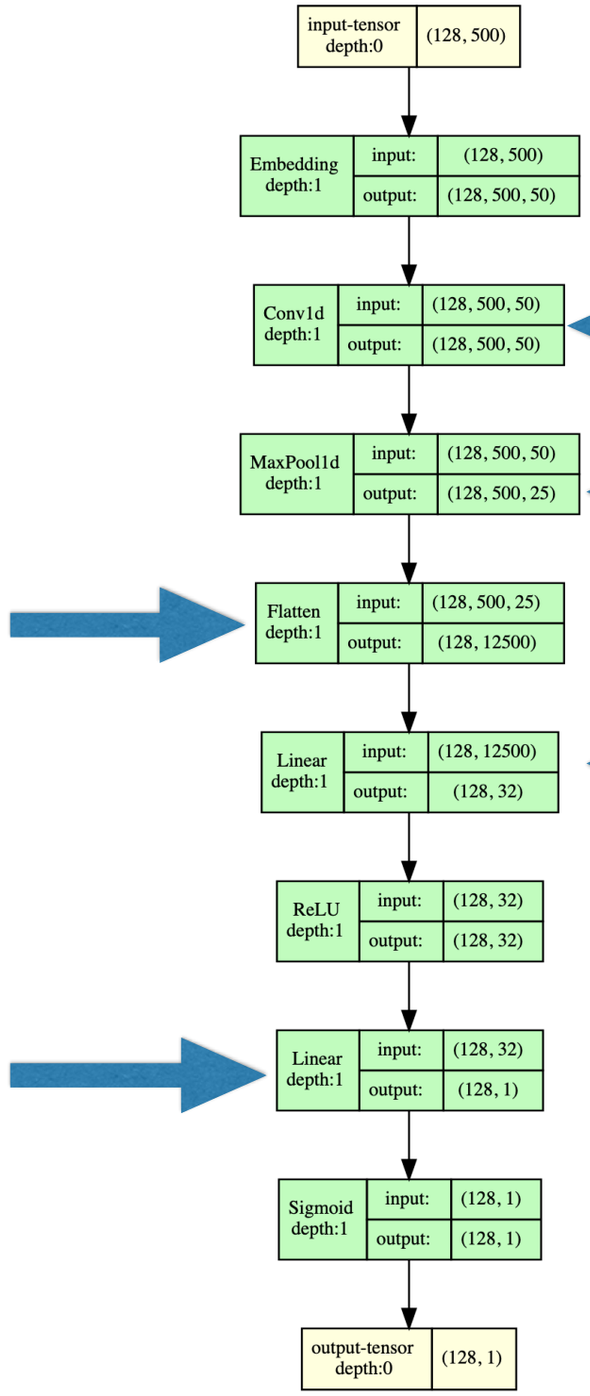
The Conv1d layer preserves the dimensions

The MaxPool layers reduces the dimension from each embedding from $50D$ to $25D$

Flatten the input from a $3D$ tensor to a $2D$ matrix

The first Linear layer reduces each instance from $12{,}500D$ to $32D$

Convolutional Neural Network



The Conv1d layer preserves the dimensions

The MaxPool layers reduces the dimension from each embedding from $50D$ to $25D$

Flatten the input from a $3D$ tensor to a $2D$ matrix

The first Linear layer reduces each instance from $12{,}500D$ to $32D$

The final Linear layer brings us down to just a single output

Code - Text Classification

https://github.com/DataForScience/AdvancedNLP

Lesson 4:
Word Embeddings

Lesson 4.1: Motivations

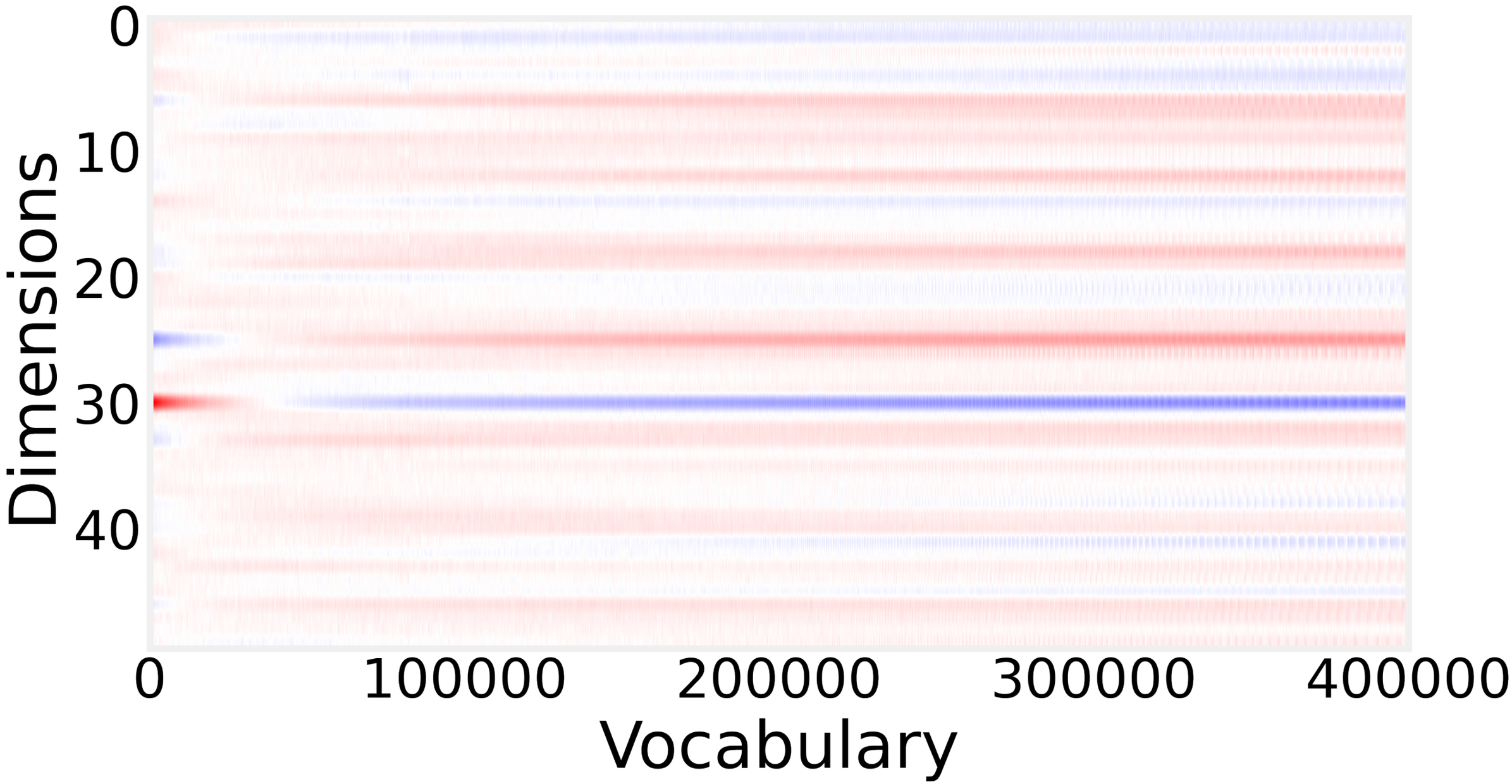
Word-Embeddings

- Word-Embeddings are simply vector representations of words.
- Typical vector representations are;
    - one-hot encoding
    - bag of words
    - TF/IDF
    - etc
- None of this representations include semantic information
- We already used an Embedding layer to map word IDs to fixed dimension vectors
- After training the network, the embedding layer contains meaningful representations of the input words

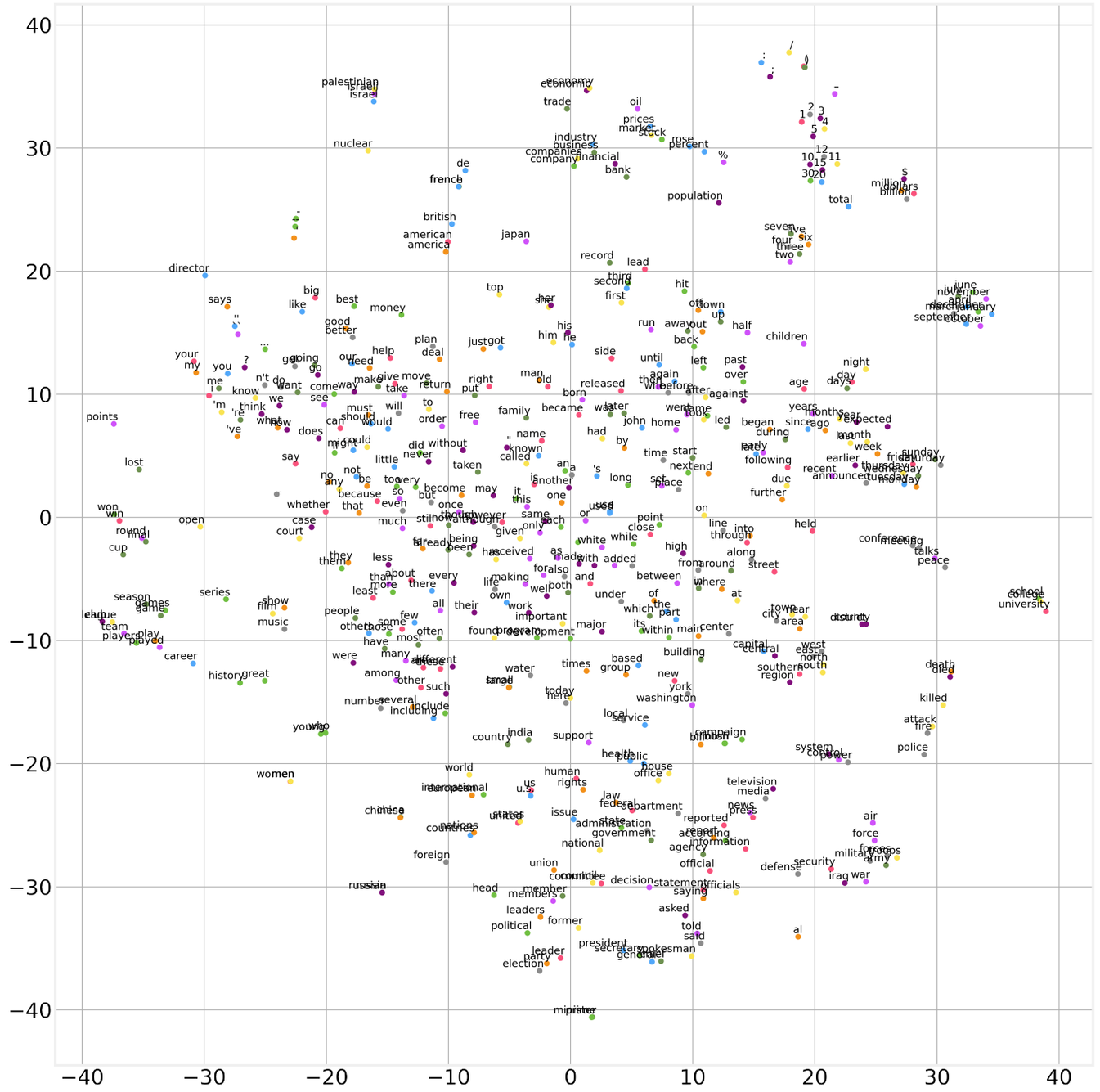
Word-Embeddings

- Different techniques were developed to generate vector representations that explicitly encode semantics and that can be reused. Two common ones are:
  - word2vec - Developed by Google using a simple Neural Network architecture.
  - GloVe - Developed by Stanford to explicitly take co-occurrences into account
- Each vector encodes information about the meaning of the word it's associated with
- Similarities between vectors match well to similarities between words
- Are useful ways of encoding words to input into a Neural Network
- Pre-trained vectors can be found online:
  - word2vec - https://sites.google.com/site/rmyeid/projects/polyglot
  - GloVe - https://github.com/stanfordnlp/GloVe

Word-Embeddings



Word-Embeddings



Lesson 4.2:
Skip-gram and Continuous Bag of Words

Word Embeddings

- The distributional hypothesis in linguistics states that words with similar meanings should occur in similar contexts.

- In other words, from a word we can get some idea about the context where it might appear.

___ ___ house __ _____.
___ ___ car __ _____.

$$\max p\left(C \mid w\right)$$

- And from the context we have some idea about possible words.

The red _____ is beautiful.
The blue _____ is old.

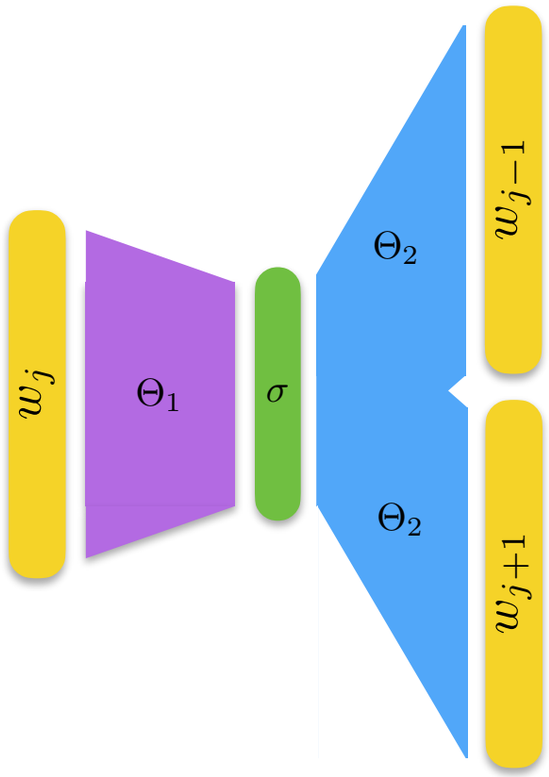
$$\max p\left(w \mid C\right)$$

word2vec

Mikolov 2013

Skipgram

$$\max p\left(C \mid w\right)$$



θ_1 word embeddings

θ_2 context embeddings

w_j one hot vector

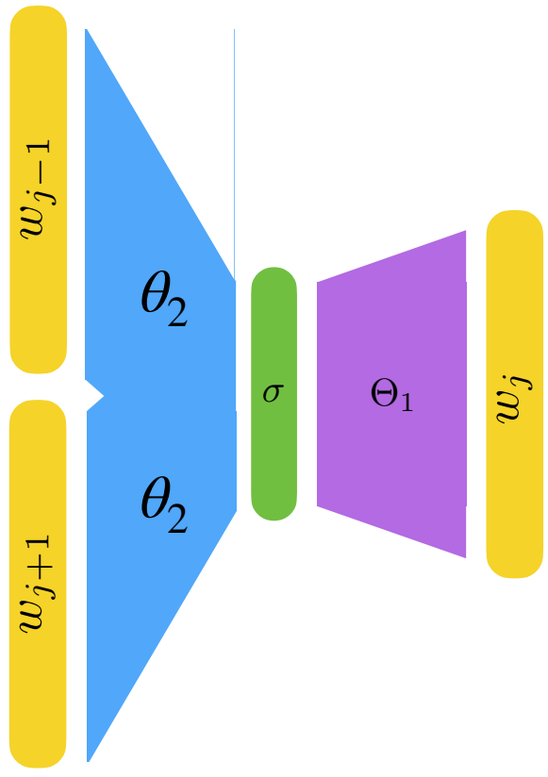
σ activation function

Word

Context

Continuous Bag of Words

$$\max p\left(w \mid C\right)$$



Context

Word

Variations

- Hierarchical Softmax:
  - Approximate the softmax using a binary tree
  - Reduces the number of calculations per training example from V to $\log_2 V$ and increases performance by orders of magnitude.
- Negative Sampling:
  - Under sample the most frequent words by removing them from the text before generating the contexts
  - Similar idea to removing stop-words — very frequent words are less informative.
  - Effectively makes the window larger, increasing the amount of information available for context

word2vec details

- The output of this neural network is deterministic:
    - If two words appear in the same context ("blue" vs "red", for e.g.), they will have similar internal representations in θ_1 and θ_2
    - θ_1 and θ_2 are vector embeddings of the input words and the context words respectively
- Words that are too rare are also removed.
- The original implementation had a dynamic window size:
    - for each word in the corpus a window size k' is sampled uniformly between 1 and k

Lesson 4.3:
Transfer Learning

Transfer Learning

https://learning.oreilly.com/library/view/java-deep-learning/9781788997454/de1d99a5-576d-45de-b77f-ee5563550894.xhtml

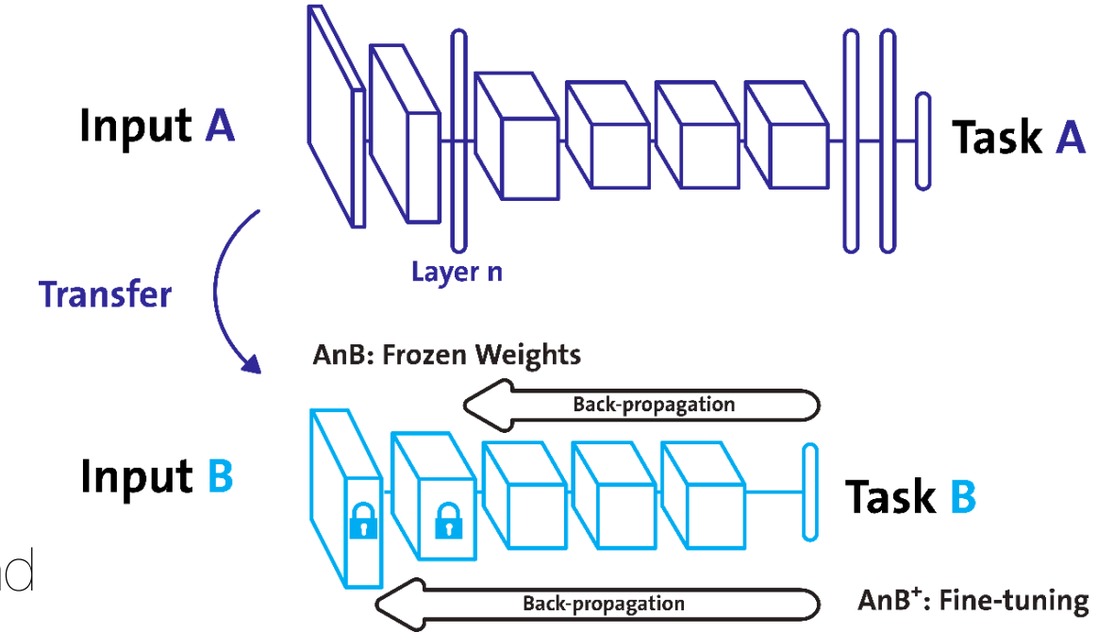
- Transfer Learning is the process of putting the knowledge learned by one network to use in another. Like when you make use concepts from a different field to solve a problem

- In a more general case, entire layers of a Deep Learning Network that was trained for Task A can be repurposed for use in Task B without any modifications

- This is particularly common in large scale systems that are extremely expensive (in both time and monty) to train from scratch

- We can take advantage of the huge amounts of work put in by Google, Stanford, etc to generate high quality embeddings to save time and effort when developing our models

- In the case of small systems with relatively few training examples, specially trained embeddings tend to over perform these high quality ones.



Code - Word Embeddings

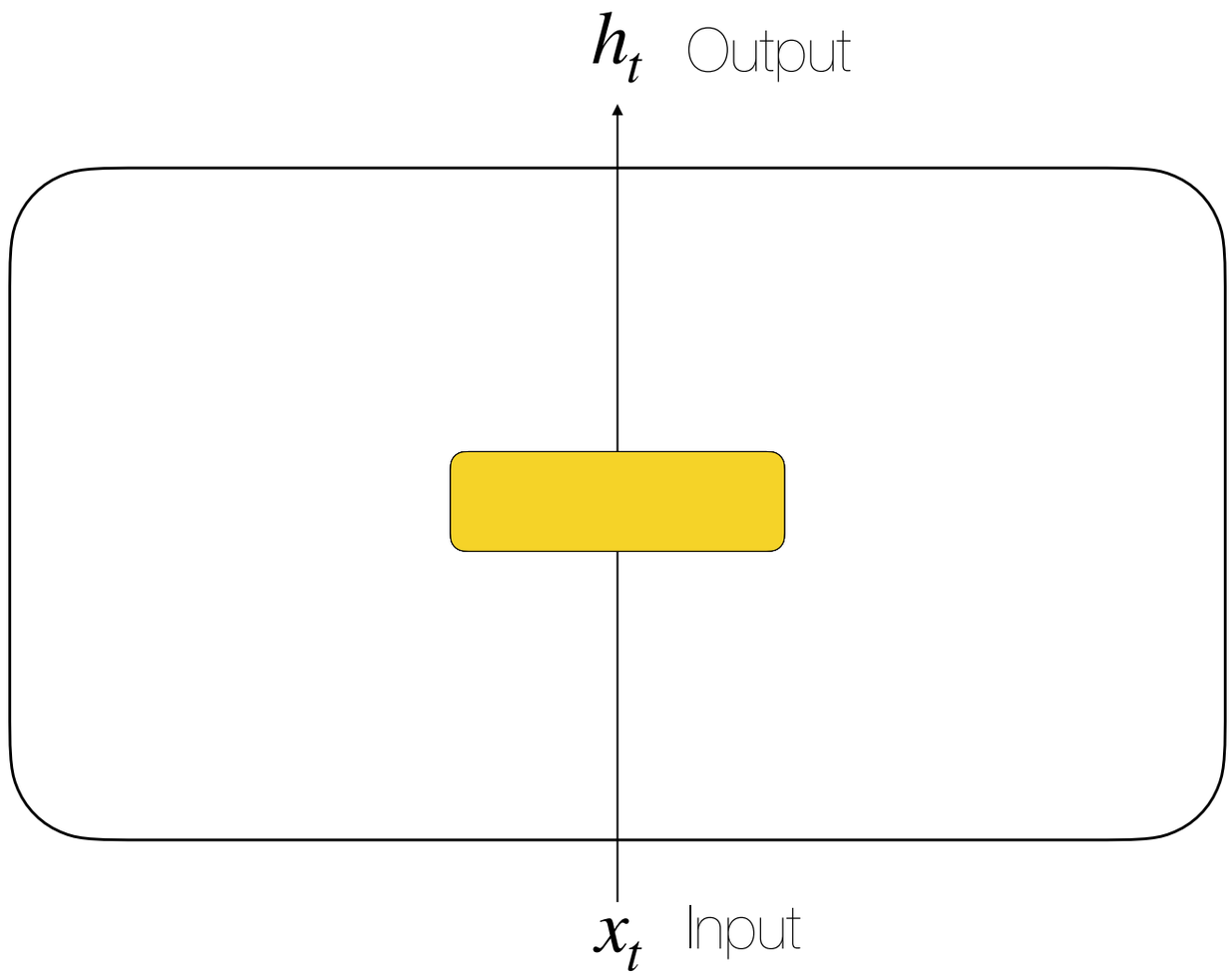
https://github.com/DataForScience/AdvancedNLP

Lesson 5:
Sequence Modeling

Lesson 5.1:
Recurrent Neural Networks

Feed Forward Networks

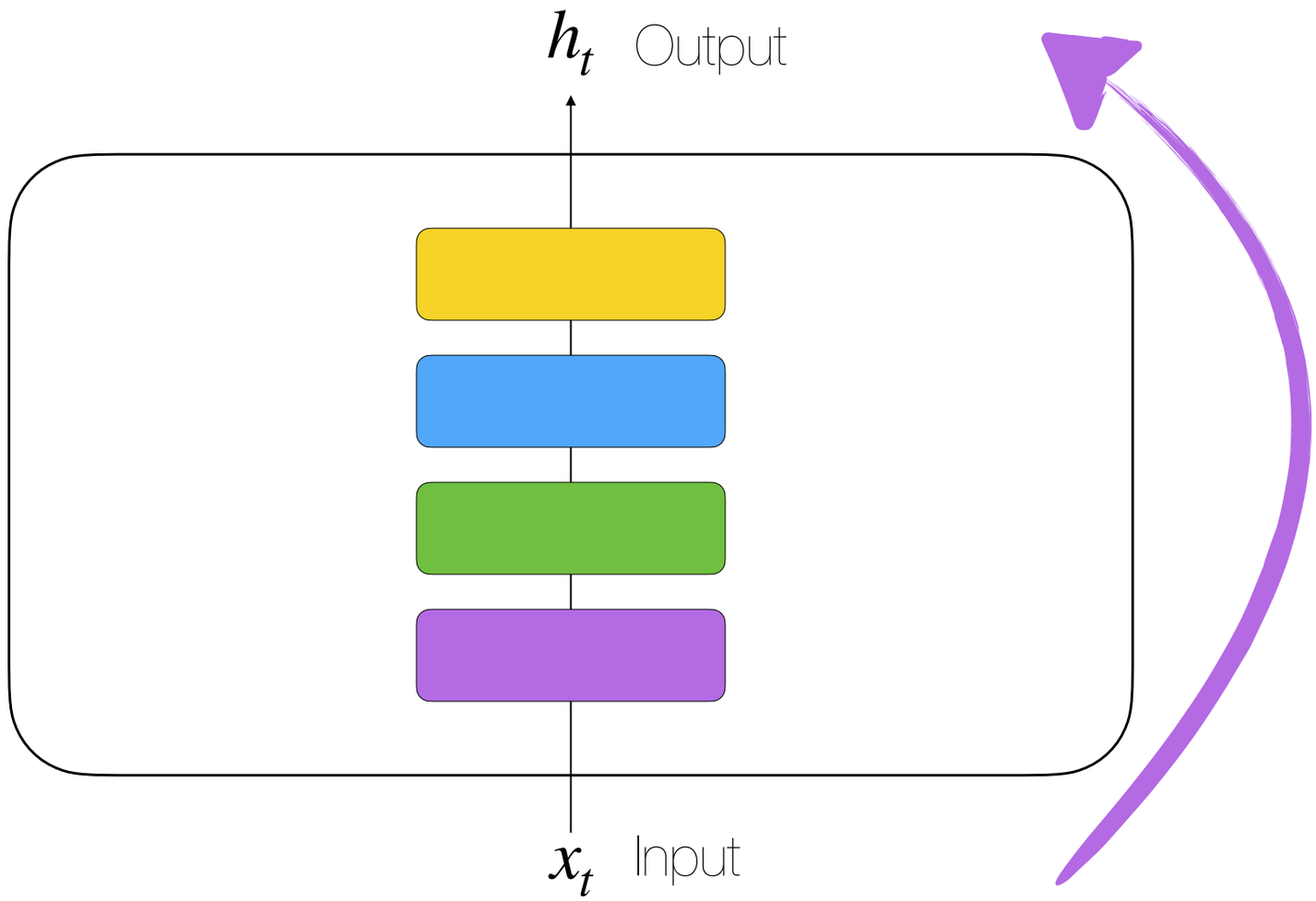
The networks we've seen so far operate in a linear fashion



$$h_t = f\left(x_t\right)$$

Feed Forward Networks

The networks we've seen so far operate in a linear fashion

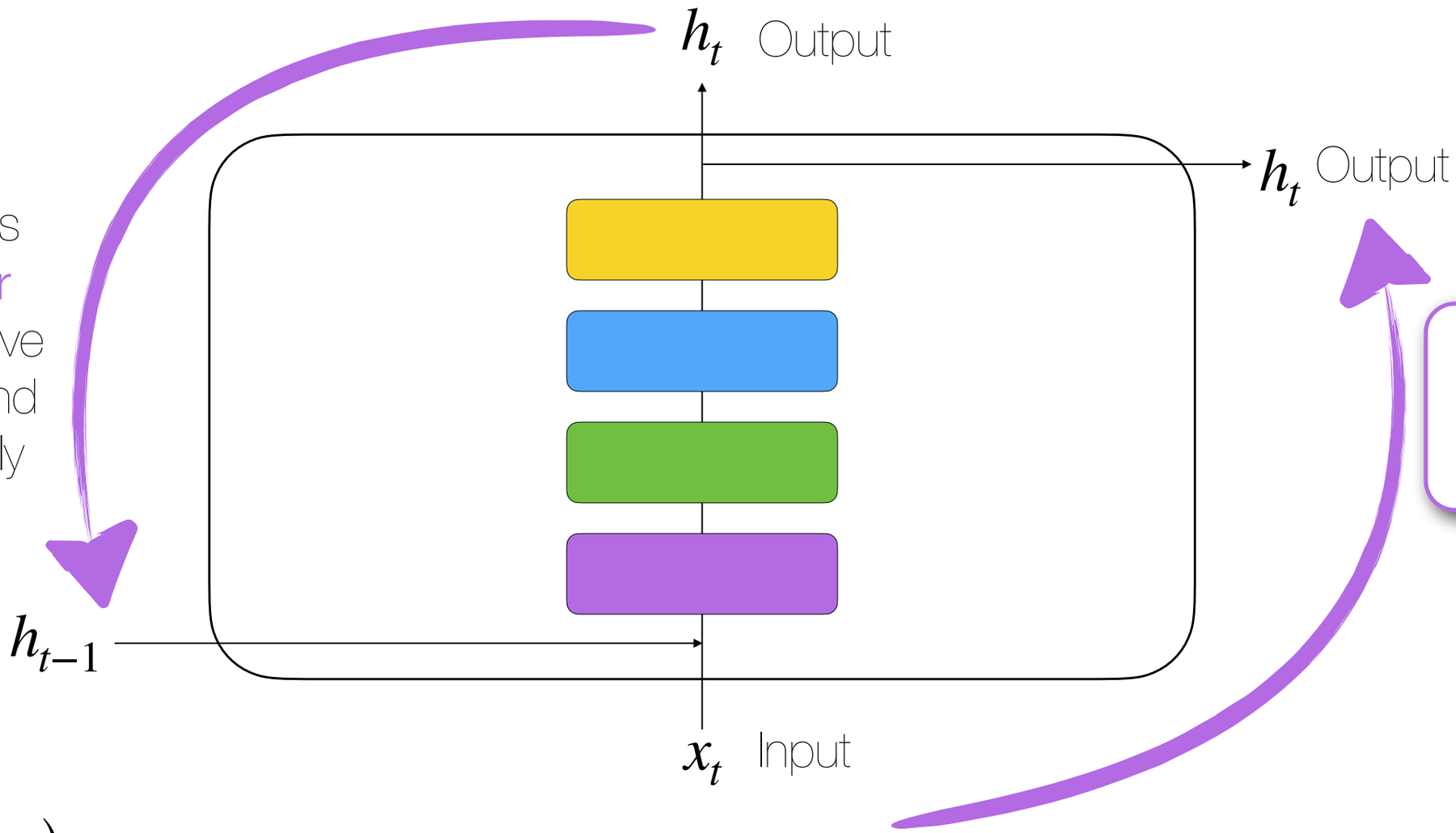


Information Flow

$$h_t = f\left(x_t\right)$$

Recurrent Neural Network (RNN)

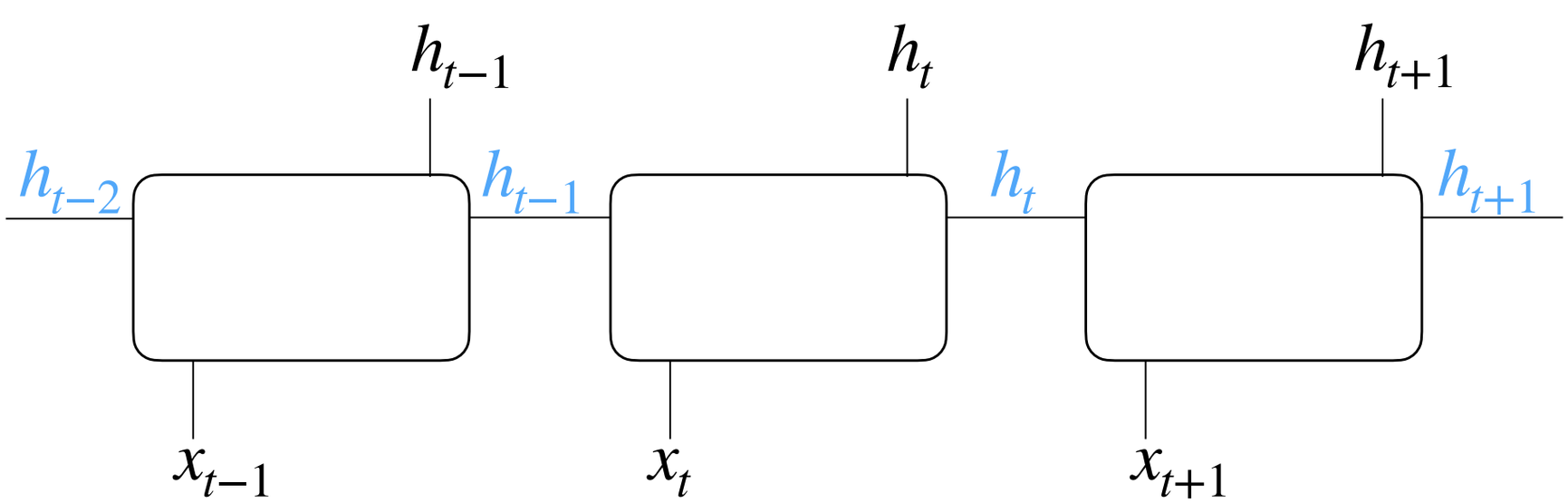
RNNs allow us to remember information we've seen before and act accordingly



$$h_t = f\left(x_t, h_{t-1}\right)$$

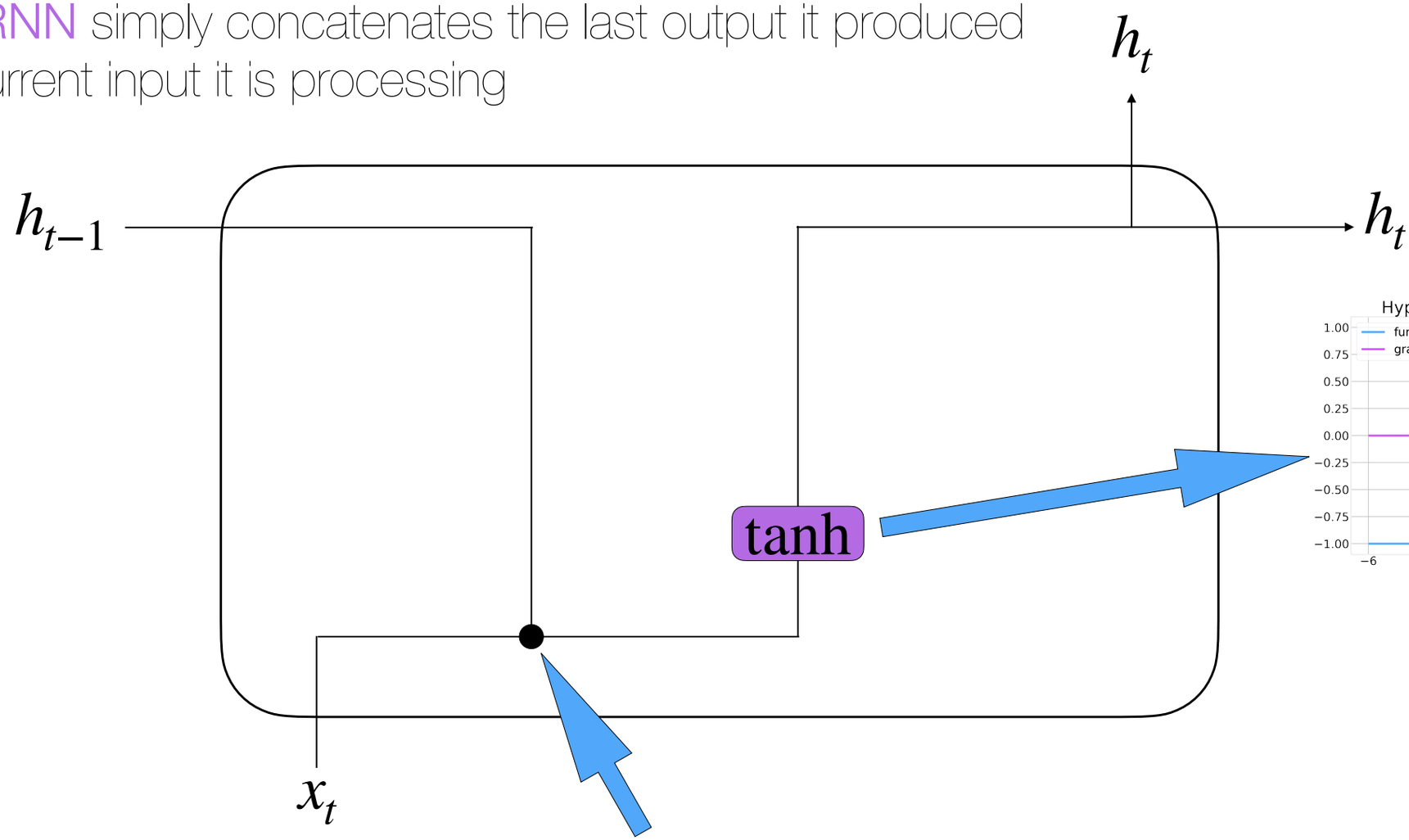
Recurrent Neural Network (RNN)

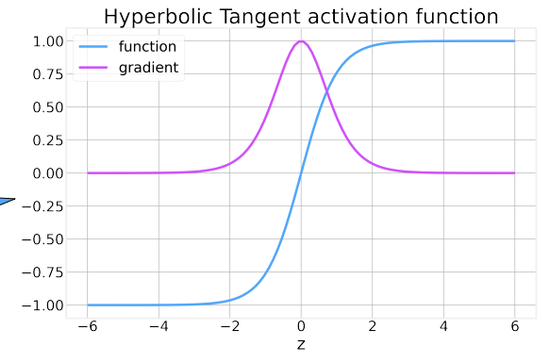
- Each output depends (implicitly) on all previous outputs.
- RNNs are particularly useful to model sequential systems, like time series, audio or streams of text
- Input sequences generate output sequences (seq2seq)



Recurrent Neural Network (RNN)

- SimpleRNN simply concatenates the last output it produced to the current input it is processing





$$h_t = \tanh\left(Wh_{t-1} + Ux_t\right)$$

Recurrent Neural Network (RNN)

Vocabulary size

```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): RNN(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

Recurrent Neural Network (RNN)

```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): RNN(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

Vocabulary size

The RNN takes the vectors in directly

Recurrent Neural Network (RNN)

```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): RNN(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

Vocabulary size

The RNN takes the vectors in directly

Output two values to make it more generic

Lesson 5.2:
Gated Recurrent Unit

Gated Recurrent Unit (GRU)

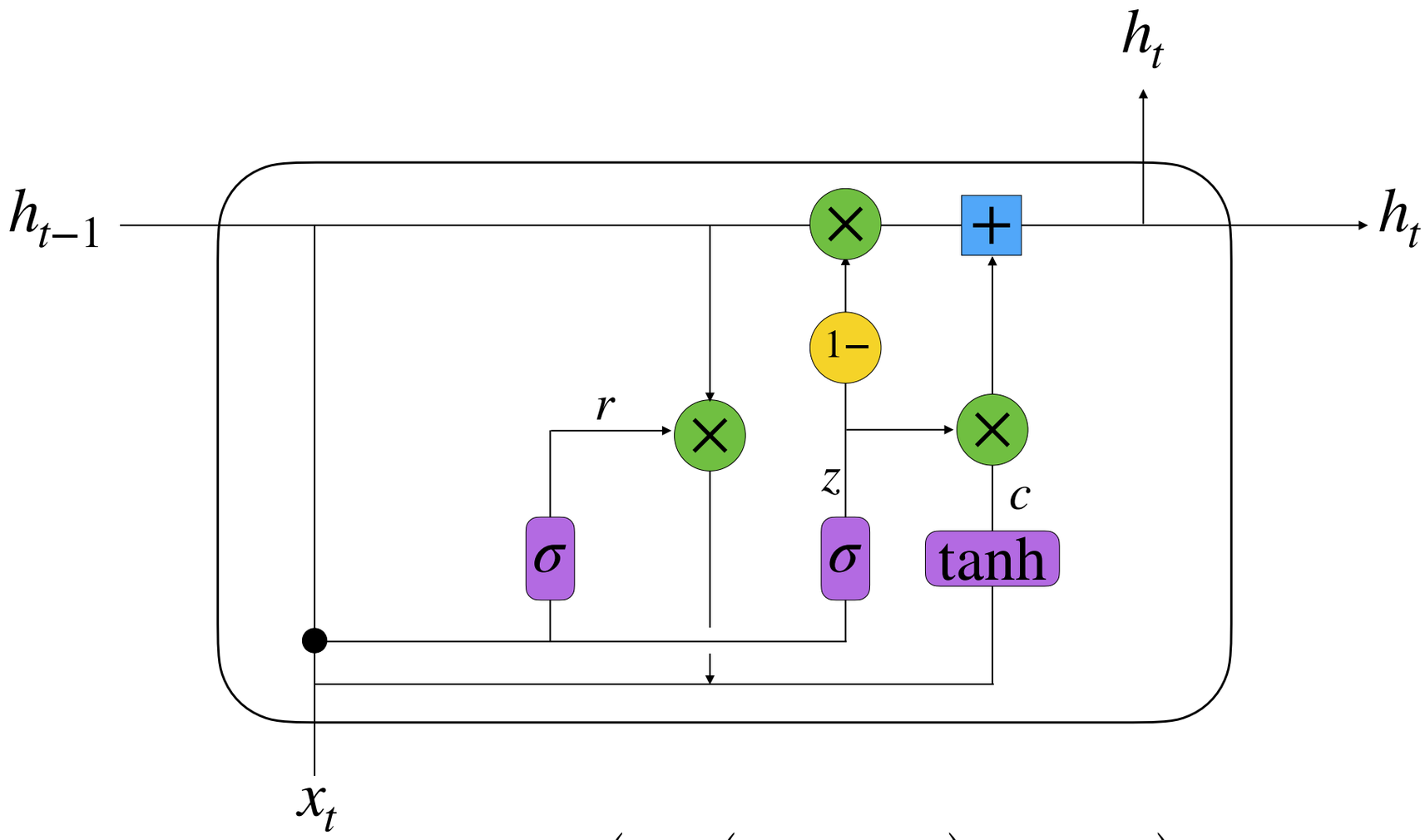
- Introduced in 2014 by K. Cho
- Meant to solve the Vanishing Gradient Problem
- Can be considered as a simplification of LSTMs
- Similar performance to LSTM in some applications, better performance for smaller datasets.

Gated Recurrent Unit (GRU)

Element wise addition

Element wise multiplication

1 minus the input



$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

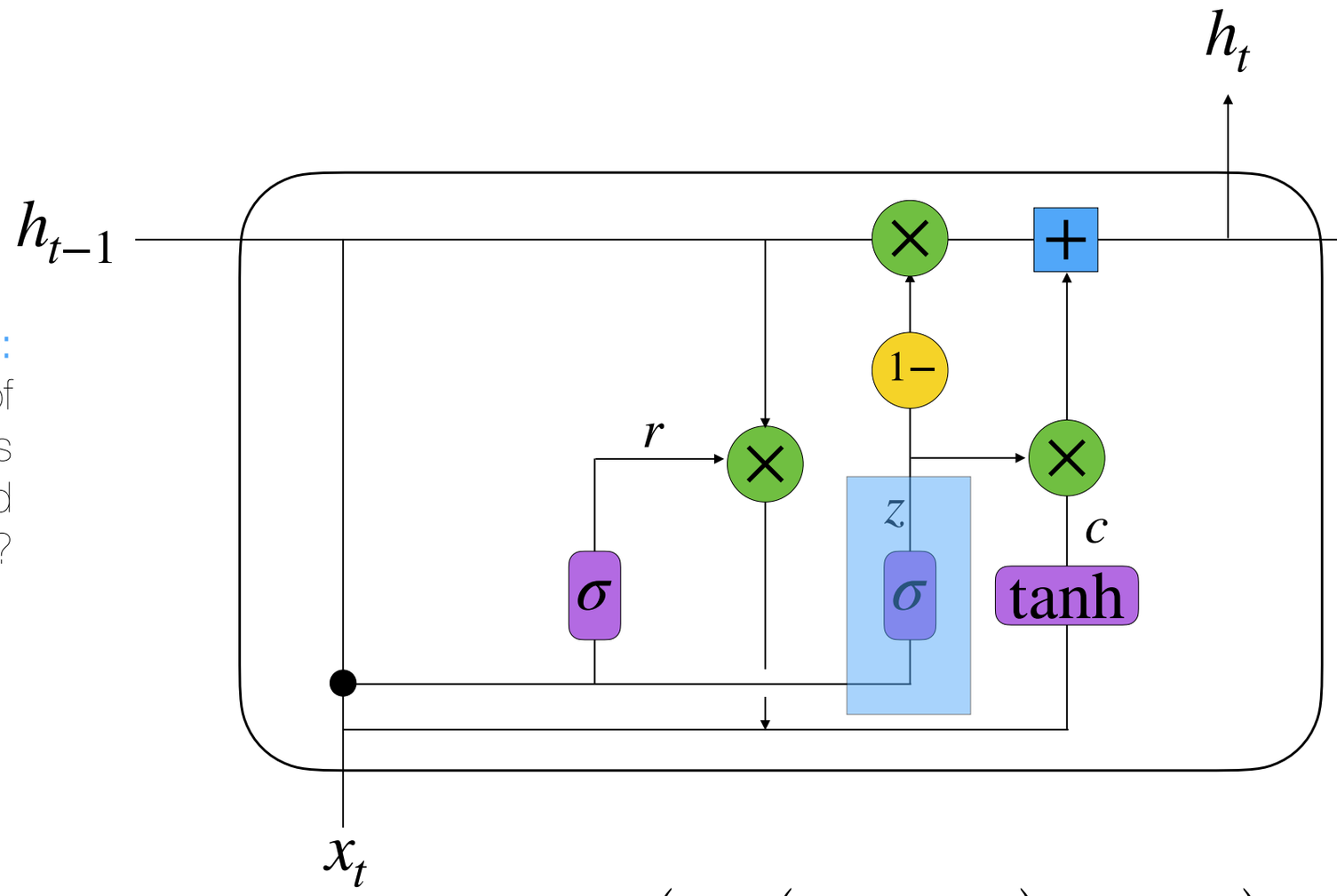
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

Gated Recurrent Unit (GRU)

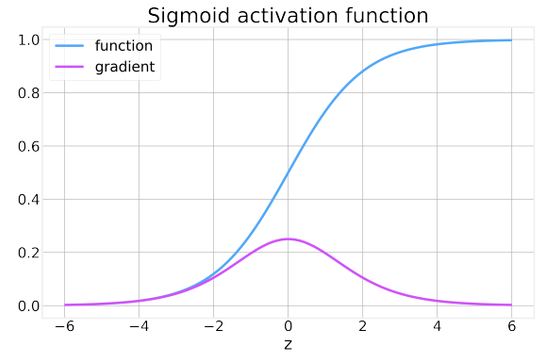
Element wise addition

Element wise multiplication

1 minus the input



Update gate:
How much of the previous state should be kept?



$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

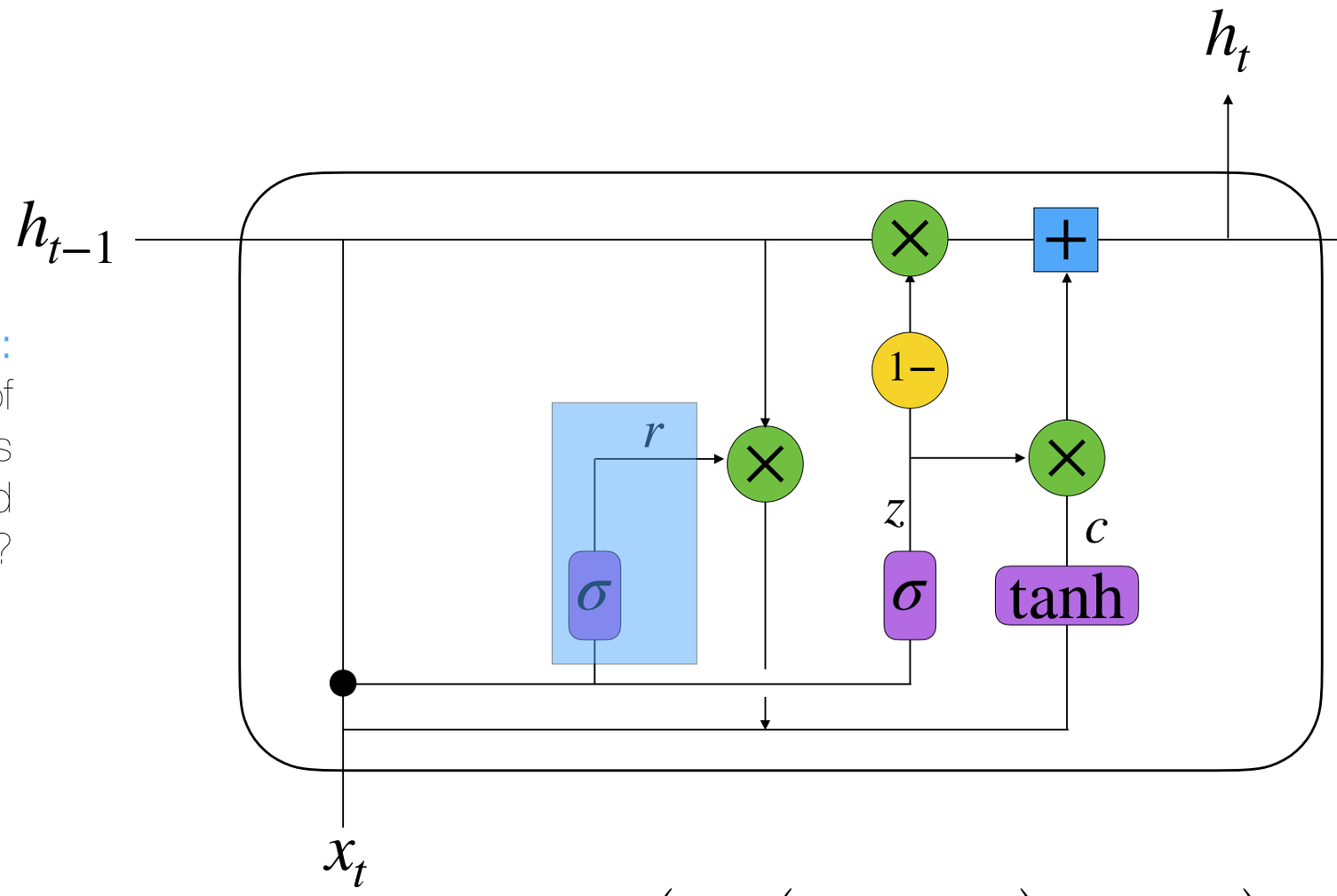
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

Gated Recurrent Unit (GRU)

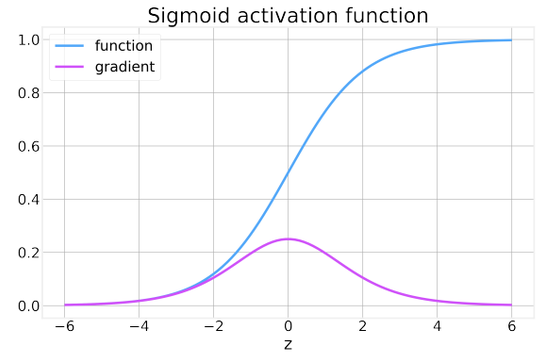
Element wise addition

Element wise multiplication

1 minus the input



Reset gate:
How much of the previous output should be removed?



$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

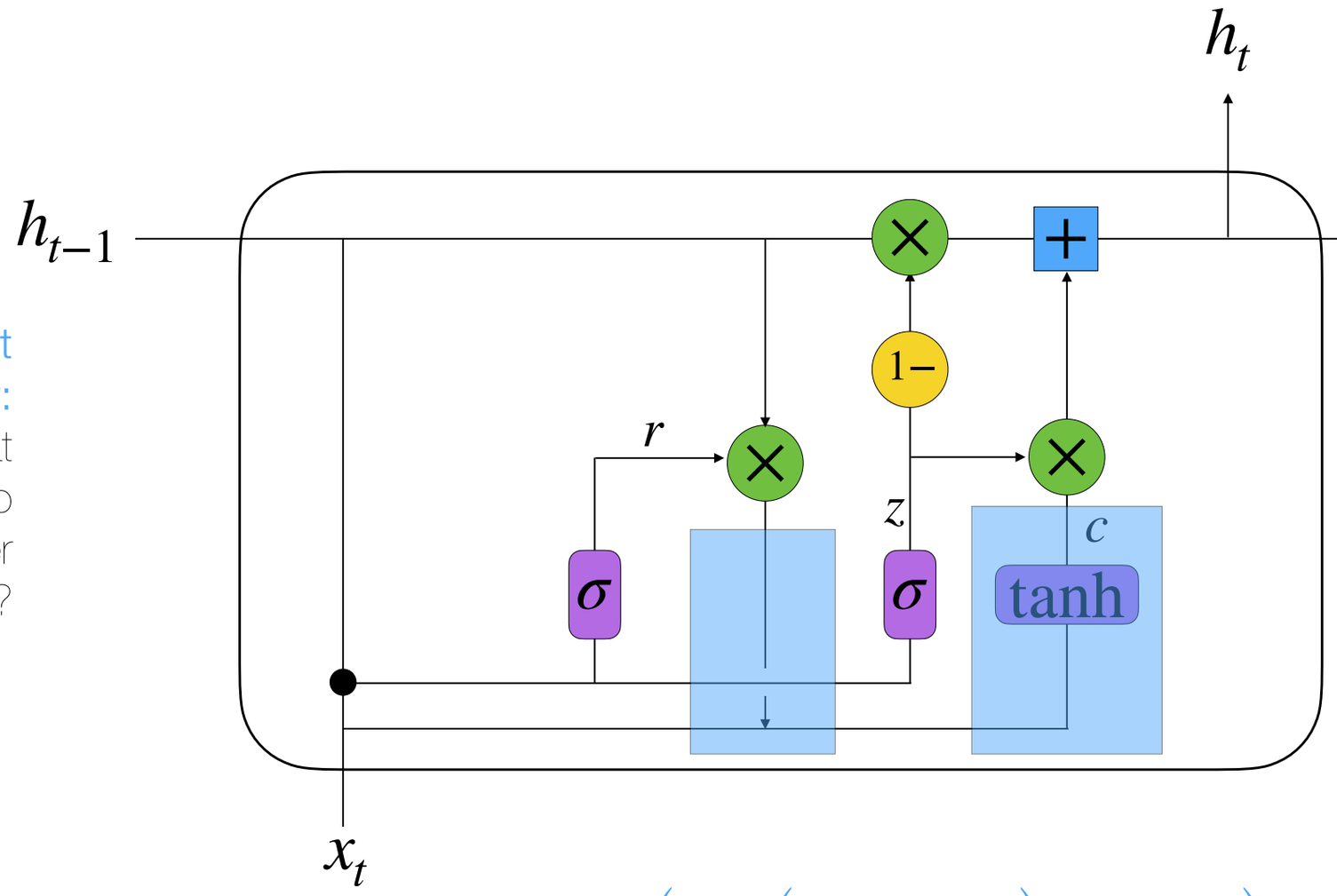
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

Gated Recurrent Unit (GRU)

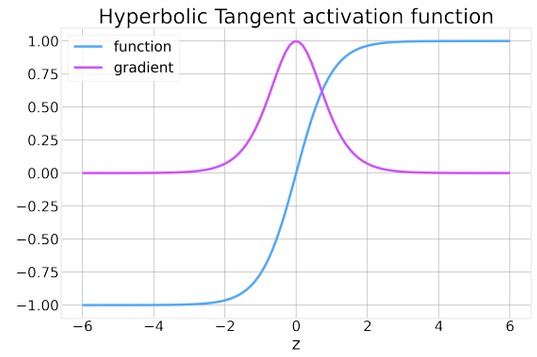
Element wise addition

Element wise multiplication

1− 1 minus the input



Current memory: What information do we remember right now?



$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

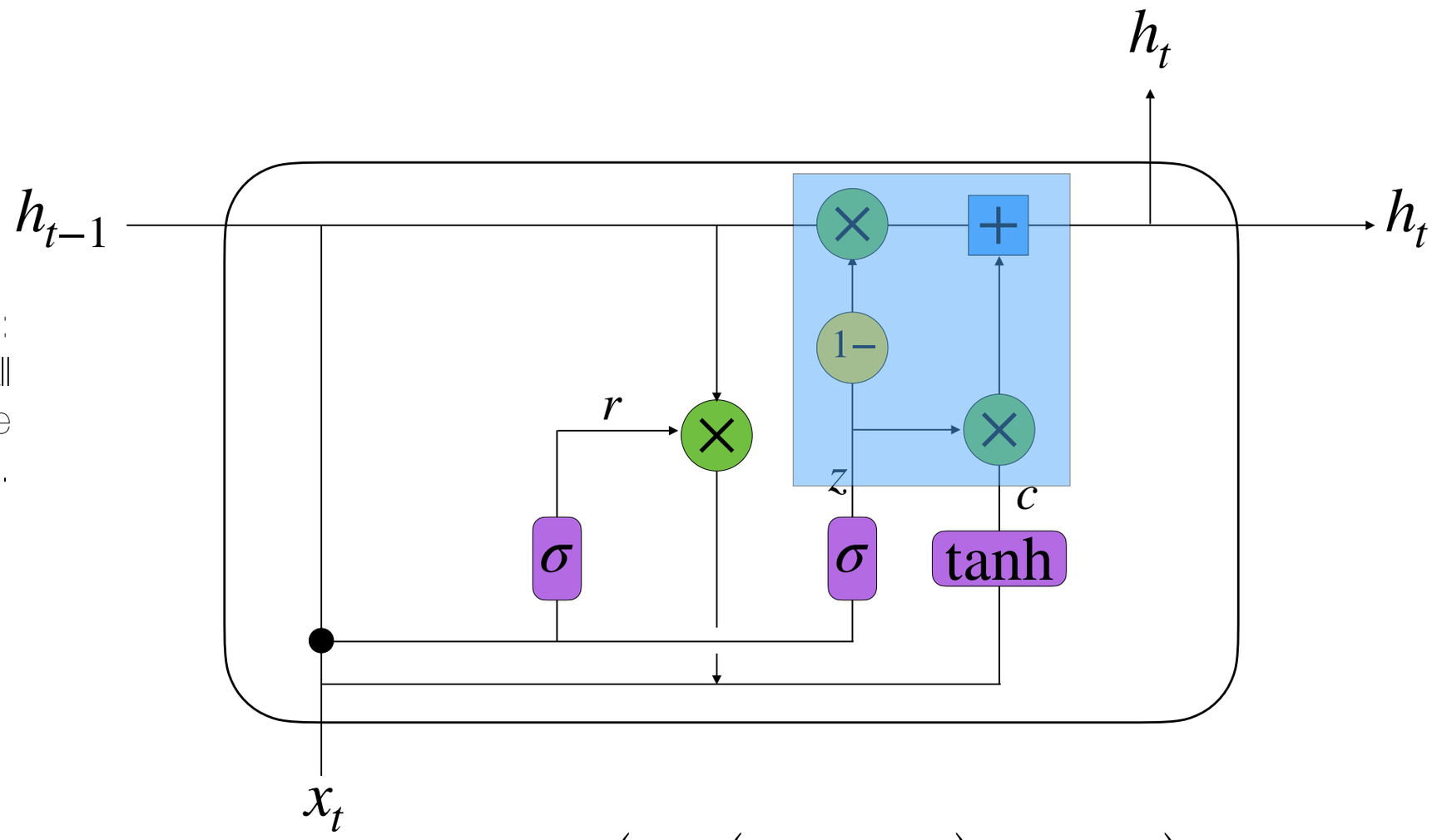
$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

Gated Recurrent Unit (GRU)

Element wise addition

Element wise multiplication

1 minus the input



Output: Combine all available information.

$$z = \sigma\left(W_z h_{t-1} + U_z x_t\right)$$

$$r = \sigma\left(W_r h_{t-1} + U_r x_t\right)$$

$$c = \tanh\left(W_c\left(h_{t-1} \otimes r\right) + U_c x_t\right)$$

$$h_t = (z \otimes c) + \left((1 - z) \otimes h_{t-1}\right)$$

Gated Recurrent Unit (GRU)

```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): GRU(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

Gated Recurrent Unit (GRU)

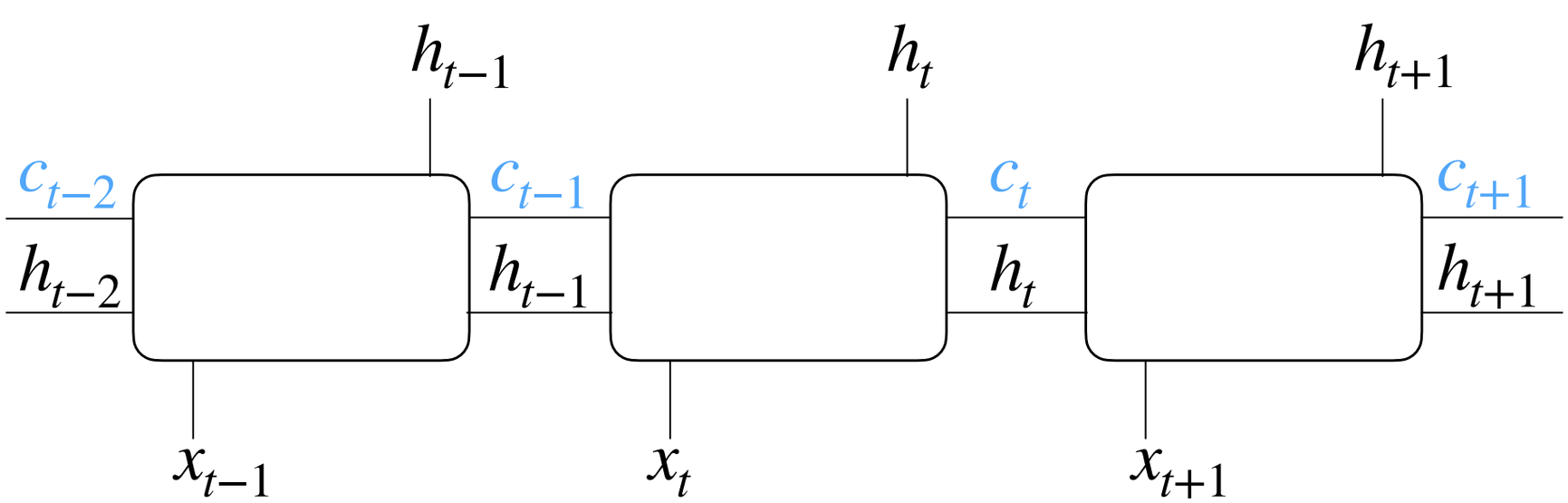
```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): GRU(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

Just plugin the GRU instead of the RNN.

Lesson 5.3:
Long-Short Term Memory

Long-Short Term Memory (LSTM)

- What if we want to keep explicit information about previous states (memory)?
- How much information is kept, can be controlled through gates.
- LSTMs were first introduced in 1997 by Hochreiter and Schmidhuber

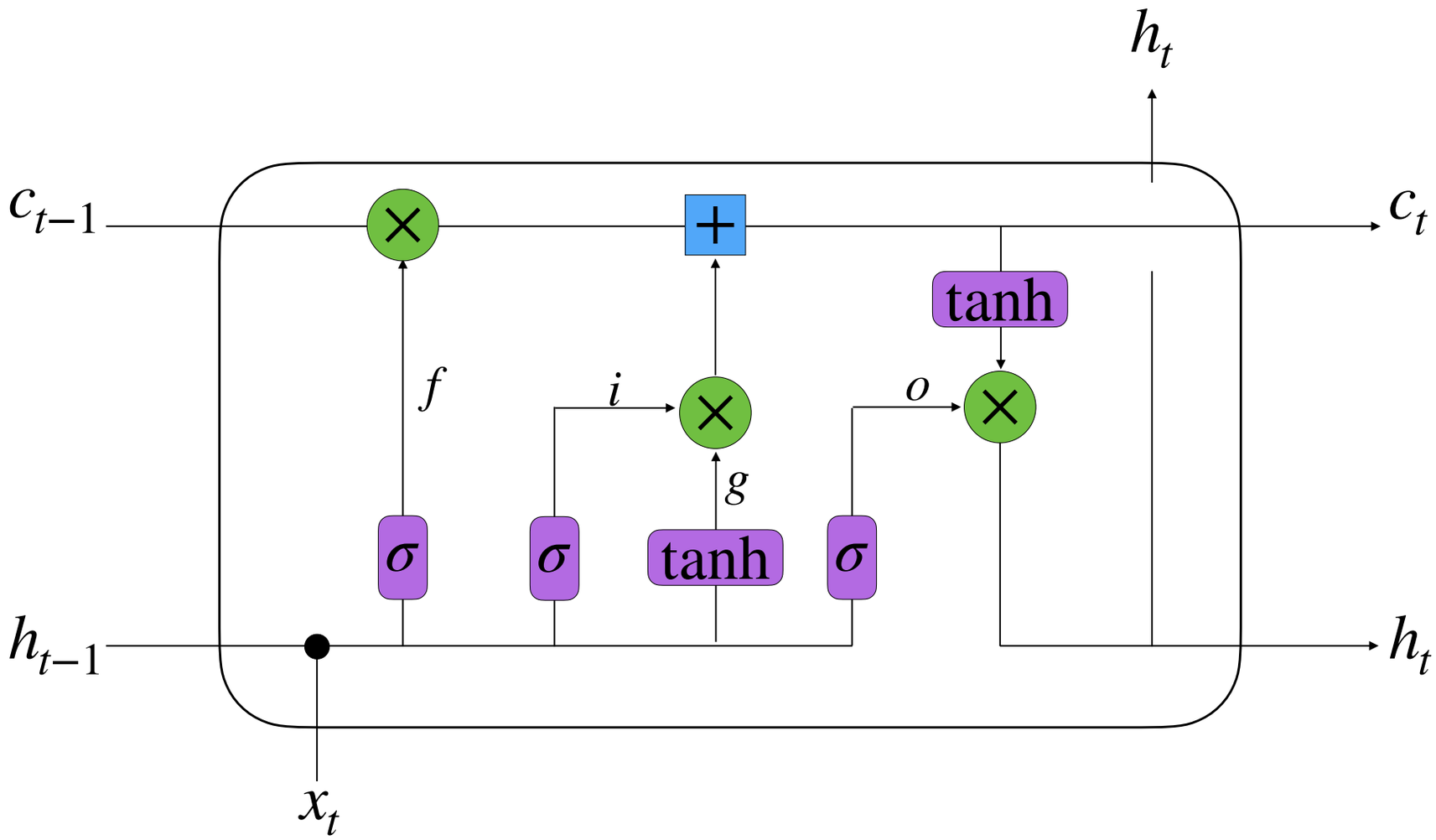


Long-Short Term Memory (LSTM)

Element wise addition

Element wise multiplication

1– 1 minus the input



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

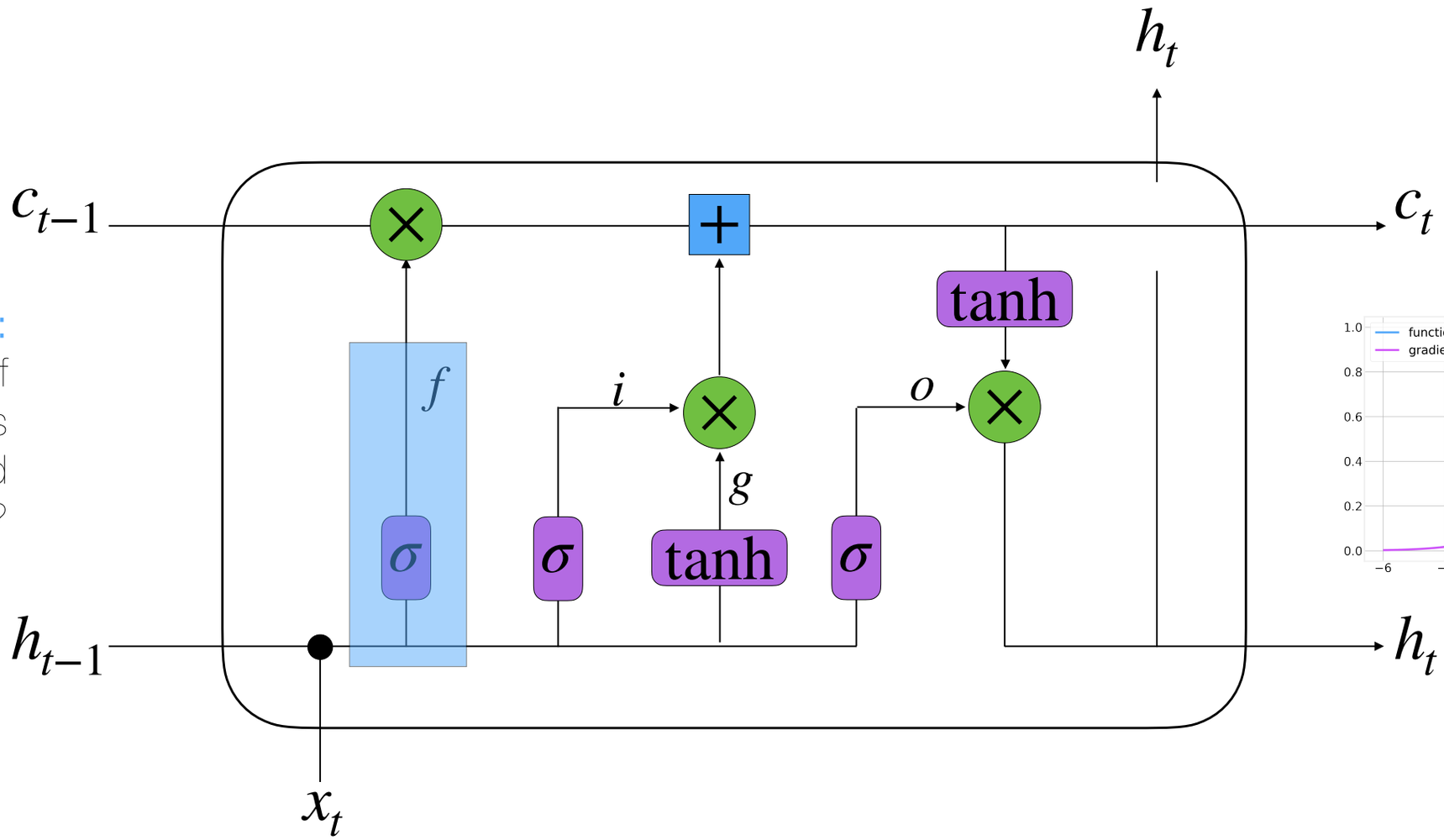
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

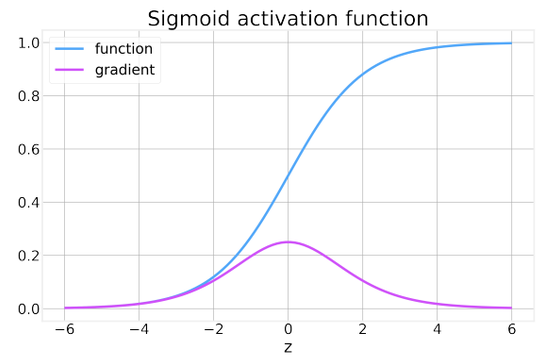
Element wise addition

Element wise multiplication

1 minus the input



Forget gate:
How much of the previous state should be kept?



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

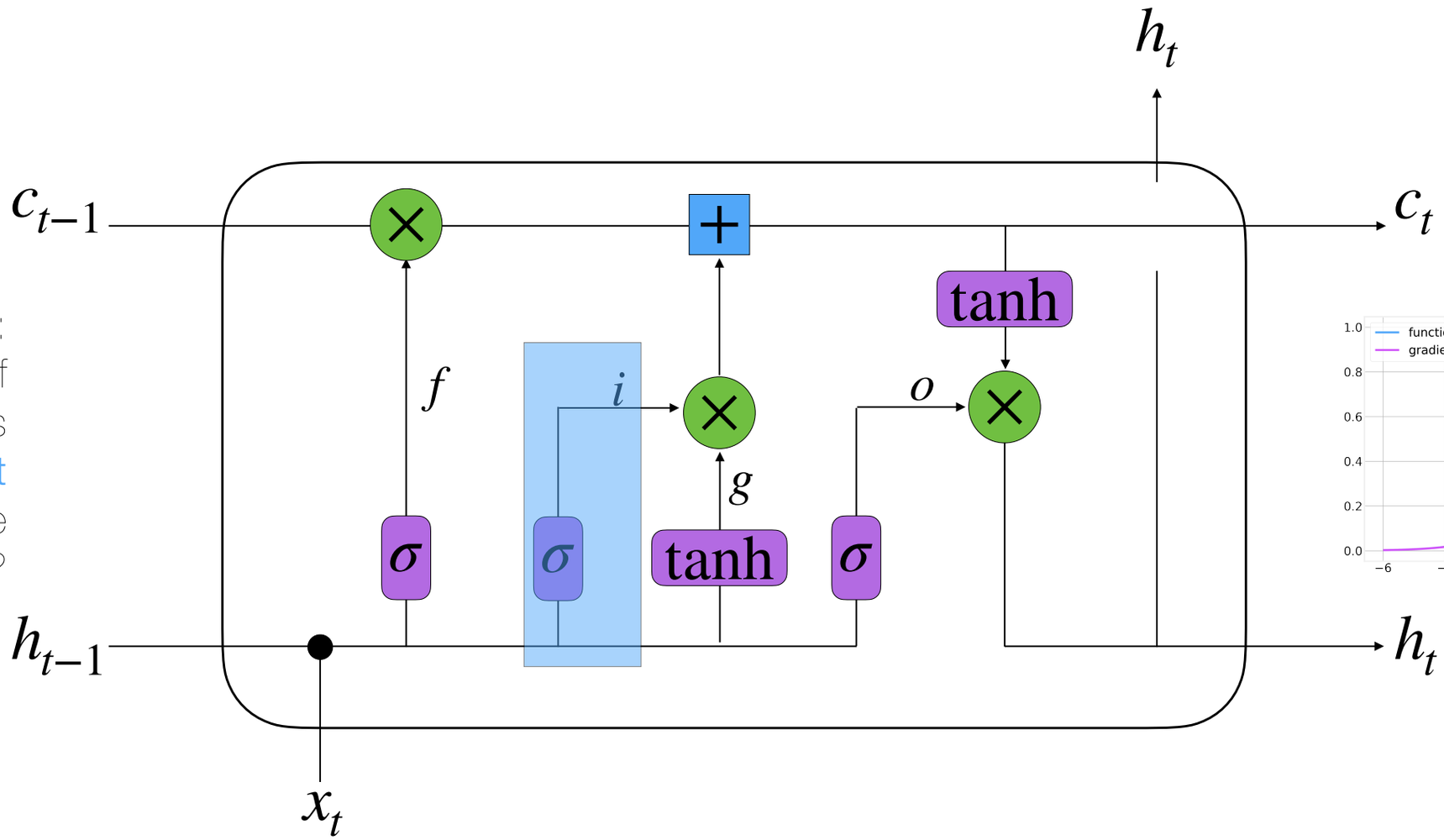
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

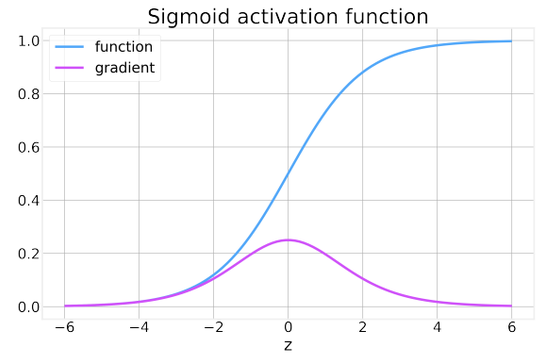
Element wise addition

Element wise multiplication

1– 1 minus the input



Input gate: How much of the previous output should be remembered?



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

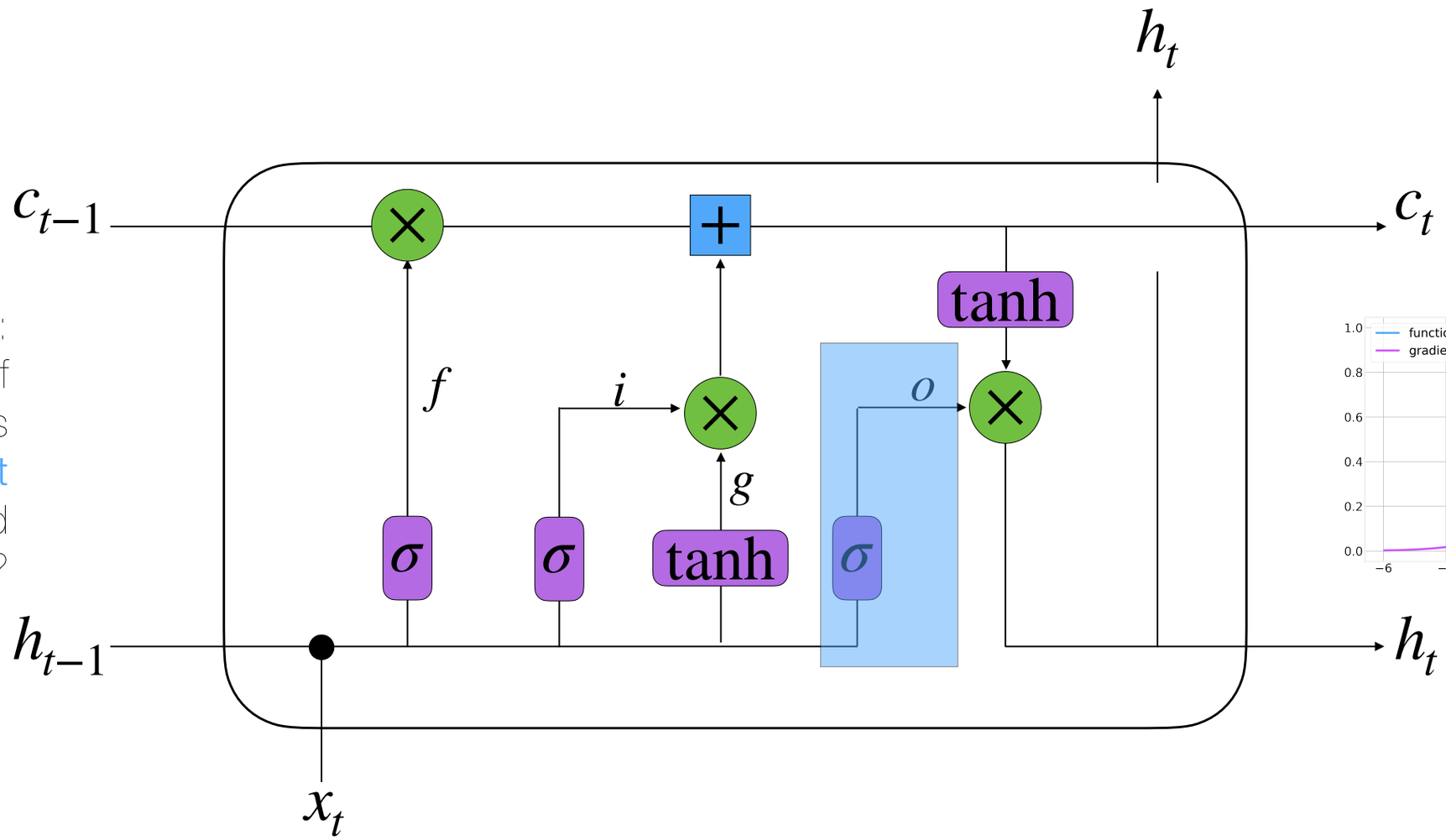
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

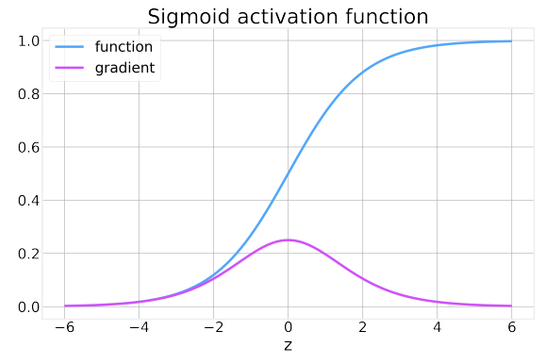
Element wise addition

Element wise multiplication

1– 1 minus the input



Output gate: How much of the previous output should contribute?



All gates use the same inputs and activation functions, but different weights

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

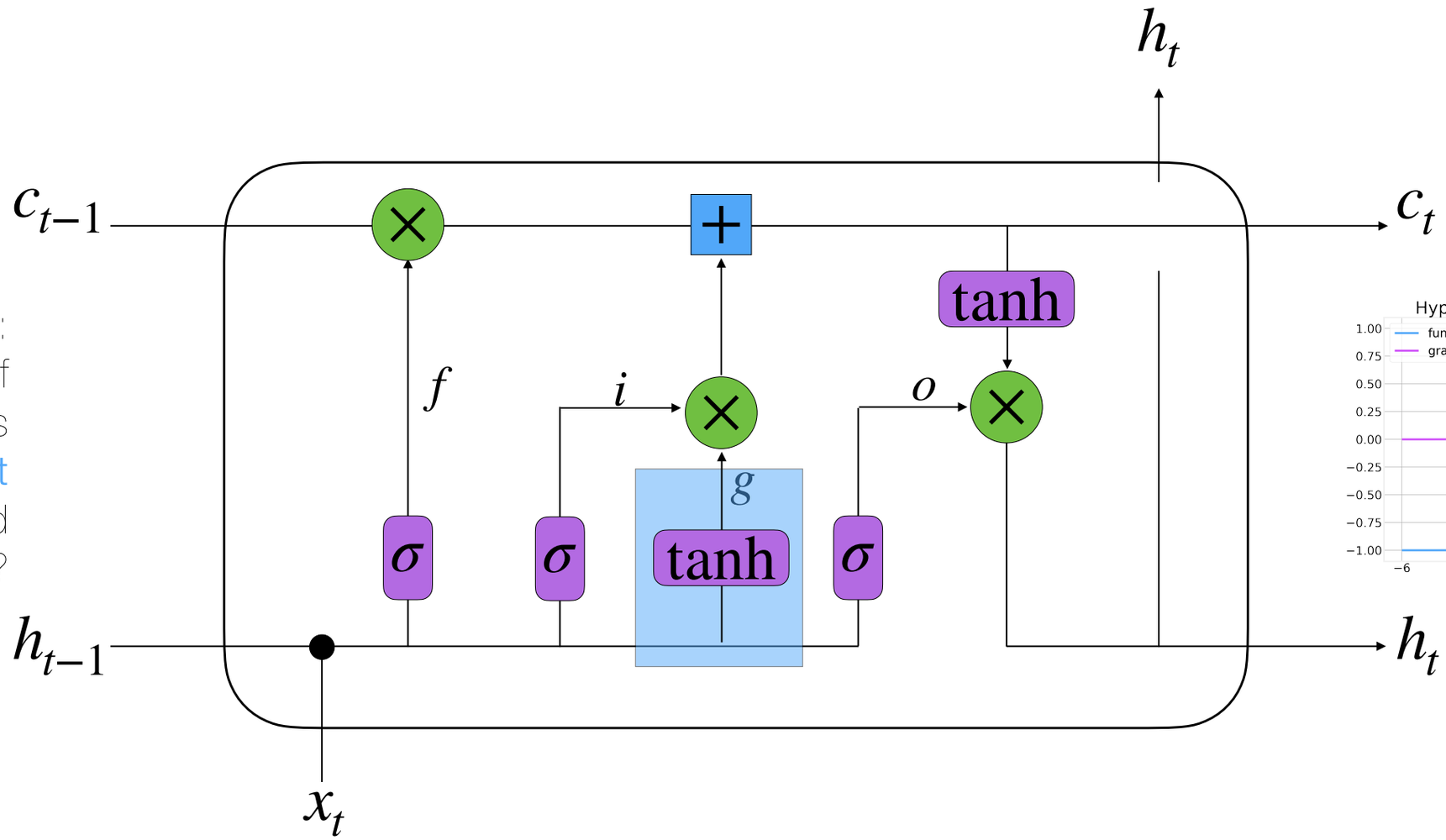
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

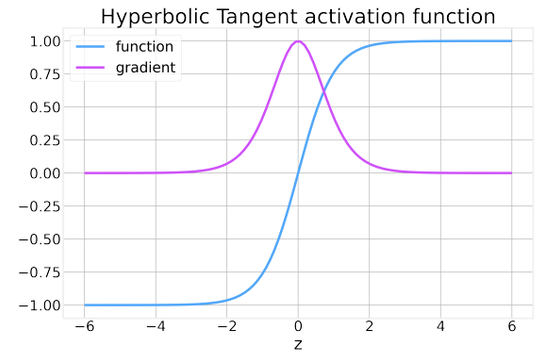
Element wise addition

Element wise multiplication

1− 1 minus the input



Output gate: How much of the previous output should contribute?



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

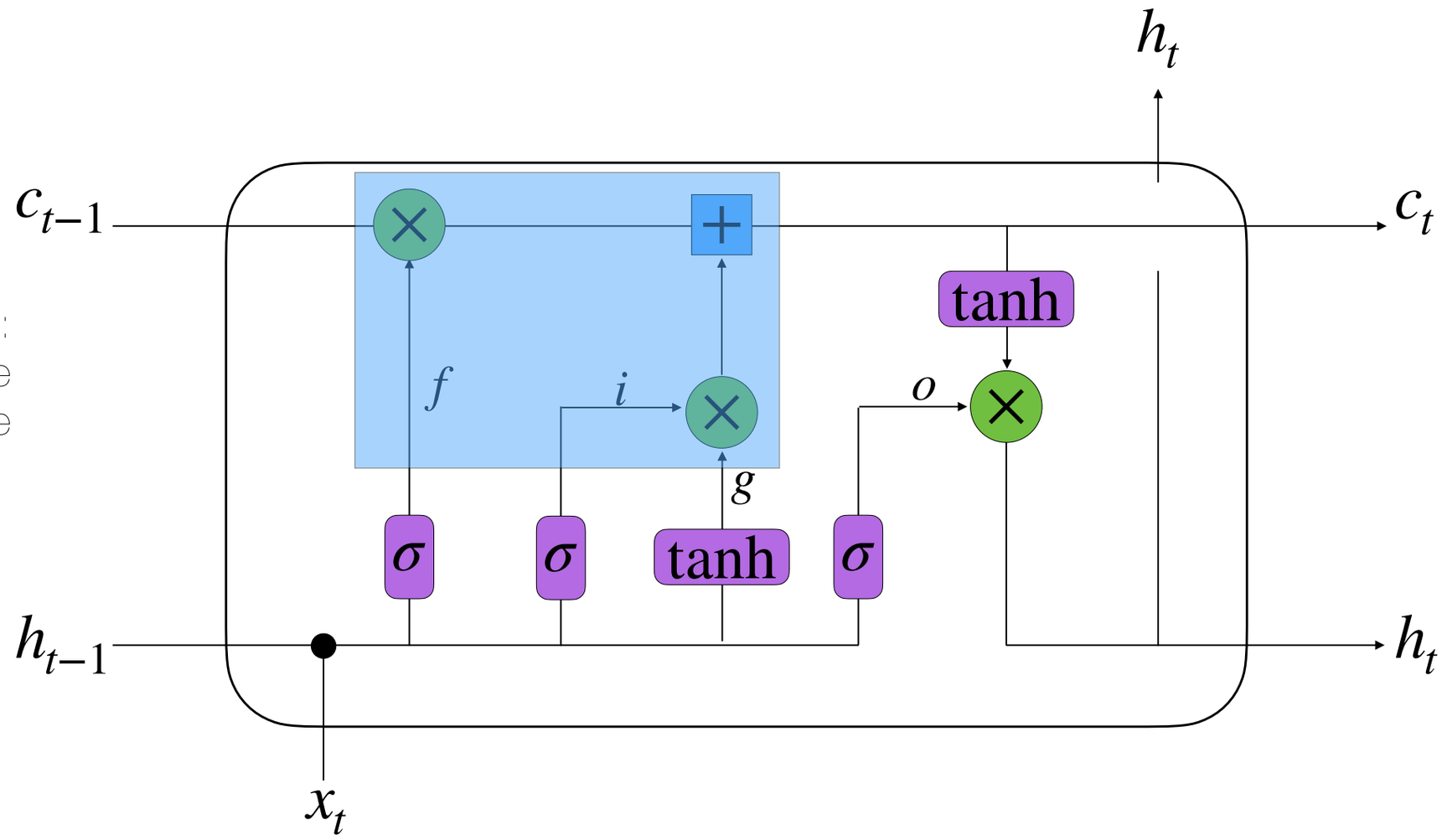
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

Element wise addition

Element wise multiplication

1– 1 minus the input



State:
Update the
current state

$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

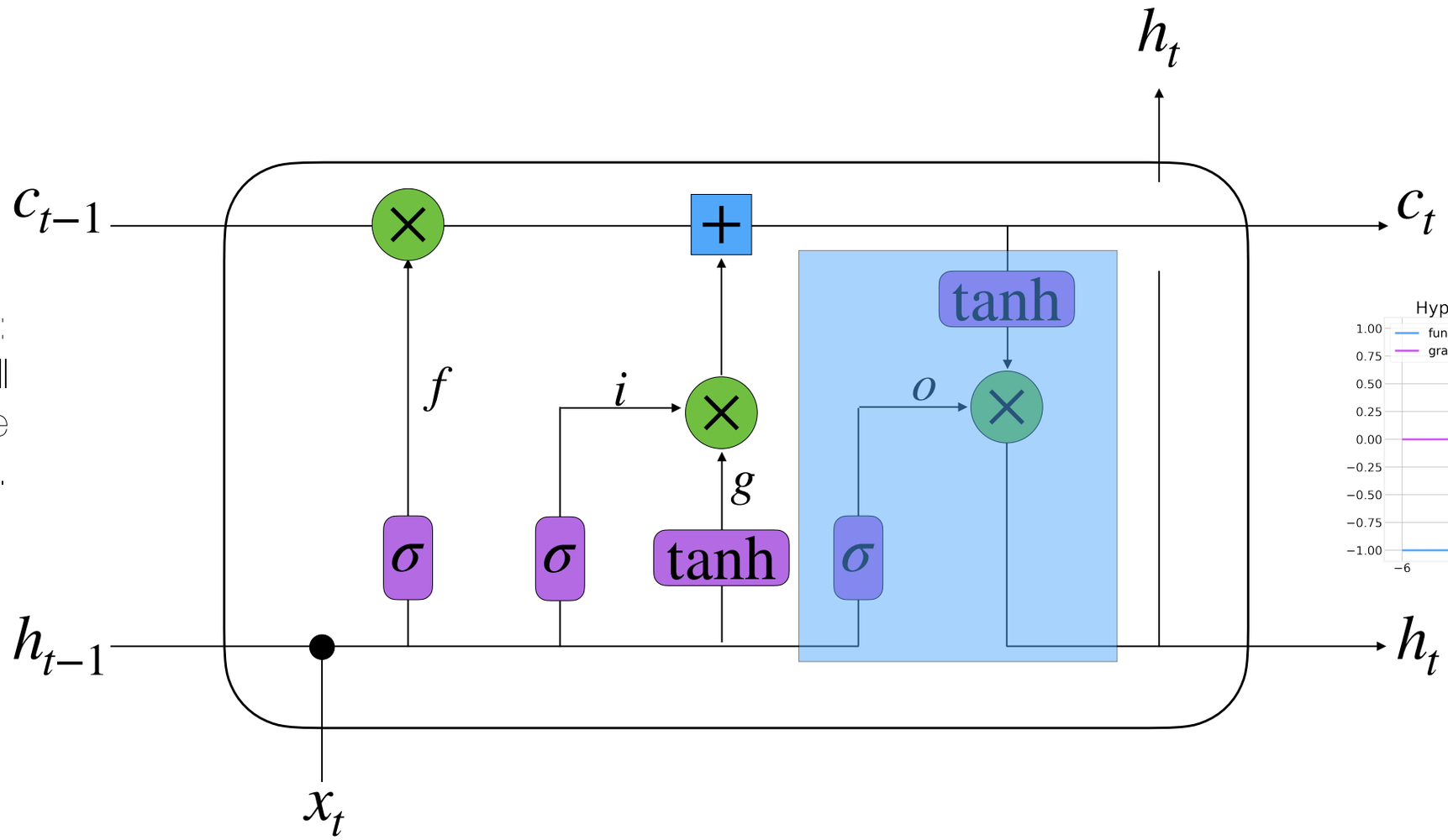
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

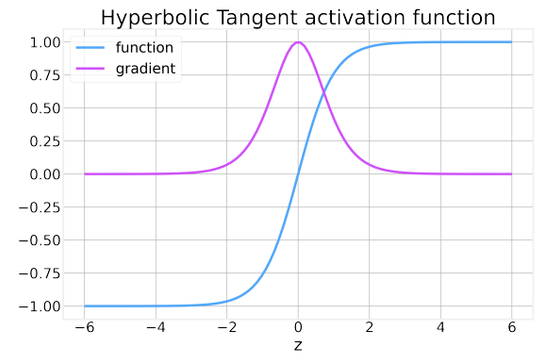
Element wise addition

Element wise multiplication

1− 1 minus the input



Output: Combine all available information.



$$f = \sigma\left(W_f h_{t-1} + U_f x_t\right) \qquad g = \tanh\left(W_g h_{t-1} + U_g x_t\right)$$

$$i = \sigma\left(W_i h_{t-1} + U_i x_t\right) \qquad c_t = \left(c_{t-1} \otimes f\right) + \left(g \otimes i\right)$$

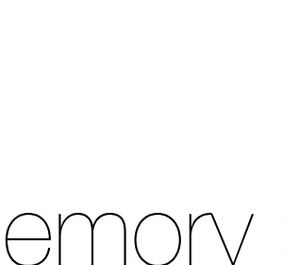
$$o = \sigma\left(W_o h_{t-1} + U_o x_t\right) \qquad h_t = \tanh\left(c_t\right) \otimes o$$

Long-Short Term Memory (LSTM)

```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): LSTM(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

Long-Short Term Memory (LSTM)

```
RNN(
  (embedding): Embedding(20002, 128)
  (rnn): LSTM(128, 256)
  (fc): Linear(in_features=256, out_features=2, bias=True)
)
```

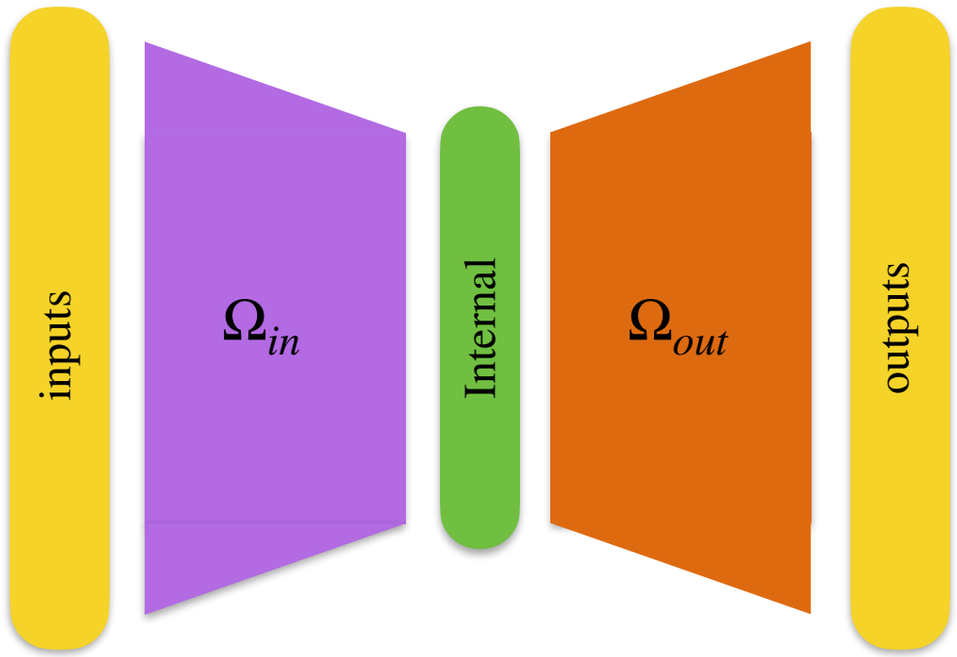


Just plugin the
GRU or RNN
with an LSTM.

Lesson 5.4:
Auto-Encoder Models

Auto-Encoders

- Auto-Encoders use the same values for both inputs and outputs
- The Internal/hidden layer(s) have a smaller number of units than the input
- The fundamental idea is that the Network needs to learn an internal representation of its inputs that is smaller but from which it is still possible to reconstruct the input.

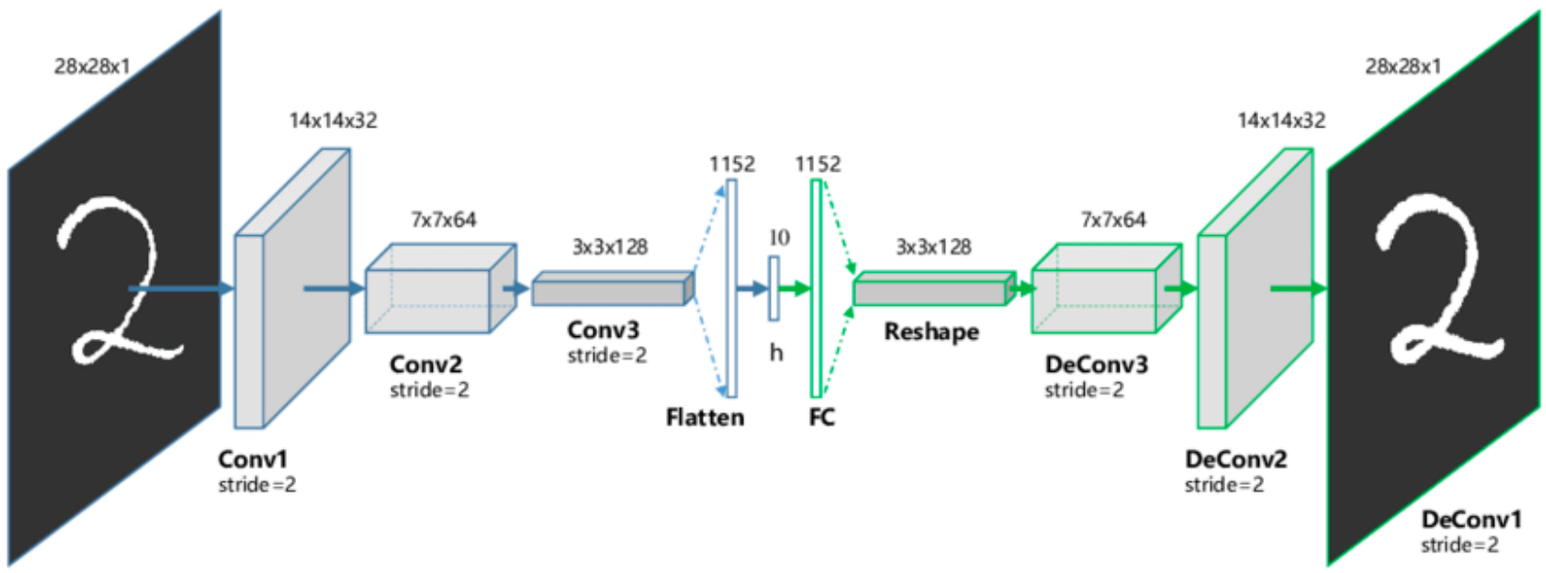


- Think of it as "zipping" and "unzipping" the input values.

Auto-Encoders

https://www.researchgate.net/figure/The-structure-of-proposed-Convolutional-AutoEncoders-CAE-for-MNIST-In-the-middle-there_fig1_320658590

- After training, the parts of the network that generate the internal representation can be used as inputs to the Networks
- This is similar to what we did when we reused the word embeddings generated by training a word2vec network
- Auto-encoders can be arbitrarily complex, including many layers between the input and the internal representation (or Code) and are often used in Image Processing to generate efficient representations of complex images



Code - Sequence Modeling

https://github.com/DataForScience/AdvancedNLP

Question

- How was the technical level?
    - 1 — Too Low (too many details)
    - 2 — Low
    - 3 — Just Right
    - 4 — High
    - 5 — Too High (too few details)

Question

- How was the level of Python code/explanations?
    - 1 — Too Low (too many details)
    - 2 — Low
    - 3 — Just Right
    - 4 — High
    - 5 — Too High (too few details)

Events

data4sci.substack.com

LangChain for Generative AI Pipelines

May 28, 2025 - 10am-2pm (PST)

Machine Learning with PyTorch for Developers

Jun 4, 2025 - 10am-2pm (PST)

Bruno Gonçalves

https://data4sci.com

info@data4sci.com

https://data4sci.com/call

Natural Language Processing (NLP) Fundamentals, 3rd Edition

https://bit.ly/NLP_LL_v3

Begin

Complete this course and earn a badge!

6h 14m • 11 sections

Natural Language Processing LiveLessons covers the fundamentals and some of the more advanced aspects of Natural Language Processing in a simple and intuitive way, empowering you to add NLP to your toolkit. Using the powerful NLTK package, it gradually moves from the basics of text representation, cleaning, topic detection, regular expressions, and sentiment analysis before moving on to the PyTorch deep learning framework to explore advanced topics such as text classification and sequence-to-sequence models. The transformer architectures underlying large language models (LLMs) like ChatGPT, Claude, and BERT are explored in depth along with some practical applications. After successfully completing these lessons you'll be equipped with a fundamental and practical understanding of the full breadth of Natural Language Processing tools and algorithms.

Course Outline

Python Data Visualization: Create impactful visuals, animations and dashboards

https://bit.ly/DataViz_LL

Begin

Complete this course and earn a badge!

6h 36m • 11 sections

Course Outline

Sneak Peek

The Sneak Peek program provides early access to Pearson video products and is exclusively available to subscribers. Content for titles in this program is made available throughout the development cycle, so products may not be complete, edited, or finalized, including video post-production editing.

Information visualization, as David McCandless aptly puts it, is a form of "knowledge compression." Our highly evolved visual processing system enables us to efficiently handle vast amounts of information. Visualization's power lies in its ability to encode data intuitively, making complex data accessible. As data grows in volume and complexity, the importance of effective visualization increases. This video explores how the human visual cortex processes colors and shapes and how we can utilize these mechanisms for effective visualization using Pythons powerful visualization libraries.

Starting with pandas and Matplotlib, two core Python libraries, we learn about the basics of Python data pre-processing and visualization before moving on to more advanced packages. Seaborn, built on top of Matplotlib, simplifies common tasks and enhances productivity. Interactive visualizations using Bokeh and Plotly are also explored. We'll use Jupyter notebooks to craft our visualizations.

Times Series Analysis for Everyone

https://bit.ly/Timeseries_LL

Begin

Complete this course and earn a badge!

6h • 14 sections

Times Series Analysis for Everyone LiveLessons covers the fundamental tools and techniques for the analysis of time series data. These lessons introduce you to the basic concepts, ideas, and algorithms necessary to develop your own time series applications in a step-by-step, intuitive fashion. The lessons follow a gradual progression, from the more specific to the more abstract, taking you from the very basics to some of the most recent and sophisticated algorithms by leveraging the statsmodels, arch, and Keras state-of-the-art models.

@bgoncalves

Course Outline