CPE 490 590: Machine Learning for Engineering Applications

14 Learning on Sequence Data

Rahul Bhadani

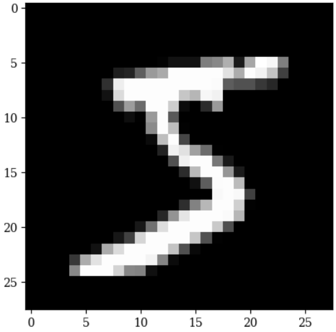
Electrical & Computer Engineering, The University of Alabama in Huntsville

Outline

Convolutional Neural Network

Motivation

- Image can be considered as a form of a matrix.



Motivation

- In order to train with Artificial Neural Network (ANN), we need to convert them 1-D array (that is flattening).

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}

$\rightarrow$

$b_1 = a_{11}$	$b_2 = a_{12}$	$b_3 = a_{13}$	$b_4 = a_{21}$	$b_5 = a_{22}$	$b_6 = a_{23}$

- In doing so, we are losing pixel-to-pixel relationship.
- In addition, working with color images would mean we have at least three channels R, G, B.
- So if we have 200×200 image size, and if we flatten it, then we will have $200 \times 200 \times 3 = 120000$ features.
- And if we have just 100 images, that's $120000 \times 100 = 12000000$ features.

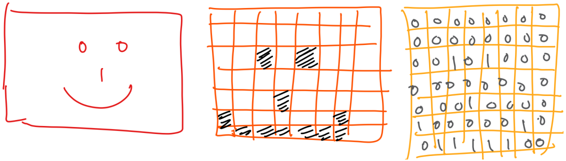
Limitations of ANN for Images

In that sense, ANN suffers from two conditions:

- Unable to preserve spatial information
- Curse of dimensionality

To preserve the pixel-to-pixel spatial relationship, we include two additional operations in the neural network:

1. Convolution
2. Pooling

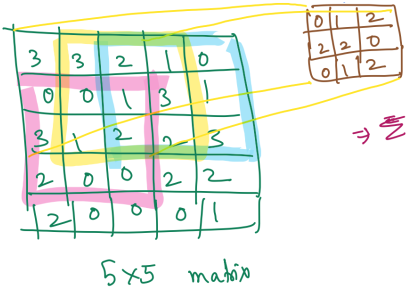


Convolution

Convolution is an operation to reduce the number of pixels through some operation that in turn reduces the number of weights to learn.

In convolution, the reduction in number of features happens by sliding a kernel across an image performing the dot product.

Kernel: It is an $n \times n$ matrix of some numbers whose dimension is usually smaller than the image size.



$$\sum \begin{bmatrix} 3\times 0 & 3\times 1 & 2\times 2 \\ 0\times 2 & 0\times 2 & 1\times 0 \\ 3\times 0 & 1\times 1 & 2\times 2 \end{bmatrix}$$

$$\Rightarrow \sum [0+3+4+0+0+0+0+1+4]$$
$$= 12$$

$$\begin{array}{cc} \begin{bmatrix} 12 & 12 & 17 \\ 10 & 17 & 19 \\ 9 & 6 & 14 \end{bmatrix} & 3\times 3 \end{array}$$

Kernels

A particular kernel usually extracts certain features from an image, or basically it performs some sort of image processing.

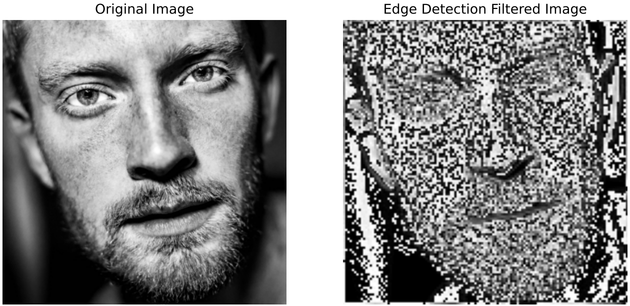
Examples of Kernels

$$\text{Edges: } \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\text{Sharpen: } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\text{Box Blur: } \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Edge-detection Example



Filter

The window we use in kernel is called a filter. The window of a certain size is slid across the image. We can apply pooling to perform operations such as minimum, median, maximum.

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

Convolutional Filters

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 2 & 1 & 0 \\
0 & 2 & 4 & 4 & 2 & 0 \\
0 & 1 & 3 & 3 & 1 & 0 \\
0 & 1 & 2 & 3 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0
\end{bmatrix}
\times
\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
=
\begin{bmatrix}
0 & & \\
 & & \\
 & &
\end{bmatrix}
$$

$$
\begin{aligned}
(0*0)+(0*1)+(0*0)+\\
(0*1)+(1*-4)+(2*1)+\\
(0*0)+(2*1)+(4*0)=0
\end{aligned}
$$

Convolutional Filters

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 & 0 \\ -2 & -5 & -5 & -2 \\ 2 & -2 & -1 & 3 \\ -1 & 0 & -5 & 0 \end{bmatrix}$$

Pooling

Pooling does a mathematical operation to the output of the convolution image. But you can also apply pooling directly to the image.

Pooling: Downsampling

Combine multiple adjacent nodes into a single node: max-pooling

$$\begin{bmatrix} \boxed{\begin{matrix} 0 & -1 \\ -2 & -5 \end{matrix}} & \begin{matrix} -1 & 0 \\ -5 & -2 \end{matrix} \\ \begin{matrix} 2 & -2 \\ -1 & 0 \end{matrix} & \begin{matrix} -1 & 3 \\ -5 & 0 \end{matrix} \end{bmatrix} \rightarrow \textbf{max} \rightarrow \begin{bmatrix} \boxed{0} & \\ & \end{bmatrix}$$

$$\begin{bmatrix} \begin{matrix} 0 & 1 \\ -2 & -5 \end{matrix} & \boxed{\begin{matrix} -1 & -0 \\ -5 & -2 \end{matrix}} \\ \begin{matrix} 2 & -2 \\ -1 & 0 \end{matrix} & \begin{matrix} -1 & 3 \\ -5 & 0 \end{matrix} \end{bmatrix} \rightarrow \textbf{max} \rightarrow \begin{bmatrix} \boxed{0} & \boxed{0} \\ & \end{bmatrix}$$

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
- Further, it protects the network from (slightly) noisy inputs

Downsampling with Stride

- Only apply the convolution to some subset of the image
- e.g., every other column and row = a stride of 2

$$\begin{bmatrix} \boxed{0} & \boxed{0} & 0 & 0 & 0 & 0 \\ \boxed{0} & \boxed{1} & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} \boxed{-2} & \\ & \end{bmatrix}$$

Downsampling with Stride

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & \\ & & \\ & & \end{bmatrix}$$

Downsampling with Stride

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 1 \\ & & \\ & & \end{bmatrix}$$

Downsampling with Stride

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 1 \\ 0 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

Operation after Pooling

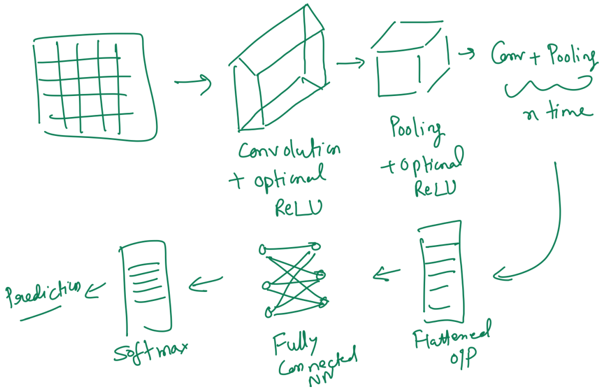
After applying convolution and pooling, we apply a non-linear activation, ReLU.

Then after that, you have more Convolution, more Pooling layers.

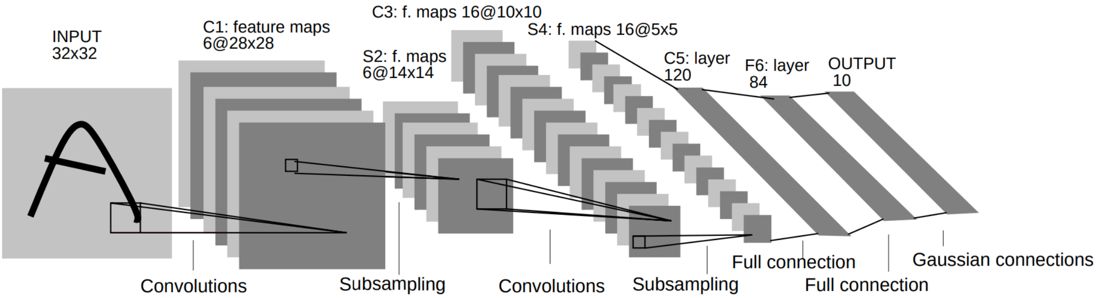
Finally, after several sequences of Convolution + Pooling + ReLU, we flatten the output from the previous layer and then we feed it to the fully connected layer (ANN).

Image Classification

Then, we apply a softmax activation if our goal is to do supervised image classification.



Lenet



Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

- One of the earliest, most famous deep learning models – achieved remarkable performance at handwritten digit recognition ($< 1\%$ test error rate on MNIST dataset)
- Used sigmoid (or logistic) activation functions between layers and mean-pooling, both of which are pretty uncommon in modern architectures

A note on multi-dimensionality and vectorization:

- Images are high-dimensional datasets. This is true even more for colored images.
- Let's say $X \in \mathbb{R}^{H \times W}$ is a matrix (for a grayscale image of height H and width W).
- After flattening, an image can be represented as a column vector $X \in \mathbb{R}^{D}$ with $D = H \times W$ elements. For a color image with 3 channels, $D = H \times W \times 3$.

Tensor

We need a concept of higher-order matrices which essentially called **Tensor**.

- $X \in \mathbb{R}^{H \times W \times D}$ is an order-3 tensor, or third-order Tensor. Another way to consider an order 3 tensor as D channels of matrices. $D = 1$ reduces it to a matrix.
- A scalar value is order-0 tensor.
- A vector is order-1 tensor.
- A matrix is order-3 tensor.

We can also have 4 dimensions if we consider opacity of an image (say in PNG).

Tensor to Vector

Given a tensor, we can arrange all numbers inside it into a long vector following a prespecified order.

Example (Matrix Vectorization - Column First Order):

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \xrightarrow{\text{In MATLAB } A(:)} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix}$$

In order to vectorize an order 3 tensor, we would first vectorize the first channel (which is a matrix), then the second channel, and so on. So we see that it is a recursion or a recursive process. The same can be applied to tensors of order > 3.

Tensor in CNN

In CNN, the convolution kernel will form an order 4 tensor.

CNN Input

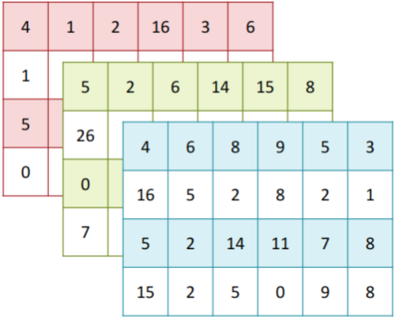
1. Depth of input: Number of channels in a colored image.
2. Number of filters: We can have more than one filter, each responsible for learning a specific feature.
3. Spatial dimensions of an image: $H \times W$.

Tensor Orders (Examples):

- Order 3 tensor: A owlch of grayscale images (or a black & white movie).
- Order 4 tensor: A owlch of colored images or a single colored movie.
- Order 5 tensor: A owlch of movies.

Tensors

Example:
$3 \times 4 \times 6$ tensor



Source: https://www.cs.cmu.edu/~mgormley/courses/10601//slides/lecture17-cnn.pdf

Further Reading

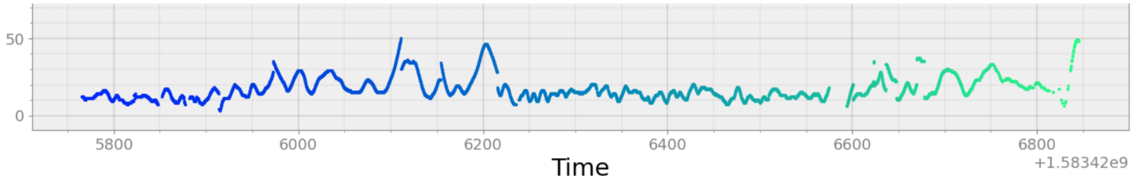
- https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf
- http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

Recurrent Neural Network

Motivation

We have seen that Convolutional Neural Networks are effective in capturing spatial relationships in data like images. Further, CNN is good for classification, and regression like tasks are complicated to implement with CNN.

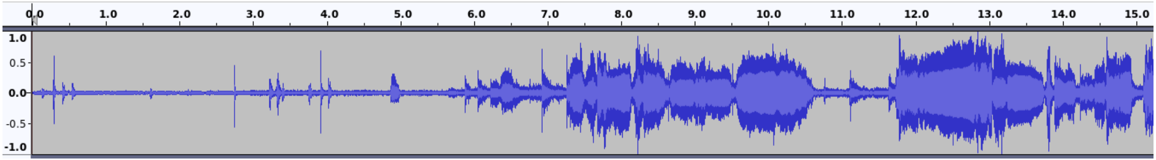
However, for data where sequential relationships are important (e.g., text, time series), we need something else.



Timeseries Data.

Some Example of Sequence Data

Speech Signal:

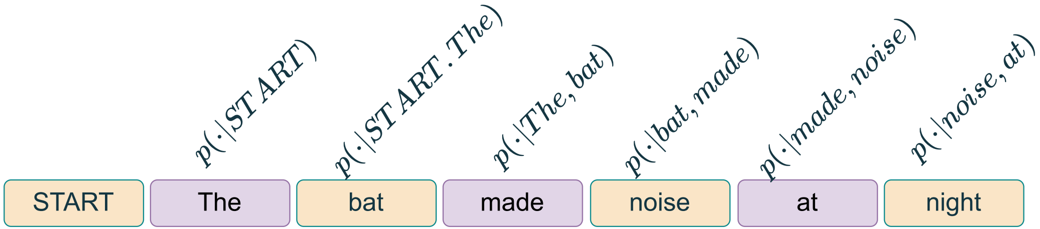


Text Sequences:

გამარჯობა! მე მიყვარს საქართველო - ჩვენი სილამაზით განთქმული ქვეყანა. საქართველო მდებარეობს კავკასიაში, შავი ზღვის აღმოსავლეთ სანაპიროზე. ჩვენი ქვეყანა ცნობილია

N-Gram Language Model

- **Goal:** Generate realistic looking sentences in human language
- **Key Idea:** Condition on the last $n-1$ words to sample the nth word



n-Gram Language Model

Question: How can we define a probability distribution over a sequence of length T?

The owl made noise at night

w_1 w_2 w_3 w_4 w_5 w_6

n - Gram Model (n = 2)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t | w_{t-1})$$

$$P(w_1, w_2, w_3, w_4, w_5, w_6) =$$

- $p(w_1)$ (The)
- $p(w_2|w_1)$ (The $\rightarrow$ owl)
- $p(w_3|w_2)$ (owl $\rightarrow$ made)
- $p(w_4|w_3)$ (made $\rightarrow$ noise)
- $p(w_5|w_4)$ (noise $\rightarrow$ at)
- $p(w_6|w_5)$ (at $\rightarrow$ night)

n-Gram Language Model

Question: How can we define a probability distribution over a sequence of length T?

The	owl	made	noise	at	night
w_1	w_2	w_3	w_4	w_5	w_6

n - Gram Model (n = 3)

$$p(w_1, w_2, ..., w_T) = \prod_{t=1}^{T} p(w_t | w_{t-1}, w_{t-2})$$

$$P(w_1, w_2, w_3, w_4, w_5, w_6) =$$

- $p(w_1)$ (The)
- $p(w_2 | w_1)$ (The $\rightarrow$ owl)
- $p(w_3 | w_2, w_1)$ (The, owl $\rightarrow$ made)
- $p(w_4 | w_3, w_2)$ (owl, made $\rightarrow$ noise)
- $p(w_5 | w_4, w_3)$ (made, noise $\rightarrow$ at)
- $p(w_6 | w_5, w_4)$ (noise, at $\rightarrow$ night)

Learning an n-Gram Model

Question: How do we learn the probabilities for the n-Gram Model?

Answer: Calculate relative frequencies from training data:

$$p(w_t | w_{t-n+1}, \ldots, w_{t-1}) = \frac{\text{count}(w_{t-n+1}, \ldots, w_t)}{\text{count}(w_{t-n+1}, \ldots, w_{t-1})}$$

Example: 3-Gram Probabilities

Training corpus with agricultural sentences:

- ... cows eat grass ... (appears 2 times)
- ... cows eat hay daily ...(appears 2 times)
- ... cows eat corn ... (appears 3 times)

Calculation: $p(\text{corn}|\text{cows eat}) = \frac{3}{7} \approx 0.43$

3-Gram Counts (context: "cows eat"):

Next Word	Count
grass	2
hay	2
corn	3

Probability Distribution:

Next Word	Probability
grass	0.29
hay	0.29
corn	0.43

Sampling from a Language Model

Process:

1. Start with initial context (n-1 words)
2. Roll weighted die for next word
3. Slide window and repeat

Sampling Demonstration:

Step	Context	Sampled Word
1	[START] The	bat
2	The bat	made
3	bat made	noise
4	made noise	at
5	noise at	night
6	at night	[END]

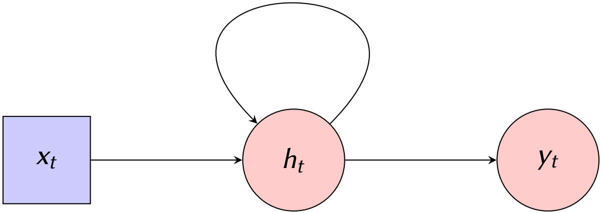
Generated Sentence:

The | bat | made | noise | at | night

Sampling introduces diversity but preserves local coherence through n-gram constraints.

Recurrent Connections and Recurrent Neural Networks

Recurrent Connection: Introduction of a cycle in a network.
Recurrent Neural Network (RNN): Any neural network that contains a cycle within its network connection.
There are many types of RNN. We will see the **Elman network** (1990, Elman).



Elman Network

The equations for the Elman network are:

$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$

where g is an activation function (e.g., tanh or sigmoid).
At the end, output layer also contains softmax: $y_t = \text{softmax}(Vh_t)$.

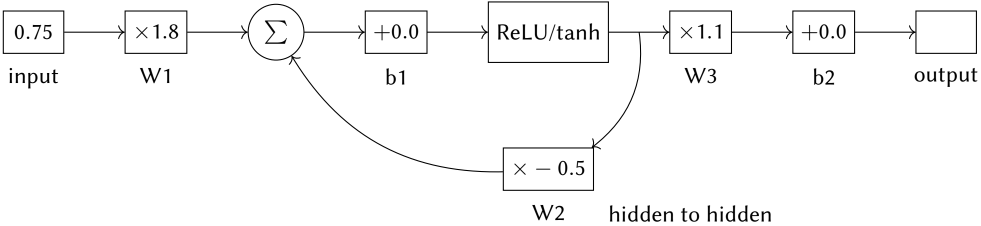
The goal is to learn the weights U, W, and V.

At the end output layer, softmax is often used.

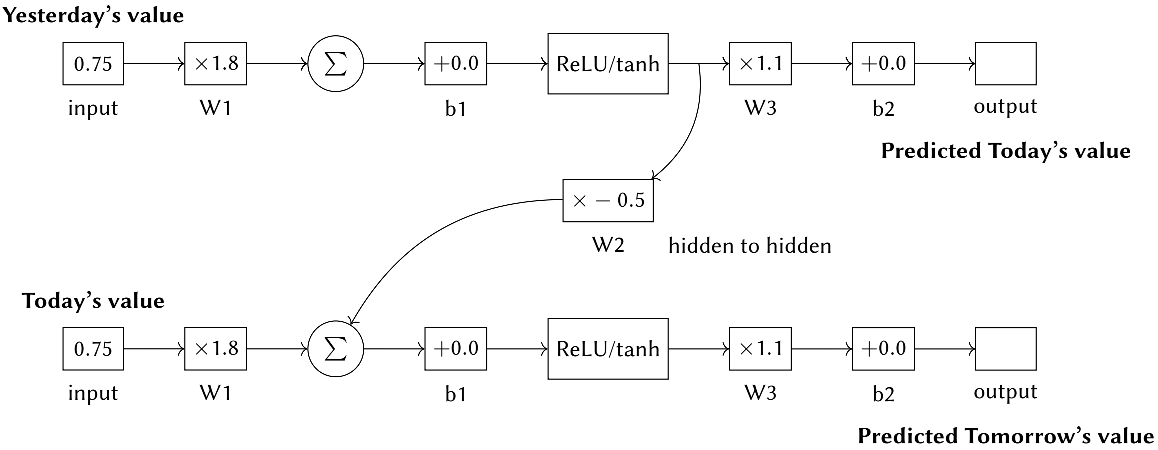
See an Example RNN Network

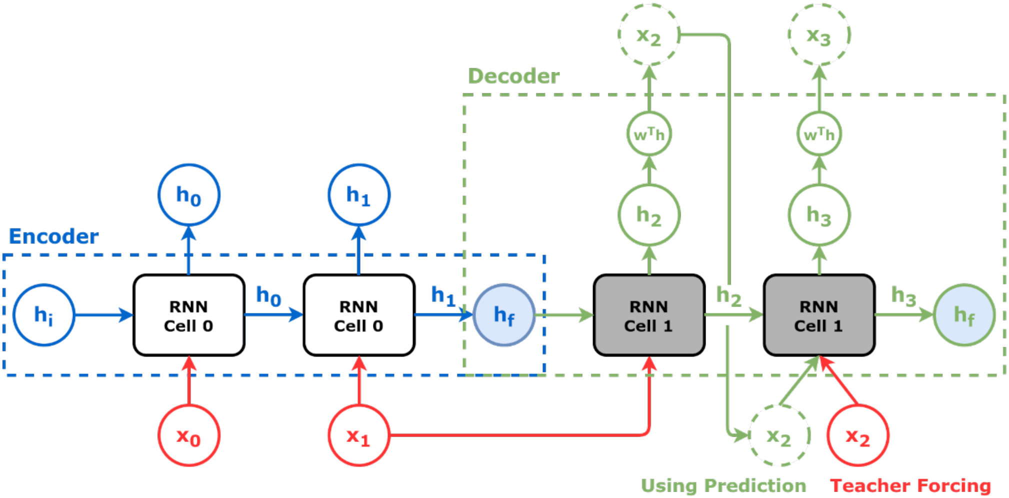
The feedback loop in Recurrent Neural Networks (RNNs) networks makes it possible to establish long-range dependence on sequential data.

To understand how these networks process sequential data over time, we can ”unroll” the network.



Unrolling the RNN Network





Source: `https://github.com/dvgodoy/dl-visuals`

Teacher Forcing

The idea that the predicted value is not used as input but the true value is used is called **teacher forcing**.

Vanishing Gradient / Exploding Gradient Problem

As we see that when we unroll the network, we have multiplication of W_2 (recurrent weight) with itself many times.

Exploding Gradient

If $W_2 > 1$, then eventually, the gradient will become really large ($W_2^{50} \gg 1$).

Vanishing Gradient

If $W_2 < 1$, then eventually, the gradient will become really small ($W_2^{50} \ll 1$).

Vanishing Gradient / Exploding Gradient Problem

Consider an RNN with ReLU activation:

$$h_t = \max(0, W_1 x_t + W_2 h_{t-1} + b_1)$$

$$y_t = W_3 h_t + b_2$$

The backpropagation through time (BPTT) is used for training sequence data.
The gradient of the loss (L) with respect to W_2 is:

$$\frac{\partial L}{\partial W_2} = \sum_{k=1}^{t} \frac{\partial L}{\partial y_k} \cdot \frac{\partial y_k}{\partial h_k} \cdot \frac{\partial h_k}{\partial W_2}$$

where k is the index for time.

Vanishing Gradient / Exploding Gradient Problem

$$\frac{\partial h_k}{\partial W_2} = \frac{\partial h_k}{\partial h_{k-1}} \cdot \frac{\partial h_{k-1}}{\partial W_2} = W_2 \cdot \frac{\partial h_{k-1}}{\partial W_2} = W_2 \cdot \left(W_2 \cdot \frac{\partial h_{k-2}}{\partial W_2} \right) = W_2^2 \cdot \frac{\partial h_{k-2}}{\partial W_2}$$

Continuing this pattern, we get:

$$\frac{\partial h_k}{\partial W_2} = W_2^{k-1} \cdot \frac{\partial h_1}{\partial W_2}$$

This shows that the gradient $\frac{\partial h_k}{\partial W_2}$ is proportional to W_2 and the previous gradients.

As you see, if $|W_2| > 1$, then the gradient grows exponentially (exploding gradient).

If $|W_2| < 1$, then the gradient vanishes.

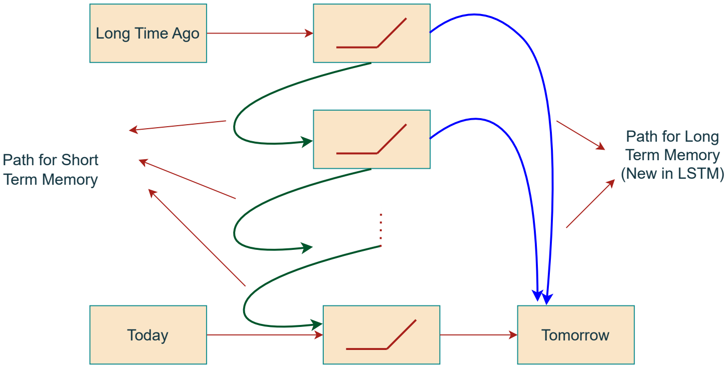
Long Short-Term Memory(LSTM)

In LSTM, we modify connections, so that there are two separate paths for long-term dependencies and short-term dependencies.

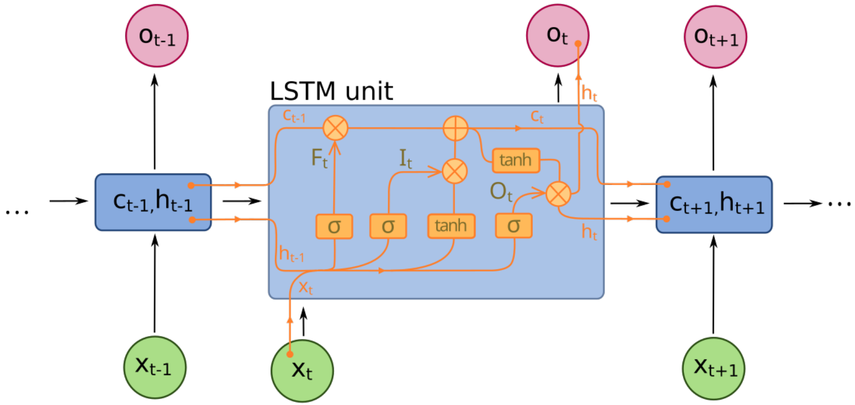
- LSTM specifically uses the tanh and sigmoid activation functions:

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

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



LSTM



Source: `https://github.com/dvgodoy/dl-visuals`

What's special in LSTM

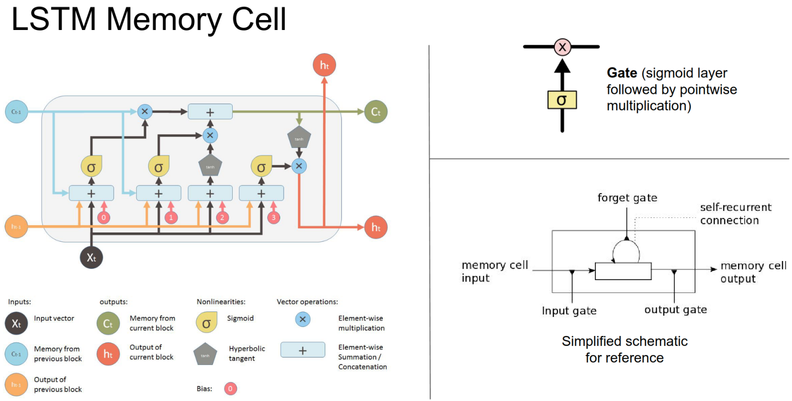
Central Idea

LSTM Introduces cell state and gating mechanisms to preserve long-term information. A memory cell (interchangeably block) which can maintain its state over time, consisting of an explicit memory (aka the cell state vector) and gating units which regulate the information flow into and out of the memory.

There are 3 kinds of gating mechanism:

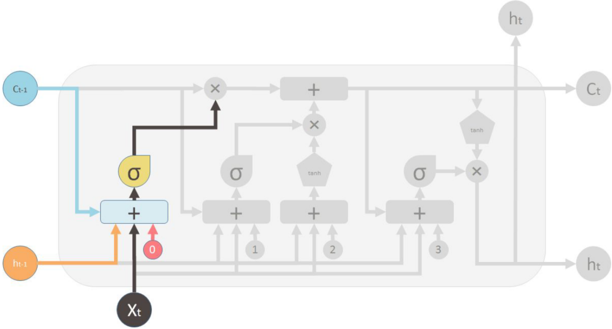
- **Forget Gate**: Decides what information to discard
- **Input Gate**: Decides what new information to store
- **Output Gate**: Decides what to output

LSTM Memory Cell

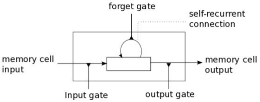


Source: https://pages.cs.wisc.edu/~shavlik/cs638/lectureNotes/Long%20Short-Term%20Memory%20Networks.pdf

Forget Gate

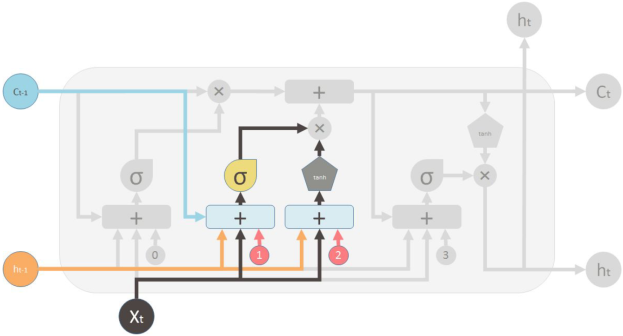


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

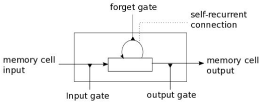


Source: https://pages.cs.wisc.edu/~shavlik/cs638/lectureNotes/Long%20Short-Term%20Memory%20Networks.pdf

Input Gate



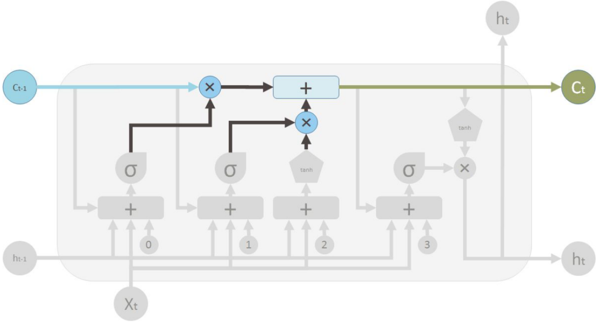
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



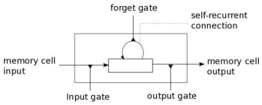
Source: https://pages.cs.wisc.edu/~shavlik/cs638/lectureNotes/Long%20Short-Term%20Memory%20Networks.pdf

Memory Update

The cell state vector aggregates the two components (old memory via the forget gate and new memory via the input gate)

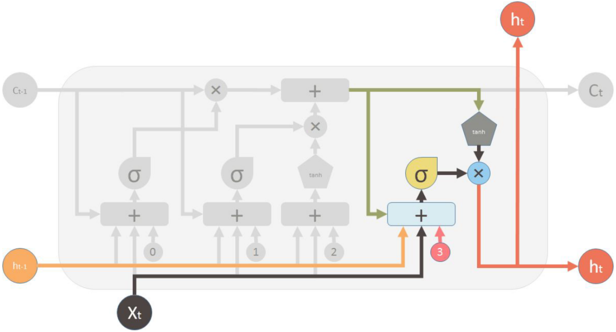


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

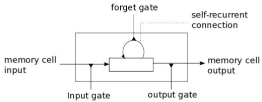


Source: https://pages.cs.wisc.edu/~shavlik/cs638/lectureNotes/Long%20Short-Term%20Memory%20Networks.pdf

Output Gate



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



Source: https://pages.cs.wisc.edu/~shavlik/cs638/lectureNotes/Long%20Short-Term%20Memory%20Networks.pdf

Transformers

Transformer

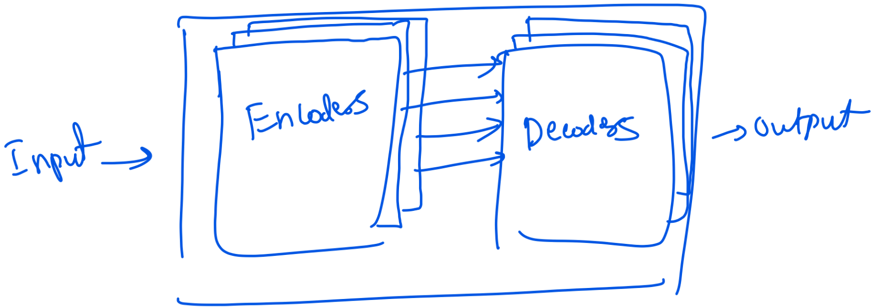
The Transformer architecture is behind models like ChatGPT and BERT.

These are generative AI models that are text based.

The Transformer was first proposed in 2017 in the paper “Attention is all you need”.

$$\boxed{\text{Input}} \longrightarrow \boxed{\text{Transformer}} \longrightarrow \boxed{\text{Output}}$$

Transformer with Encoder-decode Architecture



Transformer - Attention Mechanism

The main technology or innovation behind the transformer is the "Attention Mechanism".

An example text: *As aliens entered our planet and began to colonize Earth*

As a model generates a text word by word, it has the ability to reference or tend to words that are relevant to the generated word.

How the model learns which previous word to tend to is done via back-propagation.