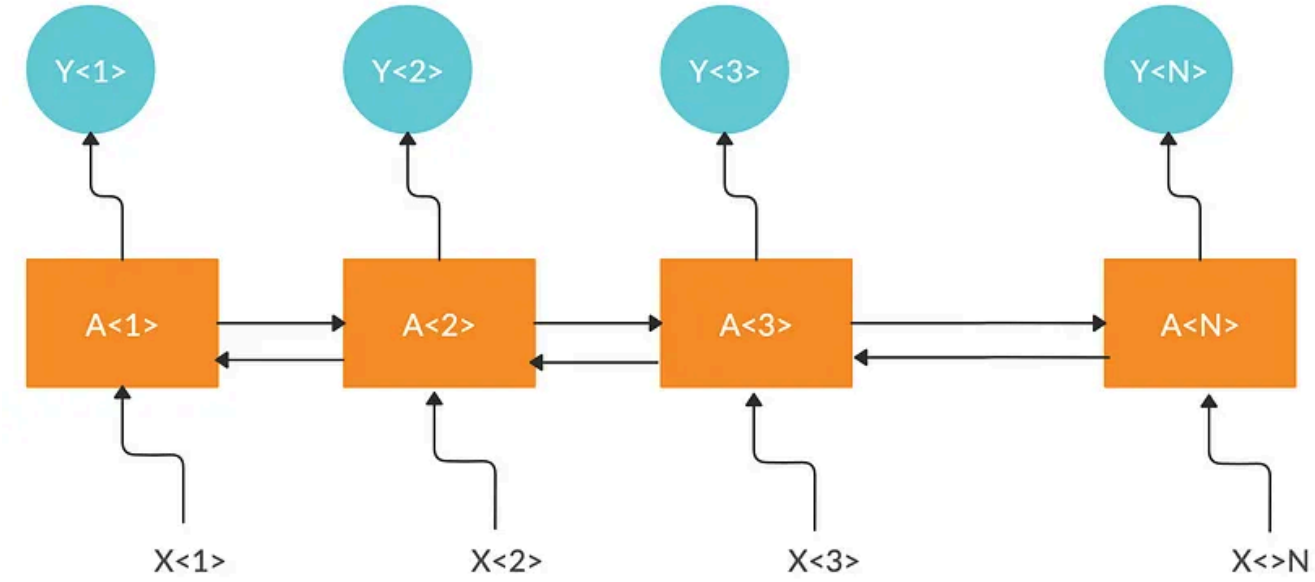
Natural Language Generation using Sequence Models

Generating texts using a Single Layer LSTM Model

Ujjwal Kumar · Follow

Published in Towards Data Science

7 min read · Jun 29, 2020



Recurrent Neural Network (Image by author)

Introduction

What if I tell you that you can enable your own device to write for you in your own style in under 100 lines of code?

The idea to make your device write on your behalf is remarkably inspiring. This practice is referred to as **Text Generation** or **Natural Language Generation**, which is a subfield of Natural Language Processing (NLP).

The fundamentals of text generation can be easily broken down into a simple supervised machine learning problem, wherein, there exists certain features (called x) with their corresponding labels (called y), and using these we can create our own prediction function which will then generate our predicted labels (called ŷ or yhat). We then map these predicted labels to the actual labels to determine the cost and optimise it using an optimisation algorithm such as Gradient Descent, RMSprop or even the Adam optimiser.

And transitively we have reduced the task of text generation to a simple prediction problem. Hence, through this, we can have a corpus of text within which we may have several sentences. We extract each sentence (say it has *n* words) and within each of these sentences, we mark the starting *n-1* words as the features (called x) and the *nth* words as the label for it (called y).

Now, let's say we have the sentence, "Deep learning has automated our world," here, the phrase "Deep learning has automated our" can be the feature (called x) and the last word "world" can be the label (called y). So in the future, whenever the device encounters the text "Deep learning has automated our" it will know to predict "world" as the next word.

Resultantly, if we train a network with a fair amount of data, we get a fairly sophisticated model that can predict the next words and *voilà*, we have taught our device to write.

Text generation using TensorFlow

We will now learn how to apply the practical method for generating new texts. For this, we will use TensorFlow 2.0 which is an open source machine learning library.

Importing necessary libraries

The steps that need to be followed for this are:

- **Data pre-processing:**

Let's take two sample sentences:

AI is the new electricity

AI is my power

(*Note that the sample taken here is infinitesimally small compared to the actual data that machine learning practitioners use to train the model. Usually, an entire corpus of a*

writers publication is used or an entire book is used as the dataset, however here, the author has only taken a small fraction of it for easy understanding.)

For simplicity and to reduce the large number of words in our collection, we can convert each word in our sentence to lowercase as this will not alter the meaning of the sentence (New, new and NEW all mean the same). We can do this by using *.lower()* in Python 3.

So we have the lowercase version of our sentences as:

ai is the new electricity

ai is my power

We now have a sample to train our model on. The next thing to do is to generate a unique token for each of the words. Repeating words in the corpus are assigned the same token. This can be easily done using the Tokenizer in TensorFlow.

For the first sentence:

ai -> 1

is -> 2

the -> 3

new -> 4

electricity -> 5

For the second sentence:

ai -> 1

is -> 2

my -> 6

power -> 7

Now, we may represent our sentences as a list of numbers:

[1, 2, 3, 4, 5] -> For the first sentence

[1, 2, 6, 7] -> For the second sentence

Generating tokens

We now generate *n-gram* sequences, wherein, the first two words of such sentences can be one sequence, and, the first three words can be the next sequence and so on. Therefore, giving our list the following possible sequences:

For the first sentence [1, 2, 3, 4, 5]

[1, 2]

[1, 2, 3]

[1, 2, 3, 4]

[1, 2, 3, 4, 5]

For the second sentence [1, 2, 6, 7]

[1, 2]

[1, 2, 6]

[1, 2, 6, 7]

We now append each of these generated sequences to another list (forming a 2-D list).

Generating n-gram sequences

One key thing to note here is that the length of each of these generated sequences are different, so to maintain order, we pre-pad with zeros, with each sequence length being equal to the longest sequence available. Since in the present case, the maximum length of the sequence is 5, we pre-pad accordingly. Padding can be done using the pad_sequences() method from Keras.

After padding each sequence will look like:

For the first sentence [1, 2, 3, 4, 5]

[0, 0, 0, 1, 2]

[0, 0, 1, 2, 3]

[0, 1, 2, 3, 4]

[1, 2, 3, 4, 5]

For the second sentence [1, 2, 6, 7]

[0, 0, 0, 1, 2]

[0, 0, 1, 2, 6]

[0, 1, 2, 6, 7]

We now know that for each of these sequences the first 4 words are the features (called x) and the last word is the label (called y). We then extract our features and their corresponding labels by doing simple NumPy array slicing.

One thing left to do is to convert our labels into one-hot encoded vectors for easy optimisation.

Say the feature is **x = [0, 1, 2, 3, 4]**

The corresponding label is **Label = [5]**

So the one-hot encoded vector of the Label is: [0, 0, 0, 0, 1] i.e. there is 1 present in the 5th position of the vector and 0 in the remaining position.

Hence, **y = [0, 0, 0, 0, 1]**

Generating features (x) and labels (y)

We are now ready with our training data which can be fed to the model.

- **Constructing the model: Single Layer LSTM Model**

We define a sequential model wherein each layer has exactly one input tensor and one output tensor.

1. The first layer is the **Embedding Layer** which would be the first layer in the network. This layer takes three arguments namely, the input dimension (the total number of unique words in our corpus); the output dimension (or the embedding dimension in which vectors will be mapped); and the input length (the length of each sequence that is fed).

2. For the second layer, we add a **Bidirectional LSTM** layer of 20 units which is efficient for taking care of the vanishing gradient problem in long sentences and dependencies.

3. And consequently, in the third layer, we add a **Dense Layer** with *p* units. Where *p* is the total number of unique words in our corpus and we apply softmax activation to each of these units.

Model construction

- **Compiling the model:**

Since the prediction of next words is a multi-class classification problem, we choose our loss function as **Categorical Cross-entropy** and the optimiser as **Adam.** By choosing this optimiser, we allow TensorFlow to adjust the learning rate on its own, which removes one hyper-parameter for us to fine tune. We then train the model for let's say 500 epochs.

Compiling the model

After fitting the training data, our model showed training accuracy close to 95% and all that is left for us to do is use this model to make predictions.

- **Predicting the next words:**

Now, in order to make predictions, we have to give our model some perception for it to know in what sense we want it to generate the texts. For this, we give an initial phrase or sentence to it, which the model then parses and makes computational predictions based upon our corpus of text that it was trained on. Accordingly, the model will predict the token for the next word and then convert that token back to the original word and finally, display the entire predicted string.

Predicting the next t words (here t=5)

Now to understand the whole process via an example, let's train the model with the following sentence:

“The scope of deep learning has been increasing at an exponential rate, the reason deep learning has bloomed is hidden in the fact that there exists a vast number of applications in todays world that we take for granted, from using Hey Siri on our iPhone (trigger word detection) to using automatic replies on our Gmail/LinkedIn (sentiment analysis); deep learning has automated our world without us even realising. The world needs deep learning to sustain as it has become necessary.”

Here, the seed to generate the new text is:

Input = scope of artificial intelligence

Whereupon, the model predicted:

Output = scope of artificial intelligence learning has automated our world

Now it may as well happen that the predicted sentence may not be syntactically valid but the overall gist of the sentence can be easily understood, thanks to our model.

To sum up

One may think that the predictions made by the model looks somewhat trivial, but it has to be kept in mind that prior to Text Generation our device was incapable of forming novel sentences, in-fact it was unaware of even the basic alphabets needed to construct words.

From making our device understand the sentiment in the text to predicting new words for a given sequence, NLP has come a long way and we can go even further with it.

Here is the link to the author’s Github repository which can be referred for the unabridged code.

Text Generation

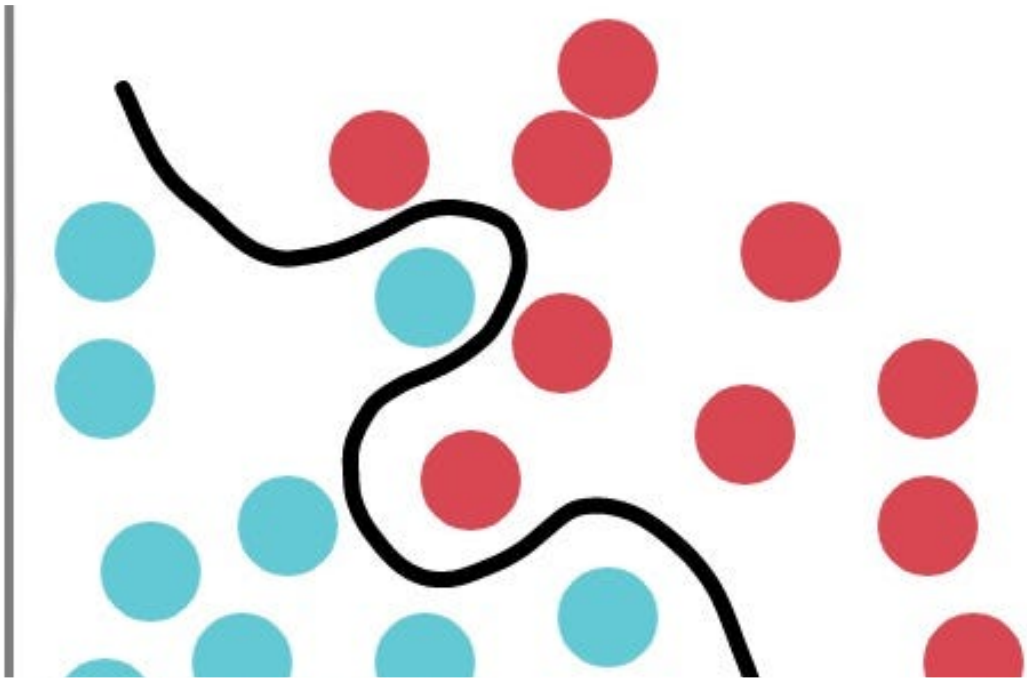
Follow

Written by Ujjwal Kumar

26 Followers · Writer for Towards Data Science

Computer Science undergraduate at SRM-IST, Artificial Intelligence Aficionado.

More from Ujjwal Kumar and Towards Data Science



Ujjwal Kumar in Towards Data Science

Impact of Regularization on Deep Neural Networks

Emphasis on Weight Decay using TensorFlow 2.0

6 min read · Jul 9, 2020

58

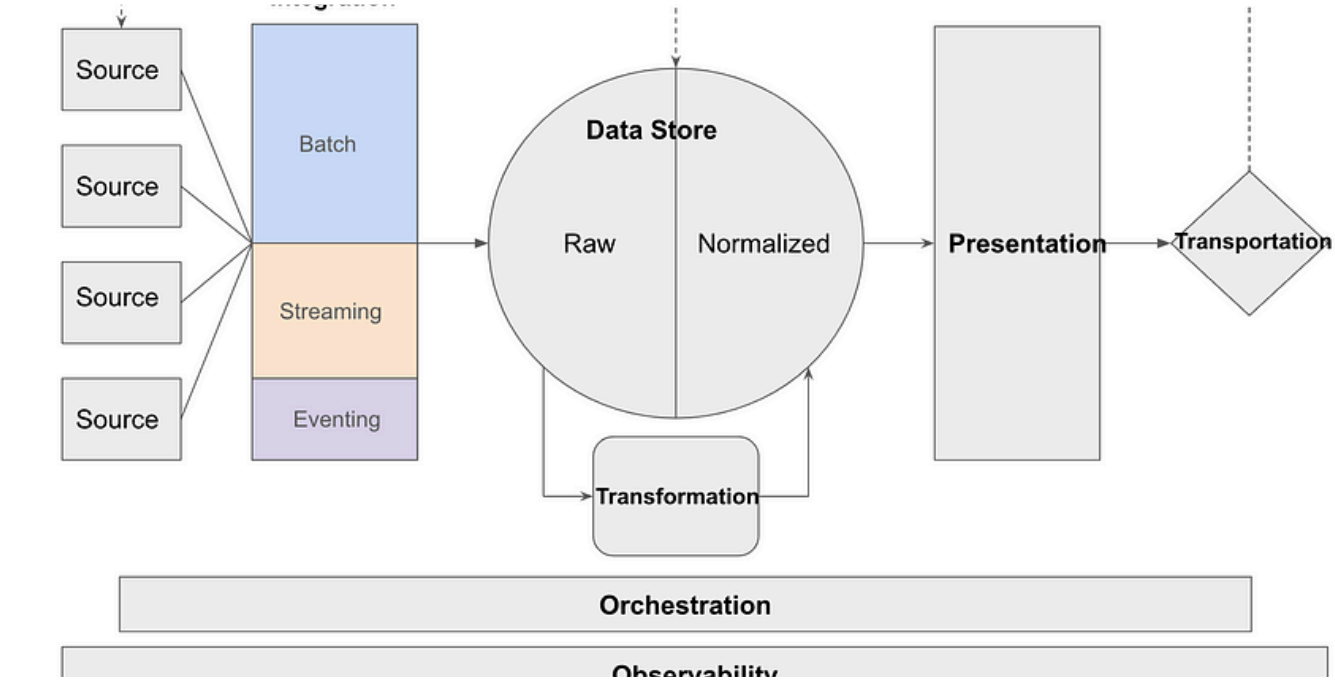
Cristian Leo in Towards Data Science

The Math behind Adam Optimizer

Why is Adam the most popular optimizer in Deep Learning? Let's understand it by diving into its math, and recreating the algorithm.

16 min read · Jan 30, 2024

2.4K 19



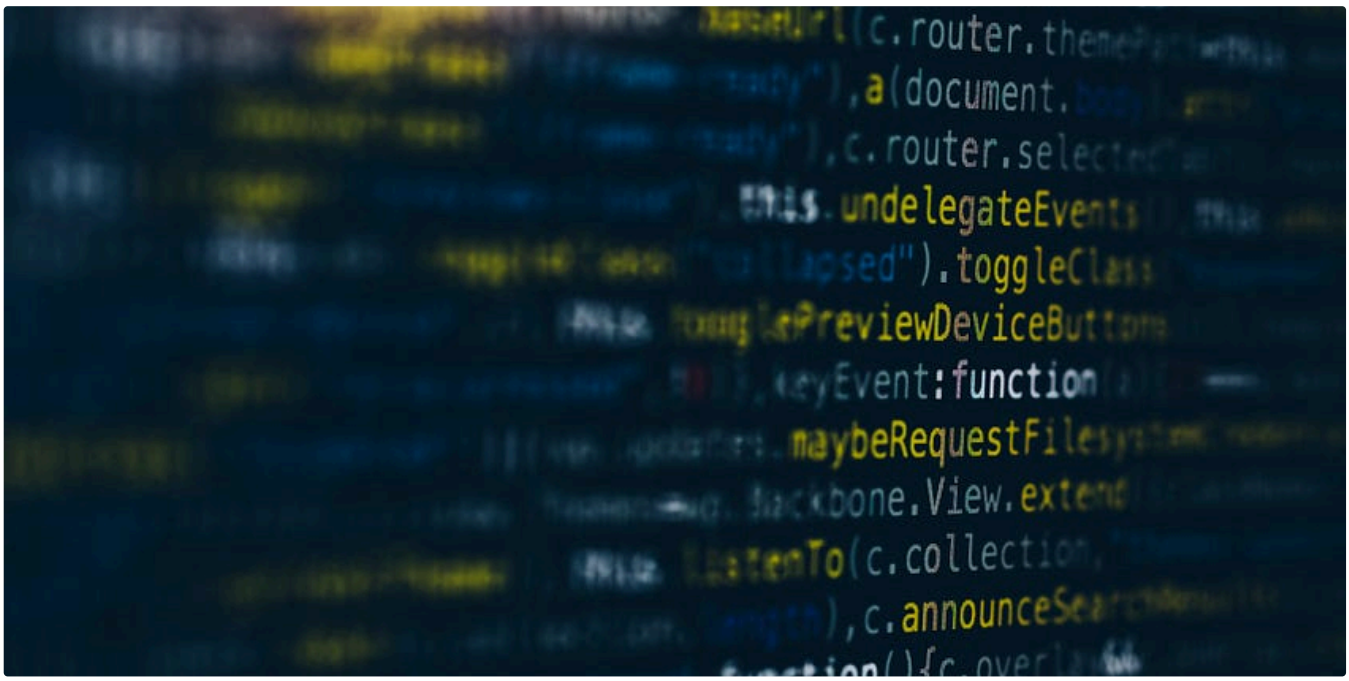
Dave Melillo in Towards Data Science

Building a Data Platform in 2024

How to build a modern, scalable data platform to power your analytics and data science projects (updated)

9 min read · Feb 6, 2024

1.6K 25



Ujjwal Kumar in Artificial Intelligence in Plain English

Less Labeled Data? Here's the Solution: The SimCLRv2

The complication of learning information from only a few labeled data has troubled machine learning researchers for a long time, especially...

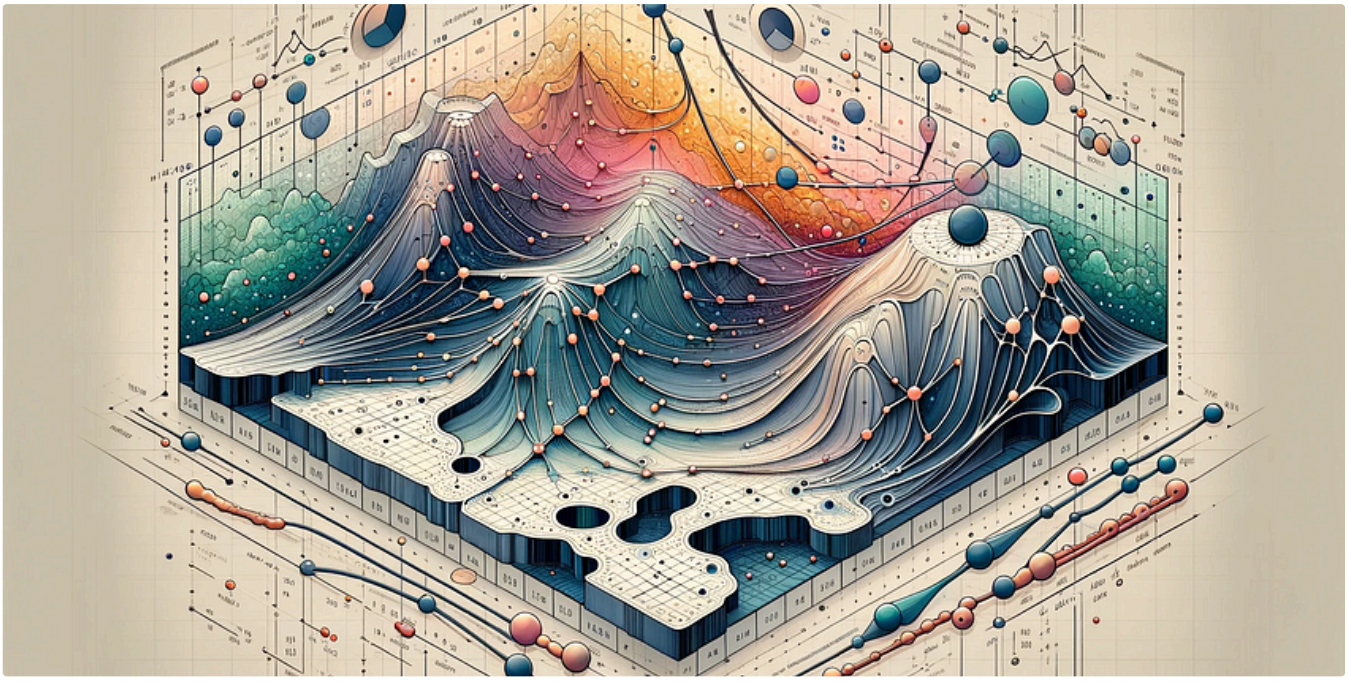
2 min read · Jun 18, 2021

82

See all from Ujjwal Kumar

See all from Towards Data Science

Recommended from Medium



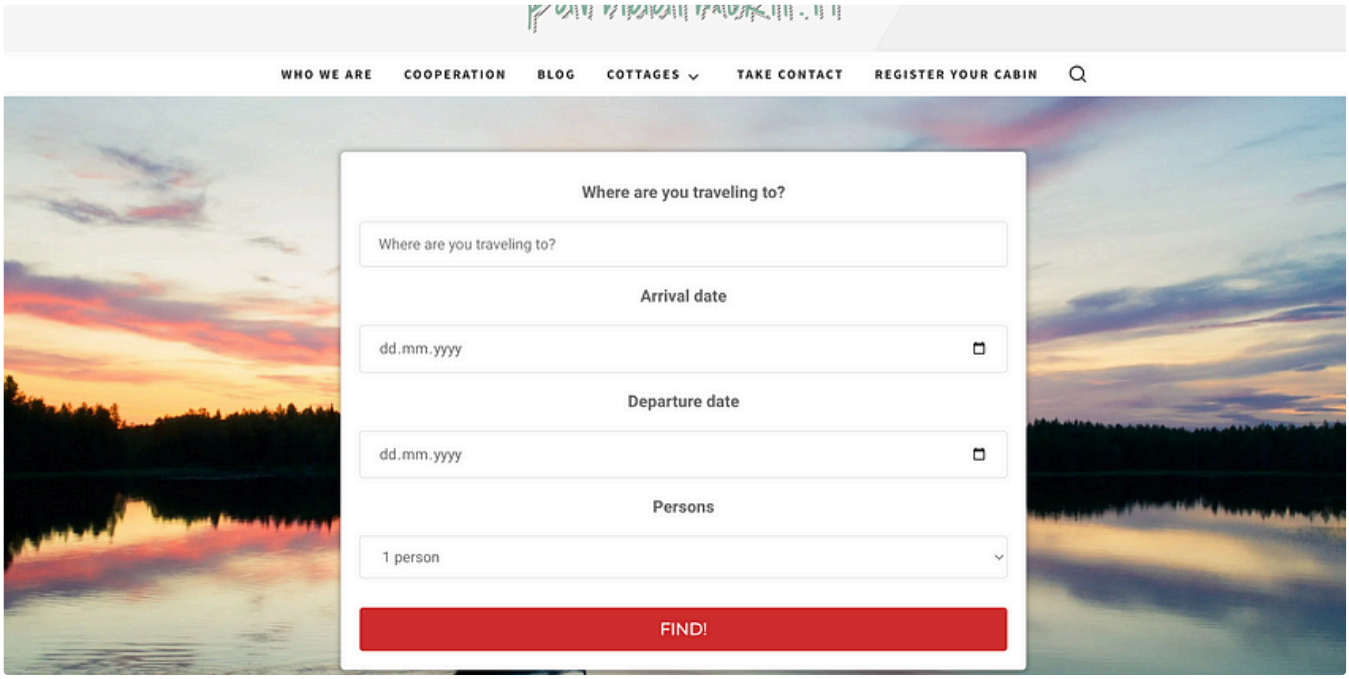
Cristian Leo in Towards Data Science

The Math behind Adam Optimizer

Why is Adam the most popular optimizer in Deep Learning? Let's understand it by diving into its math, and recreating the algorithm.

16 min read · Jan 30, 2024

2.4K 19



Artturi Jalli

I Built an App in 6 Hours that Makes $1,500/Mo

Copy my strategy!

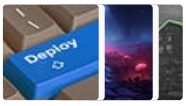
· 3 min read · Jan 23, 2024

12.1K 146

Lists

Practical Guides to Machine Learning

10 stories · 1152 saves

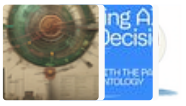


Predictive Modeling w/ Python

20 stories · 969 saves

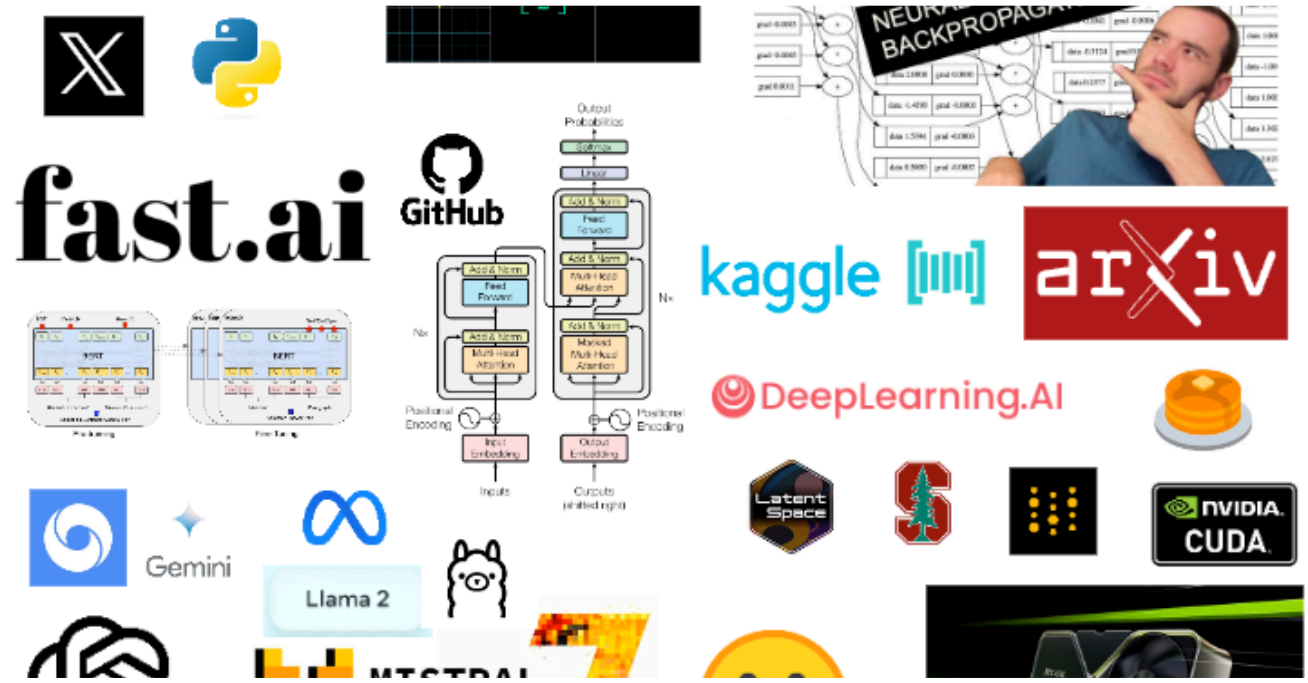
Natural Language Processing

1254 stories · 739 saves



data science and AI

40 stories · 94 saves



Benedict Neo in bitgrit Data Science Publication

Roadmap to Learn AI in 2024

A free curriculum for hackers and programmers to learn AI

11 min read · Feb 21, 2024

4.2K 51

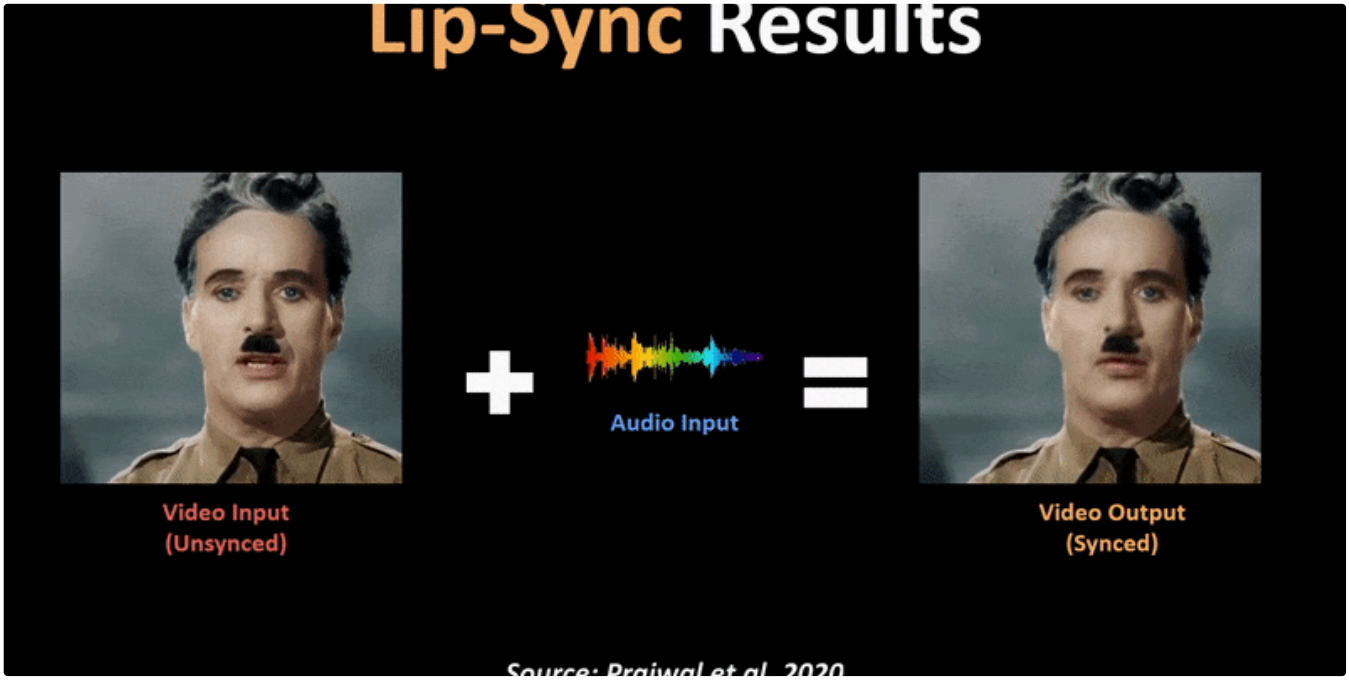
Ignacio de Gregorio

Google Has Finally Dethroned ChatGPT

They Finally Did It

· 10 min read · Feb 22, 2024

5.8K 111



Katarzyna Sornat

Deepfake video with custom text and original voice made easily

6 min read · Sep 18, 2023

23 1

Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

· 4 min read · Feb 16, 2022

76K 1068

See more recommendations