Chapter 1 : A Complete Guide on Getting Started with Deep Learning in Python

This Keras tutorial introduces you to deep learning in Python: learn to preprocess your data, model, evaluate and optimize neural networks.

There are two basic ways of initialising a neural network, either by a sequence of layers or as a graph. For building this particular neural network, we are using a Maxpooling function, there exist different types of pooling operations like Min Pooling, Mean Pooling, etc. Here in MaxPooling we need the maximum value pixel from the respective region of interest. Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector. Now, we will create an object of the sequential class below: The Conv2D function is taking 4 arguments, the first is the number of filters i. Now, we need to perform pooling operation on the resultant feature maps we get after the convolution operation is done on an image. The primary aim of a pooling operation is to reduce the size of the images as much as possible. In order to understand what happens in these steps in more detail you need to read few external resources. But the key thing to understand here is that we are trying to reduce the total number of nodes for the upcoming layers. Again, to understand the actual math behind Pooling, i suggest you to go learn from an external source, this tutorial concentrates more on the implementation part. Flattening is a very important step to understand. What we are basically doing here is taking the 2-D array, i. In this step we need to create a fully connected layer, and to this layer we are going to connect the set of nodes we got after the flattening step, these nodes will act as an input layer to these fully-connected layers. As this layer will be present between the input layer and output layer, we can refer to it a hidden layer. And the activation function will be a rectifier function. This single node will give us a binary output of either a Cat or Dog. Optimizer parameter is to choose the stochastic gradient descent algorithm. Loss parameter is to choose the loss function. Finally, the metrics parameter is to choose the performance metric. But before we do that, we are going to pre-process the images to prevent over-fitting. Overfitting is when you get a great training accuracy and very poor test accuracy due to overfitting of nodes from one layer to another. So before we fit our images to the neural network, we need to perform some image augmentations on them, which is basically synthesising the training data. We are going to do this using keras.

Chapter 2 : Python Programming Tutorials

The tutorials presented here will introduce you to some of the most important deep learning algorithms and will also show you how to run them using calendrierdelascience.com is a python library that makes writing deep learning models easy, and gives the option of training them on a GPU.

Click here to download the source code to this post. Inside this Keras tutorial, you will discover how easy it is to get started with deep learning and Python. The inspiration for this guide came from PyImageSearch reader, Igor, who emailed me a few weeks ago and asked: Hey Adrian, thanks for the PyImageSearch blog. I just call one of those functions and the data is automatically loaded for me. But how do I go about using my own image dataset with Keras? What steps do I have to take? Igor has a great point â€" most Keras tutorials you come across will try to teach you the basics of the library using an image classification dataset such MNIST handwriting recognition or CIFAR basic object recognition. To learn how to get started with Keras, Deep Learning, and Python, just keep reading! Looking for the source code to this post? Jump right to the downloads section. Writing some Keras code. And then training our networks on our custom datasets. This tutorial is not meant to be a deep dive into the theory surrounding deep learning. Our example dataset Figure 1: In fact, your training and testing splits have already been pre-split for you! Where are those helper functions loading the data from? What format should my dataset on disk be? How can I load my dataset into memory? What preprocessing steps do I need to perform? Instead, you want to work with your own custom datasets. The dataset consists of dogs, cats, and pandas. The purpose of this dataset is to correctly classify an image as containing either: Cats Dogs Pandas Containing only 3, images, the Animals dataset is meant to be an introductory dataset that we can quickly train a deep learning model on using either our CPU or GPU and still obtain reasonable accuracy. Furthermore, using this custom dataset enables you to understand: If you need to scrape images from the internet to create a dataset, check out how to do it the easy way with Bing Image Search , or the slightly more involved way with Google Images. Project structure There are a number of files associated with this project.

Chapter 3 : Keras Tutorial: How to get started with Keras, Deep Learning, and Python - PyImageSearch

Introduction. When I started my deep learning journey, one of the first things I learned was image classification. It's such a fascinating part of the computer vision fraternity and I was completely immersed in it!

Content provided by IBM. In this article, we will discuss the meaning of Deep Learning With Python. Also, we will learn why we call it Deep Learning. Deep Learning Definition To define it in one sentence, we would say it is an approach to Machine Learning. To elaborate, Deep Learning is a method of Machine Learning that is based on learning data representations or feature learning instead of task-specific algorithms. We also call it deep structured learning or hierarchical learning. For feature learning, we observe three kinds of learning: The patterns we observe in biological nervous systems inspires vaguely the deep learning models that exist. Characteristics Some characteristics of Python Deep Learning are: They use a cascade of layers of nonlinear processing units to extract features and perform transformation; the output at one layer is the input to the next. These learn multiple levels of representations for different levels of abstraction. Deep Learning uses networks where data transforms through a number of layers before producing the output. This is something we measure by a parameter often dubbed CAP. The Credit Assignment Path depth tells us a value one more than the number of hidden layers- for a feedforward neural network. Each layer takes input and transforms it to make it only slightly more abstract and composite. An Artificial Neural Network is a connectionist system. It is a computing system that, inspired by the biological neural networks from animal brains, learns from examples. Structure An Artificial Neural Network is nothing but a collection of artificial neurons that resemble biological ones. Synapses connections between these neurons transmit signals to each other. A postsynaptic neuron processes the signal it receives and signals the neurons connected to it further. A neuron can have state a value between 0 and 1 and a weight that can increase or decrease the signal strength as the network learns. We see three kinds of layers: There may be any number of hidden layers. Typically, such networks can hold around millions of units and connections. Note that this is still nothing compared to the number of neurons and connections in a human brain. At each layer, the network calculates how probable each output is. A DNN will model complex non-linear relationships when it needs to. With extra layers, we can carry out the composition of features from lower layers. Typically, a DNN is a feedforward network that observes the flow of data from input to output. It never loops back. Deep Neural Network creates a map of virtual neurons and assigns weights to the connections that hold them together. It multiplies the weights to the inputs to produce a value between 0 and 1. This is to make parameters more influential with an ulterior motive to determine the correct mathematical manipulation so we can fully process the data. Two kinds of ANNs we generally observe are: Where data can flow in any direction. Convolutional Deep Neural Networks: A deep, feedforward ANN.

Chapter 4 : Deep Learning â€“ Python Tutorial

Python is a general-purpose high level programming language that is widely used in data science and for producing deep learning algorithms. This brief tutorial introduces Python and its libraries like Numpy, Scipy, Pandas, Matplotlib; frameworks like Theano, TensorFlow, Keras.

In this part, and the next few parts, we will be considering the addition of deep learning, but, first, we have to decide what form of deep learning to use! I believe the most applicable form of machine learning to apply in this case is an evolutionary application of deep learning. The idea of Q-learning is to distribute a reward or penalty across steps for a given pass through an environment. While we could work within slices of the StarCraft II environment to rectify this issue Evolutionary algorithms are similar to reinforcement learning algorithms, so much so that I would argue that they are a form of reinforcement learning algorithms. The main idea of reinforcement learning is to reinforce good choices, through an end target result. One of the major pitfalls of reinforcement learning is that people forget that they should be just simply reinforcing an end result, and nothing else, letting the algorithm figure out how to best get to that end result without human bias. What else reinforces some end result? With an evolutionary algorithm, in the case of StarCraft II, you allow the winning algorithm to be a part of the gene pool training data , and the loser is forgotten. What I propose we do first is simply figure out attacking, or at least try. I do want to highlight that I really do not know the answer here. This is all trial and error for me now. Maybe Q-learning is the best choice, or maybe something else is, or some other form of structuring things than I will here. This is just going to be my journey through trying things and sharing it with you. Our script from before: Next, our offensive force method now becomes: I think it would more interesting to do it on a per-unit basis, but more complex. Thus, as an army, we either all attack, or not. Next, how do we make this decision? The challenge instead becomes: How do we inform the neural network? Well, in the case of this game, we have a lot of variables, and the amount of variables is We have a variable number of workers and a variable number of military units. Immediately, I would consider visualizing this data, and passing that to the network. WELL, not so fast there Johnny. We need data first. A lot of data. Mainly, these images tied to actions. So, first, we have to put in some effort to build the imagery that we want to pass to the network. To start, we need to import numpy and opencv. You may need to install these. Now for our intel method: To initially start with a blacked-out "screen" that is the size of the game map. Next, we can draw something like our Nexuses on the map with: So then we need to flip the image: Other than that, we can visualize it with: You should now see something like:

Chapter 5 : Deep learning Tutorial for Video Classification using Python

In this course, you'll gain hands-on, practical knowledge of how to use deep learning with Keras , the latest version of a cutting edge library for deep learning in Python. Learn the fundamentals of neural networks and how to build deep learning models using Keras

Key concepts in review 9. Different brands of approaches to AI 9. What makes deep learning special within machine learning 9. How to think about deep learning 9. Key enabling technologies 9. The universal machine learning workflow 9. Key network architectures 9. The space of possibilities 9. Mapping image data to vector data 9. Mapping timeseries data to vector data 9. The limitations of deep learning 9. The risk of anthropomorphizing machine learning models 9. Local generalization versus extreme generalization 9. The future of deep learning 9. Models as programs 9. Beyond backpropagation and differentiable layers 9. Automated machine learning 9. Lifelong learning and modular subroutine reuse 9. Staying up to date in a fast-moving field 9. Practice on real-world problems using Kaggle 9. Read about the latest developments on Arxiv 9. Explore the Keras ecosystem 9. Final words Appendix A: Installing Keras and its dependencies on Ubuntu A. Installing the Python scientific suite A. Setting up GPU support A. Installing Theano optional Appendix B: What are Jupyter notebooks? Why would you not want to use Jupyter on AWS for deep learning? Setting up local port forwarding B. Using Jupyter from your local browser About the Technology Machine learning has made remarkable progress in recent years. We went from near-unusable speech and image recognition, to near-human accuracy. Behind this progress is deep learningâ€”a combination of engineering advances, best practices, and theory that enables a wealth of previously impossible smart applications. About the book Deep Learning with Python introduces the field of deep learning using the Python language and the powerful Keras library. No previous experience with Keras, TensorFlow, or machine learning is required. He is the creator of the Keras deep-learning library, as well as a contributor to the TensorFlow machine-learning framework. He also does deep-learning research, with a focus on computer vision and the application of machine learning to formal reasoning.

Chapter 6 : Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python

The Keras library for deep learning in Python WTF is Deep Learning? Deep learning refers to neural networks with multiple hidden layers that can learn increasingly abstract representations of the input data.

It had many recent successes in computer vision, automatic speech recognition and natural language processing. The goal of this blog post is to give you a hands-on introduction to deep learning. This post is divided into 2 main parts. The first part covers some core concepts behind deep learning, while the second part is structured in a hands-on tutorial format. In the second part of the tutorial section 5 , we will cover an advanced technique for training convolutional neural networks called transfer learning. We will use some Python code and a popular open source deep learning framework called Caffe to build the classifier. By the end of this post, you will understand how convolutional neural networks work, and you will get familiar with the steps and the code for building these networks. The source code for this tutorial can be found in this github repository. Problem Definition In this tutorial, we will be using a dataset from Kaggle. The dataset is comprised of 25, images of dogs and cats. Our goal is to build a machine learning algorithm capable of detecting the correct animal cat or dog in new unseen images. In Machine learning, this type of problems is called classification. Classification using Traditional Machine Learning vs. Deep Learning Classification using a machine learning algorithm has 2 phases: In this phase, we train a machine learning algorithm using a dataset comprised of the images and their corresponding labels. In this phase, we utilize the trained model to predict labels of unseen images. The training phase for an image classification problem has 2 main steps: In this phase, we utilize domain knowledge to extract new features that will be used by the machine learning algorithm. In the predicition phase, we apply the same feature extraction process to the new images and we pass the features to the trained machine learning algorithm to predict the label. The main difference between traditional machine learning and deep learning algorithms is in the feature engineering. In traditional machine learning algorithms, we need to hand-craft the features. By contrast, in deep learning algorithms feature engineering is done automatically by the algorithm. Feature engineering is difficult, time-consuming and requires domain expertise. The promise of deep learning is more accurate machine learning algorithms compared to traditional machine learning with less or no feature engineering. In addition to algorithmic innovations, the increase in computing capabilities using GPUs and the collection of larger datasets are all factors that helped in the recent surge of deep learning. Artificial Neural Networks vs. Biological Neural Networks Biological Neurons are the core components of the human brain. A neuron consists of a cell body, dendrites, and an axon. It processes and transmit information to other neurons by emitting electrical signals. Each neuron receives input signals from its dendrites and produces output signals along its axon. The axon branches out and connects via synapses to dendrites of other neurons. A basic model for how the neurons work goes as follows: Each synapse has a strength that is learnable and control the strength of influence of one neuron on another. If the final sum is above a certain threshold, the neuron get fired, sending a spike along its axon. An artificial neuron has a finite number of inputs with weights associated to them, and an activation function also called transfer function. The output of the neuron is the result of the activation function applied to the weighted sum of inputs. Artificial neurons are connected with each others to form artificial neural networks. These networks have 3 types of layers: Input layer, hidden layer and output layer. In these networks, data moves from the input layer through the hidden nodes if any and to the output nodes. Below is an example of a fully-connected feedforward neural network with 2 hidden layers. Note that, the number of hidden layers and their size are the only free parameters. The larger and deeper the hidden layers, the more complex patterns we can model in theory. Activation Functions Activation functions transform the weighted sum of inputs that goes into the artificial neurons. These functions should be non-linear to encode complex patterns of the data. ReLU is the most popular activation function in deep neural networks. We need 2 elements to train an artificial neural network: In the case of image classification, the training data is composed of images and the corresponding labels. A function that measures the inaccuracy of predictions. Once we have the 2 elements above, we train the ANN using an algorithm called backpropagation together with gradient descent or one of its derivatives. For a detailed explanation of backpropagation, I recommend this article. These models are designed to emulate the behaviour of a visual cortex. CNNs perform very well on visual recognition tasks. CNNs have special layers called convolutional layers and pooling layers that allow the network to encode certain images properties. Convolution Layer This layer consists of a set of learnable filters that we slide over the image spatially, computing dot products between the entries of the filter and the input image. The filters should extend to the full depth of the input image. For example, if we want to apply a filter of size 5x5 to a colored image of size 32x32, then the filter should have depth 3 5x5x3 to cover all 3 color channels Red, Green, Blue of the image. These filters will activate when they see same specific structure in the images. Pooling Layer Pooling is a form of non-linear down-sampling. The goal of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. There are several functions to implement pooling among which max pooling is the most common one. Pooling is often applied with filters of size 2x2 applied with a stride of 2 at every depth slice. The convolutional layers are usually followed by one layer of ReLU activation functions. The convolutional, pooling and ReLU layers act as learnable features extractors, while the fully connected layers acts as a machine learning classifier. Furthermore, the early layers of the network encode generic patterns of the images, while later layers encode the details patterns of the images. Note that only the convolutional layers and fully-connected layers have weights. These weights are learned in the training phase. To implement the convolutional neural network, we will use a deep learning framework called Caffe and some Python code. We will upload our predictions to Kaggle to get the score of our prediction model. Please note, that the AMI recommended in the guide is no longer available. I prepared a new AMI amid with all the necessary software installed. After setting up an AWS instance, we connect to it and clone the github repository that contains the necessary Python code and Caffe configuration files for the tutorial. From your terminal, execute the following command. There are 4 steps in training a CNN using Caffe: Step 1 - Data preparation: In this step, we clean the images and store them in a format that can be used by Caffe. We will write a Python script that will handle both image pre-processing and storage. Step 2 - Model definition: In this step, we choose a CNN architecture and we define its parameters in a configuration file with extension. Step 3 - Solver definition: The solver is responsible for model optimization. We define the solver parameters in a configuration file with extension. Step 4 - Model training: We train the model by executing one Caffe command from the terminal. After training the model, we will get the trained model in a file with extension. After the training phase, we will use the. We will write a Python script to this. We can do this using the scp command from a MAC or linux machine. After copying the data, we unzip the files by executing the following commands: Run histogram equalization on all training images. Histogram equalization is a technique for adjusting the contrast of images. Resize all training images to a x format. Divide the training data into 2 sets: The training set is used to train the model, and the validation set is used to calculate the accuracy of the model. Store the training and validation in 2 LMDB databases. Below is the explanation of the most important parts of the code. The code for storing validation data follows the same structure. Generating the mean image of training data We execute the command below to generate the mean image of training data. We will substract the mean image from each input image to ensure every feature pixel has zero mean.

Chapter 7 : Simple Image Classification using Convolutional Neural Network â€” Deep Learning in python.

The objective of this course is to give you a holistic understanding of machine learning, covering theory, application, and inner workings of supervised, unsupervised, and deep learning algorithms.

Welcome everyone to an updated deep learning with Python and Tensorflow tutorial mini-series. Since doing the first deep learning with TensorFlow course a little over 2 years ago, much has changed. This is more of a deep learning quick start! To begin, we need to find some balance between treating neural networks like a total black box, and understanding every single detail with them. A basic neural network consists of an input layer, which is just your data, in numerical form. After your input layer, you will have some number of what are called "hidden" layers. A hidden layer is just in between your input and output layers. One hidden layer means you just have a neural network. Two or more hidden layers? Well, if you just have a single hidden layer, the model is going to only learn linear relationships. If you have many hidden layers, you can begin to learn non-linear relationships between your input and output layers. A single neuron might look as follows: So this is really where the magic happens. The idea is a single neuron is just sum of all of the inputs x weights, fed through some sort of activation function. The activation function is meant to simulate a neuron firing or not. A simple example would be a stepper function, where, at some point, the threshold is crossed, and the neuron fires a 1, else a 0. The mathematical challenge for the artificial neural network is to best optimize thousands or millions or whatever number of weights you have, so that your output layer results in what you were hoping for. Solving for this problem, and building out the layers of our neural network model is exactly what TensorFlow is for. TensorFlow is used for all things "operations on tensors. To install TensorFlow, simply do a: One such library that has easily become the most popular is Keras. Keras has become so popular, that it is now a superset, included with TensorFlow releases now! By that same token, if you find example code that uses Keras, you can use with the TensorFlow version of Keras too. In fact, you can just do something like: You can figure out your version: But, for now, woo! What exactly do we have here? The testing variants of these variables is the "out of sample" examples that we will use. Neural networks are exceptionally good at fitting to data, so much so that they will commonly over-fit the data. This typically involves scaling the data to be between 0 and 1, or maybe -1 and positive 1. In our case, each "pixel" is a feature, and each feature currently ranges from 0 to Not quite 0 to 1. It just means things are going to go in direct order. A feed forward model. Recall our neural network image? Was the input layer flat, or was it multi-dimensional? So, we need to take this 28x28 image, and make it a flat 1x Flatten This will serve as our input layer. Next, we want our hidden layers. Just like our image. The activation function is relu, short for rectified linear. Currently, relu is the activation function you should just default to. It has 10 nodes. Great, our model is done. Now we need to "compile" the model. Same thing is true for the Adam optimizer. Next, we have our loss metric. Loss is a calculation of error. It attempts to minimize loss. Again, there are many choices, but some form of categorical crossentropy is a good start for a classification task like this. Getting a high accuracy and low loss might mean your model learned how to classify digits in general it generalized Sequential a basic feed-forward model model. Flatten takes our 28x28 and makes it 1x model. Softmax for probability distribution model. Finally, with your model, you can save it super easily:

Chapter 8 : Python Deep Learning Tutorial

Welcome everyone to an updated deep learning with Python and Tensorflow tutorial mini-series. Since doing the first deep learning with TensorFlow course a little over 2 years ago, much has changed.

This is pretty handy as it confirms the structure of our network for us. Training the network Next we have to setup an optimizer and a loss criterion: The other ingredient we need to supply to our optimizer is all the parameters of our network â€” thankfully PyTorch make supplying these parameters easy by the. Module class that we inherit from in the Net class. Next, we set our loss criterion to be the negative log likelihood loss â€” this combined with our log softmax output from the neural network gives us an equivalent cross entropy loss for our 10 classification classes. During training, I will be extracting data from a data loader object which is included in the PyTorch utilities module. So by using data. We can pass a batch of input data like this into our network and the magic of PyTorch will do all the hard work by efficiently performing the required operations on the tensors. In other libraries this is performed implicitly, but in PyTorch you have to remember to do it explicitly. This is opposed to other deep learning libraries such as TensorFlow and Keras which require elaborate debugging sessions to be setup before you can check out what your network is actually producing. The second line is where we get the negative log likelihood loss between the output of our network and our target batch data. If you compare this with our review of the. Scalar variables, when we call. The next line is where we tell PyTorch to execute a gradient descent step based on the gradients calculated during the. Finally, we print out some results every time we reach a certain number of iterations: Note how you access the loss â€” you access the Variable. We access the scalar loss by executing loss. Now we have the prediction of the neural network for each sample in the batch determined, we can compare this with the actual target class from our training data, and count how many times in the batch the neural network got it right. We can use the PyTorch. Finally, after running through the test data in batches, we print out the averaged loss and accuracy: So there you have it â€” this PyTorch tutorial has shown you the basic ideas in PyTorch, from tensors to the autograd functionality, and finished with how to build a fully connected neural network using the nn. I hope it was helpful. Practical Deep Learning with PyTorch.

Chapter 9 : Deep Learning for Beginners | Deeplearning4j

A PyTorch tutorial - the basics. In this section, we'll go through the basic ideas of PyTorch starting at tensors and computational graphs and finishing at the Variable class and the PyTorch autograd functionality.

The next natural step is to talk about implementing recurrent neural networks in Keras. In a previous tutorial of mine , I gave a very comprehensive introduction to recurrent neural networks and long short term memory LSTM networks, implemented in TensorFlow. A recurrent neural network is a neural network that attempts to model time or sequence dependent behaviour â€" such as language, stock prices, electricity demand and so on. It looks like this: Unrolled recurrent neural network Here you can see that at each time step, a new word is being supplied â€" the output of the previous F i. This is because small gradients or weights values less than 1 are multiplied many times over through the multiple time steps, and the gradients shrink asymptotically to zero. LSTM networks are a way of solving this problem. These cells have various components called the input gate, the forget gate and the output gate â€" these will be explained more fully later. Here is a graphical representation of the LSTM cell: An input gate is a layer of sigmoid activated nodes whose output is multiplied by the squashed input. This variable, lagged one time step i. The mathematics of the LSTM cell looks like this: This can be expressed by: Note that the exponents g are not a raised power, but rather signify that these are the input weights and bias values as opposed to the input gate, forget gate, output gate etc. Forget gate and state loop The forget gate output is expressed as: Likewise, all the weights and bias values are matrices and vectors respectively. Now, you may be wondering, how do we represent words to input them to a neural network? The answer is word embedding. Basically it involves taking a word and finding a vector representation of that word which captures some meaning of the word. In Word2Vec, this meaning is usually quantified by context â€" i. The word vectors can be learnt separately, as in this tutorial , or they can be learnt during the training of your Keras LSTM network. We have to specify the size of the embedding layer â€" this is the length of the vector each word is represented by â€" this is usually in the region of between In other words, if the embedding layer size is , each word will be represented by a length vector i. You might be wondering where the hidden layers in the LSTM cell come from. This will further illuminate some of the ideas expressed above, including the embedding layer and the tensor sizes flowing around the network. The proposed architecture looks like the following: In other words, for each batch sample and each word in the number of time steps, there is a length embedding word vector to represent the input word. These embedding vectors will be learnt as part of the overall model learning. This output data is then passed to a Keras layer called TimeDistributed, which will be explained more fully below. This will be made more clear later. Note, you first have to download the Penn Tree Bank PTB dataset which will be used as the training and validation corpus. The text preprocessing code In order to get the text data into the right shape for input into the Keras LSTM model, each unique word in the corpus must be assigned a unique integer index. Then the text corpus needs to be re-constituted in order, but rather than text words we have the integer identifiers in order. Finally, the original text file is converted into a list of these unique integers, where each word is substituted with its new integer identifier. This allows the text data to be consumed in the neural network. This allows us to work backwards from predicted integer words that our model will produce, and translate them back to real text. This code snippet produces: The Python iterator function needs to have a form like: The initialization of this class looks like: Note, this data can be either training, validation or test data â€" multiple instances of the same class can be created and used in the various stages of our machine learning development cycle â€" training, validation tuning, test. In other words pun intended , this is the set of words that the model will learn from to predict the words coming after. To make this a bit clearer, consider the following sentence: Hopefully that makes sense. One final item in the initialization of the class needs to be discussed. In other words it is basically a data set location pointer. The second dimension is the number of words we are going to base our predictions on. First it has the batch size as the first dimension, then it has the number of time steps as the second, as discussed above. It will look something like this: Therefore, for each target word, there needs to be a 10, length vector with only one of the elements in this vector set to 1. The final step is converting each of the target words in each sample into the one-hot or categorical representation that was discussed previously. This function takes a series of integers as its first arguments and adds an additional dimension to the vector of integers â€” this dimension is the one-hot representation of each integer. Now that the generator class has been created, we need to create instances of it. As mentioned previously, we can setup instances of the same class to correspond to the training and validation data. In the code, this looks like the following: Basically, the sequential methodology allows you to easily stack layers into your network without worrying too much about all the tensors and their shapes flowing through the model. However, you still have to keep your wits about you for some of the more complicated layers, as will be discussed below. In this example, it looks like the following: The first layer in the network, as per the architecture diagram shown previously, is a word embedding layer. This will convert our words referenced by integers in the data into meaningful embedding vectors. This Embedding layer takes the size of the vocabulary as its first argument, then the size of the resultant embedding vector that you want as the next argument. Note that Keras, in the Sequential model, always maintains the batch size as the first dimension. It receives the batch size from the Keras fitting function i. The next layer is the first of our two LSTM layers. The diagram below shows what I mean: In this case, we want the latter arrangement. Well, in this example we are trying to predict the very next word in the sequence. Therefore, for both stacked LSTM layers, we want to return all the sequences. After that, there is a special Keras layer for use in recurrent neural networks called TimeDistributed. This function adds an independent layer for each time step in the recurrent model. So, for instance, if we have 10 time steps in a model, a TimeDistributed layer operating on a Dense layer would produce 10 independent Dense layers, one for each time step. The activation for these dense layers is set to be softmax in the final layer of our Keras LSTM model. For more on callbacks, see my Keras tutorial. The callback that is used in this example is a model checkpoint callback â€” this callback saves the model after each epoch, which can be handy for when you are running long-term training. The line below shows you how to do this: The next argument is the number of iterations to run for each training epoch. Likewise, a generator for the smaller validation data set is called, with the same argument for the number of iterations to run. At the end of each epoch, the validation data will be run through the model and the accuracy will be returned. Now the model is good to go! Before some results are presented â€” some caveats are required. My model parameters for the results presented below are as follows: Then a loop of dummy data extractions from the generator is created â€” this is to control where in the data-set the comparison sentences are drawn from. First, a batch of data is extracted from the generator and this is passed to the model. In other words, each word is represented by a vector of 10, items, with most being zero and only one element being equal to 1. This function identifies the index where the maximum value occurs in a vector â€” in this case the maximum value is 1, compared to all the zeros, so this is a handy function for us to use. This English word is then added to the predicted words string, and finally the actual and predicted words are returned. The output below is the comparison between the actual and predicted words after 10 epochs of training on the training data set: Comparison on the training data set after 10 epochs of training As can be observed, while some words match, after 10 epochs of training the match is pretty poor. Comparison on the test data set after 40 epochs of training Despite there not being a perfect correspondence between the predicted and actual words, you can see that there is a rough correspondence and the predicted sub-sentence at least makes some grammatical sense. So not so bad after all. However, in order to train a Keras LSTM network which can perform well on this realistic, large text corpus, more training and optimization is required. I will leave it up to you, the reader, to experiment further if you desire. However, the current code is sufficient for you to gain an understanding of how to build a Keras LSTM network, along with an understanding of the theory behind LSTM networks. Zero to Deep Learning with Python and Keras.