DOWNLOAD PDF HTML TO DEEP LEARNING

Chapter 1 : LSTM Networks for Sentiment Analysis â€” DeepLearning documentation

Deep Learning Pipelines builds on Apache Spark's ML Pipelines for training, and with Spark DataFrames and SQL for deploying models. It includes high-level APIs for common aspects of deep learning so they can be done efficiently in a few lines of code.

In this tutorial, this model is used to perform sentiment analysis on movie reviews from the Large Movie Review Dataset , sometimes known as the IMDB dataset. In this task, given a movie review, the model attempts to predict whether it is positive or negative. This is a binary classification task. While the dataset is public, in this tutorial we provide a copy of the dataset that has previously been preprocessed according to the needs of this LSTM implementation. Running the code provided in this tutorial will automatically download the data to the local directory. In order to use your own data, please use a preprocessing script provided as a part of this tutorial. Once the model is trained, you can test it with your own corpus using the word-index dictionary imdb. This means that, the magnitude of weights in the transition matrix can have a strong impact on the learning process. If the weights in this matrix are small or, more formally, if the leading eigenvalue of the weight matrix is smaller than 1. It can also make more difficult the task of learning long-term dependencies in the data. Conversely, if the weights in this matrix are large or, again, more formally, if the leading eigenvalue of the weight matrix is larger than 1. This is often referred to as exploding gradients. These issues are the main motivation behind the LSTM model which introduces a new structure called a memory cell see Figure 1 below. A memory cell is composed of four main elements: The self-recurrent connection has a weight of 1. The gates serve to modulate the interactions between the memory cell itself and its environment. The input gate can allow incoming signal to alter the state of the memory cell or block it. On the other hand, the output gate can allow the state of the memory cell to have an effect on other neurons or prevent it. Illustration of an LSTM memory cell. The equations below describe how a layer of memory cells is updated at every timestep.

Chapter 2 : Top 8 Free Must-Read Books on Deep Learning

Deploy deep learning models anywhere including CUDA, C code, enterprise systems, or the cloud. When performance matters, you can generate code that leverages optimized libraries from Intel Â® (MKL-NNN), NVIDIA (TensorRT, cuDNN), and ARM Â® (ARM Compute Library) to create deployable models with high-performance inference speed.

Consider the following sequence of handwritten digits: Most people effortlessly recognize those digits as That ease is deceptive. In each hemisphere of our brain, humans have a primary visual cortex, also known as V1, containing million neurons, with tens of billions of connections between them. And yet human vision involves not just V1, but an entire series of visual cortices - V2, V3, V4, and V5 - doing progressively more complex image processing. We carry in our heads a supercomputer, tuned by evolution over hundreds of millions of years, and superbly adapted to understand the visual world. Rather, we humans are stupendously, astoundingly good at making sense of what our eyes show us. But nearly all that work is done unconsciously. The difficulty of visual pattern recognition becomes apparent if you attempt to write a computer program to recognize digits like those above. What seems easy when we do it ourselves suddenly becomes extremely difficult. Simple intuitions about how we recognize shapes - "a 9 has a loop at the top, and a vertical stroke in the bottom right" - turn out to be not so simple to express algorithmically. When you try to make such rules precise, you quickly get lost in a morass of exceptions and caveats and special cases. Neural networks approach the problem in a different way. The idea is to take a large number of handwritten digits, known as training examples, and then develop a system which can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy. The program is just 74 lines long, and uses no special neural network libraries. But this short program can recognize digits with an accuracy over 96 percent, without human intervention. In fact, the best commercial neural networks are now so good that they are used by banks to process cheques, and by post offices to recognize addresses. As a prototype it hits a sweet spot: Of course, if the point of the chapter was only to write a computer program to recognize handwritten digits, then the chapter would be much shorter! Throughout, I focus on explaining why things are done the way they are, and on building your neural networks intuition. Perceptrons What is a neural network? Perceptrons were developed in the s and s by the scientist Frank Rosenblatt , inspired by earlier work by Warren McCulloch and Walter Pitts. So how do perceptrons work? In general it could have more or fewer inputs. Rosenblatt proposed a simple rule to compute the output. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms: Let me give an example. You like cheese, and are trying to decide whether or not to go to the festival. You might make your decision by weighing up three factors: Is the weather good? Does your boyfriend or girlfriend want to accompany you? Is the festival near public transit? You can use perceptrons to model this kind of decision-making. It makes no difference to the output whether your boyfriend or girlfriend wants to go, or whether public transit is nearby. By varying the weights and the threshold, we can get different models of decision-making. Then the perceptron would decide that you should go to the festival whenever the weather was good or when both the festival was near public transit and your boyfriend or girlfriend was willing to join you. But what the example illustrates is how a perceptron can weigh up different kinds of evidence in order to make decisions. And it should seem plausible that a complex network of perceptrons could make quite subtle decisions: What about the perceptrons in the second layer? Each of those perceptrons is making a decision by weighing up the results from the first layer of decision-making. In this way a perceptron in the second layer can make a decision at a more complex and more abstract level than perceptrons in the first layer. And even more complex decisions can be made by the perceptron in the third layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making. Incidentally, when I defined perceptrons I said that a perceptron has just a single output. In the network above the perceptrons look like they have multiple outputs. The multiple output arrows are merely a useful way of indicating that the output from a perceptron is being used as the input to several other perceptrons. Using the

bias instead of the threshold, the perceptron rule can be rewritten: Or to put it in more biological terms, the bias is a measure of how easy it is to get the perceptron to fire. And so our perceptron implements a NAND gate! The NAND example shows that we can use perceptrons to compute simple logical functions. In fact, we can use networks of perceptrons to compute any logical function at all. One notable aspect of this network of perceptrons is that the output from the leftmost perceptron is used twice as input to the bottommost perceptron. With that change, the network looks as follows, with all unmarked weights equal to -2, all biases equal to 3, and a single weight of -4, as marked: This notation for input perceptrons, in which we have an output, but no inputs, is a shorthand. To see this, suppose we did have a perceptron with no inputs. The adder example demonstrates how a network of perceptrons can be used to simulate a circuit containing many NAND gates. And because NAND gates are universal for computation, it follows that perceptrons are also universal for computation. The computational universality of perceptrons is simultaneously reassuring and disappointing. However, the situation is better than this view suggests. It turns out that we can devise learning algorithms which can automatically tune the weights and biases of a network of artificial neurons. This tuning happens in response to external stimuli, without direct intervention by a programmer. These learning algorithms enable us to use artificial neurons in a way which is radically different to conventional logic gates. Instead of explicitly laying out a circuit of NAND and other gates, our neural networks can simply learn to solve problems, sometimes problems where it would be extremely difficult to directly design a conventional circuit. Sigmoid neurons Learning algorithms sound terrific. But how can we devise such algorithms for a neural network? For example, the inputs to the network might be the raw pixel data from a scanned, handwritten image of a digit. To see how learning might work, suppose we make a small change in some weight or bias in the network. If it were true that a small change in a weight or bias causes only a small change in output, then we could use this fact to modify the weights and biases to get our network to behave more in the manner we want. For example, suppose the network was mistakenly classifying an image as an "8" when it should be a "9". We could figure out how to make a small change in the weights and biases so the network gets a little closer to classifying the image as a "9". The network would be learning. That flip may then cause the behaviour of the rest of the network to completely change in some very complicated way. So while your "9" might now be classified correctly, the behaviour of the network on all the other images is likely to have completely changed in some hard-to-control way. That makes it difficult to see how to gradually modify the weights and biases so that the network gets closer to the desired behaviour. We can overcome this problem by introducing a new type of artificial neuron called a sigmoid neuron. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output. Okay, let me describe the sigmoid neuron. In fact, there are many similarities between perceptrons and sigmoid neurons, and the algebraic form of the sigmoid function turns out to be more of a technical detail than a true barrier to understanding. How can we understand that? This shape is a smoothed out version of a step function: But you get the idea.. This linearity makes it easy to choose small changes in the weights and biases to achieve any desired small change in the output. So while sigmoid neurons have much of the same qualitative behaviour as perceptrons, they make it much easier to figure out how changing the weights and biases will change the output. How should we interpret the output from a sigmoid neuron? This can be useful, for example, if we want to use the output value to represent the average intensity of the pixels in an image input to a neural network. But sometimes it can be a nuisance. Suppose we want the output from the network to indicate either "the input image is a 9" or "the input image is not a 9". Suppose also that the overall input to the network of perceptrons has been chosen. In preparation for that, it helps to explain some terminology that lets us name different parts of a network. Suppose we have the network: As mentioned earlier, the leftmost layer in this network is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layer is called a hidden layer, since the neurons in this layer are neither inputs nor outputs. The term "hidden" perhaps sounds a little mysterious - the first time I heard the term I thought it must have some deep philosophical or mathematical significance - but it really means nothing more than "not an input or an output". The network above has just a single hidden layer, but some networks have multiple hidden layers. For

example, the following four-layer network has two hidden layers:

Chapter 3 : The limitations of deep learning

Deep learning was only a theory until recently when teams around the world started using NVIDIA GPUs. Today, it's empowering organizations to transform moonshots into real results. AI Across Industries.

It is part of a series of two posts on the current limitations of deep learning, and its future. This post is targeted at people who already have significant experience with deep learning e. We assume a lot of pre-existing knowledge. Ten years ago, no one expected that we would achieve such amazing results on machine perception problems by using simple parametric models trained with gradient descent. Now, it turns out that all you need is sufficiently large parametric models trained with gradient descent on sufficiently many examples. In deep learning, everything is a vector, i. Model inputs it could be text, images, etc and targets are first "vectorized", i. Each layer in a deep learning model operates one simple geometric transformation on the data that goes through it. Together, the chain of layers of the model forms one very complex geometric transformation, broken down into a series of simple ones. This complex transformation attempts to maps the input space to the target space, one point at a time. This transformation is parametrized by the weights of the layers, which are iteratively updated based on how well the model is currently performing. A key characteristic of this geometric transformation is that it must be differentiable, which is required in order for us to be able to learn its parameters via gradient descent. Intuitively, this means that the geometric morphing from inputs to outputs must be smooth and continuousâ€”a significant constraint. The whole process of applying this complex geometric transformation to the input data can be visualized in 3D by imagining a person trying to uncrumple a paper ball: Each movement operated by the person on the paper ball is similar to a simple geometric transformation operated by one layer. The full uncrumpling gesture sequence is the complex transformation of the entire model. Deep learning models are mathematical machines for uncrumpling complicated manifolds of high-dimensional data. All you need are spaces of sufficiently high dimensionality in order to capture the full scope of the relationships found in the original data. The limitations of deep learning The space of applications that can be implemented with this simple strategy is nearly infinite. And yet, many more applications are completely out of reach for current deep learning techniquesâ€”even given vast amounts of human-annotated data. Say, for instance, that you could assemble a dataset of hundreds of thousandsâ€”even millionsâ€”of English language descriptions of the features of a software product, as written by a product manager, as well as the corresponding source code developed by a team of engineers to meet these requirements. Even with this data, you could not train a deep learning model to simply read a product description and generate the appropriate codebase. In general, anything that requires reasoningâ€”like programming, or applying the scientific methodâ€”long-term planning, and algorithmic-like data manipulation, is out of reach for deep learning models, no matter how much data you throw at them. Even learning a sorting algorithm with a deep neural network is tremendously difficult. This is because a deep learning model is "just" a chain of simple, continuous geometric transformations mapping one vector space into another. All it can do is map one data manifold X into another manifold Y, assuming the existence of a learnable continuous transform from X to Y, and the availability of a dense sampling of X: Y to use as training data. So even though a deep learning model can be interpreted as a kind of program, inversely most programs cannot be expressed as deep learning modelsâ€”for most tasks, either there exists no corresponding practically-sized deep neural network that solves the task, or even if there exists one, it may not be learnable, i. Scaling up current deep learning techniques by stacking more layers and using more training data can only superficially palliate some of these issues. It will not solve the more fundamental problem that deep learning models are very limited in what they can represent, and that most of the programs that one may wish to learn cannot be expressed as a continuous geometric morphing of a data manifold. The risk of anthropomorphizing machine learning models One very real risk with contemporary AI is that of misinterpreting what deep learning models do, and overestimating their abilities. A fundamental feature of the human mind is our "theory of mind", our tendency to project intentions, beliefs and knowledge on the things around us. Drawing a smiley face on a rock suddenly makes it "happy"â€”in our minds. Applied to deep learning, this means that when we

are able to somewhat successfully train a model to generate captions to describe pictures, for instance, we are led to believe that the model "understands" the contents of the pictures, as well as the captions it generates. We then proceed to be very surprised when any slight departure from the sort of images present in the training data causes the model to start generating completely absurd captions. In particular, this is highlighted by "adversarial examples", which are input samples to a deep learning network that are designed to trick the model into misclassifying them. You are already aware that it is possible to do gradient ascent in input space to generate inputs that maximize the activation of some convnet filter, for instanceâ€”this was the basis of the filter visualization technique we introduced in Chapter 5 Note: Similarly, through gradient ascent, one can slightly modify an image in order to maximize the class prediction for a given class. By taking a picture of a panda and adding to it a "gibbon" gradient, we can get a neural network to classify this panda as a gibbon. This evidences both the brittleness of these models, and the deep difference between the input-to-output mapping that they operate and our own human perception. In short, deep learning models do not have any understanding of their input, at least not in any human sense. Our own understanding of images, sounds, and language, is grounded in our sensorimotor experience as humansâ€”as embodied earthly creatures. Machine learning models have no access to such experiences and thus cannot "understand" their inputs in any human-relatable way. By annotating large numbers of training examples to feed into our models, we get them to learn a geometric transform that maps data to human concepts on this specific set of examples, but this mapping is just a simplistic sketch of the original model in our minds, the one developed from our experience as embodied agentsâ€”it is like a dim image in a mirror. They were trained on a different, far narrower task than the one we wanted to teach them: Show them anything that deviates from their training data, and they will break in the most absurd ways. Local generalization versus extreme generalization There just seems to be fundamental differences between the straightforward geometric morphing from input to output that deep learning models do, and the way that humans think and learn. Aside from the different learning processes, there is a fundamental difference in the nature of the underlying representations. Humans are capable of far more than mapping immediate stimuli to immediate responses, like a deep net, or maybe an insect, would do. They maintain complex, abstract models of their current situation, of themselves, of other people, and can use these models to anticipate different possible futures and perform long-term planning. They are capable of merging together known concepts to represent something they have never experienced beforeâ€”like picturing a horse wearing jeans, for instance, or imagining what they would do if they won the lottery. This ability to handle hypotheticals, to expand our mental model space far beyond what we can experience directly, in a word, to perform abstraction and reasoning, is arguably the defining characteristic of human cognition. I call it "extreme generalization": This stands in sharp contrast with what deep nets do, which I would call "local generalization": Consider, for instance, the problem of learning the appropriate launch parameters to get a rocket to land on the moon. If you were to use a deep net for this task, whether training using supervised learning or reinforcement learning, you would need to feed it with thousands or even millions of launch trials, i. By contrast, humans can use their power of abstraction to come up with physical modelsâ€”rocket scienceâ€”and derive an exact solution that will get the rocket on the moon in just one or few trials. Similarly, if you developed a deep net controlling a human body, and wanted it to learn to safely navigate a city without getting hit by cars, the net would have to die many thousands of times in various situations until it could infer that cars and dangerous, and develop appropriate avoidance behaviors. Dropped into a new city, the net would have to relearn most of what it knows. On the other hand, humans are able to learn safe behaviors without having to die even onceâ€”again, thanks to their power of abstract modeling of hypothetical situations. In short, despite our progress on machine perception, we are still very far from human-level AI: Doing this well is a game-changer for essentially every industry, but it is still a very long way from human-level AI. To lift some of these limitations and start competing with human brains, we need to move away from straightforward input-to-output mappings, and on to reasoning and abstraction. A likely appropriate substrate for abstract modeling of various situations and concepts is that of computer programs. We have said before Note: But what if we could learn any program, in a modular and reusable way?

Chapter 4 : Neural networks and deep learning

The Deep Learning textbook is a resource intended to help students and practitioners enter the field of machine learning in general and deep learning in particular. The online version of the book is now complete and will remain available online for free.

How It Works The evolution of deep learning Deep learning is one of the foundations of artificial intelligence AI , and the current interest in deep learning is due in part to the buzz surrounding AI. Deep learning techniques have improved the ability to classify, recognize, detect and describe â€" in one word, understand. For example, deep learning is used to classify images, recognize speech, detect objects and describe content. Systems such as Siri and Cortana are powered, in part, by deep learning. Several developments are now advancing deep learning: Algorithmic improvements have boosted the performance of deep learning methods. New machine learning approaches have improved accuracy of models. New classes of neural networks have been developed that fit well for applications like text translation and image classification. We have a lot more data available to build neural networks with many deep layers, including streaming data from the Internet of Things , textual data from social media, physicians notes and investigative transcripts. Computational advances of distributed cloud computing and graphics processing units have put incredible computing power at our disposal. This level of computing power is necessary to train deep algorithms. At the same time, human-to-machine interfaces have evolved greatly as well. The mouse and the keyboard are being replaced with gesture, swipe, touch and natural language, ushering in a renewed interest in AI and deep learning. In this deep learning example, the computer program is learning to interpret animal tracks to help with animal conservation. Deep learning opportunities and applications A lot of computational power is needed to solve deep learning problems because of the iterative nature of deep learning algorithms, their complexity as the number of layers increase, and the large volumes of data needed to train the networks. The dynamic nature of deep learning methods â€" their ability to continuously improve and adapt to changes in the underlying information pattern â€" presents a great opportunity to introduce more dynamic behavior into analytics. Greater personalization of customer analytics is one possibility. Another great opportunity is to improve accuracy and performance in applications where neural networks have been used for a long time. Through better algorithms and more computing power, we can add greater depth. While the current market focus of deep learning techniques is in applications of cognitive computing, there is also great potential in more traditional analytics applications, for example, time series analysis. Another opportunity is to simply be more efficient and streamlined in existing analytical operations. Recently, SAS experimented with deep neural networks in speech-to-text transcription problems. Compared to the standard techniques, the word-error-rate decreased by more than 10 percent when deep neural networks were applied. They also eliminated about 10 steps of data preprocessing, feature engineering and modeling. The impressive performance gains and the time savings when compared to feature engineering signify a paradigm shift. Learn more about what people are saying. Looking beyond the deep learning hype Why is deep learning unequaled among machine learning techniques? This step-by-step guide compares multiple neural network models and explains how to use them. Download the paper Analyze streaming sports data with deep learning SciSports uses streaming data and applies machine learning, deep learning and artificial intelligence to capture and analyze this data, making way for innovations in everything from player recruitment to virtual reality for fans Learn more How is deep learning being used? To the outside eye, deep learning may appear to be in a research phase as computer science researchers and data scientists continue to test its capabilities. However, deep learning has many practical applications that businesses are using today, and many more that will be used as research continues. Popular uses today include: Speech Recognition Both the business and academic worlds have embraced deep learning for speech recognition.

Chapter 5 : MATLAB for Deep Learning - MATLAB & Simulink

DOWNLOAD PDF HTML TO DEEP LEARNING

These deep learning techniques are based on stochastic gradient descent and backpropagation, but also introduce new ideas. These techniques have enabled much deeper (and larger) networks to be trained - people now routinely train networks with 5 to 10 hidden layers.

Chapter 6 : The Fundamentals of Deep Learning

Deep learning is a type of machine learning that uses deep neural networks to train a computer to perform human-like tasks, like recognizing speech or images.

Chapter 7 : Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books

Deep learning, a sub-mechanism of artificial intelligence (AI), is a computing process that enables machines to find patterns in data. Deep learning sifts through data and makes connections, identifying odd trends that would be unfindable by the human senses.

Chapter 8 : Deep Learning & Artificial Intelligence Solutions from NVIDIA

Deep Learning can be used by undergraduate or graduate students planning careers in either industry or research, and by software engineers who want to begin using deep learning in their products or platforms.

Chapter 9 : What is deep learning? | SAS

Get started with deep learning via a tutorial with helpful tips and resources, aimed at data scientists and engineers who need to run deep learning applications at scale.