Convolutional and Recurrent Neural Networks

Some tasks aren't suitable for traditional neural networks and require specialized neural networks. Explore convolutional and recurrent neural networks and the types of problems they can solve.

Table of Contents

Introducing Convolutional Neural Networks (CNNs)

[Topic title: Introducing Convolutional Neural Networks (CNNs). The presenter is Jamie Campbell.] In this video, I'll discuss Convolutional Neural Networks. How they're different from regular neural networks and how they're used. Deep learning neural networks have empowered modern computing tasks. Providing breakthroughs in task processing that couldn't realistically be performed without deep learning. One such breakthrough has been convolutional neural networks – or CNNs – a special type of neural network. A convolutional layer is a neural network layer that isn't connected and includes weight-sharing units. CNNs use many copies of an identical neuron or unit. The benefit being that a network can have many neurons and take on complex computational models while minimizing the parameters taken on by the neurons. Because they're identical neurons, they share the same behavior values. Convolutional neural networks have been very popular for their implementation in image and audio processing. A big challenge with typical neural networks is that they consider an entire input all at once. For applications like image processing, for example, each of these three representations of the letter A would be treated as different inputs. Even though they're just the same character represented in different locations or angles. The solution to this involves preprocessing, which is powerful but limited in its implementation.

Or convolutional input, which grabs small portions of an image and looks at just those. Here's how the convolutional neural network processes an image. Convolution input refers to overlapping functions and input. Using a filter to obtain a portion of an image, feeding what's in the filter to a convolutional neural network neuron. Then the filter is moved, usually by a pixel. And the process is repeated until the entire image has been processed. When discussing convolutional neural networks, we speak of pooling layers, which are interwoven with convolutional layers. They're processing layers used for downsampling. If you've worked with image editing programs like Photoshop, then you'll be familiar with downsampling which is the process of taking an image file and shrinking its size. While that sounds pretty straightforward, a lot goes on in the background to shrink an image and make it look like the original at the end because downsampling requires that information, pixels, be removed from an image and you can't just randomly choose a pixel of a specific color and remove it. You have to analyze the area around that pixel and understand what the area is trying to represent. The pooling layer simplifies the input by reducing the number of parameters.

It also reduces computations and overfitting, a statistical anomaly that occurs when data noise obfuscates any relationship that the data might normally indicate. And the pooling layer is typically inserted between convolutional layers. Max pooling takes a maximum of a sample of fields to make new fields. For example, a 2x2 field of an image is analyzed and the number of pixels in that area is reduced by two on the width and height to make a new parameter and reduce a number of pixels in that area by 75%. There are some common uses for convolutional neural networks. I've already discussed image processing and that's where much of the original research for CNNs originated. Things like character recognition and image tagging are ideal for CNNs. And game playing is a popular use of CNNs. Artificial intelligence use convolutional networks to assess the current state of a game and make decisions based on that. Games like Go and some of the older Atari arcade games have been popular subjects for convolutional neural networks.

CNN Architecture

[Topic title: CNN Architecture. The presenter is Jamie Campbell.] In this video, I'll discuss the high-level architecture of convolutional neural networks. *[Basic Classification CNN Architecture]* In the convolutional neural network, there are several layers beginning with the input layer and here there are four inputs. *[x1, x2, x3, and x4.]* Then there's the convolutional layer or layers, the rectified linear unit activation layer, pooling layers, or layers for downsampling, and the fully connected layer or layers, *[y1, y2, and y3.]* then the softmax layer, which normalizes the data is next. *[Architecture Terminology]* So let's deal with the architecture terminology. The convolutional layer extracts different features from each part of the image. The filter assigns weights to any area of an image. For example, a 5x5 filter would represent a 5 pixel by 5 pixel selection of the image. The feature map is a new image that's created when the filter is applied to the input. Stride occurs when the filter moves across the image. Think of an imaginary box that surrounds a 5X5 section and moves to the left by one pixel at a time when processing is complete – by one pixel when processing is complete on the 5X5 section. Stride size is how much to move, so a stride of 1 would move the filter by 1 pixel. Pooling combines variables into one. So, for example, taking the average value of 4 pixels. *[Example CNN: LeNet-5]*

LeNet-5 was one of the first convolutional neural networks. It was developed to recognize characters from 32x32 images. And the neural network has 7 layers – convolutional layer with 5x5 filters resulting in 6 28x28 feature maps, pooling layers, which downsample images to a 14x14 feature map, 2 more convolutional layers, and 1 more pooling layer, 1 fully connected layer, and 1 output layer. *[LeNet-5 Visualized]* Here's a visualization of LeNet-5. The input is a 32x32 image, like the hand-drawn X here. A convolutional layer with 5x5 filters resulting in 6 28x28 feature maps. Pooling layers downsample the image to feature maps, two convolutional layers, one fully connected layer, and one output layer. Then finally, the softmax layer normalizes the data and the output is the letter X.

Convolution Layers

[Topic title: Convolution Layers. The presenter is Jamie Campbell.] In this video, I'll describe how convolution layers are set in convolutional neural networks. *[Applying Filters]* In convolutional neural networks, a filter is a set of weights applied to a section of an image. Filters can be visualized as small images. So a 5x5 filter is a filter that's 5 pixels high and 5 pixels wide. A third value represents the color depth. So a 5x5x3 filter is a 5x5 pixel filter with 3 color channels. To apply a filter, the dot product of the image and the filter is taken. *[Building Feature Maps]* When building feature maps, stride size is how many pixels to move the filter. A stride of one, the most popular choice, moves the input by one pixel. Think of it as having an imaginary square around a 5x5 area and then, once processing in that area is complete, you move it one pixel to the right. To build a feature map, the output of each filter is taken and an image is built. *[Weight Sharing]* Then there's the stride size. Filters are neural network units described by this image. *[The image of a neural network unit is displayed where w1 and w2 are described as the weights, pixel1 and pixel2 are described as the input attributes, and B is the described as the bias weight.]* To implement strides, the same unit is copied multiple times. Each unit shares the same weights and each unit outputs part of a feature map.

Pooling Layer

[Topic title: Pooling Layer. The presenter is Jamie Campbell.] In this video, I'll discuss pooling layers and how they're used in convolutional neural networks. In convolutional neural networks, pooling reduces the number of parameters in a network. For example, downsampled images, which when reduced in size, need to have information, that is, pixels removed. In the convolutional neural network topology, pooling layers are inserted between convolutional layers.

The advantages of pooling layers include that they reduce the amount of required computations and memory and they reduce overfitting. *[2x2 Max Pooling]* In 2x2 max pooling, for every 2x2 square, the max pixel is returned. A returned image will have 25% of the original pixels. So 75% fewer activations to compute. For example, a 32x32 pixel image will be reduced to 16x16 pixels. *[Other Pooling]* And there are other pooling types. In average pooling, the pixels we're downsampling are averaged. In global average pooling, the average of multiple feature maps is taken so they can be combined. In a basic use case, you replace a final, fully connected layer by building a feature map for each class passing all the maps in a global average pooling layer and then performing softmax on the average map to normalize the data.

CNN Training Considerations

[Topic title: CNN Training Considerations. The presenter is Jamie Campbell.] In this video, I'll discuss some training considerations for convolutional neural networks. Convolutional neural networks are a special type of neural network. A convolutional layer is a neural network layer that isn't connected and includes weight-sharing units. Convolutional neural networks are popular for several kinds of practical applications including image processing. But they require training. So here are several considerations for training. *[Preprocessing]* First, in the preprocessing phase, mean subtraction, where the mean is subtracted from each example is performed. This centers the data on the origin. Then normalization occurs where you divide subtracted data by the standard deviation. This puts all the data on the same scale. Next is zero padding where you add zeros around the data, for example, adding pixels around an image.

If a dataset is split in a training and test dataset, preprocessing should be performed after splitting occurs. Preprocessing also has to be performed on objects you want to predict. *[Initializing Weights]* When initializing weights, weights can't all be initialized to zero. And they'll always have the same values when updating. Weights can be initialized with small random numbers. But this causes high-weight variance. So, to decrease variance, you divide each weight by the square root of the number of inputs. And bias initialization may or may not affect performance. It's typically initialized to zero. But it's sometimes initialized to a big enough number to fire all the rectified linear unit, or ReLU, activations. *[Training CNNs]* In CNN training, backpropagation, which determines the amount of error that each node adds to data using gradient descent, can be used. Fully connected layers are trained like normal neural networks. Pooling layers contain no weights. So they don't need training. And convolutional layers are trained like normal neural networks. Since weights are shared, gradients are shared among them.

Regularization

[Topic title: Regularization. The presenter is Jamie Campbell.] In this video I'll discuss regularization and how it applies to convolutional neural networks. *[Regularization]* In neural network design, regularization is a method used to reduce overfitting, a statistical anomaly that occurs when data noise obfuscates any relationship that the data might normally indicate. Regularization punishes models with the potential for overfitting. It uses weight decay, which punishes high weights by decaying them over time. If an outlier causes a spike in weight, it will eventually move towards zero. If a weight truly should be high, many examples will outpace the decay.

[L1 and L2 Regularization] L1 and L2 regularization are two common methods of regularization. In L1 regularization, the weight decay is represented like this. *[lambda multiplied by the absolute value of w]* It encourages small weights, that is, weights close to 0. And small weights make noise less relevant. In L2 regularization, the weight decay is represented this way. *[lambda w square divided by 2]* It discourages the neural network's dependence on a small subset of inputs. For example, it considers more inputs more equally. Generally speaking, L2 regularization performs better. *[Other Regularization Methods]* There are other regularization methods too. Max norm constraint limits the magnitude of a unit's weights, magnitude being represented with this formula. *[magnitude of W is equal to underroot of w1 square added to w2 square that is added upto wn square. It is also known as Pythagorean theorem.]* And constraint can be enforced when updating. And dropout randomly skips some of the activation when training a dataset. The probability p can be set as a hyperparameter, so p = 0.5 is common. And dropout should not be applied when making predictions.

Regularization Methods for CNNs

[Topic title: Regularization Methods for CNNs. The presenter is Jamie Campbell.] In this video, I'll describe regularization of convolutional neural networks. In neural network design, regularization is a method used to produce overfitting – a statistical anomaly that occurs when data noise obfuscates any relationships that the data might normally indicate. Regularization punishes models with the potential for overfitting. There are several common methods for regularizing CNNs. *[Regularization Methods for CNNs]* First, you can modify datasets. By providing more training data, overfitting models may experience better performance. And, using a dropout layer, nodes are randomly dropped including the nodes' inputs and outputs.

In dropout layers, overfitting is reduced. And this is an efficient method for averaging models. Nodes become less affected by other nodes' weights. In early stopping, the number of steps for hyperparameters is tweaked. Early stopping is a regularization method that prevents overfitting with iterative methods like gradient descent. Generally the learning model is updated so better fitting occurs with the training data. And weight penalties for L1 and L2 regularization is a popular method for training models. In weight penalties, the assumption is that models with large weights are not preferable and that models with small weights are less complex. Penalties are applied to keep the weights down. Ideally no weights at all are preferred. The exception would be if there are large gradients.

Convolutional Neural Network Implementation

[Topic title: Convolutional Neural Network Implementation. The presenter is Jamie Campbell.] In this video, I'll describe implementing and training convolutional neural networks. *[The TensorFlow-Tutorial / 02_Convolutional_Neural_Network.ipynb webpage is open.]* A big challenge with typical neural networks is that they consider an entire input all at once. For applications like image processing, for example, three different handwriting samples in different locations and angles will be treated as different inputs even though they're just the same characters represented in different locations or angles. The solution to this involves preprocessing, which is a powerful but limited in its implementation or convolutional input, which grabs small portions of an image and looks at just those.

Convolutional neural networks are a special type of neural network. A convolutional layer is a neural network layer that isn't connected and includes weight-sharing units. Convolutional neural networks are popular for several kinds of practical applications, but they require training, so there are several considerations. This tutorial on GitHub is a good example of how a convolutional neural network in TensorFlow can be implemented and trained. *[He scrolls down. There is a Flowchart section.]* It implements a CNN with an accuracy nearing at 100%. And you can tweak the accuracy by using some of the suggested exercises. In the flowchart section, you can see a visual representation of how the data moves through the convolutional network. In this tutorial, an input image of 28x28 pixels is taken and filter weights of 5x5 pixels are applied on the convolutional layer 1. 14x14 pixel images are filtered on convolutional layer 2 with filter weights of 5x5 pixels. 7x7 pixel images are output to the fully connected layer with 128 neurons and after processing, sent to the output layer with 10 neurons. From that, classifications are generated, effectively the number contained in the input image, which is seven in this case. Now I recommend you try this tutorial out. There's a lot of code here and more than 50 steps in the coding process with some helpful feedback from the author.

Introducing Recurrent Neural Networks (RNNs)

[Topic title: Introducing Recurrent Neural Networks (RNNs). The presenter is Jamie Campbell.] In this video, I'll discuss Recurrent Neural Networks. Recurrent Neural Networks address the missing link between human understanding and neural network understanding. *[Recurrent Neural Networks (RNNs)]* A recurrent layer is a layer that feeds output into itself sequentially with the output of the last iteration used in the next iteration essentially retaining information. It forms a chain of units that are copies providing information to successive nodes in the network. *[Why Use RNNs?]* So why are Recurrent Neural Networks useful? Well, first, traditional neural networks don't have any memory. They forget after every step. This makes it difficult to use neural networks for time-based or

sequential data where past events affect present events. For example, time series, video sequencing, or natural language processing. *[NN Example: Machine Translation]*

Here's a neural network example for machine translation. In this example, using a neural network to translate the English phrase, the dog runs, into French. *[The example contains an English input 'The dog runs' which is translated through the Neural network into the French output 'Le chien court'.]* Classification is when there's a finite set of attributes that can be assigned to an item – for example – a pencil can have a color, a weight, a length, a diameter, and so on – a finite set of attributes. But the output can't be weighted as class because there are infinite classes. That is language translation is not often a straight one-to-one translation. And the output can't easily be mapped as regression. In other words, the model can't easily learn how wrong it is. *[RNN Example: Machine Translation]* In this Recurrent Neural Network example for machine translation, we have RNNs operating to map the translation. It's a fairly simple example because the order of the definite article the, the subject dog and the object runs appear in the same order in English and French for this example. But that's rarely the case in plain language translation. So with RNNs, the network can learn grammar rules and word associations using finite classes of words.

RNN Types

[Topic title: RNN Types. The presenter is Jamie Campbell.] In this video, I'll describe the different types of recurrent neural networks. There are different types of recurrent neural networks. *[Types of Recurrent Networks]* For example, there's one input to many outputs that is describing the input as in this graphic example. *[A diagram is displayed that shows one input and three outputs of the Recurrent Neural Network(RNN).]* There's also many inputs to a single output as in this graphic example. *[A diagram is displayed that shows one output and three inputs of the Recurrent Neural Network(RNN).]* This is commonly used for tasks like sentiment analysis where a Tweet or a Facebook post would be analyzed. And a decision would be made whether the person making the post was happy, sad, angry, frustrated, and so on. In that case, perhaps the input is product comments where a customer is commenting on a company's product. And the output would be how happy the poster is with the product.

[Many Inputs to Many Outputs] In many inputs to many outputs, the input and output can be unsynced. For example, in machine translation from one language to another where the order of the words is not as important as grammatical rules and meaning or they can be synced, for example, identifying and tagging frames in a video, which is a sequential process. *[Using RNNs For Nonsequential Input]* When using RNNs for nonsequential input, nonsequential input can be taken sequentially to use a recurrent neural network. The input can be used fully for input where features have different meanings depending on other features. For example, images where you need different items to form a face, a nose, a mouth, eyes, ears, and so on. And recurrent neural networks for images are similar in the way they work to the way convolutional neural networks work.

Long Short Term Memory (LSTMs) in TensorFlow

[Topic title: Long Short Term Memory (LSTMs) in TensorFlow. The presenter is Jamie Campbell.] In this video, I'll discuss how to implement a Long Short Term Memory network. Connecting neural network units recurrently offers limited functionality because our current neural network can only use the output from the last iteration. So they have no true memory. Long short term memory – or LSTM networks – are recurrent neural networks that can learn relationships over time. The solution, add a

memory cell. LSTM networks manage a memory cell based on inputs. Weights are used to learn what to forget and what to remember and weights are also used to learn how to use memory. In the LSTM structure, a cell represents memory. X and plus represent multiply and add gates. A sigmoid is a sigmoid or logistic unit. Tanh is a unit. And t is the current time or iteration.

In terms of how LSTM remembers, there are three main steps. First, step one deals with how much to remember. A sigmoid returns a number between 0 and 1. This is used as a percentage of what to remember *[The function is in the transcript for reference at the heading Sigmoid function.]* using this function. Step 2 deals with what to add to memory. Tanh extracts information to add using this function. A sigmoid decides the percentage of information to add using this function. *[The function is in the transcript for reference at the heading Tanh function.]* And the compiled information is added to the cell. Step 3 builds the output. Tanh extracts information from cells using this formula. The sigmoid decides the percentage of the information to add using this formula. And finally the output is sent to the subsequent input.

Sigmoid function

σ(W. [Xt, ht-1] +b)

Tanh function

tanh(w.[Xt, ht-1]+b

RNNs and LSTM for Language Modeling

[Topic title: RNNs and LSTM for Language Modeling. The presenter is Jamie Campbell.] In this video, I'll discuss an example for using RNNs and LSTM for language modeling. *[The Recurrent Neural Networks | TensorFlow webpage is open.]* Recurrent Neural Networks address the missing link between human understanding and neural network understanding. A recurrent layer is a layer that sequentially feeds output into itself with the output from the last iteration used in the next one, essentially, retaining information. But connecting neural network units recurrently offers limited functionality because RNN networks can only utilize the output from the last iteration. So they don't really have memory. Long short-term memory or LSTM networks are recurrent neural networks that can learn relationships over time. They add a memory cell and LSTM networks manage a memory cell based on inputs. Weights are used to learn what to forget and what to remember. And weights are used to learn how to utilize memory. In the LSTM structure, a cell represents memory. x and + represent multiply and add gates. Sigmoid is a logistic unit, tan h as a unit, and t is the current time or iteration. Now this tutorial demonstrates the training of a recurrent neural network for language modeling.

The goal is to fit a probabilistic model and assign probabilities to sentences. This is accomplished through prediction. What are the subsequent words given history of previous words? The model contains an LSTM layer and it computes a word at a time and determines probabilities that predict each new word in a sentence. Now the tutorial uses files from the TensorFlow models repository. *[He scrolls down and clicks the TensorFlow models repo link under the Tutorial Files section. The GitHub – tensorflow/models: Models and examples built with TensorFlow webpage opens. It includes several tabs such as Code, Issues, Wiki, and Insights. The Code tab is selected.]* So click that and then download the ZIP file here. *[He clicks the Clone or download drop-down button under the Models* *built with TensorFlow section. A pop-up box appears. It contains two buttons Open in Desktop and Download ZIP. He clicks the Download ZIP button.]* Once you have the ZIP file extracted to the home directory, *[He clicks the back arrow in the browser and goes back to the Recurrent Neural Networks | TensorFlow webpage.]* which in Windows is typically C users and whatever your username is, then download the PTB dataset from Tomas Mikolov's webpage and extract that to the home directory. *[He clicks the PTB dataset from Tomas Mikolov's webpage link under the Download and Prepare the Data section.]* The data set is a preprocessed set of data with 10,000 different words. Now let's go ahead and take a look at the directory. *[He opens the File Explorer window.]* I've already actually downloaded and unzipped these. So I'm here in this directory – that's my Users – my home directory. We have models-master and simple-examples. Those are the two folders that we need. Now I highly recommend you try this yourself and read through the excellent information on this page.

[He points at the Recurrent Neural Networks | TensorFlow webpage.] But I will demonstrate this and tell you that there are a couple of tweaks that have to be made. First the code was created for Python 2. And, in Python 3, the dictionary attribute iteritems has been changed just to items. So you have to open this file. I'll go ahead and show you. *[He switches back to the File Explorer window.]* Let's go back here. It's under models-master>tutorials>rnn, and ptb. And there's the file right there, it's ptb.word_lm.py. You open that in a text editor and modify two lines. I recommend an editor with line numbers because there are more than 500 lines of code. But you can use Search to locate the two instances of iteritems and change them to items if you like. Now, second, the code provided by the tutorial may kickback an error if you don't have a GPU. Now you can fix this by adding num_gpu to this command. Go down here. Let me say that again. Now you can fix this by adding num_ gpus to this command. *[He switches to the Recurrent Neural Networks | TensorFlow webpage. He scrolls down and highlights the commands cd models/tutorials/rnn/ptb and python ptb_word_lm.py --data_path=$HOME/simple-examples/data/ --model=small under the Run the Code section.]* This system doesn't have a CUDA enabled GPU. So I'll have to use this switch. So let's go ahead and run the code. So once you've got the models repository and PTB dataset extracted in the home directory, open a PowerShell session.

Okay, now once you're here, we want to change the home directory. Now, I'm already there, C:\Users\Jamie, but in case you want to know how to do that, cd $home in PowerShell. And then we want to change the directory that contains our code – our script – models-master/tutorials/rnn/ptb. And, finally, we want to run the Python script now. It looks like this – python ptb_word_lm.py. I'm going to put a space there and a back tick because I want a multiline command here just make it easier to look at. So you use the back tick right next to the 1 to do that. Press Enter. Okay, --data_path=$home/simple-examples/data. And then a space --model=small-rnn, _mode. Okay, going back to this space, --rnn_mode=basic. And here's what I was talking about, the GPUs, --num_gpus=0. Okay, so go ahead and press Enter. Now it's going to run. And it's going to take a while, as a matter of fact, quite a long while because I don't have a GPU on this system. So you really should try to use a GPU. And, by the way, these comments are not errors. They're just actually information telling you that there are ways to tweak your CPU if you don't have a GPU to make this run a little bit faster. So I recommend you try out this TensorFlow tutorial yourself and have some fun by tweaking the settings. And I'll just sit back and wait for this to finish. It'll take, as I said, a little while.

Exercise: Image Classification

Exercise

[Topic title: Exercise: Describe Convolution Neural Networks. The presenter is Jamie Campbell.]
Now that you learned about convolutional and recurrent neural networks, it's time to put some of that knowledge to work. In this exercise, you'll describe convolutional neural networks. You'll explain convolutional neural networks, explain common uses for CNNs, describe pooling layers and their advantages, and describe Long Short Term Memory or LSTM. At this point, you can pause this video and perform the exercise. Once you're finished, resume the video and see how I would do it.

Solution

So let's answer these questions. It's okay if you didn't do it exactly the same way. First explain convolutional neural networks. Well, a convolutional layer is a neural network layer that isn't fully connected and has weight sharing units.

Next explain common uses for CNNs. Well, first, they can be used and are commonly used with image processing as well as audio processing. Next describe pooling layers and their advantages. Well, pooling reduces the number of parameters in the network, for example, downsampled images. Next pooling layers are inserted between convolutional layers. And advantages of pooling layers include they reduce the amount of computations and memory needed and they reduce the risk of overfitting. Finally, describe Long Short Term Memory. Well, connecting neural network units recurrently is limited. The network can only use the last iteration's output. So there's no true memory if you will. The solution, add a memory cell. LSTM networks manage memory cells based on inputs. Weights are used to learn what to forget and what to remember and weights are used to learn how to use memory. I hope you found this exercise helpful.