Introduction Course to Autoencoders, VAEs, and GANs

Description

This course provides a practical introduction to Autoencoders, Variational Autoencoders (VAEs), and Generative Adversarial Networks (GANs). Learn how autoencoders compress and reconstruct data, understand the VAE training process, and apply it to image generation with TensorFlow. Explore GAN architecture, training methods, and industrial use cases, with hands-on demos for generating synthetic images.

Autoencoders and Variational Autoencoders (VAE)

Master the fundamentals of autoencoders and VAEs, and apply them to real-world image generation tasks.

- Autoencoders: Understand how autoencoders work and the challenges they face in data reconstruction

- Variational Autoencoders: Learn the core concepts and advantages of VAEs over traditional autoencoders

- VAE Training: Explore the generative training process and steps involved in building a VAE

- Image Generation: Use TensorFlow to implement VAEs for generating images with the MNIST dataset

Generative Adversarial Networks (GAN)

Master the foundations of GANs and apply them to generate synthetic images with real-world relevance.

- GAN Basics: Understand the architecture and purpose of Generative Adversarial Networks

- Training Process: Learn how GANs are trained and explore their application in industry use cases

- Image Synthesis: Generate realistic fake images using GANs in a guided hands-on demo

- Practical Insights: Gain key takeaways to apply GANs in creative and analytical domains

Conclusion

By the end of this course, learners will be equipped to build and apply autoencoders, VAEs, and GANs for real-world image generation tasks. Ideal for aspiring AI engineers, data scientists, and ML practitioners, this course combines foundational theory with practical TensorFlow demos to help professionals design, train, and implement generative models in applied AI projects.

Let us look into learning objectives.

Learning objective will be that we go to analyze the autoencoder or variational autoencoder effectiveness in the data representing the and reconstructing to focus on their structure and output quality that they have now design and implement the variational autoencoder that VAE you can see in short for a specific data modeling task to demonstrate practical skill in neural network and then and configuration and data processing on that overall how we can use it to implement a solution over there.

Analyze generative adversarial network and which is GAN to generate a unique images demonstrate the understanding of architecture and function.

We're going to use some shortcut ways of saying the name guys so VAE is virtual encoder and here we GAN is the generative adversarial network.

So we're going to use the shortcuts here to make it fast to pronounce and all do understand

---

In this lesson, we will be looking into autoencoders.

In this, we want to discuss what are autoencoders or VAE also that we are going to cover in the coming time.

Core idea behind this concept is that we should be able to create or regenerate something from scratch.

In this, we are not trying to copy a data but we are trying to generate a new data by learning from the old data.

Even if those two output and the input are exactly same, the output is generated from scratch not by copying and pasting it.

Let me make it a simple version of it.

Imagine you have a car or image of a car and you want to regenerate the image of that car.

What autoencoder will do for you, it will take that car image as an input, all the features and everything.

In the Latin space, it adds a little bit of randomness in the data.

So make sure that it has to learn from it, not memorizing it.

In simple language, reinforcement part.

So basically, in a simple way, you can say you are trying to make sure that you are not overfitting the things, you're not trying to memorizing the thing, you're trying to learn from it.

Then once the Latin space has this matrix of the spaces or we can say matrix of variance and the mean of the value, you regenerate the output area from scratch.

Now, this output image can be the same image of that car you have given or maybe an improved version of that image in a better way.

This kind of processing can help you to generate new images or say you have a very bad image which is distorted in some way and you recreate that image from scratch in a new fashion.

Let's understand this slide here.

You can see that we have encoder, decoder and Latin space at between, which is basically the code area.

Input layer have you have the input here.

So x is your input in this case.

And in the Latin space, it save the matrix which contain the mean, the average and it also save the variance that you have in the data.

Apart from that, these are more things too, but these are the two main core that followed by the decoder output.

Okay, let's continue on the PPT part.

So we have auto encoders, which is basically an unsupervised learning.

So it's an unsupervised USL part of deep learning for you.

They are used to for dimension reduction part.

So dimension reduction is a very important concept, where you're trying to reduce the dimensions and then recreating it, data compression, and the feature extraction part.

Now model consists of three main component here, an encoder, a Latin space and a decoder.

Encoder which map the input to a lower dimensional presentation, decoder, which is the mapping the lower dimension to represent back to the original output.

In between, we have Latin space or the code area where the data is, is the most compressed form.

Okay, it is you can say it is, it is reducing those features to the minimum requirement that we have to understand that data.

I want you to imagine one more scenario where you have a cat image.

Now by keeping some features of that cat image, we can memorize the entire cat completely say only by the border of that cat, only by the eyes, the nose, the mouth part of the cat.

And by saving these areas only we can save the cat image, we don't really have to save every single detail on that image.

Now on the right side, when you create the output, you the model can be a little creative and add and generate those missing areas by its own creating a new version of the image, or maybe an improved version of the image.

Moving forward, let's talk about a reconstruction part.

So it's reconstruction the image to be as close as possible to the original image indicating the network has learned a meaningful representation.

I want you to notice this left side and the right side, this is the same input and the same output, but in a different way.

These all blue dots represent the inputs or you can say the columns that you're giving as an input.

Imagine they are all the pixel matrix that you're giving.

Now the second layer is reducing so it this green layer force this model to learn more about the image or only the important features of the image and reduce the number of features you're reducing it to these many green dots.

Imagine you have to put all these dimension to this dimension now, then the last central which is a bottleneck of this entire architecture, this bottleneck will force it even more to learn even more specific details and learn more from the image than this.

So you can say by from blue to red, it's a huge gap, you're compressing it to a great extent and by doing this compression, you are learning only the important feature.

The model is learning only the important feature.

Then again, when it has to regenerate the output, it generated from the scratch by knowing the some feature and learning some features and the red dot it start generating it to the original image.

The original output can be a little different or maybe an improved version of the input part.

Okay, there are multiple things that can happen.

But by retraining it again and again, you can get a specific output too.

Let's now look into challenges in autoencoders. So challenges are also there, which are data generation part robustness in the learning part handling the variability part. So let's talk about those things. So autoencoder has been successfully in the encoder construction, and but it does have its limitations. So data generation, traditional encoder will have a limited in generating new unseen data. And this limitation was for a reason guys, as we remember, that in any model, we have something called creativity, to give a high creativity to a model that can lead to outputs, which doesn't even exist. Now, sometimes it can be a good thing, sometimes it's not a good thing.

Okay, imagine a dog with a wing, you don't want that kind of image, it doesn't look original, and you are only looking for original looking structures, right? So you want to make sure that dog does not have those wings, something like that. Robustness test is autoencoders often learn oversimplified representation. And if you have known to dysentery, this is also the same problem with dysentery, which oversimplify the data and leads to overfitting scenarios, right? So this oversimplification is always going to be a problem with n representation. So autoencoders are known to oversimplify the things missing the data, true complexity. And if you miss the true complexity of the data, you might end up miss some highly important or deep features of the image or the data itself. Now handing the variability, so autoencoders struggle with the inheritance part and they inherit the variability on randomness in the data and which is crucial for tasks like image generation and simulation. So if the inheritance part that means the parent image is trying to be copied or you are trying to model is trying to learn from the parent image, the randomness in the data can be a big challenge. I give you an example, if I have a image and I'm telling you that I have 10 images, and they belong to fish of small size only. This way, if I asked you to predict what kind of fish it could be, there's a good chance that you take at least few names correctly. At the same time, I tell you that I have 10 images, and they have images of anything on this globe on this world. Now, this is impossible for you to predict correct name, you might have some good hit, but there is a very low probability of that. So randomness in the data is a big challenge. And when you're predicting the outputs, or generating outputs like this randomness can be a very typical thing to handle. Coming to the simple part, so there's a note to that standard autoencoders are limited to the leader of development. So do read these things too. Basically, if you have a habit of reading the first paper, you will find the limitations of autoencoders is a big challenge that has been there for a long time. So we'll talk about that in the coming classes.

Let's now look into introduction to variational autoencoders.

Okay, so we are in the part where we are going to discuss the variation or the variational autoencoder the VAE.

The other autoencoder which are basically used for the same purpose I talked about to generate a new data from the old data.

So, left side is your input right side is your output and the central layer is the compressor the lattice representation of it aim of the central area is usually to understand majority of the data just by a few features and that is how it learns the most important features to remember the entire thing that is given as an input then can generate the output accordingly.

So, functions of this VAE is the design to learn an underlying probability distribution and not to compress it and to generate a new data sample that are like the training data making the powerful for generative like this imagine same thing the car image and giving the car image as an output.

So, at least the image that is giving into the output should be a car image only should not become a cat or dog and it provide a more robust and generalized way of representing the data capturing its inheritance variable and complexity of it.

Let us talk about the architecture this is very similar to what you have seen in the very first slide but little more description is there.

So, basically what you are having here is you have an input image that represent by x and at the right side the green tile that is your x bar which is basically your x hat sorry so that is basically your output.

So, the ZEC area this is where you can say it is your latent space which basically use the standard deviation and using the standard deviation part and the mean of the entire data it summarize or sum up all the mean and standardization by knowing the standard deviation by knowing the mean it knows the average and the variance in the data it saves all that into the matrix of latent space and that matrix is the main core input.

Now remember the x input was a very big input to begin with and coders reduce that to a smaller scale then this latent space reduce that by knowing only the mean and the variance of the data and saving that to the matrix then regenerating from that mean and variance on the decoder side and to reconstruct the entire image on the right side that is the entire process that we should understand.

The role of VAE so basically it goes like this the encoder then we have latent space decoders reconstruction lost because we expect that we are losing some information not all but at least some information is always gone then KL divergence terms we will talk about that also in the coming time and each component plays a district role and functionality here and the compression input data into the latent space representation that's your the first statement for the encoder and reduce the parameters that you are looking for that is mean and variance itself guys mean and variance are the main input for the latent space that you have.

The latent space it compress the representation of the data okay and again encoder and spatial feature which is data reconstruction part it save the entire matrix for you the matrix that we are talking about of mean and the variance part then we have the decoder deconstruct the data into latent space representation and show the quality of reconstruction optimal to vary performance.

So basically whatever output is bringing it should not be very far away from the mean and the average value so mean value and the variance value that we had.

Reconstruction of the loss measure the how well the decoder reconstruct the input part then the MSC part the mean square error part so which is the mean square error the total error you are getting error here will mean that whatever the input values were how far the values are from with the output side suppose the input was 20 and you are getting 21 the error will be 1.

The cross entropy loss so the randomness in the data so cross entropy is nothing then tropical part is basically where you have loss that you have in the data or randomness in the data and that randomness lead to misclassification of the decoder side.

Now the act as a guide of compass nudging the VAE towards better capturing the important features of the input then we have the last one the KL.

So basically in associate to a VAE overall loss function measure of divergence and latent space distribute for a period distribution in a simple language you can say that these are the points where we are trying to optimize the output overall and help you generalize and prevent the overfitting part now this is a very important area where we are trying to generalize the data output because we do not want a situation the model is working with the memory not learning from the data itself that will be a bad situation why because right now you are giving an input getting an output then if you change the pattern of data all of a sudden of the input data the output may vary drastically or maybe it gives the same output as old one you do not want that situation to happen.

In this lesson, we will be looking into VAE generative training process.

So VAE generative AI, how does it really work?

So what are the things that it can do?

So the first thing is data collection part that followed by the encoding.

is represented by floating numbers in a simple language which are carrying the cementing meaning at the same time.

So basically they are carrying the word and their relationship with other words.

This is a very important thing.

You can imagine a correlation between two similar inputs and two inputs which have no similarity at all.

These kinds of input understanding is very important when you are filtering out important features.

This is something you might have also heard in encoded layer in deep learning RNN LST models or you have heard about this in PCA where you are discussing the important feature filtering out of the other features.

So this is the same scenario sampling of sampling part decoding that followed by object function training and background propagation followed by generative capabilities and that again this is the last step.

Basically it's not the last step after this we have to proceed it also but this is your code in this last step.

Now the multiple step that we are having here is the basic you can say core power of generative AI capabilities and this is the very process that happens inside the VAE to process the entire things.

Now data collection part.

So how does it collect the data?

So it gathers a large data set for existing content that it has and shows the representation of the domain and type of data that we are aiming to get generated.

So that is how it really works.

So large data can be huge data where it doesn't matter.

But yes if you increase the amount of data the time to process that data will also be in correlation.

So that doesn't happen and in deep learning in autogen model training part it can take from few minutes to few hours to few days to depending on that amount of data you are willing to process to get an output.

The very pre-trained models that we have used and we have seen they have been trained for several hours several days to get to the point they are right now.

Now encoding parts.

Encoding the next time involves the encoding here.

Encoding usually is the neural network mapping the input and the latent space part.

Data comes as an input that get again the features are detected there then it goes to latent space where the mean and the variance is safe there.

So that is the latent space that we talk about.

So learning the meaning mean that is your mu that is represented here and x is your input here.

The variance is the main part standard deviation is by the sigma here.

You are squaring that to get those variance part and the entire thing with respect to Gaussian distribution is calculated.

Understand this Gaussian distribution is a very perfect case scenario in real data you don't get that Gaussian distribution kind of thing it is whenever a formula is created or a reference is created we take the best case scenario to get or it is the ideal case scenario to get a reference from but in reality the data will not show Gaussian distribution kind of thing it will be different and according to that the variance and the mean value will change they will vary a lot according to those inputs.

Sampling parts a model sample the distribution in the it learns in the latent space sampling enable the creation of the new data point from the distribution of power point and process introduction and so introduce the crucial element of this randomness again randomness is why important guys there is a reason for that randomness in that layer or at the latent space part randomness is very important if we want to prevent our model to memorizing the thing understand that it's a smart model it can memorize the entire input if it too simplifies the data the data will be simply memorized it will not try to learn from it and then generate the output in this case scenario same exit output will be generated every single time even later if you change the input image it may not change anything there they it will give you the same output. Now if you add a randomness that say by force or any way it forces the model to understand the logic behind the input and the relation between the input features so model understand it better it will not simply memorize it hence the randomness is a very crucial element in this that is the meaning of this part.

Now decoding which basically leads to your output so decoding is the another neural network generator new data sampling it maps the latent representation that is your matrix which contain the mean and variance back to the data spaces decoder learn from the mean and variance part so the sigma is that only and in it leads to the data space or data output at the end.

Let's now look into steps involved in VAE

Objective function is the training of VAE involved optimizing an objective function the object function comprise two component here minimizing the reconstruction error between the input and the generated output that you have and minimizing the athletes callback KL part which is callback liberal is a basically area where we are trying to understand the distribution of the data and how the distribution is really working all together in this case again if you take the ideal scenario it's a Gaussian distribution if you take a normal scenario okay in that case the distribution can vary to a great extent and again that leads to how much variance is there and how much you know the standard deviation steps are there how gap big gap is there in between those.

Training backpropagation so backpropagation is a very important step if you have learned the machine learning modeling you might have known this already that it goes to the back to update the weights here there right the same thing so backpropagation is the very thing that trains the model and update the model to understand the you know what to give in the output or how to give the correct output in that case and process the computed gradient in relation to the encoder and decoder and if you want to make the sentence simple that means basically when you are giving the input and giving the output at the decoder side how that is connected overall what is the point at which point it is learning properly from there okay from the input data that means it will lose some information but keep the majority of the important features information at that time the system updates the parameter to minimize the objective function there and so on it goes to the output part again again if it is required it will train from it and this process keep happening if you required but it's not like deep learning or LSTM layers that kind of thing guys it happens in a way that it goes completely from one cycle to calculate the output and if it is not okay then second image goes and then again follow this process.

The unique feature of VAE is to provide the value in the letter space enable the straightforward random sampling and interpolated between the data points and versatility allow the to generate a wide variety of data type of effectiveness to simplify this the output can vary it doesn't have to be same as the input and it can be a better output it can be a modified output it can be output which we are not expecting either imagine a cat given as an input a black cat but the output is a gray cat or brown cat with additional features in it role of VAE in generative AI guys so how does it really works so below are the two important points here as you may see so VAE are instrumental understanding and model complex data distribution on the left side that's your encoder part and the right side is VAE significantly contribute to the development of generative model in AI artificial intelligence and at the center of this image you will find something called compressor compressor is that latent space only that allow to reduce the input that is coming learn only important feature and then proceed to the output side.

The KL Divergence Term

The loss function in a VAE is composed of two main parts:

1. **The Reconstruction Loss:** This term measures how well the decoder can reconstruct the original input from the latent code. It ensures that the autoencoder is still learning a faithful representation of the data.
2. **The KL Divergence Loss:** This is the "variational" part of the loss function. Its purpose is to act as a **regularizer**. It measures the difference between the probability distribution learned by the encoder and a simple, fixed prior distribution, which is usually a standard normal (Gaussian) distribution.

The Reparameterization Trick

The reparameterization trick is a clever solution to a fundamental problem in training Variational Autoencoders (VAEs).

The core of the issue is that a VAE's encoder outputs a probability distribution, not a single point. To generate a new data point, the model must **sample** from this distribution. However, this random sampling process is not **differentiable**, which means you can't use standard backpropagation to train the model. The loss signal cannot flow back through a random operation to tell the model how to adjust its weights.

3. The reparameterization trick solves this by separating the randomness from the learned parameters. Instead of sampling z directly from the encoder's output distribution, the trick re-expresses the sampling as:
4. $z=\mu+\epsilon\cdot\sigma$
5. Here, μ and σ (the mean and standard deviation) are the learned parameters of the distribution, and ϵ is a random variable sampled from a fixed, standard normal distribution. This is the **key**. The randomness is now external, and the model's parameters are part of a differentiable, deterministic equation.
6. This makes it possible to backpropagate the error signal from the reconstruction loss back through the network and train the VAE using standard optimization techniques. In short, the reparameterization trick is what allows the VAE to be trained at all.

Let's now look into image generation. So let's discuss about the image generation parts. Image generation as we know with the name that we are trying to generate an image. One issue with image generation is that model may not generate a realistic image for us that can happen in autoencoder. But aim always going to be that we want to generate as realistic as possible for the image parts. For this we feed the data and according to the data model learns from it and generate something similar. But is at the same time it may generate something very unrealistic. Imagine a particular animal which is in only green colour but model generated in a black or blue colour. So that is one example a very small example on a bigger scale as I have talked before also that imagine a car which is flying a dog which has wings something like that. So unrealistic image generation is a big problem. We will talk about that in the coming time also. In generative AI hallucination is a big challenge and this is something similar to that. But right now let's talk about generative AI images part. So they generate unique visual art for us. They create in game asset character environment and VAE assist in generating medical images also which is definitely used for research and diagnosis part. Anomaly detection is a part where we are trying to identify anything which is out of the normal which we call outliners. They assist in spotting unusual financial transaction like fraud detection kind of thing. Enhance the security system as it is detecting the illegal activity and it does not have to be a financial activity guys it can be any sort of activity and they improve the function the overall manufacturing processing of identifying the defect in it. So this can be used in multiple direction not just one. So security finance wrong transaction anything which is about to occur natural disasters also anomaly or anything which are rare to happen and we want to detect it before it happens. Drug discovery so anything which is you know again we are trying to bring a potential output say a vaccination kind of thing. They speed up the identifying the potential drug and not only drug guys sometimes it also help us to identify a potential virus that is creating a issue. Say a particular disease is spreading and we are not sure which virus is reasonable for this or which bacteria is responsible for this. So anomaly detection can allow us to detect a particular virus which is being highlighted in each case. So they help in design molecular and with specific properties for various applications too. Data imputation so suppose you have a data which contain a lot of missing values which is very hard to handle by imputing the correct value by analyzing the data and fixing it with the most close to correct values there this can be a very helpful thing. So we will fill the missing and incomplete data set for you and we can complete the patient record in health care aiding medical profession VAE impute missing financial data for analysis VAE are invaluable where missing data complicate the decision making process. Imagine you are reading a transaction data and you don't have the detail of last four years or three years data. It can at least predict something for you that you can at least start with the research part. Drawback of VAE is there too so greatest disadvantage is that it produce blurry and unrealistic output. I talked about this in the very first slide too that it may lead sometimes to a output which is very out of the blue it's unrealistic it's blurry it's not useful at all and creativity is only okay when you are say creating an artistic image having a horse having a wing it's completely okay but when you're looking for a realistic image say a human skeleton you don't want six arms popping up there. The GAN are known for producing high quality sharp realistic outputs particularly in image generation part.

Demo: Implementing a VAE with TensorFlow for Image Generation Using the MNIST Dataset

let's now look into demo implementing a VAE with tensorflow for image generation using the MNIST data set. Welcome to demo one of generative AI and this we will discuss how we can implement the VAE. Understand few things before we move forward we have uploaded this file on Colab now we all know how to use Colab we can just upload the file by clicking on the file going on the upload part and just upload the file there.

I am doing this to make sure the execution is okay and if your hardware does not support the kind of execution that we are going to do which will take a proper RAM and GPU and all those things we require a proper support for that. So I am using this not only that from his arrow I can change the runtime type also look at GPU T4 and working on this.

So change these settings making it GPU will increase the speed of execution but this project will take some time to execute and all. I have already have a file which have the output for you so in case it takes too much time or get stuck when you are running it I will show you the output from this side. Okay so moving forward we are closing the resource part please do change the resource before you execute this file. Okay this is something that will support the speed of execution I will start with importing the important file. We are importing here numpy we all know numpy is for the array part the tensorflow use tensors at the backend which are basically arrays so having an NP imported is a good thing here. Mapplotlib is to plot any chart if you required or maybe to show you any output and tf is the tensorflow that we are using or if we want to use keras we can use directly keras from tf.keras from here only. As you can see the stick indicates that we have executed successfully and this is imported now. Now we are reading the data set from keras so tf.keras data set mnist and mnist data set is this we are loading the data creating the variable so this is my training data this is my y part this is my test data this is my y part. Now understand as we do not require the y in this case as we know that the autoencoder do not require the target column so we are ignoring the label completely this variable if you want to check name it properly y training y test and you can see the output of that label they will contain some label of the fashion data.

Following this we are having this training and test data generalized that means you are dividing the highest value of the pixel understand this in a image the image goes from 0 to 250 sorry 1 to 256 right so but when you start from 0 that goes to 0 to 255. 255 is the maximum value in the matrix that is why I am dividing the data with the maximum value here so this is a generalized scaling down method this bring the return to the range of 0 to 1. Once you done this part we will set the hyper parameter where the learning rate number of steps and batch size has been declared and here we are where learning rate is 0.001 which is usually always small number a number of steps to go with that is 100 here the number of step is the what to train the model how many times you want to train the model continuously a batch size is the how many numbers of sample you want to give as an input. Now next step to create the architecture of VAE we will define the architecture here where we start with the latent dimension that is 2. Now we are following with the encoder input which is there is a different input and shape is 28 into 28. Now before this we can do one thing just to get an idea x train is a variable name I will just create a variable right above and you can see this is the shape so we have 60000 rows 28 columns and 28 columns this is basically the pixel size so this input shape is taken from here this is what decide the input shape for you. Once you got the input shape you will go for the next part which is flatten layer now flatten layer is something that convert this 3d to 2d we require 2d shape to get an input for dense layer output of CNN layer is 4d. 4d means it is basically a 4d thing but the first value is none which is sample size the batch size so batch size is by default none so we get the rows and we get the the pixel part matrix part so this decide the input shape okay this decide the number of rows we are giving here in both cases flatten will make it 2d 2d also have the first value the batch size as 0 and the next value is the value itself okay that followed by the encoder input that you have given okay this multiplied with this and we have functional model x here now tf x layer dense we are giving 512 neurons here okay so 512 neurons are given here and you can give any number this is a by default taken so there is no logic behind this we can take a low number bigger number both are okay this is something we guess and we put the value activation is relu because it is an in between layer we will use the activation as relu only multiplying with the above layer because the functional model that goes again back to the x that is followed the next part where the dense layer is there we are giving the latin dimension which is 2 basically we are saying we are giving 2 neurons z mean sorry that followed by the next layer which is dense again latin shape and again we are giving for the x part so that goes to log variable variable here this is your last layer of this particular encoder series then defining the decoder part on this we will continue from this only so that means we say shape is latin shape and goes to the first input latin input then goes to the next layer multiplied with the latin input here then this is now going to increase we are going for the next last output i have taken 784 okay this is again a taken value when you multiply these two you will get 784 followed by sigmoid and you get this xx one more thing within this let us put one more thing that will be okay this is 784 784 is what you get when you multiply 28 and 28 okay that is what you are putting here okay hence the clarity should be there how we are getting to this number once you got the sigmoid that means you get only 0 and 1 as an output now which is the output will be 0 or 1 accepted rejected part then you get the reshaping of this reshaping the final output to 28 to 28 giving the x as an above as an input that goes to a decoder output this is your architecture complete architecture of creating the own encoded layer okay vse and once you have this you will go to the next part defining the shape of the function that function that will analyze the final value for you so sampling and in the sampling you are taking the layer itself okay keras dot layer and layer call the function the class is this layer part and then you're calling the function inside itself and the input and giving the values this is input values which is taking to the mean and the var log variance that is the shape is z mean taking the first value of the shape okay number of entries that is that goes the batch size okay the entire rows are giving as a batch size here followed by the shape of the mean the second value epsilon you are giving here and then you define the formula for this b3 values this is what is going to be written as a final output now you're connecting the encoder and decoder on this let me run these two back to back so we have the output two it may take few seconds on this okay follow by this all right next one is connecting the encoder and decoder so I am just running it to save some time to execute this part and here we go so encoder has this part okay this is input in the mean and the variance parts and the sampling area then you take the model input giving the first encoded input that you have given the output shapes is the mean variance and the output encoder that follow in the decoder part with writing inputs is going to be there which is true in this case and decoder output that followed by the next layer that followed by the last output layer once you have this shape defined you go to the next part defining the loss function compiling the model you want to compile it so vi function you're creating you're taking the model as a subclass here that is you are inheriting this class here then you're creating the self encoder decoder and the keyword argument that you're going to pass that means any variable based value that you're passing will be saved here this will be saved into a dictionary format once you do that you get the user constructor to initialize this by default the moment it's run it is going to execute this immediately then both the values will be initialized compute by the loss and you get the encoder here you take the values in the three variables then reconstruct by using the self decoder the reduced mean and the binary cross entropy giving the value

once you have this you will go to the entire formula thing here as you can see
execute this you get the final return value that goes to the next part then in the training steps
you are giving the data gradient tape which is used to calculate the gradient part the gradient
that you're that is changing here one by one is changed compute the loss of it self training
variables followed by the zipped combining these two goes into self optimization and coming to the
VAE output in case this flow is not very clear because this is pure documentation thing guys
no worries about that understand this we are basically creating a manual VAE model from our
side now tf do have its own inbuilt model that we can use directly but this is we're creating
from scratch from our side execute this then train the model then the training part you will say fit
now when you are using this manual ways of thing you call the fit directly here
you fit x train xy training train data will be there the input is the output actually so that
is what you're creating and epoch is number of steps you have given which is approximately 100
if I am recalling it correctly batch size is the number of rows you have given we execute this
it will take some time to run I think number of steps are not defined let us check on that
we have number of steps right here
okay I am good this should work now going to take some time on this
okay meanwhile it's training let's move on and discuss further steps too
it will take some time it is 100 epochs and we are hardly at 5 right now
so we are generating a manifold of digits okay so we're generating a manifold of digits by creating
a latent space grid that is the matrix that contain the variance and the mean value and
to value in the decoder to produce the images to provide the final output to it
so creating the function we are creating the random normal values giving the image in
the latent value decoder here calling the image numpy then using the ceiling method to make it
the smallest number that is possible in this fractional value that is going to give there
so an image fractional value when divided by 4 will take the lower value and that goes in the
end rows after that we are running a for loop which is range with the number of images that
you have okay subplotting that and that give you the final output this will require to wait for the
above training because after that only I can go for the final part here let's wait for it
and that will be the conclusion of this model also so let's read the conclusion
they demonstrate the successful implementation of the training of VAE model to generate the image

the managed data and the process so encompasses the several critical steps then importing the necessary libraries and model and finally generating the manifold of digits this is will be an output of image if you can look at this it is subplotting of row wise so the number of rows the 4 and the position of that particular image so first value is the number of rows second is the number of columns so 4 columns are going to be there that followed by the position of this so it is starting going to starting from 0 we required for position number 1 so the position can never be 0 that followed by the I am sure to show the image here we're giving the shape and all then axis is off and it shows the plot.show at the end so let's see if it is done now it's going to take some time so let's wait for it what I'm going to do guys it will it is going to take us a decent amount of time I can see that so I'm going to you know pause it let's wait for it to complete and then we come back actually it's almost 48 it's almost done

so guys I'm pausing it let's wait and let's come

all right so a box are completed here which took like 5 minute to complete

so depending on your speed guys if your speed is slower than this will take even more time

if you are struggling with the speed issue and you're not using colab I request you to reduce the number of epoch output clarity will not be there but at least you will have a decent you know flow of the code so you will not stuck in between going further so we'll agree with the final thing that we already discussed so we'll we'll

expect to see and plot here, the C, here is what it is generating for us, digits are there and okay it was a human written digits there and it has generated this image for you, this is your final output.

It is little blurry, what you can do to improve this, increase the number of epoch to say 500, 700 and it will definitely give you a even better result.

So definitely the number of epoch step will affect the output here, that is a requirement that we can you know we can simply fulfill by increasing the epoch and the quality of the model is also going to matter, so you can do that too.

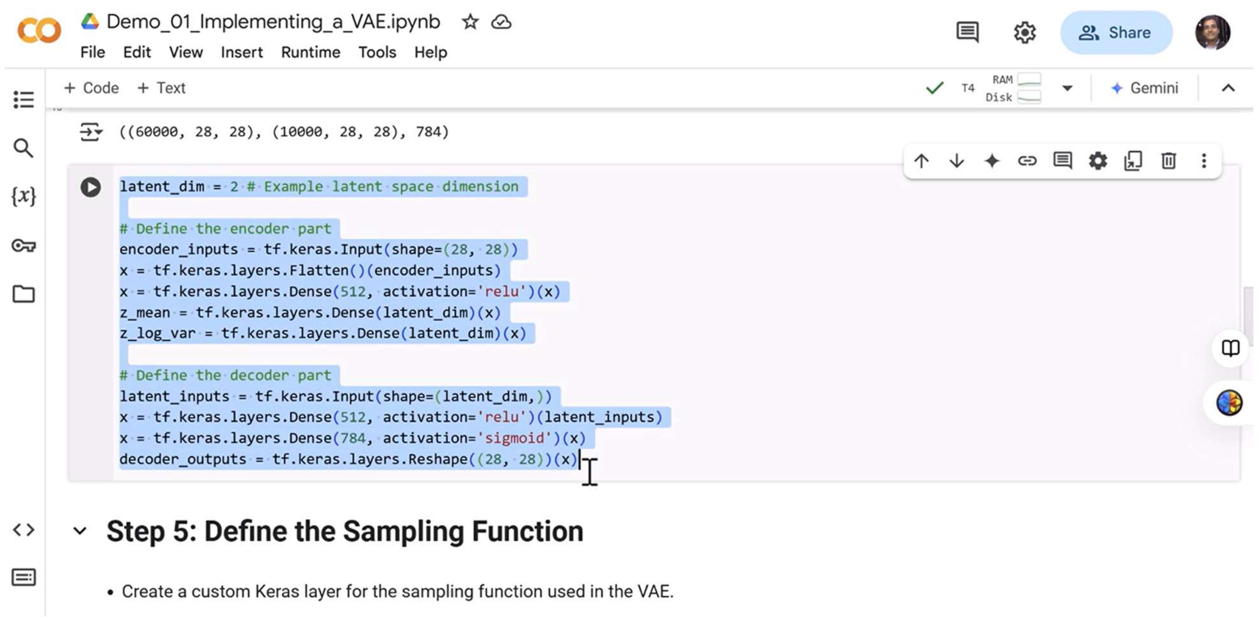
Apart from this you can use normal LSTM layers too, you do not really have to create a function like this, this was a way to teach you it from the very core but you can use normal models too or normal layers too that we already have, usually we use those layers only to generate or create a proper model for this.

So that is all for this video, see you in the next one guys, thank you.

Check this on Gemini:

```
loss = some_loss(predicted_outputs, true_outputs)
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
train_op = optimizer.minimize(loss)
```

Complete Architecture of Getting Your Own Encoded Layer



Demo_01_Implementing_a_VAE.ipynb
File Edit View Insert Runtime Tools Help
Share
+ Code + Text
T4 RAM Disk
Gemini

```
((60000, 28, 28), (10000, 28, 28), 784)
```

```python
latent_dim = 2 # Example latent space dimension

# Define the encoder part
encoder_inputs = tf.keras.Input(shape=(28, 28))
x = tf.keras.layers.Flatten()(encoder_inputs)
x = tf.keras.layers.Dense(512, activation='relu')(x)
z_mean = tf.keras.layers.Dense(latent_dim)(x)
z_log_var = tf.keras.layers.Dense(latent_dim)(x)

# Define the decoder part
latent_inputs = tf.keras.Input(shape=(latent_dim,))
x = tf.keras.layers.Dense(512, activation='relu')(latent_inputs)
x = tf.keras.layers.Dense(784, activation='sigmoid')(x)
decoder_outputs = tf.keras.layers.Reshape((28, 28))(x)
```

Step 5: Define the Sampling Function

- Create a custom Keras layer for the sampling function used in the VAE.

An epoch is one complete pass through the entire training set during the training of an AI model.

Think of it like this:

Imagine you're a student preparing for a test, and your training set is a large textbook.

- A single batch is like reading one chapter or a specific set of pages.
- A single epoch is like reading the entire textbook from cover to cover.

In TensorFlow, the training process is iterative. The model goes through the following steps repeatedly:

1. Divide the data: The complete dataset is split into smaller, manageable batches.
2. Train on a batch: The model is trained on the first batch, its parameters are updated, and its performance is evaluated.
3. Repeat for all batches: This process is repeated for every batch in the dataset until the model has seen all the data.

4. Complete an epoch: Once the model has processed every single batch, one epoch is complete.

The model usually requires many epochs to fully learn the patterns in the data and minimize its errors. After each epoch, the model's performance on the data is measured, and this helps the developer decide whether to continue training for more epochs or to stop.

Why is the ceiling method applied to fractional image values in the output?

To round up values and avoid partial pixels (Correct)

To visualize the intermediate layers

To randomize pixel intensity

To reduce loss during decoding

Correct

The ceiling method ensures that any fractional pixel values are converted into valid whole number pixel intensities.

Which TensorFlow component is used to define the encoder and decoder models in a VAE?

tf.train.Checkpoint

tf.keras.Model (Correct)

tf.data.Dataset

tf.image.resize

The correct answer is **tf.keras.Model**.

Building an Autoencoder in TensorFlow

In TensorFlow's Keras API, a tf.keras.Model is the central component for defining a neural network.[1] It provides a flexible and powerful way to build, train, and save models. For an autoencoder or a VAE, you would use it to define both the **encoder** and the **decoder** parts of the network.[2]

Here's a breakdown of how it works:

1. **Define the Encoder:** You would first create a tf.keras.Model to represent the encoder. This would consist of a series of layers (like Dense or Conv2D) that progressively reduce the dimensionality of the input data.
2. **Define the Decoder:** You would then create a separate tf.keras.Model for the decoder.[3] This model would take the compressed representation from the latent space and use a series of layers to reconstruct the output.
3. **Combine for Training:** For a VAE, you would then combine these two models within a single, custom tf.keras.Model class.[4] This allows you to define a single training loop where the input goes through the encoder and then the decoder, with the loss function measuring the difference between the final output and the original input.

While other components like tf.data.Dataset are used for data handling and tf.image.resize for preprocessing, the tf.keras.Model is the primary tool for building the network architecture itself.

The Role of the Bottleneck

The bottleneck layer in an autoencoder is a crucial design element.[2] Its purpose is to act as a constraint, forcing the model to learn a compressed and meaningful representation of the data.[3]

If the latent space is too large, the bottleneck is essentially removed. The model is no longer forced to compress the information. Instead, it can learn a simple identity function where it passes the input data directly through the network and recreates it exactly.[4] This is a form of overfitting where the model just memorizes the data rather than extracting its core features. It will fail to learn a useful representation and will not generalize well to new, unseen data.

The goal is to find the "sweet spot" where the bottleneck is small enough to force compression but large enough to capture all the essential information needed for high-quality reconstruction.

Learning Robust Features

A key challenge with basic autoencoders is that they can learn to simply "memorize" the input data rather than extracting meaningful, generalizable features.[2] This is where **denoising autoencoders** come in.

A denoising autoencoder is a variant of the standard autoencoder that is designed to learn more robust representations.[3] It works by:

1. **Corrupting the Input:** During training, a clean input image is intentionally corrupted by adding **noise** to it (e.g., random black pixels or Gaussian noise).[4]
2. **Reconstructing the Original:** The model's task is to take this noisy input and reconstruct the original, clean output.[5]

This forces the model to learn the fundamental features of the data rather than just memorizing a copy of it. To remove the noise and produce a clean output, the model must understand the underlying patterns and structure of the data.[6] This makes the learned representation much more robust and less susceptible to the kind of "memorization" that can plague a basic autoencoder.[7]

Question 13

In a VAE, KL divergence is used to:

Increase the dimensionality of the latent space

Maximize classification accuracy

Reconstruct the original input

Align the learned latent distribution with a prior

The correct answer is: **Align the learned latent distribution with a prior**.[1]

The Role of KL Divergence

The **Kullback-Leibler (KL) divergence** term is a critical part of the VAE's loss function.[2] Its primary role is to act as a **regularizer** for the latent space.

- **What it does:** The KL divergence measures the difference between the probability distribution learned by the encoder and a predefined **prior distribution**, which is typically a standard normal distribution.[3]
- **Why it's essential:** By minimizing the KL divergence, the VAE is forced to create a latent space that is well-structured and continuous.[4] This prevents the model from mapping inputs to isolated points and instead encourages the formation of smooth, meaningful clusters.

This is what gives the VAE its generative power. A continuous latent space allows you to sample a random point from the prior distribution and have the decoder produce a new, plausible output. Without the KL divergence, the latent space would be disorganized, and generation would be impossible.

What happens if the KL divergence term in a VAE is weighted too heavily?

The reconstruction becomes more accurate

The latent space becomes overly constrained, harming reconstruction

The encoder becomes non-differentiable

The decoder becomes stochastic

The correct answer is **The latent space becomes overly constrained, harming reconstruction**.

The KL Divergence and the "Prior"

The loss function in a Variational Autoencoder (VAE) is a balance between two competing goals:

1. **Reconstruction:** The model wants to accurately reproduce the input.
2. **Regularization:** The model wants to force its latent space to conform to a simple prior distribution, like a normal distribution, using the KL divergence term.

If the **KL divergence term is weighted too heavily**, the model will prioritize satisfying this regularization constraint above all else. It will force the latent space to become very close to the standard normal distribution, making the latent vectors tightly clustered around the mean. While this might be great for generation, the model will essentially ignore the reconstruction loss. This leads to a situation where the model is no longer learning how to effectively represent the original data, and the reconstructed outputs will be blurry, inaccurate, or low-quality.

The model essentially gives up on faithfully reproducing the input data in order to win the "regularization" game, which is the exact opposite of what you want in a well-trained VAE.

How does a VAE ensure sampling produces meaningful outputs?

By removing randomness from the encoding process

By regularizing the latent space to match a known prior

By using a non-probabilistic latent space

By avoiding reconstruction losses

The correct answer is **By regularizing the latent space to match a known prior.**[1]

The Role of Regularization

In a VAE, the model's ability to sample and produce meaningful outputs comes directly from its unique loss function. Unlike a standard autoencoder, a VAE's loss function includes a special term called the **Kullback-Leibler (KL) divergence.**[2]

This term's purpose is to act as a powerful **regularizer.**[3] It forces the probability distribution learned by the encoder to be very similar to a simple, well-behaved prior distribution, typically a standard normal distribution.[4]

By doing this, the VAE ensures that its latent space is:

- **Continuous:** There are no empty "voids" between learned data points.[5]
- **Smooth:** Moving from one point to a nearby point in the latent space results in a subtle, meaningful change in the generated output.[6]

This well-structured latent space is the key.[7] Once the model is trained, you can simply sample a random point from the prior (the normal distribution) and feed it to the decoder. The decoder, having learned this smooth space, will then produce a realistic and plausible new data sample.[8]