

Lecture 20.1

Revisiting Autoencoders

Okay. Hello everyone and today we'll be talking about, Variational Autoencoders.

Refer slide time :(0:24)

Acknowledgments

- Tutorial on Variational Autoencoders by Carl Doersch¹
- Blog on Variational Autoencoders by Jaan Altosaar²

¹Tutorial

²Blog

So, first let me start with a, Okay first the acknowledgment. So, a lot of material that I'm going to talk about today, is based on these two tutorials and blogs a, lot of ideas from here. And the disclaimer which I want to begin with is that, so, this is a slightly tricky topic, because it's like a marriage between, neural networks and graphical models and there are multiple perspectives from, which you can approach variation autoencoders or try to understand them. So, you'll have to be a bit patient and the entire picture would become clear only by, the last slide or last but few slides. Right? So, just be patient, I'm sure, by the end you'll understand everything, but we'll go through a lot of intermediate steps, where I'll just leave the story, at an unfinished point and then come back to it later and so. Alright?

Refer slide time :(01:10)

$X \in \mathbb{P}^n \quad X = \{x_1, x_2, \dots, x_n\}$
 $\{X = x_i\}_{i=1}^N$
Maximize $\sum_{i=1}^N \log P(X = x_i)$ over w, b, c
 $F(w, b, c, x_i)$
 μ, Σ
Maximize $\sum_{i=1}^N \log P(X = x_i)$ over θ with $\Sigma = I$
 $\mu = \theta_1 x_1 + \theta_2 x_2 + \dots$
 $= \boxed{f(\theta)(x)}$

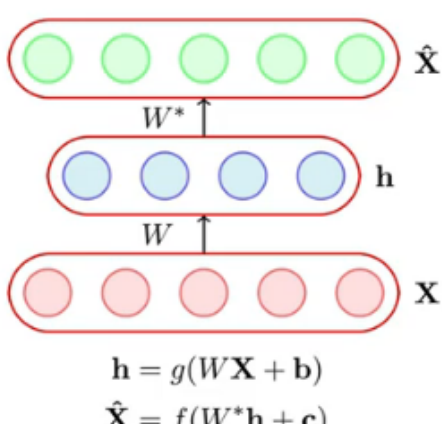
So, before we get into variational autoencoders, just are minder that over the past few weeks. We have been looking at generative models, where the idea is that, you're given a random variable X , where X

typically belongs to RN. So, X is equal to X_1, X_2, \dots, X_n . Right? And we are given a lot of training data, containing instances of X . Right? So, you have some n training samples given to you and what we are interested in is learning the Joint Distribution of X . Right? And what that means is that, for all assignments to, X , we are interested in learning this probability. And the way we go about it, is this now the standard recipe for a generative model and I will try to relate it to what we had done, for RBMs and we'll do a similar recipe in the case of variation Autoencoders, also. Right? So, what we do is, that for all the training data that is given to you, all the n training samples that, we had we want to maximize, this quantity. And this makes sense, because I have told you that this is what the data looks like here, are some instances for this data. So, if I am assigning probabilities, to these instances better assign high probabilities, to them. Right? I better maximize the probability, to the instances that I've already seen in the training data that makes sense. Now the question is, what is this optimization problem with respect to or what are the parameters of this objective function or this optimization problem. Right? So, that's the question that we need to look at. So, we looked at the similar formulae formulation in the case of RBMS, where what were the parameters of this objective function? This gives me even more incentive to release the RPM assignment, are you guys know edge state. Right? What would we do to p what did we express it as? I remember the energy function. Right? So, we're expected, express it as an energy function and what were the parameters of this energy function? W, B and C and of course the function also takes in X . Right? Because you want to compute the probability, of X taking on the value, of X . So, you better take X is the input. Right? So, these we had what we had done is there is known as parameterizing the Joint Distribution, using the certain parameters and the parameters were W, B, C and then our optimization, was with respect to these parameters. Right? Is it clear? So, that's the standard recipe that is often used that you want you're interested in a Joint Distribution, you want to learn the parameters of the Joint Distribution. So, you come up with a parametric form for it and this is the parametric form, the energy function, which had these parameters W, B, C that we use for RBMS. And now the optimization problem is maximizing, the log likelihood of the data, this is the data. And parameters of the objective function our, W, B, C is it fine. Okay? Now let's look at a more simple case. Right? That's suppose, again I'll take the same case that I want to maximize, the log likelihood of the data that is given to me and now what I'm going to assume is that this data, actually comes from a Gaussian distribution, multivariate Gaussian distribution and I'm also going to assume that the covariance matrix is identity that means I don't really need to learn anything in the covariance matrix. Now for a Gaussian distribution, what are the parameters? μ and Σ , of which I have taken Σ , off the plate. Right? I have said that Σ is I've just assumed that it's I , now can you tell me, what will this object? What this what are the parameters of this objective function? What do I need to find, such that the probabilities are maximized for the given data, μ . Right? Not Σ , because I have just simplified it by taking Σ off the plate. Right? Now that's one way of approaching this problem, the another way of approaching this problem which is something similar to what we did in the case of RBM says. I say that don't take μ as a parameter, assume that μ itself is some function, of your input. Okay? So, it is say $\theta_1 X_1$, plus $\theta_2 X_2$, plus. So, on in general I can just write it as its some function of your input, with the parameters θ . Right?

So, express μ as a parametric, form itself, but we introduced some parameters and we also have the random variables M . So, that because we are interested in computing a particular assignment. So, you better have the random variables, in the equation, does that make sense. Okay? Now if I do this, what should I optimize with respect to, θ . Right? So, now my optimization problem becomes that instead of μ , I have expressed μ itself as a function of θ . And now I want to find these details. Okay? And in

particular this function can be as complex as you want it can have as many parameters that you want. So, do you know of any such complex functions, neural networks. Right? So, this is one recipe that we are going to look at today. That you are interested in learning a Joint Distribution, you assume certain parametric form for that distribution. Right? Or rather you assume certain family for that distribution that it comes from a normal distribution, it comes from a Bernoulli distribution, it's not any arbitrary distribution, but you see it belongs to a certain family. And we are mainly going to focus on the normal distribution family. And as I said the normal distribution, has parameters mu and Sigma. So, the optimization problem should be with respect to MU and Sigma. But you are not going to do that we are going to express mu and Sigma as a complex function of some other parameters and then try to learn those parameters. Right? So, that's a very, very high-level story or recipe behind what we are going to do today. Okay? Is everyone clear with this, you get this indirect way of learning parameters of probability distribution. Okay? The distribution has parameters, the parameters are expressed as a function of some other parameters. And then you learn those parameters. Okay? And this is very commonly done, for learning generative models are learning the parameters of a Joint Distribution. Okay? So, with that background, let's so, give keep this overarching story in mind whenever we are talking about, anything on the slides that I have. Right? So, this is the background story and we'll be trying to fill, in points in this, story. Right? Gaps in the story. Okay?

Refer slide time :(07:38)




$\mathbf{h} = g(W\mathbf{X} + \mathbf{b})$
 $\hat{\mathbf{X}} = f(W^*\mathbf{h} + \mathbf{c})$

- Before we start talking about VAEs, let us quickly revisit autoencoders
- An autoencoder contains an encoder which takes the input \mathbf{X} and maps it to a hidden representation
- The decoder then takes this hidden representation and tries to reconstruct the input from it as $\hat{\mathbf{X}}$
- The training happens using the following objective function

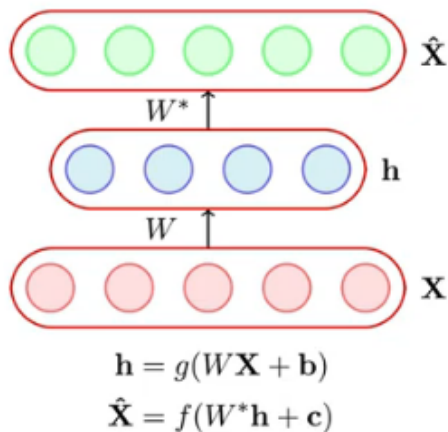
$$\min_{W, W^*, \mathbf{c}, \mathbf{b}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$

- where m is the number of training instances, $\{x_i\}_{i=1}^m$ and each $x_i \in \mathbb{R}^n$ (x_{ij} is thus the j -th dimension of the i -th training instance)



So, before we start with variational autoencoders, let's start by revisiting autoencoders. Right? So, what does an autoencoder do? It takes an input X , it maps it to a hidden representation and then tries to reconstruct, the input from the hidden representation. Right? So, you have an encoder, which does this it takes the X and maps it to a hidden representation and then you have a decoder, which tries to reconstruct the input from this hidden representation. And this is a simple feed-forward neural network. What's the training algorithm back propagation? What's the objective function squared error loss. Right? So, this is what we use ? Is the squared error loss. So, you have M training points and the data is n dimensional, for all the trained training points for every dimension, you want to minimize the difference, between the predicted value and the true value. Have done we have done this tons of time the coast. Right? M is the number of training examples and n is the dimension of the data that was given. Okay? This is what we have done many, many times, in different applications and in particularly in the context of or autoencoders.

Refer slide time :(08:46)



- But where's the fun in this ?
- We are taking an input and simply reconstructing it
- Of course, the fun lies in the fact that we are getting a good *abstraction* of the input
- But RBMs were able to do something more besides abstraction (they were able to do *generation*)
- Let us revisit *generation* in the context of autoencoders

Now what's the fun in this? I gave you an X , you gave me an \hat{X} , what's so, great about it I gave you some input, is just so, you can circuit the input and give it back to me. What's so great about it? What did we do in the process? And I want you to use some terminology that we are introduced in the case of RBMs, there were two things that we were doing for generative models, one starts with a, the other starts with G, abstraction and generation. Okay? Okay? So, now tell me what's the fun in this? What have we done in the process? I have done abstraction. Right? So, we have been able to compute, an abstract representation of the input, this representation is first of all compact, more meaningful, it ensures that the otherwise obvious differences between the inputs, no longer exist in this latent space. So, we had seen a lot of pics, of sandy beaches with a blue sky in the background, they all looked different in the pixel space, but once you compute that hidden representation, all of them would come close to each other. Right? So, the fun lies in the fact that we are learning an abstraction for the input. Okay? But RBMS were do able to do something more than abstraction, what is it that they were doing to do generation. Right? So, RBMS are also able to do generation, can you do generation, with an autoencoder, what does that even mean? What would Generation mean? What is the input and what is the output? What's the input

everyone? Hidden representation. And what's the output ? NX. Right? Whether you call it X Delta or X it's an X, weight it's the image, if your, if your network has been trained on images given some hidden representation.

Refer slide time :(10:26)

• Can we do generation with autoencoders ?

• In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a \hat{X} from it ?

• In principle, yes! But in practice there is a problem with this approach

• h is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input

$\hat{X} = f(W^*h + c)$

I want to be able to reconstruct, the image or some X tilde. Right? So, can you do this generation with autoencoders, if I want to do generation, which part of the network will go away it's useless for me, after training of course, the encoded pathway when you say, X I assume you mean the encoded battery, I don't need that, what is the input that I'll feed? H and then the decoder will take over and it will give me X tilde. Right? So, this is what the question which I was asking it's, when I say can you do generation, with an autoencoder, this is the situation that I am talking about that you remove the encoder. Now just feed an edge, to the decoder and see what it generates. So, can you do this yes. Right? I mean what's the problem with this? I just feed some edge belongs to Rd. Right? I'll just feed something yeah. So, in principle yes, I mean the mathematical operations, you can do you can take an edge and try to reconstruct, but what would happen if you did this you might get, meaningless X's, because you fed, it no we did not that's the whole point, you did not learn a probability distribution. We're asked a very similar question earlier in RBMs and we'll get that so, but I just want you to think about it. Right?

So, all of you see there's some problem with this, but probably it's still not being able to make the connection, to what the question that we asked for our games. Right? So, let's get there so, the problem is that H is actually a very high dimensional vector. Right? So, it's think of this, I mean I can't really draw high dimensional spaces. So, I'll just thought this is the high dimensional space. Okay? And now this space actually has many, many, many, many points. Right? But actually the edges, lie on a very small subspace, in this large space. Why is it so, where did the edges come from XS? So, X itself lies in a very

high dimensional space, of this high dimensional space ,the actual image is lie on a, very small, small subspace. Right? So, the images are actually on a very small subspace. So, it's quite natural that when you are trying to map them, to hidden representations they, will also belong to a very small subspace, in this large space. Right? So, now can you tell me a problem, in feeding H to the decoder and expecting it to generate an image from there, if I feed in this H, what will happen? It does not come from that space, which it has actually learned. Right? It's some, some arbitrary things it will generate an arbitrary, output do you get that. Right?

Refer slide time :(13:21)

$\hat{X} = f(W \cdot h + c)$

- Can we do generation with autoencoders ?
- In other words, once the autoencoder is trained can I remove the encoder, feed a hidden representation h to the decoder and decode a \hat{X} from it ?
- In principle, yes! But in practice there is a problem with this approach
- h is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input
- So of all the possible value of h which values should I feed to the decoder (we had asked a similar question before: slide 67, bullet 5 of lecture 19)

So, the question which I am trying to ask is, of all the possible values of H, H is a vector in R^d . Okay? It's a D dimensional vector of all the possible values of H, which ones can I feed to the decoder, what's the answer to that and give me a probabilistic answer or rather an answer in terms of probably, I want to feed the decoder those edges, which are highly likely given the exes that I had seen during race, then the moment.

Refer slide time :(13:51)

$\hat{X} = f(W^*h + c)$
 $g(WX + b)$

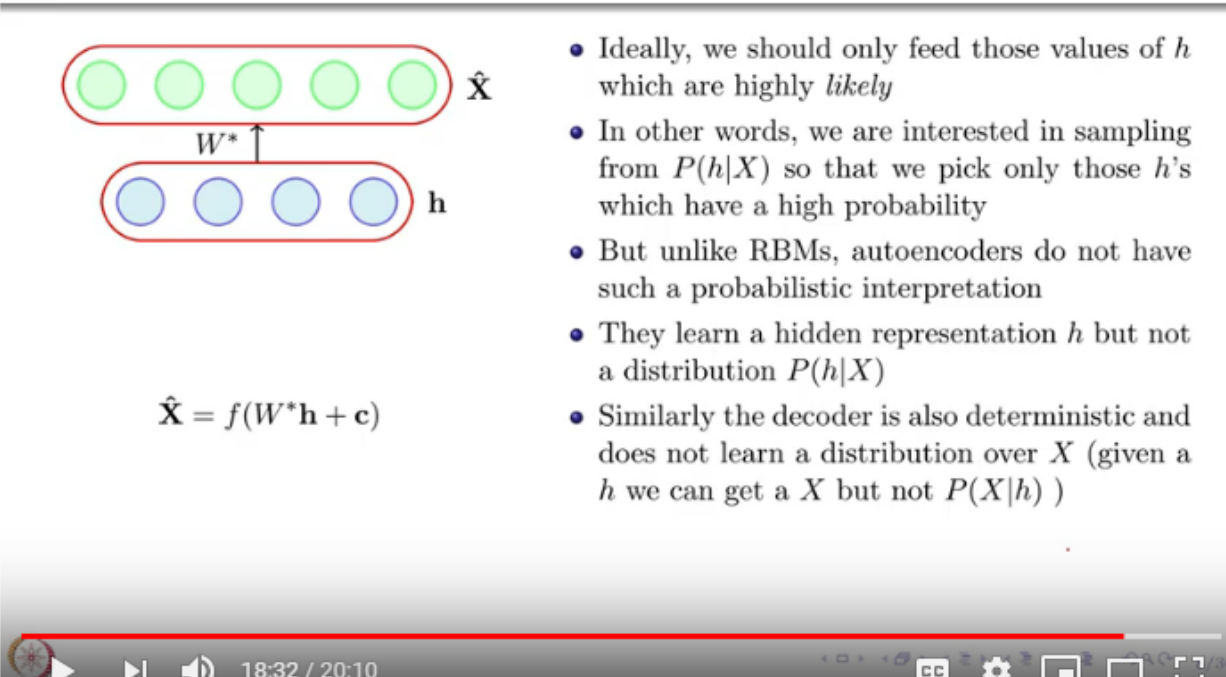
$p(h_1=1)$	$= 0.4$	1
$p(h_2=1)$	$= 0.3$	0
$p(h_3=1)$	$= 0.8$	1

I start talking about likely, what am I talking about, probability distributions. Right? So, what I want to actually do is? I want a distribution over the edges, then I want to sample from this distribution. And then take only the most likely edges. Right? So, in this case the example that I showed you, this was the manifold on, which the true etches like, if I'd learned a distribution, now it's very hard to explain this in a 2d case, but if I learn a distribution then what I wants that, it would have a very high probability for this, region of the space and almost zero everywhere else. If I had such a distribution, if I sample from this distribution ,then these are the vectors, which I am going to get and that is fine, because my model knows how to deal with that, does that make sense. Okay? So, you see the problem with autoencoders, what does it give you, a hidden representation, but that's no that's probabilistic versus, it's yeah, what's the other thing deterministic. Right? So, if I give you an X, it will give me the same hidden representation, every time. Okay?

It does not tell me a distribution, over the hidden representations is that clear. Right? For a given X, if I feed it to the encoder, I don't have the encoder equation here, but this is what IRA encoder equation, was. Right? WX plus B and once my parameters are learned, no matter how many times I feed X to this equation, I am going to get back the same H out, this is a deterministic function. Okay? However in the case of RBMS, we were actually learning a probability distribution. Right? We were learning that given an X, what's the probability distribution over the h space. Right? So, I was I was able to do this, that means if I give you an X, every time will not get back the same hidden representation, what you will get back is the probabilities of each of these hidden values taking, on a value 0 or 1 and then you sample from that distribution. Right? So, what it will tell you is that suppose there are three hidden dimensions, then what did we give you? What RBMs give you is this probability, avian remembers this that, this is the sigmoid function that we learn and suppose, it says this is 0.4, this is 0.3 and this is point 8. And now the way you construct the hidden representation, is that you sample from this distribution. Right? And you decide to set the value to 1, if you are sampling from the uniform distribution, if the number that you get

is greater than 0.4, you will set it to 0, if you get it less than point full you will set it to 1. Right? So, that way you will get the restitution, get the actual value of H. So, not getting a single H you are getting a whole bunch of H, which come from this joint distribution, does that make sense. Right? So, as opposed to Autoencoders RBMs, were actually given you a distribution, over the hidden representations. Whereas autoencoders give you one fixed hidden representation, for any given input, is that difference absolutely clear please raise your hands if it is clear. Okay? Good.

Refer slide time :(16:52)



• Ideally, we should only feed those values of h which are highly *likely*

• In other words, we are interested in sampling from $P(h|X)$ so that we pick only those h 's which have a high probability

• But unlike RBMs, autoencoders do not have such a probabilistic interpretation

• They learn a hidden representation h but not a distribution $P(h|X)$

• Similarly the decoder is also deterministic and does not learn a distribution over X (given a h we can get a X but not $P(X|h)$)

$\hat{X} = f(W^*h + c)$

Similarly the decoder, in a case of an autoencoder, is also deterministic, if I give it a hidden representation it's going to give me one X tilde back. Right? It's not going to tell me that for this hidden representation, these are all the possible X 's that I can generate. Now tie this back to the hidden representation story that we have. Right? So, hidden representations captured, things like, is this image going to be sunny, is there going to be an ocean in the background, is the B is going to be white or sandy, are there going to be trees are there going to be people and so on, if I fix the values of all this if I say that, yes it's, it's sunny, yes there's an ocean no it's not a, white beach I know there are no trees and yes there are people, even if I fix this latent distribution, there is a whole bunch of images, which can come from this latent representation, it's not that there is a one-to-one mapping, between this latent representation and the set of images that you can draw. Right? For the same description, sunny beach with white sand and people on it I can get many, many possible images. So, what I have is a distribution over the input space, given the latent representation and not a deterministic function. Right? This probabilistic interpretation was captured by RBMs, but not by autoencoders, although both give you an abstraction, RBMs were also able to do

generation, various auto encoders cannot do that does, Even completely get this part of the story, the difference the I'll not call it a limitation, but how auto-encoders differ from RBMs, although you could potentially use both, to learn an abstract representation. Okay? Okay?

Refer slide time :(18:36)

RBM \rightarrow ~~Z~~ V, H
VAE \rightarrow X, Z

We will now look at variational autoencoders which have the same structure as autoencoders but they learn a distribution over the hidden variables

$P(h|X)$
 $P(X|h)$

So, now that's going to be the basis for the rest of the discussion, what we are going to look at now is something known as variational autoencoders, which are the same structure as an autoencoder. But instead of learning a fixed representation, they learn a distribution over the representation, space. Right? So, now we can have a variational Autoencoders, which will give us the two quantities that we are interested in, P of H given X and P of X, given H. Now in the remainder of the slides, just to be consistent. So, unfortunately when people talk about, RBMs they use they don't even use X, energy they use V and H, but for some reasons when they talk about VAE is, they use X and Z. Right? So, for the remainder of this, talk to be consistent with the literature, on VAEs I will also stick to X and Z, but the mapping to RBMs that X are the visible variables and H is RB or the Z's are the hidden variables. Right? So, just be comfortable with this transition and at some places I might have, again used H by mistake instead of Z. Right?