Autoencoder

Autoencoder

Az autoencoder egy unsupervised learning technika, amit representation learning-re tudunk használni, vagyis arra, hogy az adatok valamilyen jellegzetességét megtanuljuk. Tehát csak egy $\mathcal{D} = \{x^{(1)}, \ldots, x^{(1)}\}$ adathalmazunk van, nincsenek labelek.

Ez egy olyen modell, aminek a célja az, hogy a bemenetet lemásolja, tehát egy olyan függvényt szeretne megtanulni, ami minél jobban közelíti az identitás függvényt, és így a kimenet nagyon hasonló lesz a bemenethez. Tulajdonképpen tekinthető supervised learning problémának is, egyszerűen a label-ek maguk a képek.

Egy olyan neurális hálót szeretnénk konstruálni, ami valamilyen formában tömöríti az adatokat és ezt úgy fogjuk tenni, hogy valamilyen megszorítást teszünk még pluszban a hálóra.

Az autoencoder tulajdonképpen két függvényből áll, mindkettőt egy neurális hálóval modellezzük. Az első részt encodernek hívjuk. Ez a bemenetet leképzi egy látens térbe, vagyis egy f függvényt modellez, ha x a bemenet, akkor a kimenet $h = f(x)$ a látens térbeli reprezentáció.

A második rész pedig a decoder, ez pedig veszi a látens térbeli reprezentációt, és ebből szeretné a bemenetet rekonstruálni. Ez egy g függvényt ír le, ha h a látens térbeli vektor, akkor a kimenete az autoencodernek $r = g(h)$.

Vagyis az egész háló leírható a $g \circ f$ összetett függvénnyel, és azt szeretnénk, hogy $\hat{x} = g(f(x))$ minél közelebb legyen az x-hez.

Az encoder és decoder függvény is egy paraméterezett függvény, és a loss függvényt, ami azt méri, hogy milyen jól sikerült rekonstruálni a bemenetet szeretnénk minimalizálni, ezek szerint a paraméterek szerint.

Struktúra szempontjából a legegyszerűbb autoencoder a vanilla autoencoder, ami összesen három rétegből áll, vagyis egy közbülső réteg van.

Természetesen nem muszáj ilyen egyszerűnek lennie, állhat sok rétegből, amit úgy hívunk, hogy deep autoencoder, illetve lehet konvolúciós is a háló.

Tömörítés

Tehát az autoencoder tulajdonképpen nem más, mint egy tömörítési algoritmus, ahol a betömörítő és kitömörítő függvények adatspecifikusak, veszteségesek, és automatikusan tanulhatóak az adathalmaz alapján, nem egy előre megtervezett algoritmus.

Az autoencoder adatspecifikus, vagyis ez azt jelenti, hogy csak olyan bemenetet tud tömöríteni, ami hasonló ahhoz, mint amin tanítva volt. Ez különbözik például az mp3 tömörítéstől, ahol nincsen megkötés arra, hogy milyen jellegű hangot kell tömöríteni. Ha az autoencodert például arcképeken tanítottuk, akkor rosszul fog működni például egy olyan képen, amint egy fa van, mert azok a jellemzők, amiket megtanult arc-specifikusak.

Az autoencoder veszteséges, vagyis a kimenet rosszabb minőségű lesz, mint a bemenet.

Hasznos, hogy automatikusan tanulhatóak a függvények, mert könnyű tanítani, egyedül elegendő mennyiségű adatra van szükség, nem kell kézzel manipulálni.

Sajnos nem jók tömörítésben, az a tény, hogy az autoencoder adatspecifikus, már mutatja, hogy adattömörtésre nem célszerű. A másik, hogy például fénykép tömörítésben nehéz olyan autoencodert betanítani, ami jobb eredményt ér el, mint például a JPEG tömörítés.

Ennek ellenére vannak alkalmazásai, később látni fogunk párat.

Korlátozások

Önmagában annak, hogy egyszerűen lemásoljuk a bemenetet nincsen sok értelme. A lényeg a tömörítésen van. Azt reméljük, hogy a látens reprezentáció segít felderíteni az adathalmaz rejtett összefüggéseit.

Ahhoz, hogy a háló valóban tömörítse az adatokat és ne csak lemásolja, valamiféle korlátozásokat kell tennünk a hálóra. Ugyanis ha például úgy konstruáljuk a hálót, mint ahogy az ábrán látható, ugyanannyi a látens tér dimenziója, mint a bemenet dimenziója (vagy nagyobb), akkor a háló egyszerűen meg tudja jegyezni a bemeneti értékeket, vagyis semmi új információt, összefüggést nem fedezett fel.

Az ideális autoencoder egyrészt érzékeny eléggé a bemenetre ahhoz, hogy elég pontosan vissza tudja azt állítani, viszont valahogyan el kell érni azt, hogy ne egyszerűen lemásolja a bemenetet.

Épp ezért a legtöbb esetben a loss függvény, amit minimalizálni akarunk két részből áll. Az egyik a reconstruction loss $\mathcal{L}(x, g(f(x)))$, ami azért felel, hogy minél pontosabban visszaállítsa a bemenetet, illetve egy regularizer, ami megakadályozza, hogy a model memorizálja az adatokat.

Undercomplete autoencoder

A legegyszerűbb módja a megszorításnak az, hogy a közbülső réteg(ek) dimenziójának korlátozzuk. Ezáltal limitálva van az információ, ami átáramolhat a hálón, és így a model a bemenet legfontosabb tulajdonságait tudja csak megtanulni. Ezt úgy is hívjuk, hogy bottleneck.

Mivel a neurális hálók képesek nemlineáris összefüggéseket megtanulni, ezért Ezt a modellt tekinthetjük úgy, mint a PCA egy nemlineáris, erősebb általánosítása. Míg a PCA egy alacsonyabb dimenziós hipersíkot keres, ami jól leírja az adathalmazt, addig az autoencoder képes megtanulni egy nemlineáris sokaságot, ami jellemzi az adathalmazt.

Az undercomplete autoencoder modellben nincsen külön regularization term, a modellt egyszerűen a reconstruction loss alapján tanítjuk. Tehát az egyedüli mód arra, hogy megakadályozzuk azt, hogy a modell egyszerűen memorizálja a bemenetet az, hogy eléggé lekorlátozzuk a a hidden layer-ek dimenzióját.

A deep autoencodert modellezünk, akkor még jobban vigyázni kell, mert annak ellenére, hogy mondjuk a bottleneck csak egy neuronból áll, lehet, hogy mégis meg tudja jegyezni a háló a bemenetet, ha közben megtanul valamilyen függvényt, ami az adatokat el tudja képezni egy indexre.

Regularized autoencoder

Ennél valamivel bonyolultabb módok is vannak arra, hogy megakadályozzuk, hogy a modell memorizálja a bemenetet. Nem a hidden layer-ek dimenzióját korlátozzuk, hanem ezek a modellek használnak egy regularizer-t, tehát a loss függvény picit bonyolultabb. A gyakorlatban kétféle regularized autoencoderrel szoktunk találkozni: az egyik a sparse autoencoder, a másik pedig a denoising autoencoder.

Sparse autoencoder-ek esetén egyáltalán nem követeljük meg azt, hogy a hidden layerek dimenziója kisebb legyen. Helyette úgy konstruáljuk meg a loss függvényt, hogy büntetjük a layer-eken belüli aktivációkat. Vagyis olyan encoding és decoding függvényeket szeretnénk, ami kevés neuront aktivál. Azt mondjuk, hogy egy neuron aktív, ha közel van 1-hez, és inaktív, ha 0-hoz van közel.

A képen látható egy általános sparse autoencoder, ahol a homályos neuronok nem aktiválódnak. Fontos megjegyezni, hogy az, hogy melyik neuron aktiválódik adat-specifikus: ha más az input, más neuronok fognak aktiválódni.

Tehát a háló egyes neuronokat érzékennyé tesz az adathalmaz különböző tulajdonságaira. Míg az undercomplete autoencoder az egész hálót használja, addig a sparse autoencoder csak egy részét, a bemeneti adattól függően.

Vagyis külön tudjuk kezelni a látens tér reprezentációt és a regularizációs tagot, meg tudjuk tetszés szerint választani a látens tér dimenzióját, aszerint, hogy a bemenet alapján mire van szükség, és szabályozni tudjuk a modellt egy sparsity constraint segítségével.

Ez a sparsity constraint általában két féle. Az egyik az L_1 regularization, ahol a loss függvény:

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i |a_i^{(h)}|$$

vagyis a h layer-ben vesszük mindegyik aktiváció abszolútértékét, ezeket összeadjuk, és megszorozzuk még egy konstans λ paraméterrel.

A másik pedig a Kullback-Leibler divergencia segítségével van definiálva.

Kullback-Leibler divergencia

Ez egy mérőszám arra, hogy mennyire különbözik két eloszlás egymástól. Relatív entrópiának is szokták nevezni, jele D_{KL}.

Ha X és Y folytonos valószínűségi változók, melyeknek a sűrűségfüggvénye $p(x)$, illetve $q(x)$, akkor

$$D_{\mathrm{KL}}(p\|q) = \int p(x) \cdot \log \frac{p(x)}{q(x)} \, dx = \int p(x) \cdot \big(\log p(x) - \log q(x) \big) \, dx$$

Vagyis várható értékké átírva:

$$D_{\mathrm{KL}}(p\|q) = E_{x\sim p} \log \frac{p(x)}{q(x)}$$

Csak akkor van definiálva, ha az teljesül, hogy minden x-re ha $q(x) = 0$, akkor abóól következik, hogy $p(x) = 0$. Ha $p(x) = 0$, akkor azt a tagot 0-nak vesszük, mivel $x \log(x) \to 0$, ha $x \to 0$.

Persze nem csak folytonos valószínűségi változókra lehet definiálni a Kullback-Leibler divergenciát, annyi a különbség, hogy ott szumma van az integrál helyett.

Nem távolságfüggvény, mert nem szimmetrikus és a háromszög-egyenlőtlenség sem teljesül. Viszont a Jensen-egyenlőtlenség segítségével bebizonyítható, hogy a Kullback-Leibler divergencia mindig nemnegatív, és pontosan akkor 0, ha a két eloszlás megegyezik.

Sparse autoencoder continued

Először rögzítünk egy ρ sparsity paramétert, ami tipikusan egy kicsi, pozitív szám, mondjuk 0.05. Jelölje $a_j^h(x)$ a hidden layerben a j-edik neuron aktivációját, ha a háló bemenete x volt.

Továbbá legyen

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} (a_j^{(h)}(x^{(i)})),$$

vagyis mindegyik elemre a training setből vesszük a j-edik neuron aktivációját és ezeknek vesszük az átlagát. Azt szeretnénk, hogy mindegyik neuronra a hidden layer-ben $\hat{\rho}_j$ közel legyen a ρ értékhez. Ahhoz, hogy ez teljesüljön, a legtöbb neuron aktivációjának nagyon kicsinek kell lennie.

A regularization term ebben az esetben

$$\sum_{j=1}^{s_h} D_{\mathrm{KL}}(\rho \| \hat{\rho}_j)$$

ahol s_h a neuronok száma a hidden layerben, a szumma végigmegy a hidden layeren, és a Kullback-Leibler divergencia pedig két Bernoulli eloszlás között van, az egyiknek a várható értéke ρ, a másiké pedig $\hat{\rho}_j$, vagyis az egyik ρ valószínűséggel vesz fel 1-et, $1-\rho$ valószínűséggel 0-t, hasonlóan a másik.

Két Bernoulli eloszlás között a Kullback-Leibler divergencia:

$$D_{\mathrm{KL}}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

Vagyis a második tag

$$\sum_{j=1}^{s_h} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

Korábban volt, hogy a Kullback-Leibler divergencia pontosan akkor 0, ha a két eloszlás megegyezik, vagyis akkor, ha $\rho = \hat{\rho}_j$. Vagyis a loss függvény:

$$\mathcal{L}(x, \hat{x}) + \beta \sum_{j=1}^{s_h} D_{\mathrm{KL}}(\rho \| \hat{\rho}_j),$$

ahol a β egy konstans paraméter, ami kontrollálja, hogy a második tagnak mekkora súlya van.

Denoising autoencoder

Ennél nincsen regularization tag, viszont a loss függvény sem a megszokott. Azt csinálju, hogy elrontjuk a bementetet, például zajosítjuk a képet, és a feladata a hálónak most az, hogy az eredeti zajmentes képet állítsa elő.

Ekkor a modell nem tudja memorizálni a training data-t, mert a bemenet és a kimenet nem ugyanaz. Ehelyett a modell egy vektormezőt tanul meg, ami a bemenetet egy alacsonyabb dimenziós sokaság felé képzi. Ha a sokaság jól leírja a zajmentes adathalmazt, akkor sikeresen eltüntettük a zajt.

Variational autoencoder

Generative model

Minden fajta generative model célja, hogy a training set (az adathalmaz halmaz, amin tanítjuk) eloszlását megtanulja. Ez azért jó, mert akkor ebből az eloszlásból tudunk mintát venni, és ezáltal új adatot tudunk generálni, ami hasonlít az eredeti adatokra, de nem egyezik meg velük.

Sajnos nem mindig lehet a pontos eloszlást megtanulni, tehát egy olyan eloszlást szeretnénk modellezni, ami minél jobban közelíti a valódi eloszlást.

Ha például egy ilyen modell az MNIST adathalmazon van tanítva, akkor nyilván nagy valószínűségűnek kellene lennie egy olyan képnek, amin egy számjegy van, de kis valószínűségű legyen például egy olyan, ami egy random zagyvaság.

Ezeket a modelleket sok féleképpen lehet alkalmazni, például lehet nem valódi emberi képeket, generálni, vagy 3D növényeket kreálni, és azokkal megtölteni egy erdőt egy videojátékban, vagy taníthatujk kézzel írt szövegeken és tudunk generálni újabb kézzel írt szöveget, stb.

Az egyik ilyen népszerű generative modell a variational autoencoder.

Latent variable model

Ha az egyes pixelek függetlenek lennének, akkor minden egyes pixel sűrűségfüggvényét kell külön-külön megtanulni, ami egyszerű, és a mintavétel is egyszerű, minden pixelből függetlenül veszünk mintát.

Azonban azokon a képeken, amin tanítjuk a modellt egyértelműen van kapcsolat a pixelek között, például ha a kép bal oldali felét nézzük és az úgy néz ki, mint egy 4-es kezdete, akkor a kép jobb oldalán nem lehetséges, hogy például egy 0-nak a végét lássuk.

Ezeket a kapcsolatokat a látens tér írja le. Erre gondolhatunk úgy, mint egy R^k-dimenziós vektortérre, ahol mindegyik vektor k db lényeges információt tárol arról, hogy hogyan kell

egy egy számjegyet megrajzolni. Például az első dimenzió lehet, hogy melyik számjegyet rajzoljuk, a második a szélessége, stb.

Feltesszük, hogy a folyamat, ami generálja a számjegyeket, két részből áll. Először eldönti, hogy milyen tulajdonságai legyenek a számjegynek, amit rajzolni szeretne, majd megrajzolja.

Problem scenario

Van egy adathalmazunk, $\mathcal{D} = \{x^{(1)}, x^{(1)}, \ldots, x^{(n)}\}$, amiről feltesszük, hogy mindegyik eleme egy független minta, egy változatlan, $p^*(x)$ eloszlásból. Vagyis az adathalmaz független, azonos eloszlású. Ez a training set.

Feltesszük azt is, hogy az adathalmaz egy véletlen folyamattal volt generálva, egy z folytonos valószínűségi változó segítségével, amit nem tudunk megfigyelni.

A folyamat úgy zajlik, hogy először generálunk egy $z^{(i)}$ vektort a $p(z)$ prior eloszlásból, és ebből van generálva egy $x^{(i)}$ vektor egy $p(x|z)$ feltételes eloszlás segítségével.

A teljes valószínűség tétele alapján a marginális eloszlás $p(x) = \int p(x, z)dz = \int p(x|z)p(z)dz$. A cél, hogy minden egyes $x^{(i)}$ elemre az adathalmazból, a $p(x^{(i)})$ minél nagyobb legyen. Vagyis amit csinálni szeretnénk, az tulajdonképpen egy maximum-likelihood becslés.

Intractabilities

A legnagyobb probléma a maximum likelihood becsléssel, hogy a marginális eloszlás tipikusan nehezen kezelhető az integrál miatt. Ha az integrált nem tudjuk kiszámolni, akkor nem tudjuk a paraméterei szerint deriválni, és optimalizálni.

Egyébként a marginális eloszlás nehezen kezelhetősége a $p(z|x)$ eloszlás nehezen kezelhetőségével függ össze. Ugyanis az együttes eloszlást hatékonyan ki lehet számolni, és a Bayes-tétel alapján

$$p(x|z) = \frac{p(x, z)}{p(x)}.$$

Monte-Carlo módszerrel meg lehet próbálni becsülni: vegyünk N db mintát a prior eloszlásból: $\{z_1, \ldots, z_n\}$, és becsüljük a marginális eloszlást: $p(x) \approx \frac{1}{N} \sum_{i=1}^{N} p(x|z_i)$.

Azonban ezzel pedig az a gond, hogy mivel az x valószínűségi változó dimenziója nagy, ezért nagyon sok mintát kellene vennünk a prior eloszlásból ahhoz, hogy egy viszonylag jó közelítését kapjuk a marginális eloszlásnak.

Vagyis olyan módszert szeretnénk találni, ami akkor is jól működik, ha nehezen kezelhető az integrál, illetve, ha akkora az adathalmaz, hogy mintavételezést használó megoldások túl lassúak.

A prior és feltételes eloszlás

Kérdés, hogy hogyan definiáljuk a látens változókat? A döntések, amiket a modellnek meg kell hoznia, mielőtt elkezdené rajzolni a számjegyet elég bonyolultak. Ki kell találnia, hogy melyik számjegyet rajzolja, a szöget, stílust, stb. Ami rosszabb, hogy ezek a tulajdonságok összefügghetnek, például ha valaki gyorsan ír, akkor a számjegy dőltebb lehet és vékonyabb is.

Ideális esetben el szeretnénk kerülni, hogy saját magunk határozzuk el minden egyes koordinátáról a látens térben, hogy milyen tulajdonságot kódol, és azt is el szeretnénk kerülni, hogy az egyes összefüggéseket a koordináták között mi írjuk le.

A VAE ezek egyikét sem teszi, hanem azt mondja, hogy úgy kaphatjuk a látens tér egy elemét, hogy az $\mathcal{N}(0, I)$ (standard normális) eloszlásból mintát veszünk, vagyis $p(z) = \mathcal{N}(0, I)$, ahol I az egységmátrix.

Ez azért tud működni, mert ha d dimenzióban veszünk egy tetszőleges eloszlást, akkor azt meg tudjuk kapni úgy, hogy egy szükségszerűen bonyolult függvényt alkalmazunk a d-dimenziós standard normális eloszlásra.

Tehát ha van egy neurális hálónk, ami a látens térbeli vektorból csinál egy számjegyet, akkor képzelhetjük úgy, hogy a háló első néhány rétege megtanulja azt a függvényt, ami a valódi látens eloszlásra képezi a normális eloszlást, és a normális eloszlásból mintavételezett z vektorokat elképzi olyan látens vektorokba, amire a modellnek szüksége van, majd ezeket a vektorokat képzi egy számjegybe.

Feltesszük, hogy a feltételes eloszlás egy paramétercsaládból származik, valami θ ismeretlen paraméterekkel, és a célunk az, hogy a θ paramétereket beállítsuk úgy, hogy minden egyes training set-beli kép valószínűsége minél nagyobb legyen.

Konkrétan a $p_\theta(x|z)$ feltételes eloszlás egy normális eloszlás lesz, $\mathcal{N}(f(z, \theta), \sigma^2 I)$, ahol σ egy hiperparaméter. A neurális hálónk kimenete az $f(z, \theta)$ függvény.

Ebben a modellben az a jó, hogy a prior és feltételes eloszlások viszonylag egyszerű, viszont attól még a marginális eloszlás lehet negyon bonyolult, ezért jól lehet használni arra, hogy a valódi eloszlást közelítsük vele.

Shortcut

A marginális eloszlás helyett a logaritmusával fogunk dolgozni, annak a maximalizálása a kérdéses helyeken ekvivalens a marginális eloszlás maximalizálásával. Mivel a training setről feltettük, hogy i.i.d, ezért a $\sum_{i=1}^{n} \log p_\theta(x^{(i)})$ összeget szeretnénk maximalizálni. Így elég mindegyik adathalmazbeli képpel külön foglalkozni.

Korábban már említve volt, hogy lehetne úgy közelíteni a marginális eloszlást, hogy veszünk sok mintát a prior eloszlásból, és kiszámoljuk az $\frac{1}{N} \sum_{i=1}^{N} p(x^{(i)}|z_i)$, ezzel viszont az volt a baj, hogy túl sok mintát kellene venni.

Kérdés, hogy gyorsabbá lehet-e tenni ezt az eljárást. A tapasztalat azt mutatja, hogy a legtöbb z értékre $p_\theta(x^{(i)}|z)$ nagyon kicsi, vagyis a marginális eloszlás becsléséhez semmivel sem járul hozzá.

A VAE mögötti kulcs ötlet az, hogy olyan z értékeket szeretnénk mintavételezni, amire a feltételes valószínűség nagy, vagyis aminél valószínű, hogy a látens vektor ezen értékeiből keletkezett a vizsgált $x^{(i)}$ számjegy, és csak ezekre szeretnénk az átlagot kiszámolni, ami közelíti a marginális eloszlást.

Ez azt jelenti, hogy szükségünk van egy új eloszlásra, $q(z|x^{(i)})$, ami egy eloszlást ad a látens téren, és megmondja, hogy mely z értékek azok, amik valószínűleg a képet generálták. Reményeink szerint azon z-k halmaza, amikre ez a valószínűség nagy, sokkal kisebb, mint

azon z-k halmaza, amik a prior eloszlás szerint valószínűek. Ekkor például a $E_{z\sim q}p(x^{(i)}|z)$ várható értéket relatíve könnyedén ki tudjuk számolni.

Vehetnénk a $p_\theta(z|x^{(i)})$ posterior eloszlást, ami pontosan megmondja, azonban feltettük, hogy az nehezen kezelhető. Ezért azt szeretnénk, hogy a q eloszlás minél jobban közelítse a posterior eloszlást.

Kérdés persze az, hogy ha z vektort úgy kapjuk, hogy egy tetszőleges eloszlásból mintavételezzük, aminek a sűrűségfüggvénye ez a q, ami nem a standard normális, akkor az miért segít a marginális eloszlás optimalizálásában.

Az első dolog, amire szükség van, hogy kapcsolatba hozzuk a marginális eloszlást a könnyen számolható $E_{z\sim q}p(x^{(i)}|z)$ várható értékkel. Azt majd később látni fogjuk, hogy a q honnan jön.

Itt már látszódik, hogy miért VAE-nak hívják a modellt, gyakorlatilag megkaptuk az autoencoder-eknél látott struktúrát, az encoder rész modellezi a q eloszlást és a decoder rész pedig a $p_\theta(x|z)$ eloszlást.

Reforming the Kullback-Leibler divergence

Most egy csomó számolás fog következni. Először a $q(z|x^{(i)})$ és $p_\theta(z|x^{(i)})$ eloszlások közötti KL-divergenciát kezdjük el vizsgálni és átalakítani. A korábbiak alapján:

$$D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}(\log q(z|x^{(i)}) - \log p_\theta(z|x^{(i)})).$$

Most alkalmazzuk a Bayes-tételt a $p_\theta(z|x^{(i)})$ eloszlásra:

$$p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)} \mid z)p(z)}{p_\theta(x^{(i)})}.$$

Vagyis, logaritmust véve:

$$\log p_\theta(z|x^{(i)}) = \log p_\theta(x^{(i)} \mid z) + \log p(z) - \log p_\theta(x^{(i)}).$$

Ezt beírva a fenti egyenlet jobb oldalába:

$$D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log q(z|x^{(i)}) - \log p_\theta(x^{(i)}|z) - \log p(z) + \log p_\theta(x^{(i)})\Big).$$

A várható értékből az utolsó tag kivihető, mert nem függ z-től, tehát a következőt kaptuk:

$$D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log q(z|x^{(i)}) - \log p_\theta(x^{(i)}|z) - \log p(z)\Big) + \log p_\theta(x^{(i)}).$$

Ezt átrendezve, a jobb oldalon az utolsó tagot átvive a bal oldalra, és mindkét oldal (-1)-szeresét véve:

$$\log p_\theta(x^{(i)}) - D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log p_\theta(x^{(i)}|z) + \log p(z) - \log q(z|x^{(i)})\Big).$$

A jobb oldalt még tovább tudjuk rendezni. Ugyanis ha megfigyeljük, akkor

$$D_{KL}(q_\phi(z|x^{(i)})\|p(z)) = E_{q(z|x^{(i)})}\Big(\log q(z|x^{(i)}) - p(z)\Big),$$

és ez megjelenik az előző egyenlet jobb oldalán. Tehát végül azt kapjuk, hogy

$$\log p_\theta(x^{(i)}) - D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log p_\theta(x^{(i)}|z)\Big) - D_{KL}(q_\phi(z|x^{(i)})\|p(z)).$$

Ez az egyenlet a lényege a VAE-nak, vizsgáljuk meg, hogy mit jelent. A bal oldalon megjelent az, amit maximalizálni szeretnénk. Ezen kívül van még egy hiba tag, egy Kullback-Leibler divergencia. Minél kisebb ez a tag, annál inkább teljesül, hogy a q eloszlásból mintavételezve olyan z értéket kapunk, ami képes az $x^{(i)}$-t reprodukálni.

Ha a bal oldalt maximalizálni akarjuk, akkor azzal egyszerre maximalizáljuk az értéket, amit szeretnénk, és közelítjuk a posterior eloszlást a q eloszlással.

ELBO

Most vizsgáljuk meg a jobb oldalt. Ez alapján megérthetjük, hogy miben különbözik az variational autoencoder az autoencoder-től. Vizsgáljuk meg a látens teret.

A legnagyobb előnye a VAE-nak az, hogy egy sima látens térbeli reprezentációját tudjuk megtanulni a bemeneti adatoknak. A sima autoencodernél csak egy olyan encoding függvényt kell megtanulni, aminek a segítségével rekonstruálni tudjuk a bemenetet. A bal oldali ábrán látjuk, hogy mi történik, ha csak a reconstruction loss-ra koncentrálunk. Ekkor a különböző osztályokat sikerült szeparálnunk egymástól, ami megkönnyíti a rekonstrukciót. De az a gond, hogy vannak olyan részei a látnes térnek, ami egyáltalán nem reprezentálja egyik megfigyelt adatot sem.

Ha ezzel szemben meg csak a Kullback-Leibler divergenciára fókuszálunk, vagyis arra, hogy a látens eloszlás közel legyen a prior eloszláshoz, akkor meg mindegyik megfigyelést ugyanazzal az eloszlással, a standard normálissal próbálunk leírni. Ekkor pedig a modell úgy képzeli, hogy midnegyik megfigyelésnek ugyanazok a karakterisztikái, vagyis nem tudtuk jellemezni az eredeti adathalmazt.

Ellenben ha a kettőt együtt optimalizáljuk, akkor egyrészt bátorítjuk a modellt arra, hogy az adathoz tartozó látens eloszlást egy olyannal írja le, ami közel van a prior eloszláshoz, de mégis a másik tag meg arra bátorítja, hogy mégis különbözzenek, ha szükséges.

A VAE-nál általában q eloszlást is egy normális eloszlásnak szokták választani, vagyis $q(z|x^{(i)}) = \mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)}))$. Ezt az eloszlást is egy neurális háló paraméterezi, az ismeretlen paramétereket ϕ-vel jelöljük, a μ és a Σ dereminisztikus, ϕ-től függő függvények, amiket a háló megtanul. A Σ-ról azt is fel szokták tenni, hogy diagonális mátrix.

Az egyenlet jobb oldalát úgy szokták hívni, hogy ELBO, jelölése:

$$\mathcal{L}(\theta, \phi, x^{(i)}) = E_{q_\phi(z|x^{(i)})} \log p_\theta(x^{(i)}|z) - D_{KL}(q_\phi(z|x^{(i)})\|p(z)).$$

Úgy is szokták hívni, hogy Variational Lower Bound. Miért alsó korlát: azért, mert a bal oldalon szereplő Kullback-Leibler divergencia nemnegatív. Ha a jobb oldalt maximalizáljuk, akkor ezzel egy csapásra maximalizáljuk a bal oldalt, és a jobb oldalt tudjuk kezelni, a gradiens módszerrel, mint látni fogjuk.

Azért jó, hogy a q_ϕ eloszlást normális eloszlásnak választjuk, mert két normális eloszlás közötti Kullback-Leibler divergenciát zárt alakra tudjuk hozni. Legyenek $\mathcal{N}(\mu_0, \Sigma_0)$ és $\mathcal{N}(\mu_1, \Sigma_1)$ tetszőleges normális eloszlások. Ekkor a Kullback-Leibler divergenciájuk:

$$D_{KL}(\mathcal{N}(\mu_0, \Sigma_0) \| \mathcal{N}(\mu_1, \Sigma_1)) =$$
$$= \frac{1}{2}\Big(\operatorname{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\Big(\frac{\det \Sigma_1}{\det \Sigma_0}\Big)\Big),$$

ahol a k az eloszlás dimeziója. A mi esetünkben:

$$D_{KL}(\mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)})) \| \mathcal{N}(0, I)) =$$
$$= \frac{1}{2}\Big(\operatorname{tr}(\Sigma_\phi(x^{(i)})) + (\mu_\phi(x^{(i)})^T (\mu_\phi(x^{(i)}) - k + \log\det(\Sigma_\phi(x^{(i)})\Big)$$

Az első tag, a várható érték egy picit bonyolultabb. Lehetne azt csinálni, hogy közelítjük mintavételezéssel, tehát veszünk sok-sok z_i mintát a q eloszlásból, és azokat átlagoljuk. Azonban kimérték azt, hogy egy darab mintát veszünk a q-ból, és vesszük arra a z-re a $\log p_\theta(x^{(i)}|z)$, már az is elég jó közelítése lesz a várható értéknek.

Tehát ha nézzük a várható értéket az ELBO-nál a jobb oldalon, akkor ott a logaritmusa szerepel egy normális eloszlás sűrűségfüggvényének. Ha kiszámoljuk a logaritmust, akkor van egy konstans tag, ami nem értdekes, és az ellentettje az $x^{(i)}$ és az $f(z, \theta)$ vektorok távolság négyzetének. Tehát ez tényleg reconstruction loss, azt szeretné, hogy a távolság a bemenet és a kimenet között minél kisebb legyen.

A θ és a ϕ paramétereket szeretnénk megtalálni, tehát ezek szerint szeretnénk a gradiensét venni az ELBO-nak. Mivel a második tag nem függ θ-tól, ezért

$$\nabla_\theta \mathcal{L}(\theta, \phi, x^{(i)}) = \nabla_\theta E_{q_\phi(z|x^{(i)})} \log p_\theta(x^{(i)}|z).$$

Mivel $q_\phi(z|x^{(i)})$ sem függ θ-tól, ezért

$$\nabla_\theta E_{q_\phi(z|x^{(i)})} \log p_\theta(x^{(i)}|z) = E_{q_\phi(z|x^{(i)})}(\nabla_\theta \log p_\theta(x^{(i)}|z)) \simeq$$

$$\simeq \nabla_\theta \log p_\theta(x^{(i)}|z),$$

ahol ez az utolsó sor Monte-Carlo becslése az előzőnek, z ott egy random minta a $q_\phi(z|x^{(i)})$ eloszlásból.

A ϕ szerint véve a gradienst, a második taggal nincsen gond. azonban Az első tagnál most nem vihetjük be a gradienst az integrálba, valami mást kell kitalálni.

Reparametrization trick

Először szemléletesen nézzük, hogy mit jelent, mi a baj, ha ϕ szerint vesszük a gradienst. Az a probléma, hogy amikor a backpropagation-t végezzük, akkor annak át kell mennie egy olyan rétegen, ami mintavétellel ad egy z vektort a $q_\phi(z|x^{(i)})$ eloszlásból, egy nem folytonos művelet.

Az ötlet az, hogy újraparaméterezzük a z valószínűségi változót, mint az $x^{(i)}$, a ϕ és egy új valószínűségi változó, ε függvénye, és így már ki tudjuk számolni az f függvény ϕ szerinti gradiensét.

Általában, ha z egy valószínűségi változó, és g egy differenciálható és invertálható transzformáció, amire $z = g(\varepsilon, \phi, x)$, ahol a ε valószínűségi változó $r(\varepsilon)$ eloszlása nem függ x-től vagy ϕ-től, akkor

$$E_{q_\phi(z|x^{(i)})}(f(z)) = E_{r(\varepsilon)}(f(g(\varepsilon, \phi, x))).$$

Innentől kezdve ugyanúgy el tudunk járni, mint az első esetben, a gradienst be tudjuk vinni a várható értékbe. Szemléletesen az történt, hogy lett egy új input réteg, ami az ε-et adja.

A mi esetünkben a posterior eloszlás $\mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)}))$, legyen $\varepsilon \sim \mathcal{N}(0, I)$, vagyis standard normális eloszlású. Ekkor $z = \mu_\phi(x^{(i)} + \Sigma_\phi^{1/2}(x^{(i)}) \cdot \varepsilon$ teljesül, ami egy differenciálható, invertálható függvény, tehát így már működik a backpropagation.

Testing

Ha már betanítottuk a hálót, és tesztelni akarjuk, vagyis újabb képet szeretnénk generálni, akkor az egész encoder részre, beleértve az összeadást és szorzást, ami megváltoztatja a z eloszlását. Egyszerűen veszünk egy mintát a standard normális eloszlásból, és azt átküldjük a decoder-en.

Az alábbi ábrán az MNIST adathalmazon betatnított VAE által generált kimenetet mutatja, itt egy két dimenziós normális eloszlásból vett mintát, egy rácson, és itt látszik a kimenet. Ezen is látszik, hogy a különböző számjegyek különböző területeken élnek, folytonosan mennek át egymásba.