Математичка гимназија

Матурски рад
из
информатике

Варијациони аутоенкодери

Ученик:
Иван Поп-Јованов, IVб

Ментори:
Јелена Хаџи-Пурић
Петар Величковић

Београд, Мај 2018.

Садржај

1 Увод

Последњих деценија, начини решавања проблема у информатици се све више окрећу машинском учењу. Вештачка интелигенција има технолошки потенцијал да буде боља од људи у већини интелектуалних активности. Неуралне мреже боље играју шах, боље препознају компликоване позадинске релације података.

Иако је вештачка интелигенција надмашила људе у неким погледима, много је већа количина области у којима људиски мозак и даље побеђује. Једна од тих области је способност да репродукује податке које среће у природи. Човек може да наслика слику дрвета које је видео у парку, али још боље од тога, човек може да наслика слику дрвета које никад није видео, само на основу његовог знања о разним параметрима који моделују изглед дрвета. Човек такође може да компонује музичко дело које никад није чуо или да напише књигу коју никад није прочитао. Ово успева узимајући правилности из других музичких дела и књига које је искусио у животу.

Вештачка интелигенција је веома добра у препознавању тога да је нешто дрво, али када би тражили од ње да генерише сасвим нову слику дрвета, не би била сигурна како то да уради. Да би уклонили овуману вештачке интелигенције људи су осмислили неке од могућих модела који сагледају проблем из различитих углова. Овај рад има за циљ да опише и објасни једну групу приступа решавању овог типа проблема.

Рад ће започети основним уводом у генералне технике машинског учења, а касније ће се фокусирати на генеративно моделовање и на приступ варијационог закључивања.

2 Машинско учење

2.1 Мотивација

Начин на који се рачунари користе за решавање проблема се може у већини случајева свести на експлицитне алгоритме који у израчунавању користе неке већ познате информације о проблему.

Постоје неки проблеми које је веома тешко, ако не и немогуће, решавати на овакав начин. Један од примера је препознавање цифара писаних слободном руком са фотографије. Покушаји решавања овог проблема експлицитним програмирањем веома брзо наилазе на зид бесконачних случајева, подслучајева и изузетака. Са друге стране, људски мозак нема никакве проблеме да увиди позадинске правилности у подацима и да веома брзо и ефикасно научи да их разликује.

Машинско учење је област вештачке интелигенције која покушава да имитира начин на који људски мозак учи да решава проблеме. Један од приступа овоме су **неуронске мреже** које се базирају на биолошкој архитектури људског мозга.

Да би неуралном мрежом решили неки проблем, потребна је **велика количина података**. Мрежа која учи да решава малопре поменут проблем препознавања цифара писаних слободном руком захтева да јој дамо што више примера фотографија цифара за које смо означили која се цифра налази на њима. База података која се најчешће користи за решавање овог проблема је МНИСТ база која садржи 60 хиљада слика писаних цифара величине 28x28 пиксела.

Овакве базе података постоје и за многе друге проблеме који се решавају неуронским мрежама. Неки примери проблема су препознавање објекта на слици, препознавање гласа особе, препознавање теме текста, самовозећи аутомобили, филтрирање спам мејлова итд.

Машинско учење се обично дели у три категорије:

Учење са надзором - Мрежи су дати парови улаза и излаза за које она мора да нађе правилности. Овде спада поменути проблем препознавања цифара.

Учење без надзора - Мрежи су дати подаци који нису означени за које мора сама да пронађе позадински модел који их дефинише. Пример овога би био да су мрежи дате мало пре поменуте слике цифара али без ознака која је која цифра и уместо да се тражи од мреже да их препознаје и разликује, од ње се захтева да генерише нове слике цифара које човек не би могао да разликује од правих. Једним примером ове мреже се бави овај рад.

Учење са појачањем - Мрежа је агент којем је дато неко интерактивно окружење (на пример видео игрица) у ком она мора да доноси одлуке. На основу повратних информација о томе да ли је та одлука била добра или не, мрежа постаје све боља у решавању датог проблема.

Један подскуп проблема учења без надзора се решава **генеративним моделовањем**. Ово су проблеми у којима се тражи од мреже да генерише податке који прате исте правилности као реални подаци. Неки од примера су генерисање фотографија лица људи, генерисање људског гласа, генерисање музике у стилу одређеног композитора, генерисање текста који изгледа као дијалог итд.

У решавању оваквих проблема постоје две водеће архитектуре: ГАНови и Варијациони аутоенкодери.

ГАНови вуку инспирацију из теорије игара и генеришу податке коришћењем две мреже, једну која покушава да генерише што реалније податке, и другу која покушава да што боље препозна да ли је податак стваран или генерисан. Њиховим такмичењем и једна и друга мрежа ће постајати све боља у свом циљу. Тако се добијају подаци које је веома тешко разликовати од реалних.

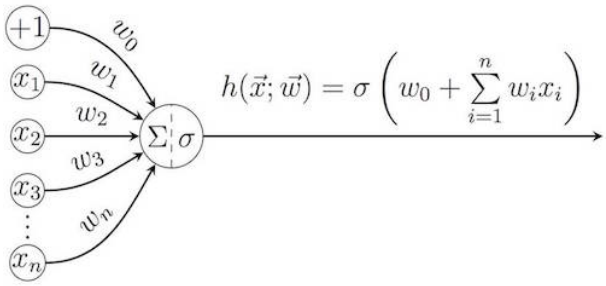
Варијациони аутоенкодери са друге стране користе теорију информација, статистику и варијационо закључивање. О оваквим мрежама ће бити више говора касније у овом раду.

Слика изнад је пример резултата мреже која учи да стил неког уметника пренесе на дату фотографију.

2.2 Неуронске мреже

2.2.1 Неурон

Неурон је структура која узима улазни вектор $\vec{x} = [x_1, x_2, ..., x_n]$ и тежински вектор $\vec{w} = [w_0, w_1, w_2, ..., w_n]$ и примењује активациону функцију σ на линеарну комбинацију $w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$ где је w_0 слободан члан.



Активациона функција може да буде само идентитет $\sigma(z) = z$, али обично се за њу узима нека функција која ће да дода нелинеарност у систем. Неки од честих примера су:

Сигмоидне функције:

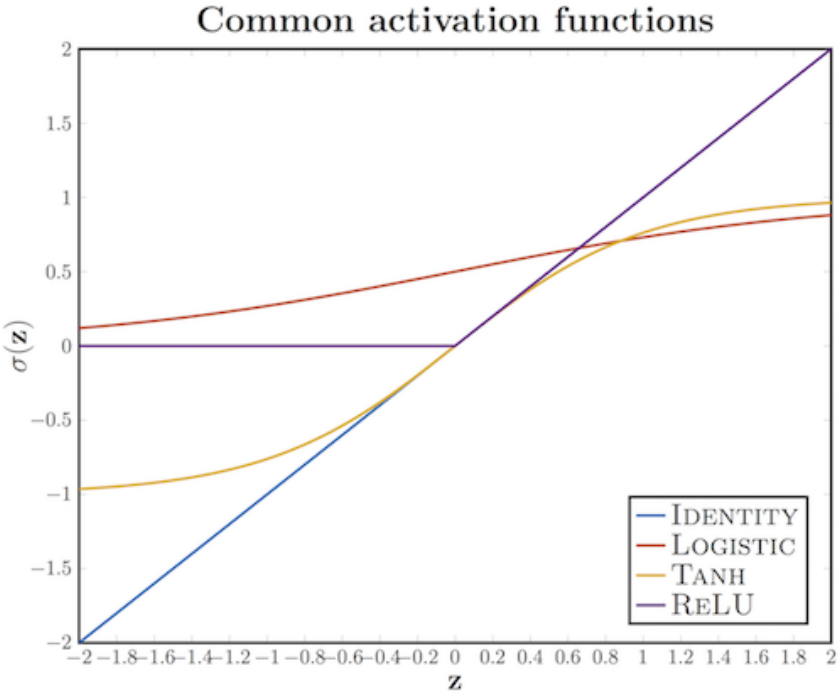
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\sigma(z) = \tanh(z)$$

или исправљена линеарна функција (ReLU):

$$\sigma(z) = \max(0, z)$$

која се доста користи зато што се брзо израчунава, а доноси таман довољно нелинеарности у алгоритам.



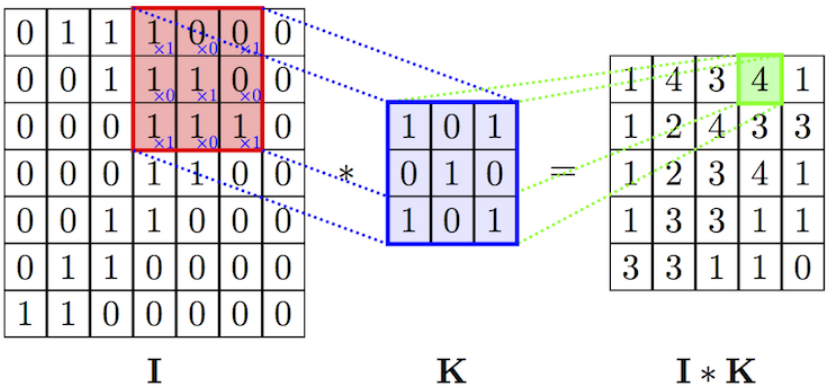
2.2.2 Слој неурона

Неуронске мреже се праве тако што се неурони групишу у слојеве и онда ти слојеви ређају у низ. Први и последњи слојеви се зову улазни и излазни слој. Слојеви између њих се зову скривени слојеви. Ако мрежа има више од једног скривеног слоја, назива се дубоком.

Слојеви се спајају тако што неурони једног слоја узимају за узлаз излазе једног или више неурона претходних слојева. Ако сваки неурон неког слоја узима за улаз излазе свих неурона претходног слоја, тај слој зовемо **густо повезаним**.

Неуронске мреже се могу поделити и на рекурентне и нерекурентне у зависности од тога да ли у њиховој конфигурацији слојева постоје циклуси.

Један користан пример слоја неурона је **конволуциони слој**. Он је направљен са циљем да буде оптимизован за ситуације у којима су подаци на којима тренирамо мрежу фотографије. Користи чињеницу да пиксели слике нису одвојени подаци сваки за себе, него је битна позиција сваког пиксела. Две фотографије које се састоје од скроз истих пиксела само у различитом распореду не морају да буду исте.



Неурони у овом слоју нису повезани са свим неуронима претходног слоја, него само са онима који му се налазе у непосредној близини. Ово се реализује померањем мале матрице (која се обично зове кернел) преко улазне слике и користећи вредности унутар те матрице као улазе за неурон.

Овакви модели обично налазе кернеле који су корисни за налажење специфичних детаља везаних за слике, као што су ивице објеката, ћошкови, градијенти и слично.

2.2.3 Тренинг мреже

Процес тренирања неуронске мреже обухвата проналажење оптималних тежинских вектора за све неуроне. Ово се реализује минимизовањем **функције губитка** L која представља удаљеност резултата мреже од стварних вредности на скупу примера на којима су нам познате. Скуп улаза за које су позната решења зове се **тренинг сет** и што он више примера обухвата то ће мрежа бити боље истренирана.

2.3 Градијентни спуст

Минимизовање функције губитка L се своди на налажење парцијалног извода

$$\frac{\partial L}{\partial w_{i\to j}}$$

за тежину која спаја неурон i и неурон j. Даље се свака тежина у мрежи смањи за њен одговарајући парцијални извод помножен са стопом учења η која се обично динамички варира. Овај алгоритам се зове **градијентни спуст** и он се користи за налажење локалног минимума функције губитка L.

2.3.1 Пропагација уназад

Парцијални изводи потребнии за градијентни спуст се налазе алгоритмом пропагације уназад. Пропагација уназад за неки пример i из тренинг сета се реализује тако што се прво улаз x_i пропусти кроз мрежу и излаз мреже се упореди са одговарајућим жељеним излазом y_i.

Налажење парцијалног извода неке тежине $w_{i\to j}$ се дели на два случаја.

У првом случају тражена тежина се налази у излазном слоју. Из правила о изводу сложене функције следи

$$\frac{\partial L}{\partial w_{i\to j}} = \frac{\partial z_j}{\partial w_{i\to j}} \frac{\partial a_j}{\partial z_j} \frac{\partial L}{\partial a_j}$$

Где је z_j вредност неурона пре примењивања функције σ, а a_j после ($a_j = \sigma(z_j)$).

$$\frac{\partial z_j}{\partial w_{i\to j}} = a_i$$

$$\frac{\partial a_j}{\partial z_j} = \sigma'(z_j)$$

Ако се функција губитка L за излаз a_j и тражени излаз t_j дефинише као $\frac{1}{2}(a_j - t_j)^2$ онда важи и

$$\frac{\partial L}{\partial a_j} = a_j - t_j$$

и ако се уведе нова ознака

$$\delta_j = (a_j - t_j)\sigma'(z_j)$$

онда важи

$$\frac{\partial L}{\partial w_{i\to j}} = a_i \delta_j$$

У другом случају тражена тежина се налази у неком од скривених слојева.

$$\frac{\partial L}{\partial w_{i\to j}} = \frac{\partial z_j}{\partial w_{i\to j}} \frac{\partial a_j}{\partial z_j} \frac{\partial L}{\partial a_j}$$

$$\frac{\partial z_j}{\partial w_{i\to j}} = a_i$$

$$\frac{\partial a_j}{\partial z_j} = \sigma'(z_j)$$

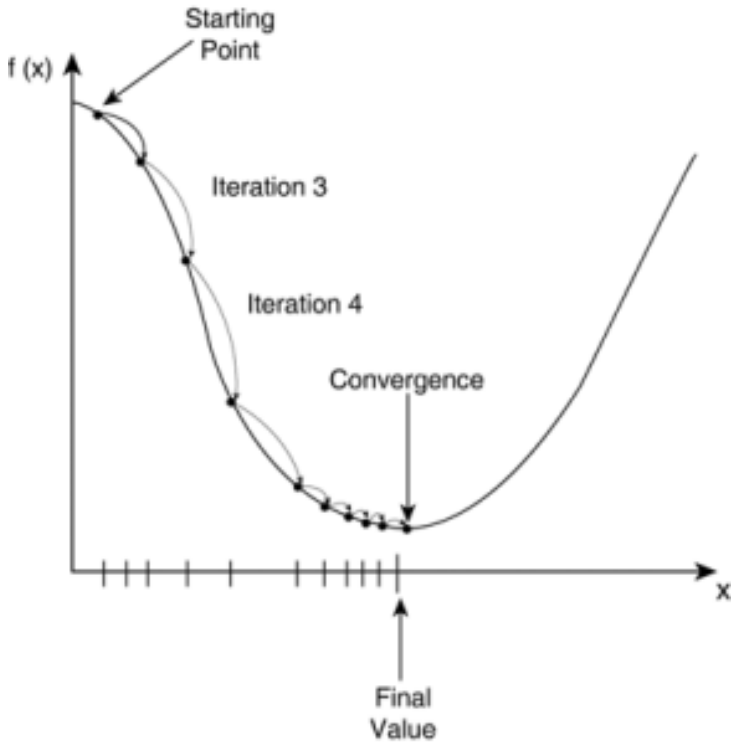
$$\frac{\partial L}{\partial a_j} = \sum_k (\frac{\partial z_k}{\partial a_j} \frac{\partial a_k}{\partial z_k} \frac{\partial L}{\partial a_k}) = \sum_k (w_{j \to k} \delta_k)$$

и дефинише се

$$\delta_j = \sigma'(z_j) \sum_k (w_{j \to k} \delta_k)$$

И онда се промене вредности тежина градијентним спустом:

$$w_{i \to j} \leftarrow w_{i \to j} - \eta \frac{\partial L}{\partial w_{i \to j}}$$

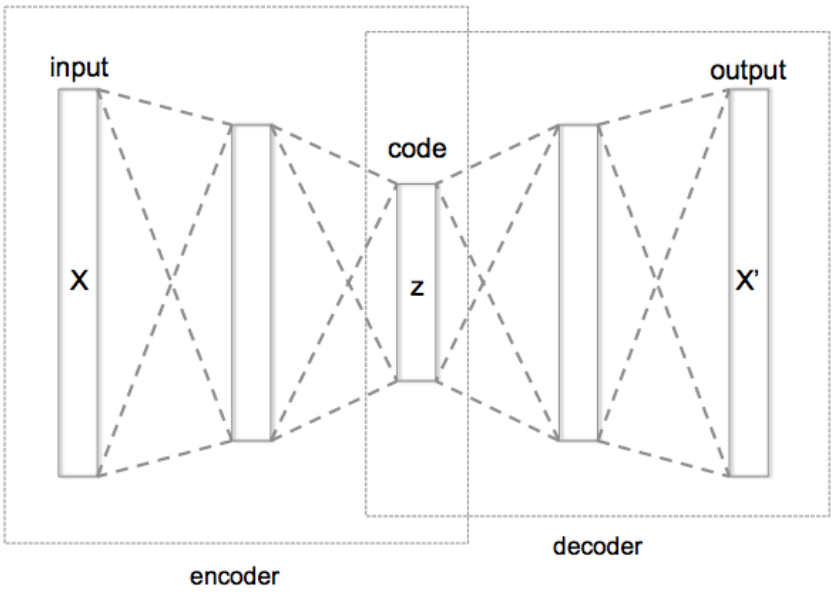


Слика изнад представља визуалну репрезентацију градијентног спуста.

3 Аутоенкодер

У већини случајева подаци узети из стварног света могу да изгледају компликовано иако се у позадини могу описати са само неколико параметара. Ако се узму на пример фотографије неке коцкице за бацање из различих углова, без знања шта да очекујемо на фотографији, потребно је сачувати вредност сваког пиксела слике. Међутим, ако се у обзир узме информација да је на слици коцкица за бацање и ми знамо како та коцкица изгледа, потребно је само сачувати параметре угла из ког је узета фотографија, и могуће је генерисати изнова целу фотографију. Ову чињеницу користи аутоенкодер.

Аутоенкодер је посебна структура неуралне мреже која за циљ има да научи репрезентацију сета података. Реализује се тако што је скривени слој ужи од улазног, а излазни исте ширине као улазни. Мрежа се тренира да на излазу генерише идентичну копију свог улаза. Ово тера мрежу да научи да прикаже улазне податке кодом ширине најужег скривеног слоја. Ако мрежа успе да само на основу неколико променљиви реконструише слику, значи да је нашла од којих параметара зависи садржај слике. На пример ако се тражи од ње да реконструише слике малопре поменуте коцкице на основу само три променљиве, мрежа би успела да научи да искористи те променљиве да представи ротацију коцкице, а све остало би знала да генерише из ротације.



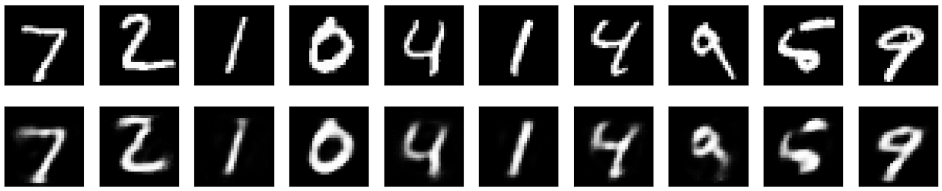
Део аутоенкодера пре кода је енкодер, а део после декодер. **Енкодер** учи да издвоји најкорисније информације из улаза, а **декодер** учи да на основу тих информација што боље реконструише улаз.

3.1 Имплементација аутоенкодера

Ово је пример најпростијег аутоенкодера имплементираног у Керасу.

```
input_img = Input(shape=(784,))
encoded = Dense(32, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)

encoder = Model(input_img, encoded)

encoded_input = Input(shape=(32,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))
```

Резултати овог кода:



У првом реду су неке од слика на којима је мрежа тренирана, а у другом реду су реконструисане слике.

Очигледно је да је мрежа научила да улазне слике које садрже 784 параметара (пиксели слике 28x28) представи коришћењем само 32 параметра свог скривеног слоја.

Ова мрежа се даље може унапредити додавањем више слојева у енкодер и у декодер. Такође се уместо густо повезаних слојева могу користити и конволуциони слојеви који би искористили чињеницу да мрежа ради са фотографијама да још више унапреде резултате.

4 Варијациони Аутоенкодер

Аутоенкодер се даље може унапредити додавањем варијационог закључивања у причу. Овај метод се своди на статистичку представу проблема. Нека је x улазни податак, а z његова скривена репрезентација, где је димензија z много мања од димензије x. Њихова заједничка расподела може да се представи као

$$p(x, z) = p(x|z)p(z)$$

Одатле се генерисање података ради тако што се прво извуче $z_i \sim p(z)$, а онда одатле и $x_i \sim p(x|z)$.

Енкодер са друге стране има за циљ да представи расподелу $p(z|x)$. По Бајесовој формули знамо

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

Проблем овде настаје због тога што је за рачунање $p(x)$ потребно решити интеграл

$$p(x) = \int p(x|z)p(z)dz$$

који у већини случајева није решив. Уместо тога се расподела $p(z|x)$ апроксимира расподелом $q_\lambda(z|x)$ где су λ параметри претпостављене расподеле. Налажење одступања наше апроксимације $q_\lambda(z|x)$ од реалне расподеле $p(z|x)$ налазимо **Кулбек-Лајблеровим** (КЛ) **растојањем**:

$$D_{KL}(q_\lambda(z|x)||p(z|x)) = E_q \log \frac{q_\lambda(z|x)}{p(z|x)}$$

(Где је E_q скраћена ознака која означава $E_q \log \frac{q(x)}{p(x)} = \sum q(x) \log \frac{q(x)}{p(x)}$ инспирисана паралелом са математичким очекивањем.)

$$= E_q \log \frac{q_\lambda(z|x)p(x)}{p(z, x)}$$

$$= E_q \log q_\lambda(z|x) - E_q \log p(z, x) + \log p(x)$$

Циљ је наћи такво λ да КЛ растојање буде минимално. Проблем је у томе што то растојање идаље није могуће израчунати зато што садржи $p(x)$. До решења се долази ако се једначина преуреди на следећи начин:

$$\log p(x) = D_{KL}(q_\lambda(z|x)||p(z|x)) + (E_q \log p(z, x) - E_q \log q_\lambda(z|x))$$

Уводи се нова ознака

$$ELBO(\lambda) = E_q \log p(z, x) - E_q \log q_\lambda(z|x)$$

Тако да малопређашња једначина сада постаје

$$\log p(x) = D_{KL}(q_\lambda(z|x)||p(z|x)) + ELBO(\lambda)$$

Пошто је $\log p(x)$ константа, а D_{KL} увек веће од нуле, да би минимизовали КЛ растојање довољно је да максимизујемо $ELBO(\lambda)$. Овиме се избегава рачунање $\log p(x)$. Сада само остаје да се израчуна $ELBO(\lambda)$.

$$ELBO(\lambda) = E_q \log p(z, x) - E_q \log q_\lambda(z|x)$$

$$= E_q \log \frac{p(z,x)}{q_\lambda(z|x)} = E_q \log \frac{p(x|z)p(z)}{q_\lambda(z|x)}$$

$$= E_q \log p(x|z) - E_q \log \frac{q_\lambda(z|x)}{p(z)}$$

$$= E_q \log p(x|z) - D_{KL}(q_\lambda(z|x)||p(z))$$

У овој једначини, $E_q \log p(x|z)$ представља стандардну функцију губитка L неуронске мреже. Остаје само да се израчуна КЛ растојање и наш модел је спреман. Овде у причу улазе варијациони аутоенкодери.

Варијациони аутоенкодер је неуронска мрежа која има исту архитектуру као аутоенкодер, али поставља неке претпоставке о дистрибуцији скривених променљиви. Користи приступ варијационог закључивања за учење скривених репрезентација модела.

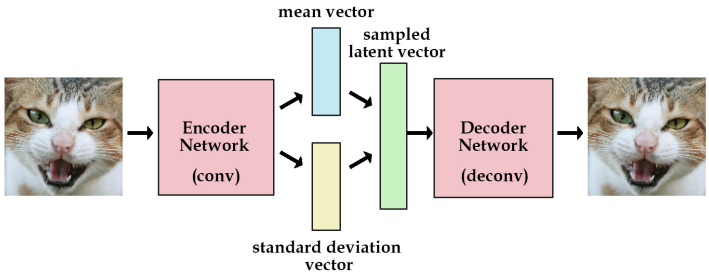
Енкодер у обичном аутоенкодеру је добар у прављењу кодова који су корисни за реконструкцију слике. Са друге стране, није могуће сампловати насумичан код и очекивати да декодер од њега направи смислени излаз. Променљиве у коду аутоенкодера не морају да буду континуалне и обично имају огромне празнине за које није везана ни једна особина података. Када би сваки део кода пратио непрекидну расподелу, било би могуће правити мале промене на коду које утичу на излазне податке на смислен начин. Ово би омогућило узимање насумичног кода који прати расподелу скривених варијабли из ког декодер може да генерише податак који има смисла.

Варијациони аутоенкодер се обично реализује тако што се пође од претпоставке да се свака скривена променљива која моделује податке понаша по нормалној расподели. Неуронална мрежа се натера да генерише скривене кодове који прате нормалну расподелу додавањем додатног ограничења на функцију губитка. Ово ограничење је Кулбек-Лајблерово (КЛ) растојање између нормалне расподеле и расподеле скривених променљиви које генерише енкодер

$$D_{KL}(q(z|x)||p(z))$$

где је $q(z|x)$ расподела генерисаних вектора скривених променљиви z, а $p(z)$ претпоставка о расподели скривених променљиви, која је у овом случају стандардна нормална расподела. Минимизовање овог растојања натераће енкодер да генерише кодове који прате нормалну расподелу.

Да би овакав модел радио, прво је потребно направити неколико малих измена на структури мреже. Уместо да енкодер избацује низ скривених променљиви, сада избацује низове параметра дистрибуције сваке променљиве. У овом случају ти параметри су очекивање и стандардна девијација нормалне расподеле. Њих даље користи да би сампловао код који прослеђује декодеру.



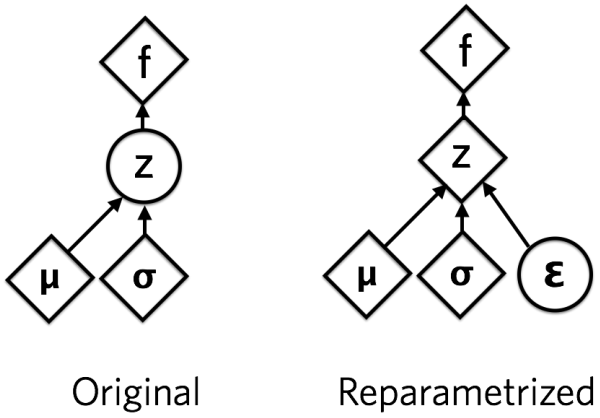
Оваква архитектура мреже поставља проблем за коришћење алгоритма пропагације уназад. Налажење парцијалних извода функције губитка у односу на тежине енкодера се отежава чињеницом да смо увели насумичан фактор у израчунавање, а он нема извод.

Овај проблем се решава **триком репараметризације**, који представља скривени вектор z на другачији начин.

$$z = \mu + \sigma \odot \epsilon$$

Где је μ очекивање расподеле, σ стандарна девијација а ϵ случајна величина

$$\epsilon \sim N(0, 1)$$



Овиме је дефинисана функција која од параметара зависи детерминистички. Дакле, могуће је наћи њен извод у односу на параметре μ и σ.

Остаје само да се $D_{KL}(q(z|x)||p(z))$ изрази преко μ и σ.

$$D_{KL}(q(z|x)||p(z)) = \sum_{i=0}^{n}(\sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1)$$

Овиме је завршено рачунање функције губитка за варијациони аутоенкодер.

4.1 β - варијациони Аутоенкодер

Иако је варијациони аутоенкодер добар у проналажењу параметара који моделују податаке, може се десити да нађени параметри буду замршени. На пример, ако мрежа учи да генерише слике људског лица, може се десити да нађе да је један од параметара ротација лица, али да се померањем овог параметра мења и боја очију. Један од могућих узрока оваквог понашања била би ситуација где је у тренинг сету сасвим случајно већина лица окренутих на десно плавоока. Мрежа ће претпоставити да је ово правилност, иако то и не мора да буде. Овај проблем се решава тако што се КЛ растојање у функцији губитка помножи неким новим коефицијентом β који се унапред фиксира.

Ако се параметар постави на $\beta = 0$, мрежа ће се понашати као обичан аутоенкодер. Ако се стави да је $\beta = 1$, понашаће се као обичан варијациони аутоенкодер.

За $\beta > 1$ мрежи се поставља већа битност на налажење смислених и независних параметара модела а мању на прецизно реконструисање податка. Ово се објашњава чињеницом да $N(0, 1)$ у матрици коваријансе има нуле свуда ван главне дијагонале.

Тражењем од кодова да прате ову расподелу форсира се да њихови елементи немају међусобну коваријансу. Очигледно је да ни превелики параметар β није добар по модел зато што се губи квалитет генерисаних података.

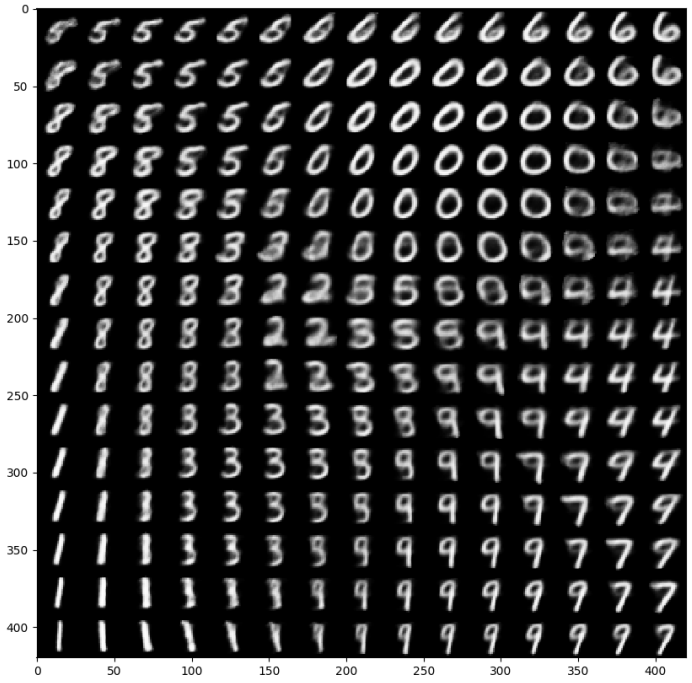
4.2 Имплементација β-варијационог аутоенкодера

Ово је пример имплементације варијационог аутоенкодера у Керасу.

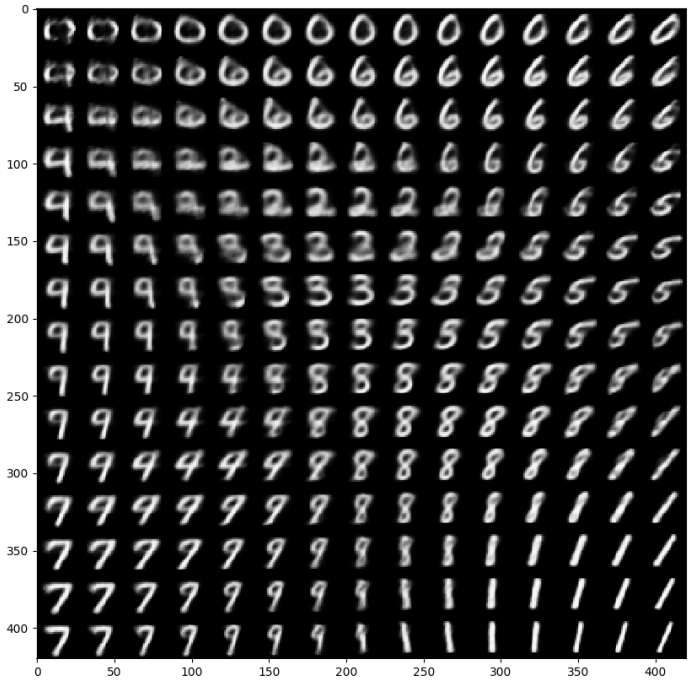
```
batch_size = 100
original_dim = 784
latent_dim = 2
intermediate_dim = 256
epochs = 100
epsilon_std = 1.0
beta = 2

x = Input(batch_shape=(batch_size, original_dim))
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_sigma = Dense(latent_dim)(h)

def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=(batch_size, latent_dim),
                              mean=0., stddev=epsilon_std)
    return z_mean + K.exp(z_log_sigma) * epsilon

z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_sigma])

decoder_h = Dense(intermediate_dim, activation='relu')
decoder_mean = Dense(original_dim, activation='sigmoid')
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)

vae = Model(x, x_decoded_mean)

encoder = Model(x, z_mean)

decoder_input = Input(shape=(latent_dim,))
_h_decoded = decoder_h(decoder_input)
_x_decoded_mean = decoder_mean(_h_decoded)
generator = Model(decoder_input, _x_decoded_mean)

def vae_loss(x, x_decoded_mean):
    xent_loss = original_dim * metrics.binary_crossentropy(x, x_decoded_mean)
    kl_loss = - 0.5 * K.mean(1 + z_log_sigma - K.square(z_mean) - K.exp(z_log_sigma), axis=-1) * beta
    return K.mean(xent_loss + kl_loss)
```

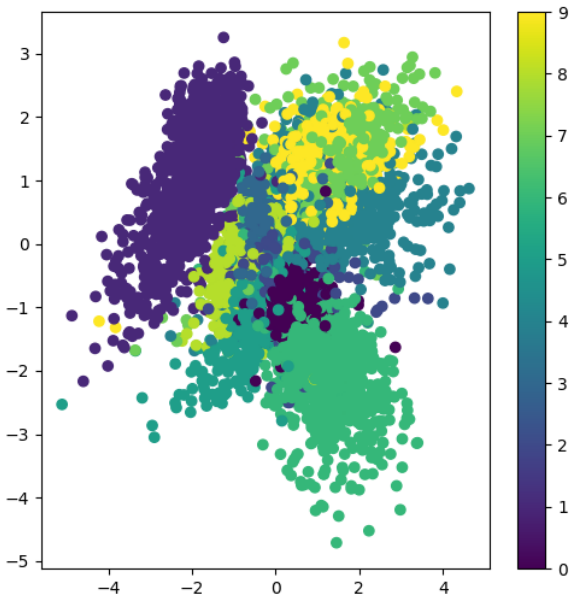
Ова мрежа учи да генерише цифре на основу само две променљиве у скривеном слоју. Посебну пажњу треба обратити на функцију vae_loss, која представља функцију губитка за ову мрежу. У њој се види имплементација варијационог закључивања и сама суштина варијационих аутоенкодера.

Чињеница да су узете само две промењиве за скривени слој дозвољава да се научена репрезентација цифара представи у две димензије. Овај приказ се налази на наредној страни.

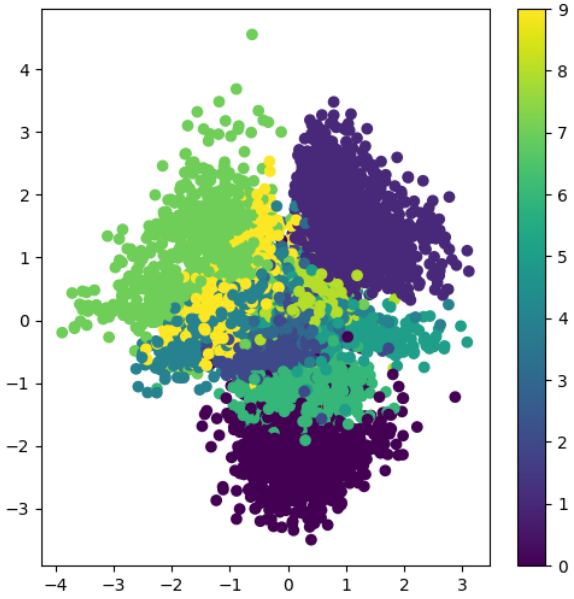


Изнад је резултат за $\beta = 1$ а испод за $\beta = 2$.





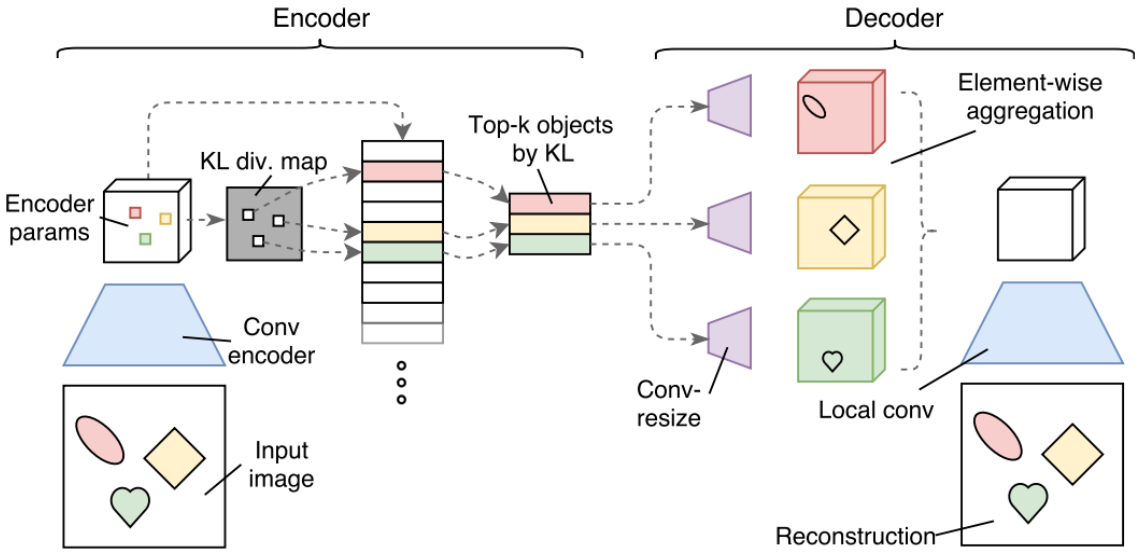
Дводимензионална структура података такође омогућава визуалну репрезентацију расподеле различитих класа. Осе датог графика представљају две променљиве најужег слоја. Сваки од ових обојених кластера представља једну цифру. Кластери који су близу једни другима на овом графику представљају цифре сличне визуалне структуре. На овој страни је дат пар графика ових кластера за примере са претходне стране.



5 Мулти-ентитетски варијациони аутоенкодер

Проблем препознавања и генерисања цифара је проблем који се бави само једним објектом на фотографији. Већина проблема у стварном свету се састоји од више различитих ентитета. У природи људског мозга је да свет око себе посматра објектно оријентисано. Представа о објектима се у машинском учењу углавном решава инжењерски, тако што се неке представе о објектима мануално кодирају. Истраживање у области неуралних мрежа би се значајно убрзало кад би се мреже саме научиле да податке представљају као више објеката организованих у хијерархијске структуре.

Модел за решавање овог проблема предложен је у раду The Multi-Entity Variational Autoencoder, Charlie Nash et al.[1]



У овом моделу енкодер се тренира да избацује велике количине кандидат-објеката од којих се узима K најбољих који се прослеђују декодеру за одвојену реконструкцију. Након тога се реконструисани објекти поново спајају у једну слику. Кандидат-објекти се налазе тако што улаз прође кроз конволуциону мрежу. Свака просторна локација излаза се третира као могући објекат. За сваки кандидат објекат се рачуна КЛ растојање од $N(0, 1)$ и узима се K објеката са највећим растојањем. Ово ради зато што ће конволуциона мрежа морати да сачува највише информација тамо где се налазе објекти, а неће имати потребе да чува информације на пољима на којима се налази само позадина.

6 Закључак

Овај рад је описао неке теоријске основе за машинско учење, као и објаснио варијационо закључивање и његову корист у генеративном моделовању и описао неке од могућих примена. Варијациони аутоенкодери су област која идаље мучи многе који покушшају да јој приђу. Главни узрок за ово је то што се у њиховом дефинисању мешају многи појмови из статистике као и из машинског учења, што уме веома често да буде конфузно. Ја се надам да сам успео што боље да објасним инспирацију иза оваквих модела, као и њихову имплементацију и примену.

За крај, желим да изразим захвалност својим менторима, Јелени Хаџи-Пурић и Петру Величковићу, за издвојено време и стрпљење без ког вероватно не бих имао довољно разумевање неких идеја о којима сам писао у овом раду. Такође, без Петрових предавања о машинском учењу на Недељи информатике, никад не бих увидео колико је ова област заправо занимљива и вероватно би још увек мислио да је превише компликована и досадна. Уз то, хтео бих да се захвалим и свим професорима информатичких предмета који су ми предавали у Математичкој гимназији, без којих би мој пут ка знању био само безцилно лутање.

7 Литература

[1] http://charlienash.github.io/assets/docs/mevae2017.pdf

[2] https://en.wikipedia.org/wiki/Machine_learning

[3] https://en.wikipedia.org/wiki/Autoencoder

[4] https://en.wikipedia.org/wiki/Artificial_neural_network

[5] https://cambridgespark.com/content/tutorials/deep-learning-for-complete-beginners-recognising-handwritten-digits/index.html

[6] https://blog.keras.io/building-autoencoders-in-keras.html

[7] https://github.com/cazala/mnist