

APPLICATIONS



OF DATA SCIENCE

Intro to Autoencoders

Applications of Data Science - Class 19

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2022-05-30

APPLICATIONS



OF DATA SCIENCE

Stacked Autoencoders (SAE)

(Heavily inspired by Geron 2019)

APPLICATIONS

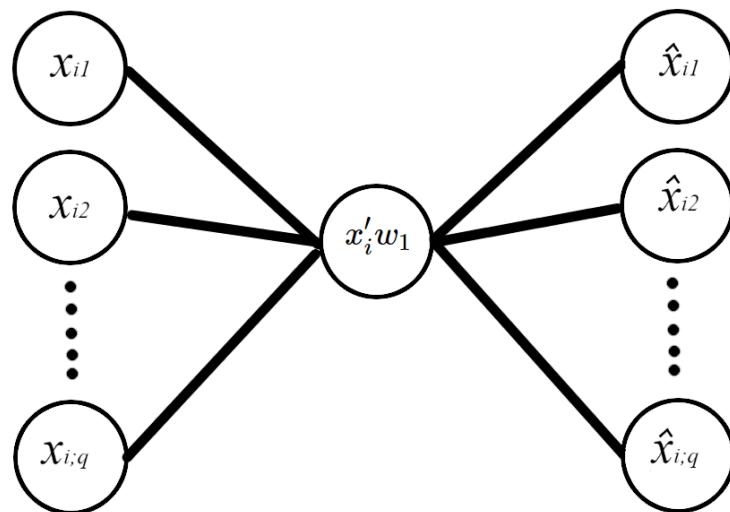


OF DATA SCIENCE

Remember me? 😢

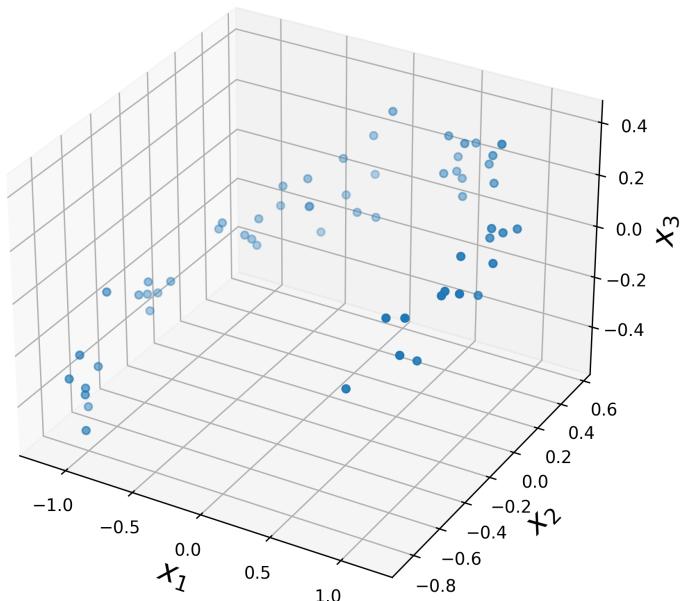
$$\max_{w_1} w_1' X' X w_1 \text{ s.t. } \|w_1\| = 1$$

PCA is a linear encoder. Sort of.



NN "PCA"

```
x_train = generate_3d_data(60)  
x_train = x_train - x_train.mean(axis=0, keepdims=0)
```



```
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=2)  
pca.fit(X_train)
```

```
## PCA(n_components=2)
```

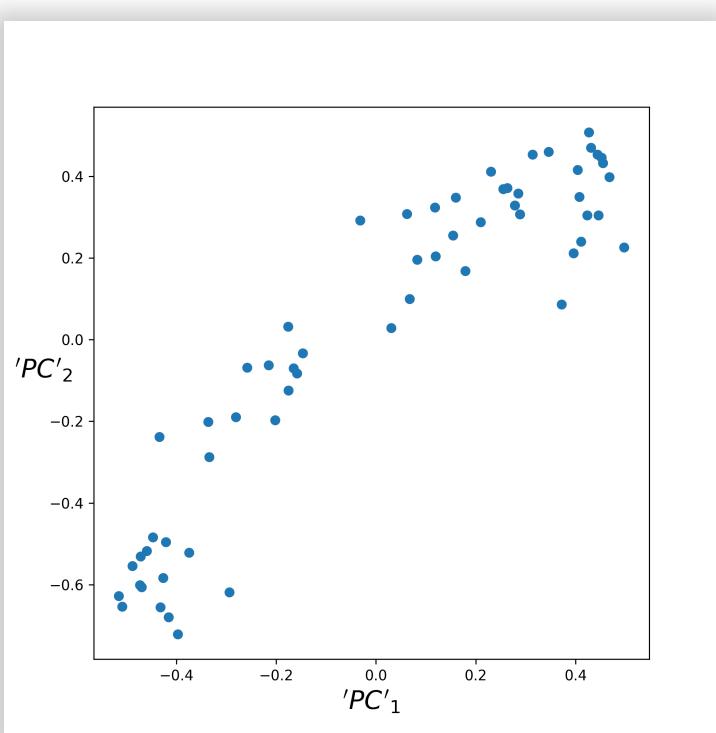
```
from tensorflow.keras import Sequential  
from tensorflow.keras.layers import Dense, Flatten, Reshape  
  
encoder = Sequential([Dense(2, input_shape=[3])])  
decoder = Sequential([Dense(3, input_shape=[2])])  
autoencoder = Sequential([encoder, decoder])  
  
autoencoder.compile(loss='mse', optimizer='adam')  
  
history = autoencoder.fit(X_train, X_train, epochs=20, verbose=0)
```

```
codings = encoder.predict(X_train)  
pcs = pca.transform(X_train)  
  
print(f'codings shape: {codings.shape}, pcs shape: {pcs.shape}')
```

```
## codings shape: (60, 2), pcs shape: (60, 2)
```

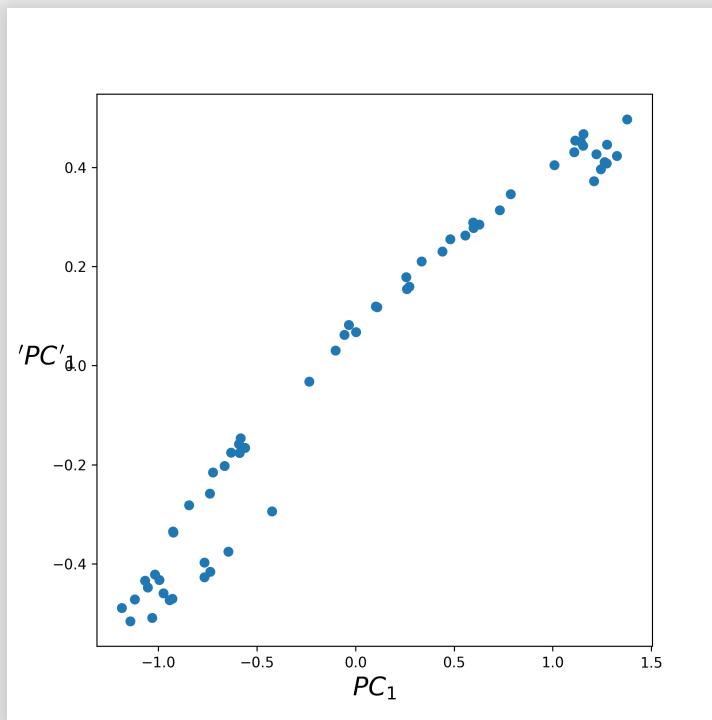
How did NN "PCA" do?

```
plt.scatter(codings[:, 0], codings[:, 1])
plt.xlabel("$'PC'_1$", fontsize=18)
plt.ylabel("$'PC'_2$", fontsize=18, rotation=0)
plt.show()
```



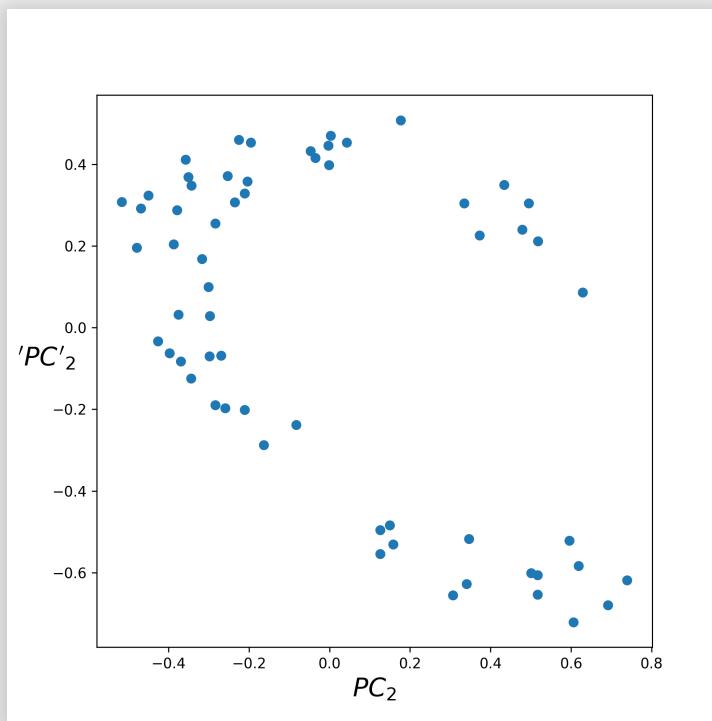
Is it PCA though?

```
plt.scatter(pcs[:, 0], codings[:, 0])
plt.show()
```



Is it PCA though?

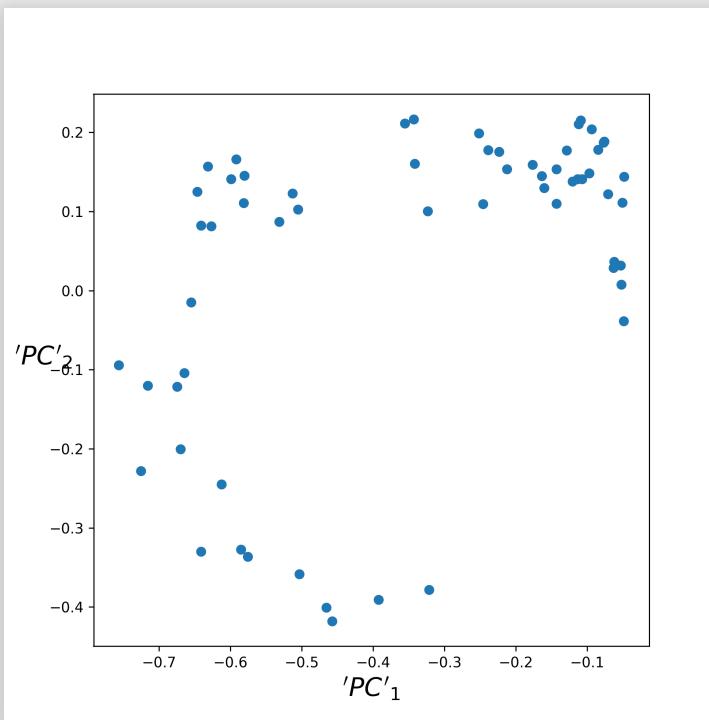
```
plt.scatter(pcs[:, 1], codings[:, 1])
plt.show()
```



Can you make it PCA?

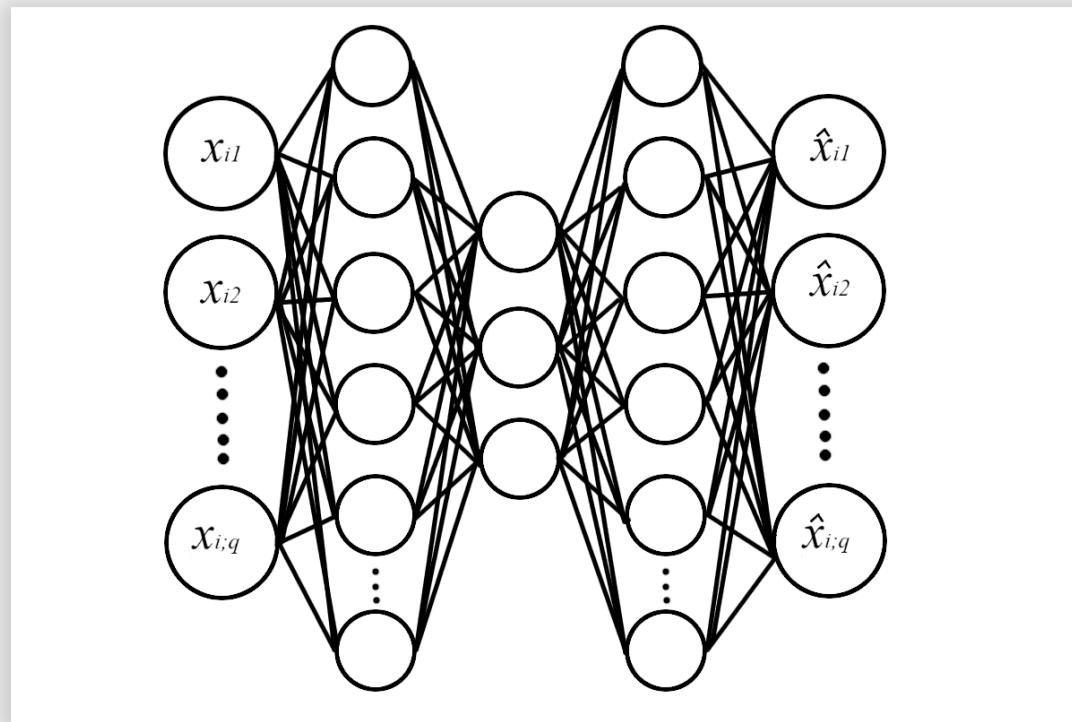
Will non-linearity help?

```
encoder = Sequential([Dense(20, input_shape=[3], activation='relu',
                           Dense(2))
decoder = Sequential([Dense(20, input_shape=[2], activation='relu',
                           Dense(3)])
autoencoder = Sequential([encoder, decoder])
autoencoder.compile(loss='mse', optimizer='adam')
history = autoencoder.fit(X_train, X_train, epochs=20, verbose=0)
```



Stacked Autoencoders

Stacked autoencoders have been around for NLPCA for over 30 years (see e.g. [Kramer 1991](#)).



SAE on FNIST

```
from tensorflow.keras.datasets import fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = X_train.astype(np.float32) / 255
X_test = X_test.astype(np.float32) / 255

stacked_encoder = Sequential([
    Flatten(input_shape=[28, 28]),
    Dense(100, activation="relu"),
    Dense(30, activation="relu"),
])
stacked_decoder = Sequential([
    Dense(100, activation="relu", input_shape=[30]),
    Dense(28 * 28, activation="sigmoid"), # make output 0-1
    Reshape([28, 28])
])
stacked_ae = Sequential([stacked_encoder, stacked_decoder])

stacked_ae.compile(loss='mse', optimizer='adam')

history = stacked_ae.fit(X_train, X_train, epochs=20, verbose=1)
```

Can it reconstruct?

```
def show_reconstructions(sae, images, n_images=5):
    reconstructions = sae.predict(images[:n_images])
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plt.imshow(images[image_index], cmap='binary')
        plt.axis('off')
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plt.imshow(reconstructions[image_index], cmap='binary')
        plt.axis('off')

show_reconstructions(stacked_ae, X_test)
plt.show()
```



Can it "denoise"?

```
shoe_encoded = stacked_encoder.predict(X_test[np.newaxis, 0,:])  
shoe_encoded  
  
## array([[ 2.92097   ,  10.338918  ,  0.          ,  1.4428661 ,  2.6388376 ,  
##          2.3139865 ,  2.5651445 ,  3.1734047 ,  3.2220273 ,  4.8254766 ,  
##          3.1547084 ,  3.3593066 ,  1.6458672 ,  2.6469245 ,  4.6263247 ,  
##          3.5413108 ,  1.239067  ,  9.445068  ,  0.75500596,  5.3597875 ,  
##          0.          ,  4.605066  ,  10.533057 ,  4.6313896 ,  6.31067 ,  
##          3.82158   ,  6.353969  ,  3.82279   ,  3.3340364 ,  0.9221499 ]  
##          dtype=float32)  
  
shoe1 = stacked_decoder.predict(shoe_encoded) [0]  
  
## WARNING:tensorflow:5 out of the last 7 calls to <function Model.make_p  
  
shoe2 = stacked_decoder.predict(shoe_encoded + np.random.normal(sca  
shoe3 = stacked_decoder.predict(shoe_encoded + np.random.normal(sca
```

```
fig, axes = plt.subplots(1, 3, figsize=(6, 3))
axes[0].imshow(shoe1, cmap="binary")
axes[1].imshow(shoe2, cmap="binary")
axes[2].imshow(shoe3, cmap="binary")
_ = axes[0].axis('off')
_ = axes[1].axis('off')
_ = axes[2].axis('off')
plt.show()
```



Can it average?

```
shirt_encoded = stacked_encoder.predict(X_test[np.newaxis, 1,:])
mean_encoded = (shoe_encoded + shirt_encoded) / 2

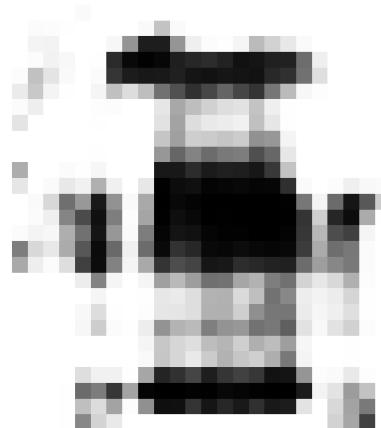
fig, axes = plt.subplots(1, 3, figsize=(6, 3))
_ = axes[0].imshow(X_test[0,:], cmap="binary")
_ = axes[1].imshow(X_test[1,:], cmap="binary")
_ = axes[2].imshow(stacked_decoder.predict(mean_encoded)[0], cmap="binary")
_ = axes[0].axis('off')
_ = axes[1].axis('off')
_ = axes[2].axis('off')
plt.show()
```



Can it generate?

```
X_test_compressed = stacked_encoder.predict(X_test)
mins = X_test_compressed.min(axis=0)
maxs = X_test_compressed.max(axis=0)

_ = plt.imshow(stacked_decoder.predict(
    np.random.uniform(mins, maxs, size=30) [np.newaxis, :]
) [0], cmap="binary")
_ = plt.axis('off')
plt.show()
```



Variational Autoencoders (VAE)

APPLICATIONS

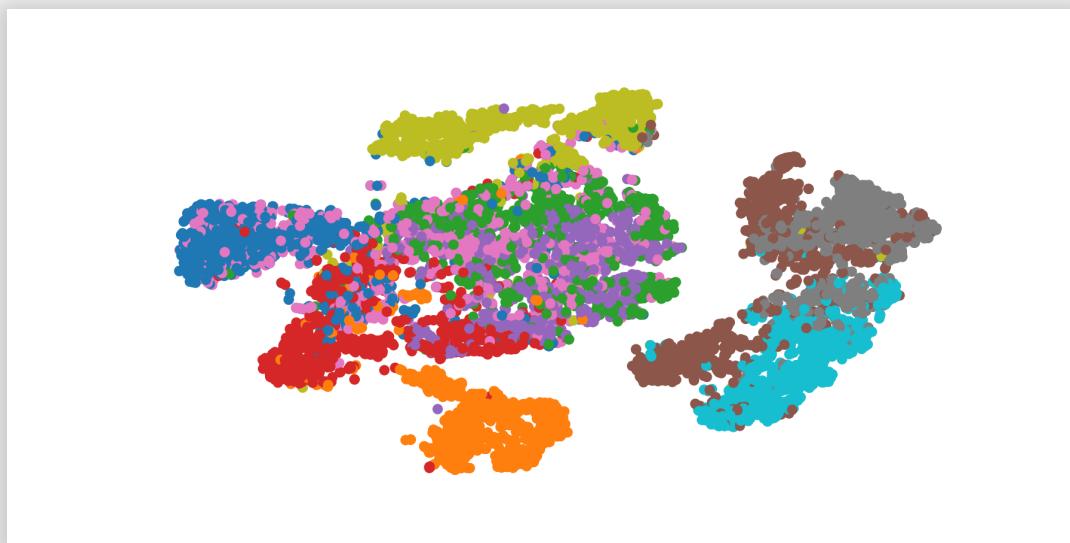


OF DATA SCIENCE

How does SAE latent space look like?

```
from sklearn.manifold import TSNE

tsne = TSNE()
X_test_2D = tsne.fit_transform(X_test_compressed)
X_test_2D = (X_test_2D - X_test_2D.min()) / (X_test_2D.max() - X_t
    _ = plt.scatter(X_test_2D[:, 0], X_test_2D[:, 1], c=y_test, s=10,
    _ = plt.axis('off')
plt.show()
```



Road to VAE

- The latent space has many "holes", it is "irregular"
- If we sample from these areas and decode, we cannot expect to get a meaningful element in the input space, result would be **unlikely**
- And why should we? Reconstruction loss is "king", it is overfitting! We need to regularize.



Road to VAE

In a landmark paper, [Kingma & Welling \(2013\)](#) suggest:

- Suppose there's a latent variable (LV) z
- The **probabilistic** encoder is defined by $p(z|x)$
- The **probabilistic** decoder is defined by $p(x|z)$
- Make $p(z)$, the prior of the LV, $N(0, I)$ - so the posterior encoder $p(z|x)$ cannot "run away"
- Make $p(x|z)$ be $N(f_\theta(z), \sigma^2 I)$ where f would be modeled by DNN
- Goal: make the x 's produced by the decoder **likely**, i.e. maximize the marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

But what is the posterior $p(z|x)$?

Variational Inference (VI)

- A technique to approximate complex distributions (posteriors)
- Set a parametrised family of distribution $q(z)$ (for example the family of Gaussians)
- Look for the best approximation of our target distribution among this family: $q(z) \approx p(z|x)$
- The best element in the family is one that minimizes:

$$D_{KL}(q(z)||p(z|x)) = \int q(z) \log \frac{q(z)}{p(z|x)}$$

- Mark q as $q(z|x)$, so that this "family" would be $N(g_\eta(x), h_\psi(x))$
- This is how it will depend on the data x , and g, h will be found with the encoder network, minimizing the KL loss



Our encoder finds means and variances for a distribution! When we have them we would sample from that distribution.

Now dig this:

$$\begin{aligned} D_{KL}(q(z|x)||p(z|x)) &= \int q(z|x) \log \frac{q(z|x)}{p(z|x)} \\ &= E_q[\log(q(z|x)) - \log(p(z|x))] \\ &= E_q[\log(q(z|x)) - \log \frac{p(z)p(x|z)}{p(x)}] \\ &= E_q[\log(q(z|x)) - \log(p(z))] \\ &\quad - E_q[\log(p(x|z))] + \log(p(x)) \\ &= D_{KL}(q(z|x)||p(z)) - E_q[\log(p(x|z))] + \log(p(x)) \end{aligned}$$

But we want to maximize $p(x)$!

When writing:

$$\log(p(x)) - D_{KL}(q(z|x)||p(z|x)) = E_q[\log(p(x|z))] - D_{KL}(q(z|x)||p(z))$$

we see that maximizing log-likelihood $\log(p(x))$ combined with minimizing KL-divergence $D_{KL}(q(z|x)||p(z|x))$ boils down to maximizing the RHS or maximizing the **Evidence Lower BOund**:

$$ELBO(\theta, \eta, \psi | X) = E_q[\log(p(x|z))] - D_{KL}(q(z|x)||p(z))$$

Which is called like this since $D_{KL} \geq 0$ always, so
 $\log(p(x)) \geq ELBO$.

Turning this to a loss function we need to minimize:

$$-ELBO(\theta_f, \theta_g, \theta_h | X) = -E_q[\log(p(x|z))] + D_{KL}(q(z|x)||p(z))$$

Almost there (I)

Fortunately D_{KL} for two Gaussians has closed form:

For $p_1 = N(\mu_1, \Sigma_1)$ and $p_2 = N(\mu_2, \Sigma_2)$:

$$D_{KL}(p_1 \parallel p_2) = \frac{1}{2} \left[\log \frac{|\Sigma_2|}{|\Sigma_1|} - n + \text{tr} \Sigma_2^{-1} \Sigma_1 + (\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) \right]$$

In our case $\mu_1 = g_\eta(x) = \mu$, $\Sigma_1 = h_\psi(x) = \Sigma$, where Σ is diagonal with σ_i^2 on its diagonal, $\mu_2 = \vec{0}$, $\Sigma_2 = I$, so:

$$D_{KL}(q(z|x) \parallel p(z)) = -\frac{1}{2} \sum_i [1 + \gamma_i - \exp(\gamma_i) - \mu_i^2]$$

where $\gamma_i = \log \sigma_i^2$ for better stability. See [here](#) for a full proof.

Here i goes over number of neurons in the latent layer, D .

Almost there (II)

Since we assumed $p(x|z)$ is $N(f_\theta(z), \sigma^2 I)$:

$$-E_q[\log(p(x|z))] = C \cdot E_q[||x - f_\theta(z)||^2] = C \cdot E_q[||x - \hat{x}||^2],$$

where C is some constant dependent on σ^2 .

We use standard mini-batch SGD for approximation that expectation.

So our final loss optimized with SGD, is the sum of MSE and KL:

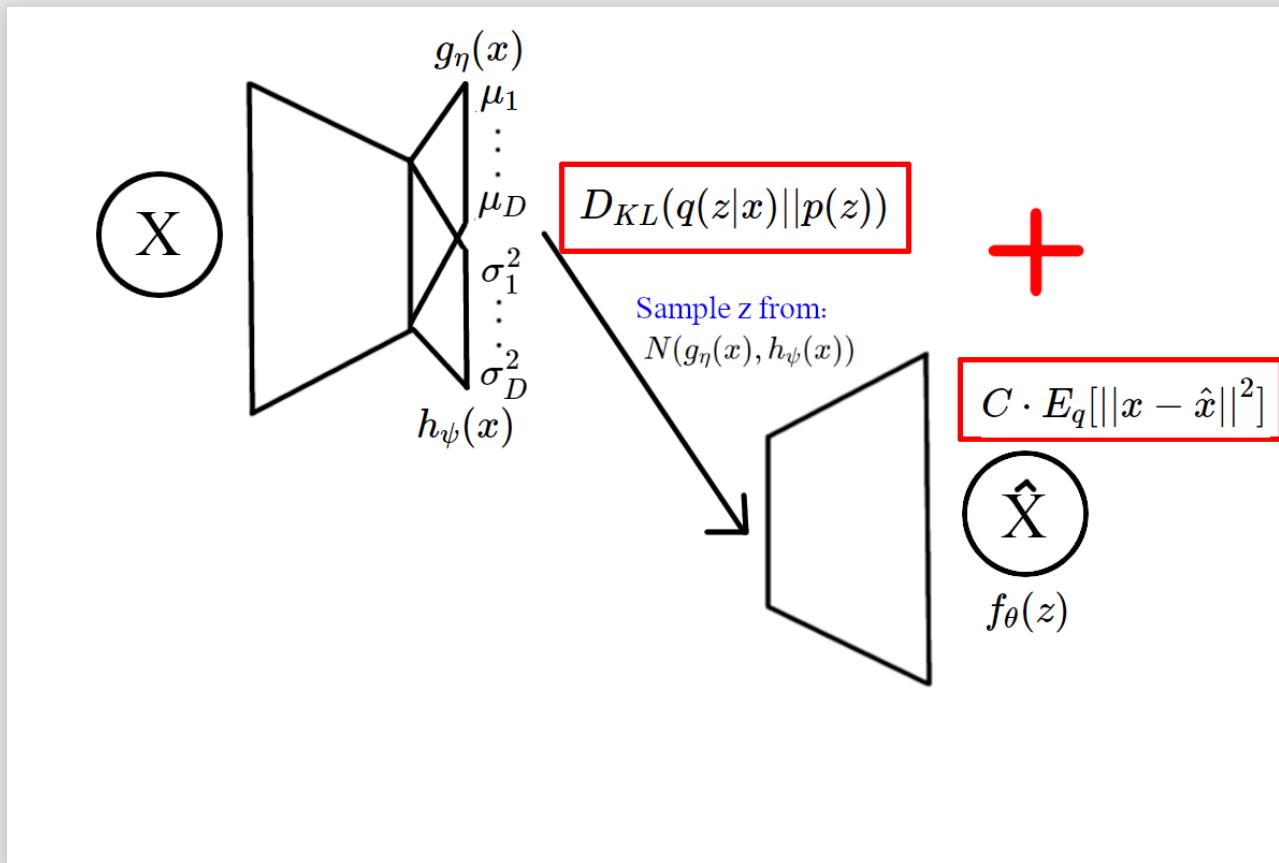
VAE-Loss($\theta, \eta, \psi | X$)

$$= C \cdot \sum_{batch} (x_{batch} - \hat{x}_{batch})^2 - \frac{1}{2} \sum_i [1 + \gamma_i - \exp(\gamma_i) - \mu_i^2]$$



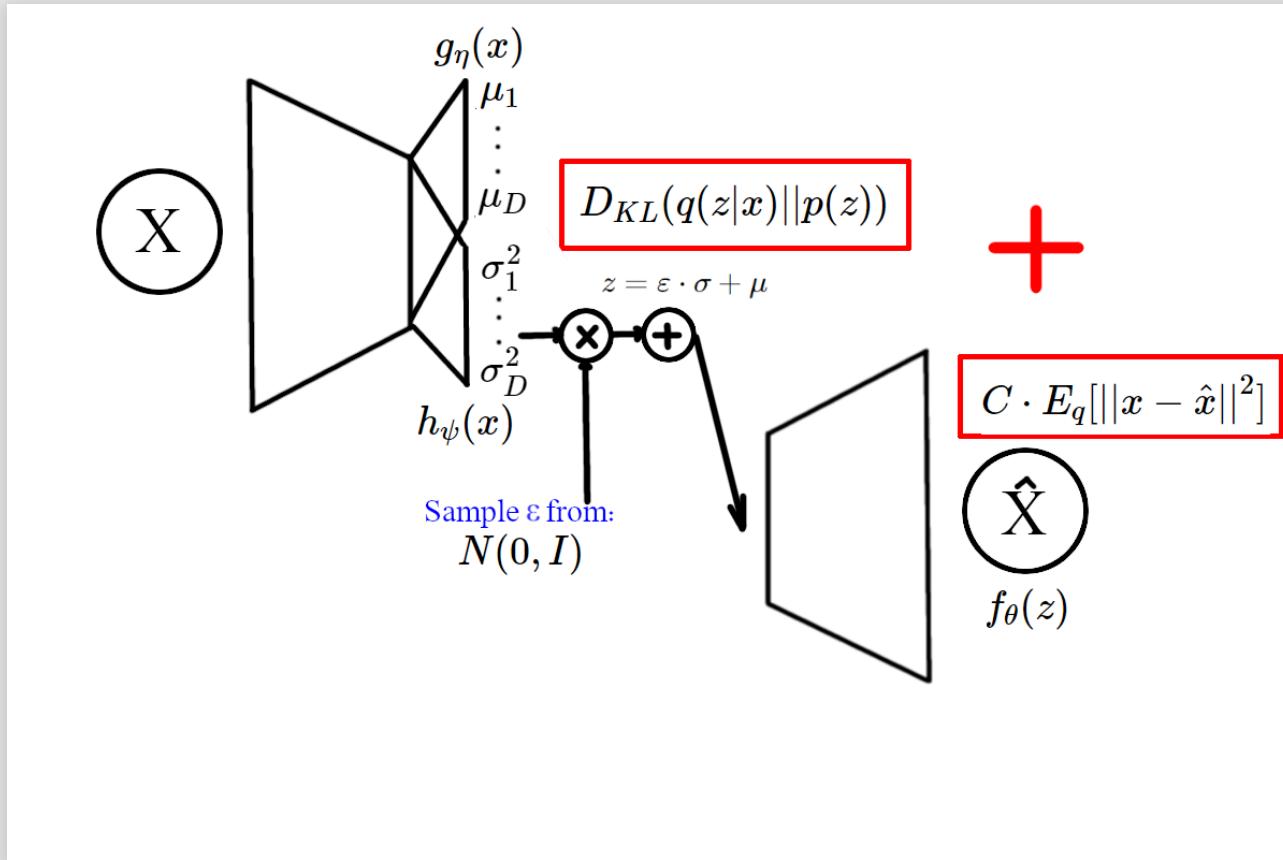
Reconstruction loss + Regularization!

Still not there



💡 Why wouldn't this work?

Reparametrisation trick



In terms of $\gamma = \log \sigma^2$, this means: $z = \varepsilon \cdot \frac{\exp(\gamma)}{2} + \mu$

VAE in Keras

```
from tensorflow.keras.layers import Layer, Input
from tensorflow.keras.losses import mean_squared_error as mse
from tensorflow.keras import Model
import tensorflow.keras.backend as K

# need a sampling layer which takes \mu and \gamma, samples \varphi
# and turns it to N(\mu, "\gamma")
class Sampling(Layer):
    def call(self, inputs):
        mean, log_var = inputs
        return K.random_normal(tf.shape(log_var)) * K.exp(log_var) / D = 30

inputs = Input(shape=[28, 28])
z = Flatten()(inputs)
z = Dense(150, activation='relu')(z)
z = Dense(100, activation='relu')(z)
codings_mean = Dense(D)(z)
codings_log_var = Dense(D)(z)
codings = Sampling()([codings_mean, codings_log_var])
variational_encoder = Model(
    inputs=[inputs], outputs=[codings_mean, codings_log_var, codir
)
```

```

decoder_inputs = Input(shape=[D])
x = Dense(100, activation='relu')(decoder_inputs)
x = Dense(150, activation='relu')(x)
x = Dense(28 * 28, activation='sigmoid')(x) # make output 0-1
outputs = Reshape([28, 28])(x)
variational_decoder = Model(inputs=[decoder_inputs], outputs=[outputs])

_, _, codings = variational_encoder(inputs)
reconstructions = variational_decoder(codings)
variational_ae = Model(inputs=[inputs], outputs=[reconstructions])

kl_loss = -0.5 * K.sum(
    1 + codings_log_var - K.exp(codings_log_var) - K.square(codings_
axis=-1))

# if inputs/outputs were flat I think it would work
# reconstruction_loss = mse(inputs, outputs) * 784
# vae_loss = K.mean(reconstruction_loss + kl_loss)
# variational_ae.add_loss(vae_loss)

variational_ae.add_loss(K.mean(kl_loss) / 784.)
variational_ae.compile(loss='mse', optimizer='adam')

history = variational_ae.fit(X_train, X_train, epochs=10, verbose=1)

```

Can it reconstruct?

```
show_reconstructions(variational_ae, X_test)  
plt.show()
```



Do you see the difference in comparison to SAE?

Can it "denoise"?

```
shoe_encoded = variational_encoder.predict(X_test[np.newaxis, 0,:])
shoe1 = variational_decoder.predict(shoe_encoded)[0]
shoe2 = variational_decoder.predict(shoe_encoded + np.random.normal(0, 0.1, shoe_encoded.shape))
shoe3 = variational_decoder.predict(shoe_encoded + np.random.normal(0, 0.5, shoe_encoded.shape))

fig, axes = plt.subplots(1, 3, figsize=(6, 3))
_ = axes[0].imshow(shoe1, cmap='binary')
_ = axes[1].imshow(shoe2, cmap='binary')
_ = axes[2].imshow(shoe3, cmap='binary')
_ = axes[0].axis('off')
_ = axes[1].axis('off')
_ = axes[2].axis('off')
plt.show()
```

Can it average?

```
shirt_encoded = variational_encoder.predict(X_test[np.newaxis, 1, :]
mean_encoded = (shoe_encoded + shirt_encoded) / 2

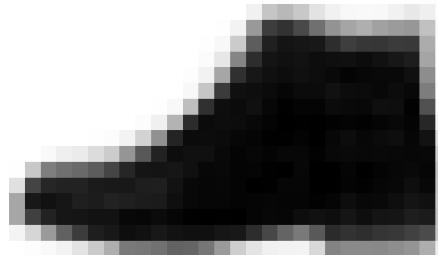
fig, axes = plt.subplots(1, 3, figsize=(6, 3))
_ = axes[0].imshow(X_test[0, :], cmap='binary')
_ = axes[1].imshow(X_test[1, :], cmap='binary')
_ = axes[2].imshow(variational_decoder.predict(mean_encoded)[0], cmap='binary')
_ = axes[0].axis('off')
_ = axes[1].axis('off')
_ = axes[2].axis('off')
plt.show()
```



Can it generate?

```
X_test_compressed = variational_encoder.predict(X_test) [0]
mins = X_test_compressed.min(axis=0)
maxs = X_test_compressed.max(axis=0)

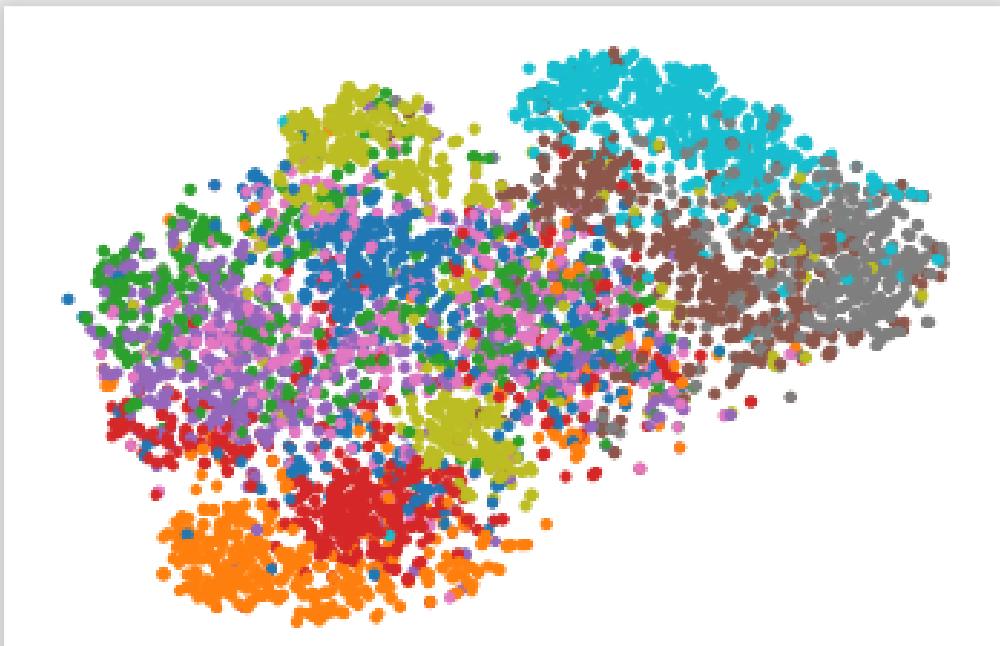
_ = plt.imshow(variational_decoder.predict(
    np.random.uniform(mins, maxs, size=D) [np.newaxis, :])
    ) [0], cmap='binary')
_ = plt.axis('off')
plt.show()
```



Latent space slightly more regular

```
from sklearn.manifold import TSNE

tsne = TSNE()
X_test_2D = tsne.fit_transform(X_test_compressed)
X_test_2D = (X_test_2D - X_test_2D.min()) / (X_test_2D.max() - X_t
    _ = plt.scatter(X_test_2D[:, 0], X_test_2D[:, 1], c=y_test, s=10,
    _ = plt.axis('off')
plt.show()
```

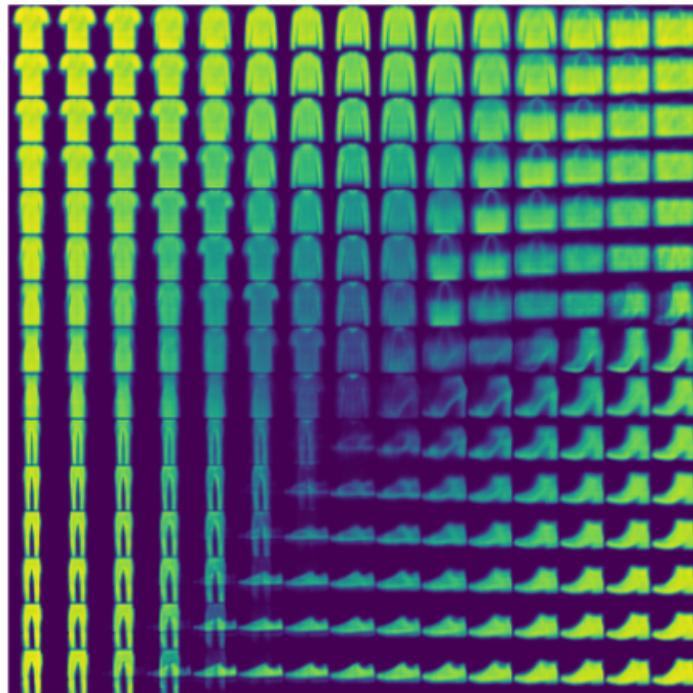


For a 2D latent space we could actual "travel" within the space:

```
n = 15 # figure with 15x15 items
item_size = 28
figure = np.zeros((item_size * n, item_size * n))
# We will sample n points within [-2, 2]
grid_x = np.linspace(-2, 2, n)
grid_y = np.linspace(-2, 2, n)

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        x_decoded = variational_decoder.predict(z_sample)
        item = x_decoded[0]#.reshape(item_size, item_size)
        figure[i * item_size: (i + 1) * item_size,
               j * item_size: (j + 1) * item_size] = item

plt.figure(figsize=(10, 10))
plt.imshow(figure)
plt.axis('off')
plt.show()
```



What's next?

GANs!



Source: [Karras 2017]

Source: [Karras, ICLR 2018](#)