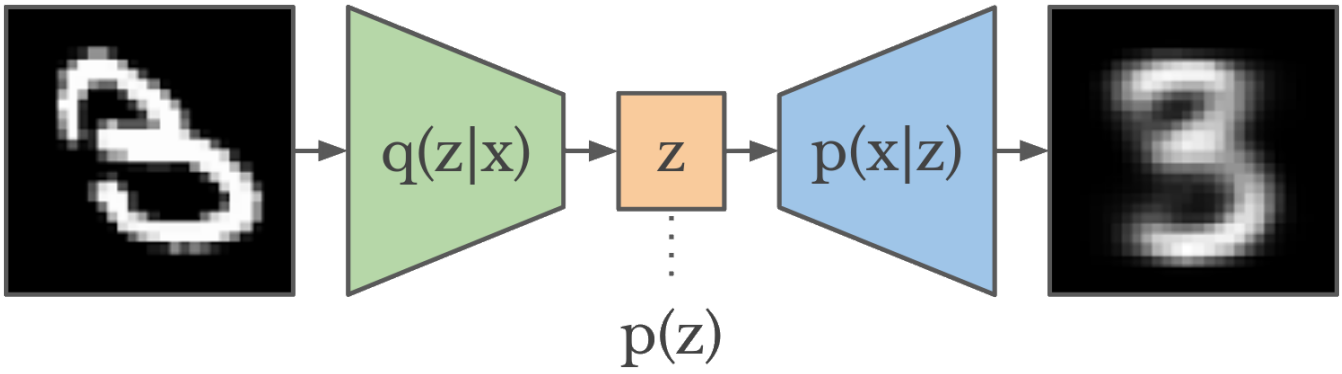
Building Variational Auto-Encoders in TensorFlow

Variational Auto-Encoders (VAEs) are powerful models for learning low-dimensional representations of your data. TensorFlow's distributions package provides an easy way to implement different kinds of VAEs.

In this post, I will walk you through the steps for training a simple VAE on MNIST, focusing mainly on the implementation. Please take a look at Kevin Frans' post for a higher-level overview.

Defining the Network

A VAE consist of three components: an encoder $q(z|x)$, a prior $p(z)$, and a decoder $p(x|z)$.



The encoder maps an image to a proposed distribution over plausible codes for that image. This distribution is also called the *posterior*, since it reflects our belief of what the code should be for (i.e. after seeing) a given image.

```
import tensorflow as tf
tfd = tf.contrib.distributions

def make_encoder(data, code_size):
  x = tf.layers.flatten(data)
  x = tf.layers.dense(x, 200, tf.nn.relu)
  x = tf.layers.dense(x, 200, tf.nn.relu)
  loc = tf.layers.dense(x, code_size)
  scale = tf.layers.dense(x, code_size, tf.nn.softplus)
  return tfd.MultivariateNormalDiag(loc, scale)
```

The prior is fixed and defines what distribution of codes we would expect. This provides a soft restriction on what codes the VAE can use. It is often just a Normal distribution with zero mean and unit variance.

```
def make_prior(code_size):
  loc = tf.zeros(code_size)
  scale = tf.ones(code_size)
  return tfd.MultivariateNormalDiag(loc, scale)
```

The decoder takes a code and maps it back to a distribution of images that are plausible for the code. It allows us to reconstruct images, or to generate new images for any code we choose.

```
import numpy as np

def make_decoder(code, data_shape):
  x = code
  x = tf.layers.dense(x, 200, tf.nn.relu)
  x = tf.layers.dense(x, 200, tf.nn.relu)
  logit = tf.layers.dense(x, np.prod(data_shape))
  logit = tf.reshape(logit, [-1] + data_shape)
  return tfd.Independent(tfd.Bernoulli(logit), 2)
```

Here, we use a Bernoulli distribution for the data, modeling pixels as binary values. Depending on the type and domain of your data, you may want to model it in a different way, for example again as a Normal distribution.

The `tfd.Independent(..., 2)` tells TensorFlow that the inner two dimensions, width and height in our case, belong to the same data point, even though they have independent parameters. This allows us to evaluate the probability of an image under the distribution, not just individual pixels.

Reusing Model Parts

We would like to use the decoder network twice, for computing the reconstruction loss described in the next section, as well as to decoder some randomly sampled codes for visualization.

In TensorFlow, if you call a network function twice, it will create two separate networks. TensorFlow templates allow you to wrap a function so that multiple calls to it will reuse the same network parameters.

```
make_encoder = tf.make_template('encoder', make_encoder)
make_decoder = tf.make_template('decoder', make_decoder)
```

The prior has no trainable parameters, so we do not need to wrap it into a template.

Defining the Loss

We would like to find the network parameters that assign the highest likelihood to our data set. However, the likelihood of a data point depends on the best code for it, which we don’t know during training.

Instead, we train the model using the evidence lower bound (ELBO), an approximation to the data likelihood.

$$\ln p(x) \geq \mathbb{E}_{q(z|x)}[\ln p(x|z)] - D_{KL}[q(z|x)||p(z)]$$

The important detail here is that the ELBO only uses the likelihood of a data point *given our current estimate of its code*, which we can sample.

```
data = tf.placeholder(tf.float32, [None, 28, 28])

prior = make_prior(code_size=2)
posterior = make_encoder(data, code_size=2)
code = posterior.sample()

likelihood = make_decoder(code, [28, 28]).log_prob(data)
divergence = tfd.kl_divergence(posterior, prior)
elbo = tf.reduce_mean(likelihood - divergence)
```

An intuitive interpretation is that maximizing the ELBO maximizes the likelihood of the data given the current codes, while encouraging the codes to be close to our prior belief of how codes should look like.

Running the Training

We just maximize the ELBO using gradient descent. This works because the sampling operations are implemented using the reparameterization trick internally, so that TensorFlow can backpropagate through them.

```
optimize = tf.train.AdamOptimizer(0.001).minimize(-elbo)
```

Moreover, we sample a few random codes from the prior to visualize the corresponding images that the VAE has learned. This is why we used `tf.make_template()` above, allowing us to call the decoder network again.

```
samples = make_decoder(prior.sample(10), [28, 28]).mean()
```

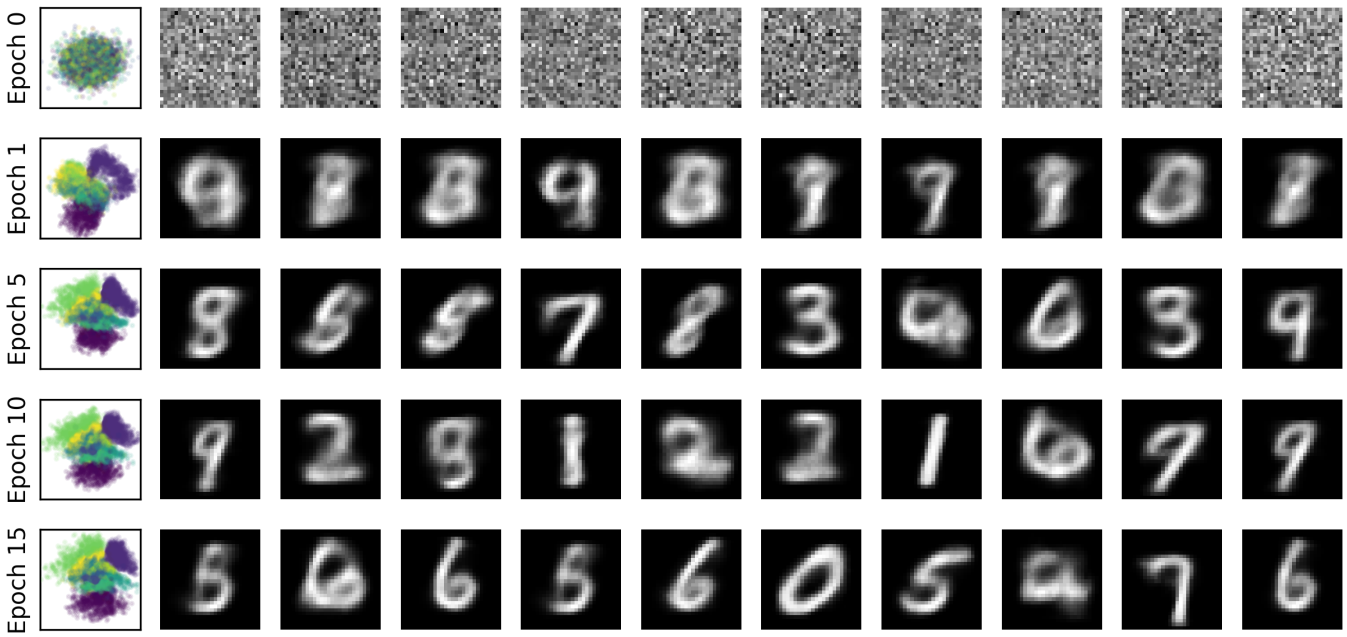
Finally, we load the data and create a session to run the training:

```
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('MNIST_data/')

with tf.train.MonitoredSession() as sess:
  for epoch in range(20):
    test_elbo, test_codes, test_samples = sess.run(
        [elbo, code, samples], {data: mnist.test.images})
    print('Epoch', epoch, 'elbo', test_elbo)
    plot_codes(test_codes)
    plot_sample(test_samples)
    for _ in range(600):
      sess.run(optimize, {data: mnist.train.next_batch(100)[0]})
```

That's it! If you want to play around with the code, take a look at the full code example. It also contains the plotting omitted in this post. Here are the latent codes, color-coded by the labels, as well as the decoded samples from the prior for the first few epochs of training.



As you can see, the latent space quickly separates into clusters for some of the different digits. If you use more dimensions for the code and larger networks, you will also see the generated images getting sharper.

Conclusion

We've learned to build a VAE in TensorFlow and trained it on MNIST digits. As a next step, you can run the code yourself and extend it, for example using a CNN encoder and deconv decoder. As always, if you have any question, please ask them below.


```
@misc{hafner2018tfdistvae,
  author = {Hafner, Danijar},
  title = {Building Variational Auto-Encoders in TensorFlow},
  year = {2018},
  howpublished = {Blog post},
  url = {https://danijar.com/building-variational-auto-encoders-in-ter
}
```

Thanks for the great tutorial!

I am confused by this two lines:

```
logit = tf.layers.dense(x, np.prod(data_shape))
logit = tf.reshape(logit, [-1] + data_shape)
```

I suppose data_shape = [28,28],
so the first line is to make the shape of 'logit' to be 28*28, and then why you reshape 'logit' to be [27,27] ?

Thank you!

Reply • Share ›

Puyuan Peng ➚ Puyuan Peng • a month ago

Also, about `logit = tf.layers.dense(x, np.prod(data_shape))`
while there is no activation function (e.g. sigmoid), how can you make sure that every entry of logit is between 0-1 so it can be the 'p' of a Bernoulli distribution

Thanks!

Reply • Share ›

Danijar Mod ➚ Puyuan Peng • a month ago

The -1 in a reshape operation is a placeholder for a dimension that is computed automatically. This works for both np.reshape() and tf.reshape(). We reshape from [N, 784] to [N, 28, 28], where N is the batch size. Since we specify all other dimensions, we can pass -1 instead of N and have it compute the correct length automatically.

Logits are by definition unnormalized log probabilities. The Bernoulli distribution applies a softmax function to them to arrive at normalized probabilities.

Reply • Share ›

Puyuan Peng ➚ Danijar • a month ago

From https://www.tensorflow.org/... it seems that it should be sigmoid(logits) rather than softmax?

Reply • Share ›

Puyuan Peng ➚ Danijar • a month ago

Thanks for your reply! This is really helpful!!

Reply • Share ›

Nikhil Oswal • 3 months ago

Hello, Can you help with plotting the loss function for this without using tensorboard

Thanks

Reply • Share ›

Erin Fowler • 5 months ago

I got some errors. Not with your code but the mnist dataset. that it wasn't the right shape and/or part of the code needed a tensor handler... not to mention warnings that i should download https://github.com/tensorfl... and use that instead of the

from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('MNIST_data/')

I am still very new to python and deep learning and although I understand the logic I always get held up it seems with getting data into the right format. Which is frustrating.

If you know how to get tf.keras.datasets.mnist into the proper format... or any other help it would be appreciated

Reply • Share ›

Мирољуб Арбутина • a year ago

Hi Danijar, nice tutorial.
Any particular reason you choose

code_size=2 i.e. how do we get an optimal embedding dimension size(latent dim)? They could have been 1D or 3D or nD vectors?

Also I'm a bit puzzled by the Bernoulli output. Does the TF calculates log_prob with all the [28x 28]D Bernoulli distribs per sample or does he use the reduced 2D.
Could you have used there a binary_crossetropy or generative loss as in the article.
Thanks

Reply • Share ›

Danijar Mod ➚ Мирољуб Арбутина • a year ago

The task becomes easier with a larger code size. I chose 2 in the post because it makes it easy to visualize. When you have larger codes, you need to project them down into a 2D or 3D for visualization. On MNIST, I've found 8 to be a reasonable size for the code.

The pixels are modeled as independent Bernoulli variables. If you just created a `tfd.Bernoulli()` and evaluate the log probability of an image under it, TensorFlow would return a tensor with the probabilities for every pixel separately. The `tfd.Independent()` tells TensorFlow to return a scalar probability per image. As the name "independent" suggests, this single probability is just the product of the pixel probabilities.

2 Reply • Share ›

Мирољуб Арбутина ➚ Danijar • a year ago

Thanks for the clarifications.

Reply • Share ›

ALSO ON **DANIJAR HAFNER**

Why Mean Squared Error?

4 comments • a year ago

Danijar — I see, the central limit theorem says that the sum of many independent variables tends towards being Gaussian distributed. ...

Introduction to Recurrent Networks in TensorFlow

54 comments • 3 years ago

Akshit Arora — Hi!I am interested in using sigmoid cross entropy loss and calculating auc instead of accuracy since I want

Variable Sequence Lengths in TensorFlow

91 comments • 3 years ago

Rohit Gupta — If you are using a seq2seq model then the encoder inputs must be padded with zeros on the left and the decoder

Patterns for Fast Prototyping with TensorFlow

3 comments • 8 months ago

richdev — Great article !!

Danijar Hafner

mail@danijar.com
danijar
danijarh

Researcher aiming to build intelligent machines based on concepts of the human brain.