Variational Inference in TensorFlow

Danijar Hafner · Stanford CS 20 · 2018-02-16
University College London, Google Brain

Outline

Variational Inference

Tensorflow Distributions

VAE in TensorFlow

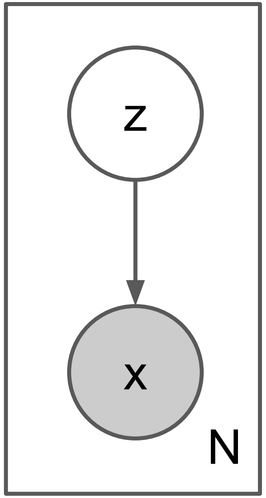
Variational Inference

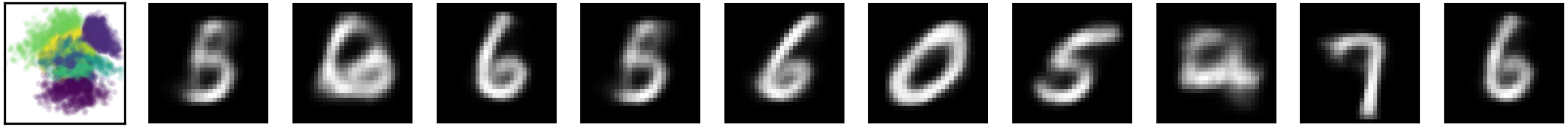
Learning unknown variables

Images consist of millions of pixels, but there is likely a more compact representation of the content (objects, positions, etc)

Finding the mapping to this representation allows us to find semantically similar images and even to generate new images

We call the compact representation z and its corresponding pixels x, and this is the same structure for every of our N images





Learning unknown variables

Assume model where data x was generated from latent variables z

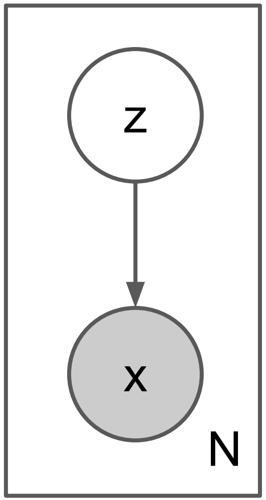
We want to find the latent variables that explain our data best

But we cannot directly maximize the data likelihood since it depends on the latent variables and we don't know their values

$$\theta^* = \text{argmax } P_\theta(x) = \text{argmax} \int P_\theta(x|z)P(z)\, dz$$

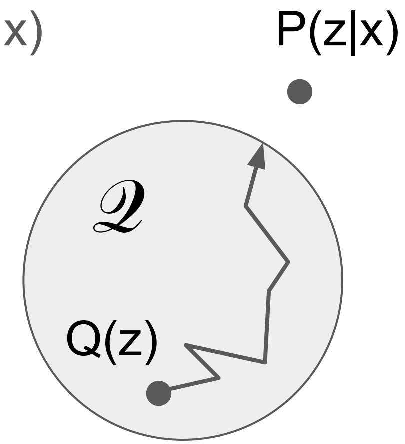
We define a prior assumption P(z) about how z is distributed

We are interested in the posterior belief P(z|x) that depends on the corresponding data point x



Variational lower bound: Overview

Iteratively optimize approximate posterior Q(z) until Q(z) ≈ P(z|x)



Objective for Q(z) is the variational lower bound

$$
\begin{aligned}
\ln P(x) &\geq E_{Q(z)}[\ln P(x,z) - \ln Q(z)] \\
&= E_{Q(z)}[\ln P(x|z) + \ln P(z) - \ln Q(z)] \\
&= E_{Q(z)}[\ln P(x|z)] - D_{KL}[Q(z)||P(z)]
\end{aligned}
$$

This is a lower bound since the KL is non-negative

Choose Q(z) ϵ $\mathscr{Q}$ that allows differentiable sampling, often Gaussian distribution

KL becomes a regularization terms; computed analytically or sampled

Variational lower bound: Algorithm

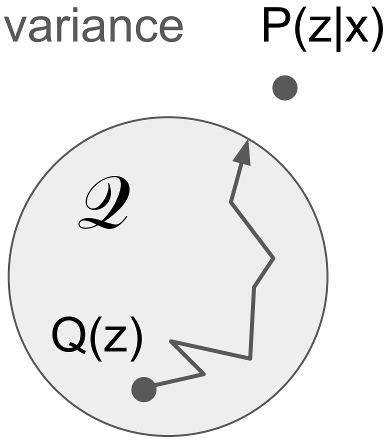
Initialize fixed prior on latents P(z), for example zero mean unit variance

Initialize network weights θ and sufficient stats μ and σ of Q(z)

Remember objective $\ln P(x) \geq E_{Q(z)}[\ln P_\theta(x|z)] - D_{KL}[Q(z)||P(z)]$

Iterate until convergence:

1. Sample a $z \sim Q(z)$ as $z = \sigma\varepsilon + \mu$ with $\varepsilon \sim N(0, 1)$
2. Compute data likelihood $P_\theta(x|z)$ using neural network
3. Compute KL between Q(z) and P(z) analytically
4. Perform gradient ascent on objective to optimize θ, μ, σ



Amortized inference: Overview

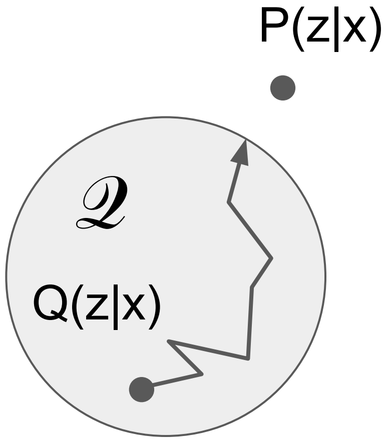
We could learn sufficient stats of Q(z) for every data point via gradient descent

But need multiple gradient steps for every data point, even during evaluation

Instead, learn the result of this process using an encoder network Q(z|x)

Assume similarity in how latents are inferred for all data points

Backpropagate to optimize encoder weights instead of posterior sufficient statistics



Amortized inference: Algorithm

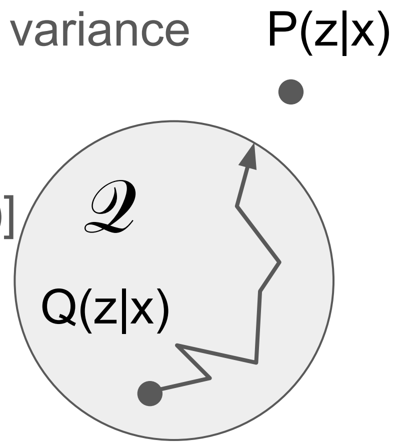
Initialize fixed prior on latents $P(z)$, for example zero mean unit variance

Initialize encoder weights ϕ and decoder weights θ

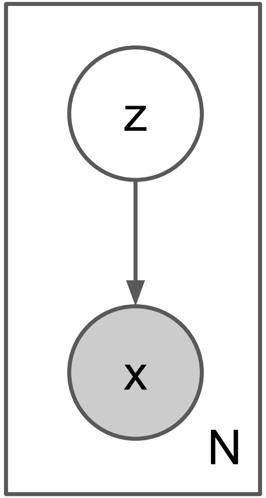
Remember objective $\ln P(x) \geq E_{Q(z)}[\ln P_\theta(x|z)] - D_{KL}[Q_\phi(z|x)||P(z)]$

Iterate until convergence:

1. Select data point x and compute $Q_\phi(z|x)$ using encoder
2. Sample a $z \sim Q(z|x)$ as $z = \sigma\varepsilon + \mu$ with $\varepsilon \sim N(0, 1)$
3. Compute data likelihood $P_\theta(x|z)$ using decoder
4. Compute KL between $Q(z)$ and $P(z)$ analytically
5. Perform gradient ascent on objective to optimize θ and ϕ



Variational Auto-Encoder

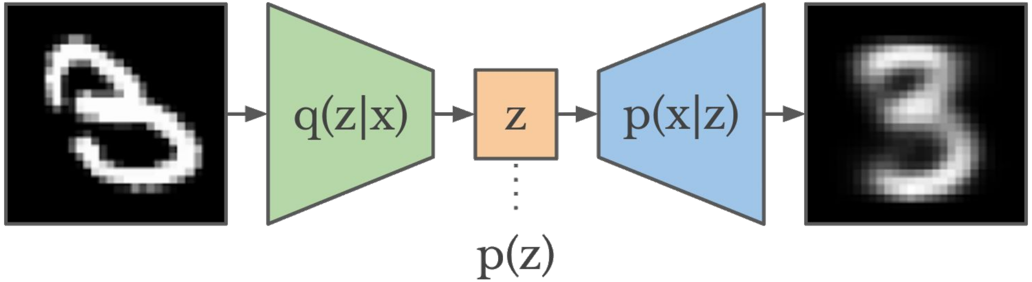


Encoder for amortized inference $Q(z|x)$

Decoder for generative model $P(x|z)$

Variational lower bound objective $E_{Q(z|x)}[\ln P(x|z)] - D_{KL}[Q(z|x)||P(z)]$

Trained end-to-end via gradients by reparameterized sampling



Kingma et al. 2014, Rezende et al. 2014

Bayesian Neural Network

Independent latent $Q(\theta)$ is diagonal Gaussian

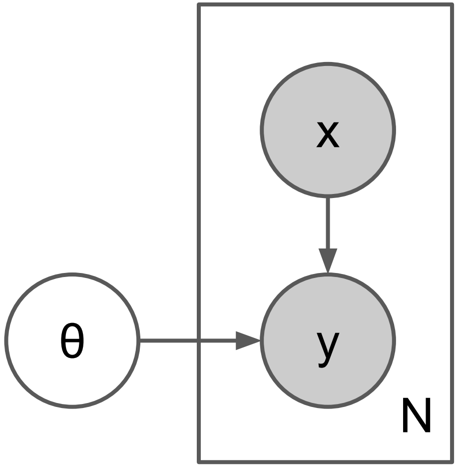
Conditional generative model $P_\theta(y|x)$

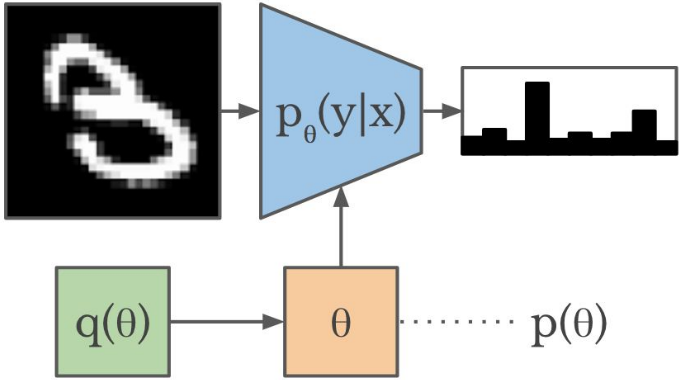
Variational lower bound objective
$E_{Q(\theta)}[\ln P_\theta(y|x)] - D_{KL}[Q(\theta)||P(\theta)]$

Divide KL term by the data set size since parameters are shared for whole data set

Trained end-to-end via gradients by reparameterized sampling

Blundell et al. 2015





TensorFlow Distributions

TensorFlow Distributions: Overview

Probabilistic programming made easy!

```
tfd = tf.contrib.distributions
```

TensorFlow Distributions: Overview

Probabilistic programming made easy!

```
tfd = tf.contrib.distributions
```

```
mean = tf.layers.dense(hidden, 10, None)
stddev = tf.layers.dense(hidden, 10, tf.nn.softplus)
dist = tfd.MultivariateNormalDiag(mean, stddev)
```

TensorFlow Distributions: Overview

Probabilistic programming made easy!

```
tfd = tf.contrib.distributions

mean = tf.layers.dense(hidden, 10, None)
stddev = tf.layers.dense(hidden, 10, tf.nn.softplus)
dist = tfd.MultivariateNormalDiag(mean, stddev)

samples = dist.sample()
dist.log_prob(samples)
```

TensorFlow Distributions: Overview

Probabilistic programming made easy!

```
tfd = tf.contrib.distributions

mean = tf.layers.dense(hidden, 10, None)
stddev = tf.layers.dense(hidden, 10, tf.nn.softplus)
dist = tfd.MultivariateNormalDiag(mean, stddev)

samples = dist.sample()
dist.log_prob(samples)

other = tfd.MultivariateNormalDiag(
    tf.zeros_like(mean), tf.ones_like(stddev))
tfd.kl_divergence(dist, other)
```

TensorFlow Distributions: Regression example

```
tfd = tf.contrib.distributions

hidden = tf.layers.dense(inputs, 100, tf.nn.relu)
mean = tf.layers.dense(hidden, 10, None)
dist = tfd.MultivariateNormalDiag(mean, tf.ones_like(mean))
```

TensorFlow Distributions: Regression example

```
tfd = tf.contrib.distributions

hidden = tf.layers.dense(inputs, 100, tf.nn.relu)
mean = tf.layers.dense(hidden, 10, None)
dist = tfd.MultivariateNormalDiag(mean, tf.ones_like(mean))

loss = -dist.log_prob(label)  # Squared error
optimize = tf.train.AdamOptimizer().minimize(loss)
```

TensorFlow Distributions: Classification example

```
tfd = tf.contrib.distributions

hidden = tf.layers.dense(inputs, 100, tf.nn.relu)
logit = tf.layers.dense(hidden, 10, None)
dist = tfd.Categorical(logit)
```

TensorFlow Distributions: Classification example

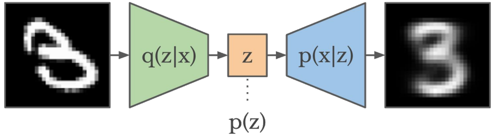
```
tfd = tf.contrib.distributions

hidden = tf.layers.dense(inputs, 100, tf.nn.relu)
logit = tf.layers.dense(hidden, 10, None)
dist = tfd.Categorical(logit)

loss = -dist.log_prob(label)  # Cross entropy
optimize = tf.train.AdamOptimizer().minimize(loss)
```

VAE in TensorFlow

VAE in TensorFlow: Overview

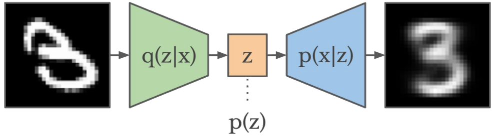



```
tfd = tf.contrib.distributions

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

prior = make_prior()
posterior = make_encoder(images)
dist = make_decoder(posterior.sample())
```

VAE in TensorFlow: Overview



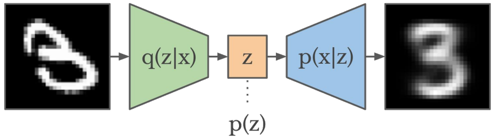

```
tfd = tf.contrib.distributions

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

prior = make_prior()
posterior = make_encoder(images)
dist = make_decoder(posterior.sample())

elbo = dist.log_prob(images) - tfd.kl_divergence(posterior, prior)
optimize = tf.train.AdamOptimizer().minimize(-elbo)
samples = make_decoder(prior.sample(10)).mean()  # For visualization
```

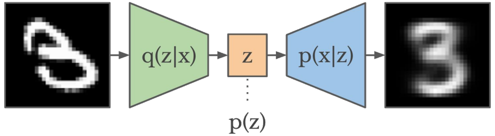
VAE in TensorFlow: Prior & encoder




```
def make_prior(code_size=2):
  mean, stddev = tf.zeros([code_size]), tf.ones([code_size])
  return tfd.MultivariateNormalDiag(mean, stddev)

def make_encoder(images, code_size=2):
  images = tf.layers.flatten(images)
  hidden = tf.layers.dense(images, 100, tf.nn.relu)
  mean = tf.layers.dense(hidden, code_size)
  stddev = tf.layers.dense(hidden, code_size, tf.nn.softplus)
  return tfd.MultivariateNormalDiag(mean, stddev)
```

VAE in TensorFlow: Networks



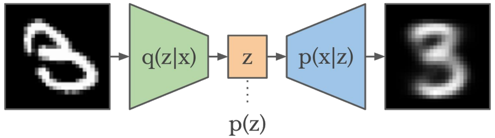

```
def make_decoder(code, data_shape=[28, 28]):
  hidden = tf.layers.dense(code, 100, tf.nn.relu)
  logit = tf.layers.dense(hidden, np.prod(data_shape))
  logit = tf.reshape(logit, [-1] + data_shape)
  return tfd.Independent(tfd.Bernoulli(logit), len(data_shape))
```

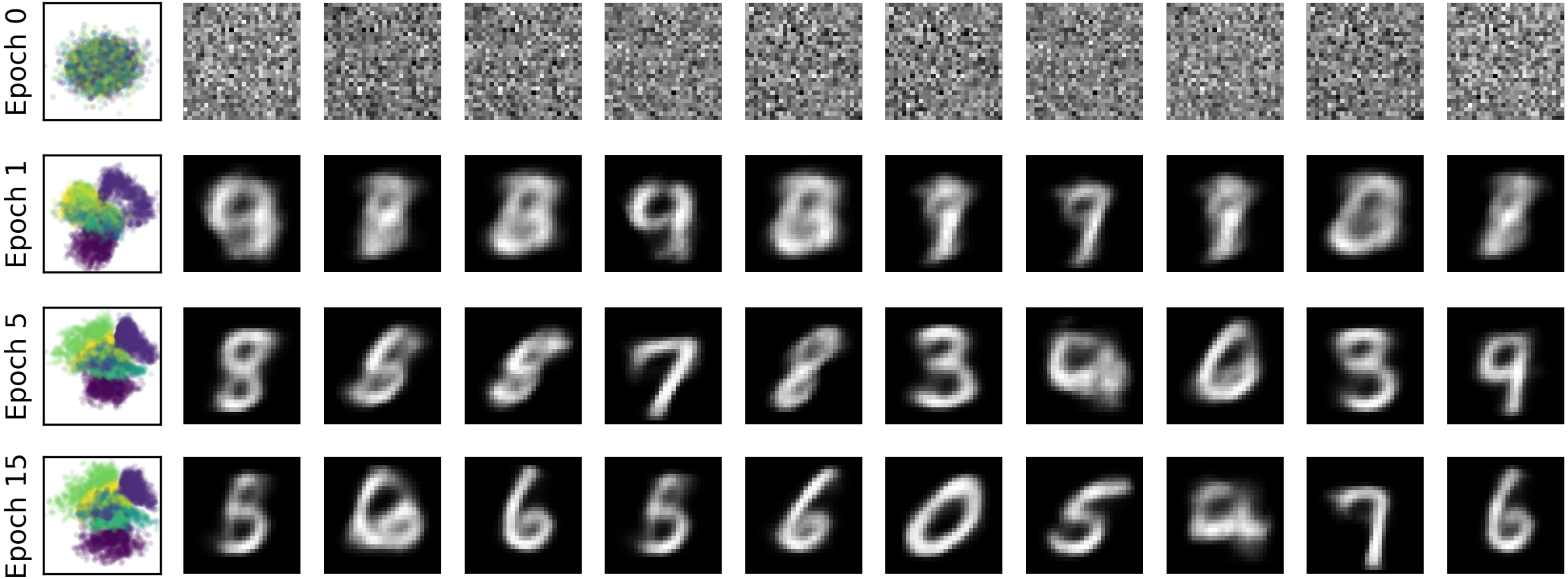
The tfd.Independent(dist, 2) tells TensorFlow to treat the two innermost dimensions as data dimensions rather than batch dimensions

This means dist.log_prob(images) returns a number per images rather than per pixel

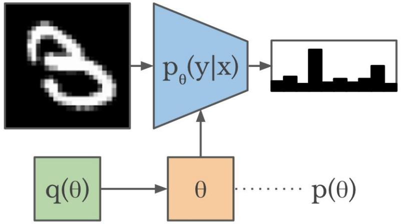
As the name tfd.Independent() says, it's just summing the pixel log probabilities

VAE in TensorFlow: Results



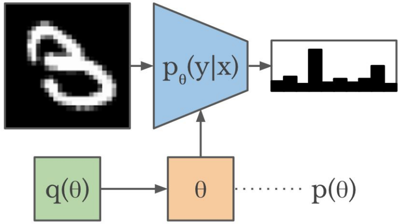


Bonus: BNN in TensorFlow




```
def define_network(images, num_classes=10):
  mean = tf.get_variable('mean', [28 * 28, num_classes])
  stddev = tf.get_variable('stddev', [28 * 28, num_classes])
  prior = tfd.MultivariateNormalDiag(
      tf.zeros_like(mean), tf.ones_like(stddev))
  posterior = tfd.MultivariateNormalDiag(mean, tf.nn.softplus(stddev))
```

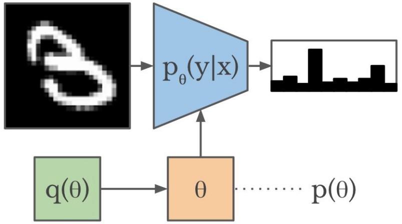
Bonus: BNN in TensorFlow




```
def define_network(images, num_classes=10):
  mean = tf.get_variable('mean', [28 * 28, num_classes])
  stddev = tf.get_variable('stddev', [28 * 28, num_classes])
  prior = tfd.MultivariateNormalDiag(
      tf.zeros_like(mean), tf.ones_like(stddev))
  posterior = tfd.MultivariateNormalDiag(mean, tf.nn.softplus(stddev))

  bias = tf.get_variable('bias', [num_classes])  # Or Bayesian, too
  logit = tf.nn.relu(tf.matmul(posterior.sample(), images) + bias)
  return tfd.Categorical(logit), posterior, prior
```

Bonus: BNN in TensorFlow




```
def define_network(images, num_classes=10):
  mean = tf.get_variable('mean', [28 * 28, num_classes])
  stddev = tf.get_variable('stddev', [28 * 28, num_classes])
  prior = tfd.MultivariateNormalDiag(
      tf.zeros_like(mean), tf.ones_like(stddev))
  posterior = tfd.MultivariateNormalDiag(mean, tf.nn.softplus(stddev))

  bias = tf.get_variable('bias', [num_classes])  # Or Bayesian, too
  logit = tf.nn.relu(tf.matmul(posterior.sample(), images) + bias)
  return tfd.Categorical(logit), posterior, prior

dist, posterior, prior = define_network(images)
elbo = (tf.reduce_mean(dist.log_prob(label)) -
        tf.reduce_mean(tfd.kl_divergence(posterior, prior))
```

That's all