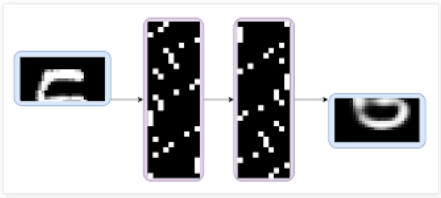
Eric Jang

Technology, A.I., Careers

Tuesday, November 8, 2016

Tutorial: Categorical Variational Autoencoders using Gumbel-Softmax



In this post, I discuss our recent paper, Categorical Reparameterization with Gumbel-Softmax, which introduces a simple technique for training neural networks with discrete latent variables. I'm really excited to share this because (1) I believe it will be quite useful for a variety of Machine Learning research problems, (2) this is my first published paper ever (on Arxiv, and submitted to a NIPS workshop and ICLR as well).

The TLDR; if you want categorical features in your neural nets, just let `sample = softmax((logits+gumbel noise)/temperature)`, and then backprop as usual using your favorite automatic differentiation software (e.g. TensorFlow, Torch, Theano).

You can find the code for this article here

Introduction

One of the main themes in Deep Learning is to "let the neural net figure out all the intermediate features". For example: training convolutional neural networks results in the self-organization of a feature detector hierarchy, while Neural Turing Machines automatically "discover" copying and sorting algorithms.

The workhorse of Deep Learning is the backpropagation algorithm, which uses dynamic programming to compute parameter gradients of the network. These gradients are then used to minimize the optimization objective via gradient descent. In order for this to work, all of the layers in our neural network — i.e. our learned intermediate features — must be continuous-valued functions.

What happens if we want to learn intermediate representations that are discrete? Many "codes" we want to learn are fundamentally discrete - musical notes on a keyboard, object classes ("kitten", "balloon", "truck"), and quantized addresses ("index 423 in memory").

We can use stochastic neural networks, where each layer compute the parameters of some (discrete) distribution, and its forward pass consists of taking a sample from that parametric distribution. However, the difficulty is that we can't backpropagate through samples. As shown below, there is a stochastic node (blue circle) in between $f(z)$ and θ.

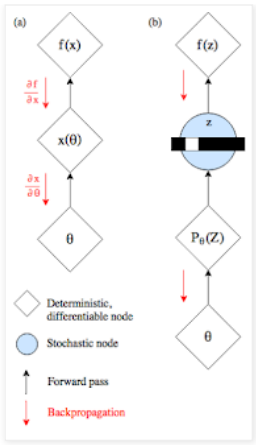
RSS

Subscribe to Email Updates

Email address... Submit

Blog Archive

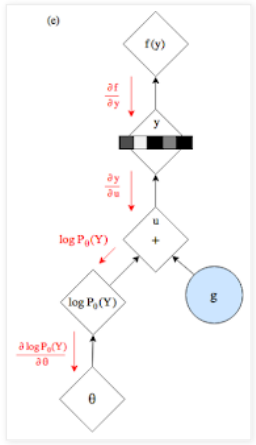
▶ 2019 (7)
▶ 2018 (11)
▶ 2017 (3)
▼ 2016 (11)
▼ November (1)
Tutorial: Categorical Variational Autoencoders usi...
▶ September (2)
▶ August (1)
▶ July (3)
▶ June (4)



Left: in continuous neural nets, you can use backprop to compute parameter gradients. Right: backpropagation is not possible through stochastic nodes.

Gumbel-Softmax Distribution

The problem of backpropagating through stochastic nodes can be circumvented if we can re-express the sample $z \sim p_\theta(z)$, such that gradients can flow from $f(z)$ to θ without encountering stochastic nodes. For example, samples from the normal distribution $z \sim \mathcal{N}(\mu, \sigma)$ can be re-written as $z = \mu + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$. This is also known as the "reparameterization trick", and is commonly used to train variational autoencoders with Gaussian latent variables.



The Gumbel-Softmax distribution is reparameterizable, allowing us to avoid the stochastic node during backpropagation.

The main contribution of this work is a "reparameterization trick" for the categorical distribution. Well, not quite - it's actually a re-parameterization trick for a distribution that we can *smoothly deform* into the categorical distribution. We use the Gumbel-Max trick, which provides an efficient way to draw samples z from the Categorical distribution with class probabilities π_i:

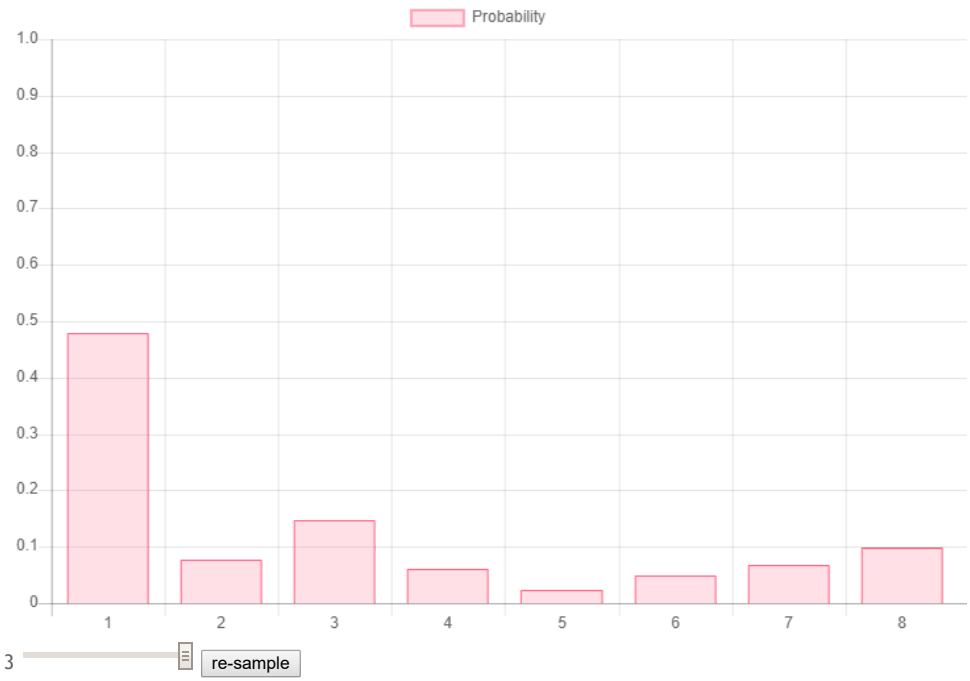
$$z = \texttt{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right)$$

argmax is not differentiable, so we simply use the softmax function as a continuous approximation of argmax:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \qquad \text{for } i = 1, \ldots, k.$$

Hence, we call this the Gumbel-*Soft*Max distribution*. τ is a temperature parameter that allows us to control how closely samples from the Gumbel-Softmax distribution approximate those from the categorical distribution. As $\tau \to 0$, the softmax becomes an argmax and the Gumbel-Softmax distribution becomes the categorical distribution. During training, we let $\tau > 0$ to allow gradients past the sample, then gradually anneal the temperature τ (but not completely to 0, as the gradients would blow up).

Below is an interactive widget that draws samples from the Gumbel-Softmax distribution. Keep in mind that samples are vectors, and a one-hot vector (i.e. one of the elements is 1.0 and the others are 0.0) corresponds to a discrete category. Click "re-sample" to generate a new sample, and try dragging the slider and see what samples look like when the temperature τ is small!



TensorFlow Implementation

Using this technique is extremely simple, and only requires 12 lines of Python code:

```python
def sample_gumbel(shape, eps=1e-20):
  """Sample from Gumbel(0, 1)"""
  U = tf.random_uniform(shape,minval=0,maxval=1)
  return -tf.log(-tf.log(U + eps) + eps)

def gumbel_softmax_sample(logits, temperature):
  """ Draw a sample from the Gumbel-Softmax distribution"""
  y = logits + sample_gumbel(tf.shape(logits))
  return tf.nn.softmax( y / temperature)

def gumbel_softmax(logits, temperature, hard=False):
  """Sample from the Gumbel-Softmax distribution and optionally discretize.
  Args:
    logits: [batch_size, n_class] unnormalized log-probs
    temperature: non-negative scalar
    hard: if True, take argmax, but differentiate w.r.t. soft sample y
  Returns:
    [batch_size, n_class] sample from the Gumbel-Softmax distribution.
    If hard=True, then the returned sample will be one-hot, otherwise it will
    be a probabilitiy distribution that sums to 1 across classes
  """
  y = gumbel_softmax_sample(logits, temperature)
  if hard:
    k = tf.shape(logits)[-1]
    #y_hard = tf.cast(tf.one_hot(tf.argmax(y,1),k), y.dtype)
    y_hard = tf.cast(tf.equal(y,tf.reduce_max(y,1,keep_dims=True)),y.dtype)
    y = tf.stop_gradient(y_hard - y) + y
  return y
```

gumbel-softmax.py hosted with ♡ by GitHub view raw

Despite its simplicity, Gumbel-Softmax works surprisingly well - we benchmarked it against other stochastic gradient estimators for a couple tasks and Gumbel-Softmax outperformed them for both Bernoulli (K=2) and Categorical (K=10) latent variables. We can also use it to train semi-supervised classification models much faster than previous approaches. See our paper for more details.

Categorical VAE with Gumbel-Softmax

To demonstrate this technique in practice, here's a categorical variational autoencoder for MNIST, implemented in less than 100 lines of Python + TensorFlow code.

In standard Variational Autoencoders, we learn an encoding function that maps the data manifold to an isotropic Gaussian, and a decoding function that transforms it back to the sample. The data manifold is projected into a Gaussian ball; this can be hard to interpret if you are trying to learn the categorical structure within your data.

First, we declare the encoding network:

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt
import numpy as np
```

```
%matplotlib inline
slim=tf.contrib.slim
Bernoulli = tf.contrib.distributions.Bernoulli

K=10 # number of classes
N=30 # number of categorical distributions

# input image x (shape=(batch_size,784))
x = tf.placeholder(tf.float32,[None,784])
# variational posterior q(y|x), i.e. the encoder (shape=(batch_size,200))
net = slim.stack(x,slim.fully_connected,[512,256])
# unnormalized logits for N separate K-categorical distributions (shape=(batch_size*N,K))
logits_y = tf.reshape(slim.fully_connected(net,K*N,activation_fn=None),[-1,K])
q_y = tf.nn.softmax(logits_y)
log_q_y = tf.log(q_y+1e-20)
```

gs-encoder.py hosted with ♡ by GitHub view raw

Next, we sample from the Gumbel-Softmax posterior and decode it back into our MNIST image.

```
# temperature
tau = tf.Variable(5.0,name="temperature")
# sample and reshape back (shape=(batch_size,N,K))
# set hard=True for ST Gumbel-Softmax
y = tf.reshape(gumbel_softmax(logits_y,tau,hard=False),[-1,N,K])
# generative model p(x|y), i.e. the decoder (shape=(batch_size,200))
net = slim.stack(slim.flatten(y),slim.fully_connected,[256,512])
logits_x = slim.fully_connected(net,784,activation_fn=None)
# (shape=(batch_size,784))
p_x = Bernoulli(logits=logits_x)
```

gs-decoder.py hosted with ♡ by GitHub view raw

Variational autoencoders minimizes reconstruction error of the data by maximizing an expectedlower bound (ELBO) on the likelihood of the data, under a generative model $p_\theta(x)$. For a derivation, see this tutorial on variational methods.

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(y|x)}[\log p_\theta(x|y)] - KL[q_\phi(y|x)||p_\theta(y)]$$

```
# loss and train ops
kl_tmp = tf.reshape(q_y*(log_q_y-tf.log(1.0/K)),[-1,N,K])
KL = tf.reduce_sum(kl_tmp,[1,2])
elbo=tf.reduce_sum(p_x.log_prob(x),1) - KL
```

gs-kl.py hosted with ♡ by GitHub view raw

Finally, we run train our VAE:

```
loss=tf.reduce_mean(-elbo)
lr=tf.constant(0.001)
train_op=tf.train.AdamOptimizer(learning_rate=lr).minimize(loss,var_list=slim.get_model_variables())
init_op=tf.initialize_all_variables()
# get data
data = input_data.read_data_sets('/tmp/', one_hot=True).train

BATCH_SIZE=100
NUM_ITERS=50000
tau0=1.0 # initial temperature
np_temp=tau0
np_lr=0.001
ANNEAL_RATE=0.00003
MIN_TEMP=0.5

dat=[]
sess=tf.InteractiveSession()
sess.run(init_op)
for i in range(1,NUM_ITERS):
  np_x,np_y=data.next_batch(BATCH_SIZE)
  _,np_loss=sess.run([train_op,loss],{x:np_x,tau:np_temp,lr:np_lr})
  if i % 100 == 1:
    dat.append([i,np_temp,np_loss])
  if i % 1000 == 1:
    np_temp=np.maximum(tau0*np.exp(-ANNEAL_RATE*i),MIN_TEMP)
    np_lr*=0.9
  if i % 5000 == 1:
    print('Step %d, ELBO: %0.3f' % (i,-np_loss))
```

gs-train.py hosted with ♡ by GitHub view raw

...and, that's it! Now we can sample randomly from our latent categorical code and decode it back into MNIST images:

Code can be found here. Thank you for reading, and let me know if you find this technique useful!

Acknowledgements

I'm sincerely grateful to my co-authors, Shane Gu and Ben Poole for collaborating with me on this work. Shane introduced me to the Gumbel-Max trick back in August, and supplied the framework for comparing Gumbel-Softmax with existing stochastic gradient estimators. Ben suggested and implemented the semi-supervised learning aspect of the paper, did the math derivations in the Appendix, and helped me a lot with editing the paper. Finally, thanks to Vincent Vanhoucke and the Google Brain team for encouraging me to pursue this idea.

*Chris J. Maddison, Andriy Mnih, and Yee Whye Teh at Deepmind have discovered this technique independently and published their own paper on it - they call it the "Concrete Distribution". We only found out about each other's work right as we were submitting our papers to conferences (oops!). If you use this technique in your work, please cite both of our papers! They deserve just as much credit.

23 Apr 2017: Update - Chris Maddison and I integrated these distributions into TensorFlow's Distributions sub-framework. Here's a code example of how to implement a categorical VAE using the distributions API.

Posted by Eric at 11:00 PM

Labels: AI, Statistics

17 comments:

Viktor Yanush November 9, 2016 at 11:05 AM

Could you please compare your model to model called "Discrete Variational Autoencoder" and give some thoughts on the difference and similarity of models?

https://arxiv.org/abs/1609.02200

Reply

Replies

Unknown November 14, 2016 at 11:44 AM

+1

Eric April 23, 2017 at 3:12 PM

Sure. The Discrete VAE paper (https://arxiv.org/abs/1609.02200), despite its name, is not the first paper to implement discrete variational autoencoders with stochastic discrete latent variables. Prior work includes NVIL (https://arxiv.org/pdf/1402.0030), DARN, and MuProp. Discrete VAE presents a model that counts technically as a VAE, but its forward pass is not equivalent to the model described in the other papers. In Discrete VAE, the forward sampling is autoregressive through each binary unit, which allows every discrete choice to be marginalized out in a tractable manner in the backward pass. Because the forward pass is different, the optimization objective is different, which makes it harder to compare (we are optimizing different models). The non-discrete Gumbel-Softmax relaxation also technically results in optimizing a different model as well, but since it's merely a relaxation of the original model, we can still evaluate it the same way.

Whereas DARN, MuProp, NVIL, Straight-Through Gumbel-Softmax present a way to train the same forward model, Discrete VAE optimizes a new objective altogether. It's an open question what the

"right forward pass" is, but it makes it hard to compare Discrete VAE with other work since they have different forward passes and optimization strategies.

Reply

Anonymous November 9, 2016 at 10:39 PM

Great tutorial!

I am wondering what happens if the number of categories is extremely large, i.e., 1 million. Gradients need to calculated for all the \PI s and g s of a large number?

Thank you, and looking forward to hearing.

Reply

Replies

Eric November 10, 2016 at 5:55 PM

If categories are large, you will need a more efficient encoding of samples from the categorical distribution than one-hot vectors, otherwise you will have a rank>1e6 matrix multiply. A reparameterization trick for other encodings of vectors might be worth pursuing.

Reply

Unknown November 14, 2016 at 6:01 PM

Hi Eric,

Agree with the posters above me -- great tutorial!

I was wondering how this would be applied to my use case: suppose I have two dense real-valued vectors, and I want to train a VAE s.t. the latent features are categorical and the original and decoded vectors are close together in terms of cosine similarity. I'm guessing that I have to change the first term of the ELBO function, since `p_x.log_prob(x)` isn't what I care about (is that right?). Any thoughts on what the modified version would be?

Thanks

Reply

Ero Gol November 17, 2016 at 5:02 AM

I still don't see why we cannot train the same network by enforcing the latent space a one-hot vector for the example above. So if the backprop is the problem, you can flow the error through the argmax node and you can learn the parameters still. Could you give more details what is the differentiating factor of your method. Also cloud you explain what is z values on the last figure?

Reply

Unknown December 23, 2016 at 1:56 PM

More stuff about gumbel-sigmoid here - https://github.com/yandexdataschool/gumbel_lstm/blob/master/demo_gumbel_sigmoid.ipynb

@Ero Gol
afaik, unlike max, argmax (index of maximum) will have zero/NA gradient by definition since infinitely small changes in the vector won't change index of the maximum unless there are two exactly equal elements.

Reply

Brandy Lehmann January 25, 2017 at 2:06 AM

This comment has been removed by a blog administrator.

Reply

Unknown March 6, 2017 at 3:48 PM

Hi Eric,

I'm a student researcher at MIT doing work on the application of GAN to a discrete data set. I was wondering if there's any chance we could hop on a call so you could explain this methodology to me further?

Reply

Replies

Eric April 23, 2017 at 3:13 PM

Sure. Please send me an email (you can find it on my website, evjang.com)

Reply

Gökçen Eraslan March 31, 2017 at 11:47 AM

p_theta(y) prior here in KL divergence code is just a categorical with equal 1/K probabilities, right?

Reply

Unknown April 14, 2017 at 5:15 PM

Thank you Eric for this detailed introduction. Super helpful!

-Yixing

Reply

Gopal Sharma May 22, 2017 at 6:30 PM

What are the pros and cons of using q(y|x) as discrete distribution?

Reply

Prasanna Gyawali August 7, 2017 at 2:01 PM

This comment has been removed by the author.

Reply

Unknown October 2, 2017 at 3:19 AM

Thanks Eric, this is very exciting work!

I had a question about the loss function. The latent loss here seems to be the KL-divergence between two categorical distributions (without tau). However, the density of the GS distribution in your paper involves tau. Does it simplify somehow in the KL divergence, or did you use the categorical distribution for the loss function instead?

Thanks a lot in advance!

Reply

Replies

Eric November 29, 2017 at 4:25 PM

We used a categorical KL instead of the KL of the Gumbel-Softmax distribution. Maddison et al. 2017 (Concrete Distribution) use the latter, which involves tau.

Reply

Enter your comment...

Comment as: yakimonofr@gı ▾ Sign out

Publish Preview Notify me

Comments will be reviewed by administrator (to filter for spam and irrelevant content).

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)