Bayesian Layers: A Module for Neural Network Uncertainty

Dustin Tran*, Mike Dusenberry*, Mark van der Wilk†, Danijar Hafner*
* Google Brain † PROWLER.io

Overview

- Software for research with Bayesian neural nets, GPs, flows are limiting. Scale is one challenge.
- Bayesian Layers extend neural net libraries with layers capturing uncertainty—including 1. weights, 2. functions, 3. activations, and 4. propagating uncertainty from input-to-output.
- We fit a 10B-parameter Bayesian neural net (Bayesian Transformer) over 512 TPUv2 cores.
- Current work is solving challenges with billion parameter BNNs, GPs, and flows.

Bayesian LSTM Example

```
batch_size = 256
features, labels = load_dataset(batch_size)
lstm = layers.LSTMCellReparameterization(512)
output_layer = tf.keras.layers.Dense(labels.shape[-1])
state = lstm.zero_state(batch_size, dtype=tf.float32)
nll = 0.
for t in range(features.shape[1]):
  net, state = lstm(features[:, t], state
  logits = output_layer(net)
  nll += tf.losses.softmax_cross_entropy(
    labels=labels[:, t],
    logits=logits)

kl = sum(lstm.losses)
loss = nll + kl
train_op = tf.AdamOptimizer().minimize(los
```

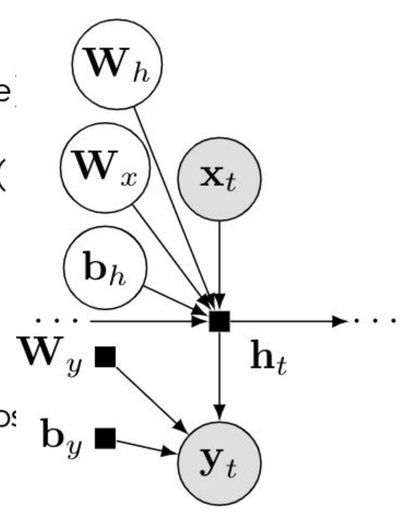
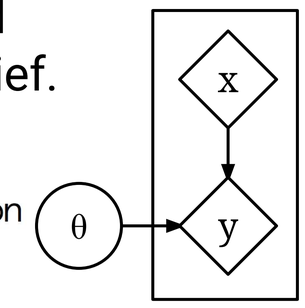


Figure: Bayesian LSTM

1 Bayesian Neural Network Layers

BNN Layers map tensor to tensor and internally sample from the weight belief.



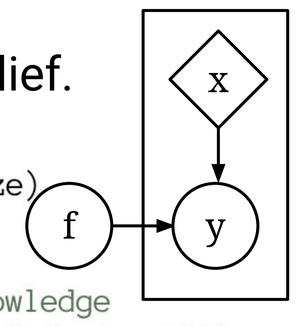

```
if FLAGS.be_bayesian:
  Conv2D = layers.Conv2DReparameterization
else:
  Conv2D = tf.keras.layers.Conv2D

model = tf.keras.Sequential([
    Conv2D(32, kernel_size=5, strides=1, padding="SAME"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation("relu"),
    Conv2D(32, kernel_size=5, strides=2, padding="SAME"),
    tf.keras.layers.BatchNormalization(),
    ...
])
```

Figure: Bayesian CNN

2 Gaussian Process Layers

GP layers map tensor to tensor and internally sample from the function belief.



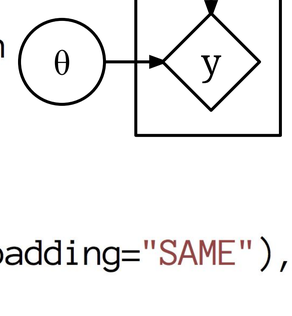

```
batch_size = 256
features, labels = load_spatial_data(batch_size)

model = tf.keras.Sequential([
  tf.keras.layers.Flatten(),  # no spatial knowledge
  layers.SparseGaussianProcess(units=256, num_inducing=512),
  layers.SparseGaussianProcess(units=256, num_inducing=512),
  layers.SparseGaussianProcess(units=10, num_inducing=512),
])
predictions = model(features)
neg_log_likelihood = tf.losses.mean_squared_error(labels=labels,
    predictions=predictions)
kl = sum(model.losses)
loss = neg_log_likelihood + kl
train_op = tf.train.AdamOptimizer().minimize(loss)
```

Figure: Deep GP

3 Stochastic Output Layers

Stochastic output layers map tensors to random variables. Internally they do deterministic computation.




```
x = PositionalEmbedding(max_length=128*128*3)(x)
x = tf.keras.layers.Dropout(0.3)(x)
for _ in range(10):
  y = LocalAttention1D(hparams, attention_type="local_mask_right",
                       q_padding="LEFT", kv_padding="LEFT")(x)
  x = LayerNormalization()(tf.keras.layers.Dropout(0.3)(y) + x)
  y = tf.keras.layers.Dense(x, 512, activation=tf.nn.relu)
  y = tf.keras.layers.Dense(512, activation=None)(y)
  x = LayerNormalization()(tf.keras.layers.Dropout(0.3)(y) + x)
x = layers.MixtureofLogistic(3, num_components=5)(x)
x = layers.Discretize(x)
```

Figure: PixelCNN++ Likelihood

4 Reversible Layers

Reversible layers map random variables to random variables. Internally, they do deterministic computation.


```
batch_size = 256
features = load_cifar10(batch_size)

model = tf.keras.Sequential([
  layers.RealNVP(MADE(hidden_dims=[512, 512])),
  layers.RealNVP(MADE(hidden_dims=[512, 512],
                      order='right-to-left')),
  layers.RealNVP(MADE(hidden_dims=[512, 512])),
])
base = ed.Normal(loc=tf.zeros([batch_size, 32*32*3]), scale=1.)
outputs = model(base)
loss = -tf.reduce_sum(outputs.distribution.log_prob(features))
train_op = tf.train.AdamOptimizer().minimize(loss)
```

Figure: RealNVP

Contact: trandustin@google.com Twitter: @dustinvtran