Bayesian Layers
A Module for Neural Network Uncertainty

Dustin Tran
Michael W. Dusenberry
Mark van der Wilk
Danijar Hafner

1Google Brain 2 PROWLER.io

Summary
Coding up uncertainty models at CIFAR+ scale leads to bad times.

How do you build a Bayesian ResNet? Uncertainty for model-based RL? Support GPU/TPU and flexible training?

Core contributions:
• No reinventing the wheel. No new language. No new abstractions.
• Extend neural network library semantics to distributions over functions.
• Everything composes!

What goes into a neural network library?

In all libraries, layers are the core building block. Optimizers, losses, metrics, and GPU/TPU strategies are designed around them.

Initializers.
Regularizers.
Compositionality.

Example: Deep Gaussian Process

```python
...init_({
    units,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
})

def loss_fn(features, labels):
    predictions = model(features)
    nll = tf.reduce_mean((labels-predictions).mean())
    kl = sum(model.losses) / dataset_size
    return nll + kl

model.compile('adam', loss_fn)
model.fit(features, labels, batch_size=64, epochs=100)
```

Example: Bayesian LSTM

```python
lstm = ed.layers.LSTMCellReparameterization(512)
output_layer = tf.keras.layers.Dense(10)

def loss_fn(features, labels, dataset_size):
    state = lstm.get_initial_state(features)
    nll = 0.
    for t in range(features.shape[1]):
        net, state = lstm(features[:, t], state)
        nll += tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(
                labels[:, t, logits]))
    kl = sum(lstm.losses) / dataset_size
    return nll + kl
```

Bayesian Layers

There are several design considerations for uncertainty models.

Computing the integral. We need to compute often-intractable integrals. For example in Bayesian neural networks:

$$ ELBO(\theta) = \int q(\theta) \log p(y \mid f_\theta(x)) \, d\theta - KL[q(\theta) \parallel p(\theta)], $$

$$ q(y \mid x) = \int q(\theta)p(y \mid f_\theta(x)) \, d\theta. $$

Make each estimator its own Layer.

Type Signature. Bayesian layers are drop-in replacements.

```python
...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(64, kernel_size=5, strides=2, padding='SAME'),
    tf.keras.layers.BatchNormalization(),
])
```

Current Work

Uncertainty baselines. High-quality, SOTA implementations of uncertainty models. (note: We already have CIFAR & ImageNet SOTA!)

• How does model uncertainty correlate with generalization?
• How do we combine SGD randomization with explicit prior/posterior randomization?
• How can uncertainty models achieve not just better robustness and reliability but better test accuracy?

Code: [github.com/google/edward2](https://github.com/google/edward2)