Deep Probabilistic Programming Languages: A Qualitative Study
Guillaume Baudart, Martin Hirzel, Louis Mandel, IBM Research

Deep Probabilistic Programming Languages

DeepPPLs, which have emerged just recently, aim to combine the benefits of probabilistic programming and deep learning models.

These languages can be used to define:
- Probabilistic deep models: weight uncertainty in deep neural networks
- Deep probabilistic models: probabilistic models using deep learning

Goal: Compare and characterize DeepPPLs, focusing on two languages:
- Edward, based on TensorFlow (static computation graph)
- Pyro, based on PyTorch (dynamic computation graph)

Probabilistic Deep Model

Example: Bayesian Multi-Layer Perceptron (MLP) with weight uncertainty
Lift network parameters to random variables

```
# Model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.l1 = torch.nn.Linear(nx, nh)
        self.l2 = torch.nn.Linear(nh, ny)
    def forward(self, x):
        h = torch.relu(self.l1(x))
        yhat = self.l2(h)
        return torch.log_softmax(yhat)
mlp = MLP()
```

```
def model(x, y):
    priors = {
        'by': Normal(loc=torch.zeros(ny, nh), scale=torch.ones(ny, nh)),
        'W2': Normal(torch.zeros(nh, ny), torch.ones(nh, ny))
    }
    z = pyro.sample('z', dist.Normal(z_mu, z_sigma))
    return mlp(z)
```

```
def predict(x):
    z_samples = ([z for _ in range(args.num_samples)])
    yhat = mlp(z_samples)
    return np.argmax(mean(yhat, axis=1))
```

(a) Pyro

Deep Probabilistic Model

Networks capture complex probabilistic dependencies with deep learning models

```
# Model
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.l1 = torch.nn.Linear(nx, nh)
        self.l2 = torch.nn.Linear(nh, nz)
        self.z_sigma = torch.nn.Linear(nh, nz)
    def forward(self, x):
        hidden = torch.relu(self.l1(x))
        z_mu = self.l2(hidden)
        z_sigma = softplus(self.z_sigma(hidden))
        return z_mu, z_sigma
encoder = Encoder()
```

```
def predict(x):
    x = tf.cast(x, tf.float32)
    hidden = tf.nn.relu(tf.matmul(x, theta['W1']) + theta['b1'])
    mu = tf.matmul(hidden, theta['W2']) + theta['b2']
    return mu
```

(b) Edward