Regression with Bayesian neural networks

Juho Lee
Grad school of AI
Samsung AI expert course

Heterogeneous regression

Prediction form:

$$y = \mu(x) + \sigma(x) \cdot \varepsilon.$$

Prediction at this point Uncertainty at this point

C. F. : Homogeneous regression (e.g., least square):

$$y = \mu(x) + \sigma \cdot \varepsilon.$$

Prediction at this point Shared uncertainty level

Objective function?

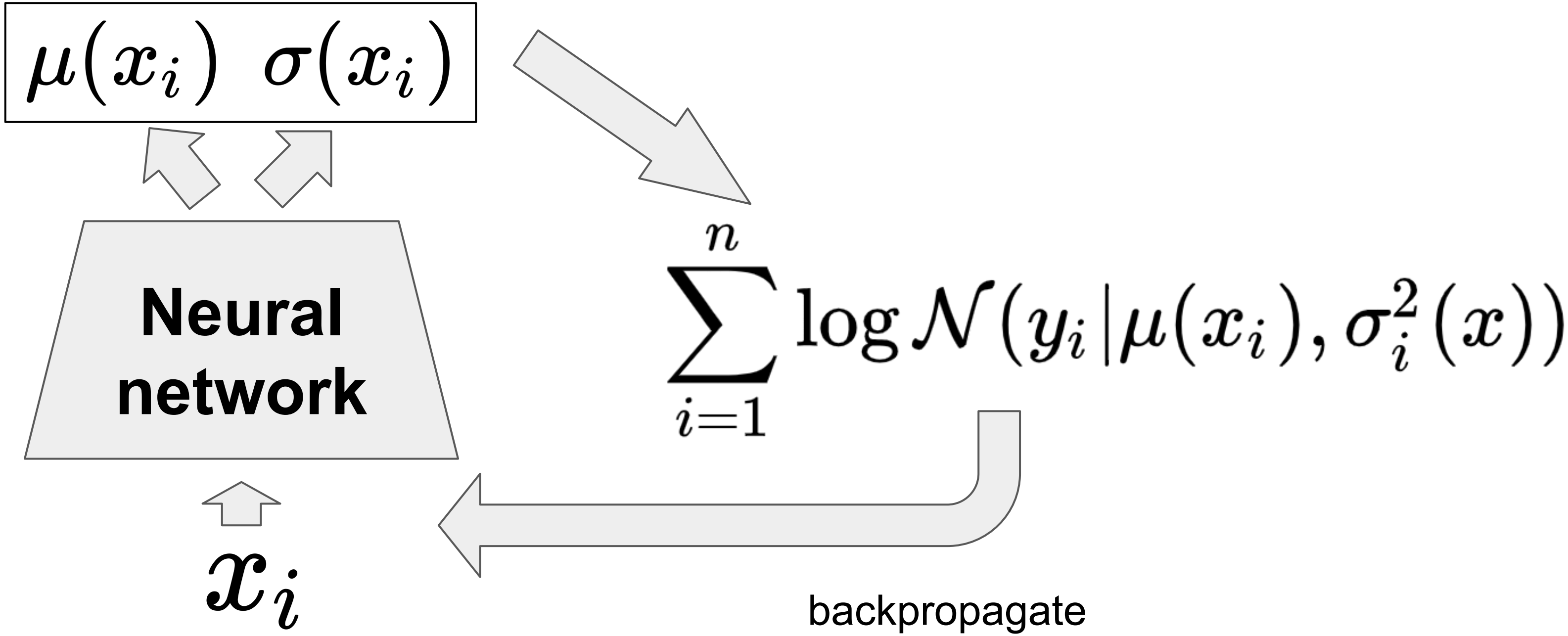
Probabilistic interpretation:

$$y_i \sim \mathcal{N}(\mu(x_i), \sigma_i^2(x))$$

Objective function: maximum likelihood!

$$\sum_{i=1}^{n} \log \mathcal{N}(y_i | \mu(x_i), \sigma_i^2(x))$$

Problem setting



`dense` layer in `tensorflow`

```python
def dense(x, num_units, name='dense', activation=None):
    with tf.variable_scope(name, reuse=tf.AUTO_REUSE):
        W = tf.get_variable('W', [x.shape[1], num_units])
        b = tf.get_variable('b', [num_units],
                initializer=tf.zeros_initializer())
        x = tf.matmul(x, W) + b
        if activation == 'relu':
            x = tf.nn.relu(x)
        return x
```

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

$$\boxed{\mathbb{R}^{N \times d_{\text{in}}}} \quad \boxed{\mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}} \quad \boxed{\mathbb{R}^{d_{\text{out}}}}$$

Bayes Dense?

- Keep the distribution of W and b
- The distributions? - Gaussians
- To define Gaussians, we need means and covariances.

`bayes_dense` layers in `tensorflow`

```
def bayes_dense(x, num_units, name='dense', gamma=1.0, activation=None):
    with tf.variable_scope(name, reuse=tf.AUTO_REUSE):
        W_mu = tf.get_variable('W_mu', [x.shape[1], num_units])
        W_rho = tf.nn.softplus(
                tf.get_variable('W_rho', [x.shape[1], num_units],
                    initializer=tf.random_uniform_initializer(-3., -2.)))
        b_mu = tf.get_variable('b_mu', [num_units],
                initializer=tf.zeros_initializer())
        b_rho = tf.nn.softplus(
                tf.get_variable('b_rho', [num_units],
                    initializer=tf.random_uniform_initializer(-3., -2.)))
```

$$q(\mathbf{W}) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{W}}, \boldsymbol{\rho}^2_{\mathbf{W}}) = \prod_{i,j} \mathcal{N}(\mu_{w_{ij}}, \rho^2_{w_{ij}})$$

$$q(\mathbf{b}) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{b}}, \boldsymbol{\rho}^2_{\mathbf{b}}) = \prod_{j} \mathcal{N}(\mu_{b_j}, \rho^2_{b_j})$$

Computing outputs in `bayes_dense`

```
# sample
W = W_mu + W_rho * tf.random.normal(W_mu.shape)
b = b_mu + b_rho * tf.random.normal(b_mu.shape)

x = tf.matmul(x, W) + b
if activation == 'relu':
    x = tf.nn.relu(x)
```

$$\mathbf{W} = \boldsymbol{\mu}_{\mathbf{W}} + \boldsymbol{\varepsilon} \odot \boldsymbol{\rho}_{\mathbf{W}}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\mathbf{b} = \boldsymbol{\mu}_{\mathbf{b}} + \boldsymbol{\varepsilon} \odot \boldsymbol{\rho}_{\mathbf{b}}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

KL-divergence

```
# kl divergence
kld_W = tf.reduce_sum(kl_divergence(Normal(W_mu, W_rho), Normal(0., gamma)))
kld_b = tf.reduce_sum(kl_divergence(Normal(b_mu, b_rho), Normal(0., gamma)))
kld = kld_W + kld_b
```

$$p(\mathbf{W}) = \mathcal{N}(\mathbf{0}, \gamma\mathbf{I}) \qquad p(\mathbf{b}) = \mathcal{N}(\mathbf{0}, \gamma\mathbf{I})$$

$$\mathrm{KL}[q(\mathbf{W})q(\mathbf{b})\|p(\mathbf{W})p(\mathbf{b})] = \mathrm{KL}[q(\mathbf{W})\|p(\mathbf{W})] + \mathrm{KL}[q(\mathbf{b})\|p(\mathbf{b})]$$

Objective function for Bayes neural net

$$-\sum_{i=1}^{n} \log \mathcal{N}(y_i | \mu(x_i), \sigma^2(x_i)) + \mathrm{KL}[q(\mathbf{W}) \| p(\mathbf{W})]$$

$$-\frac{1}{n}\sum_{i=1}^{n} \log \mathcal{N}(y_i | \mu(x_i), \sigma^2(x_i)) + \frac{1}{n}\mathrm{KL}[q(\mathbf{W}) \| p(\mathbf{W})]$$

$$-\frac{1}{n}\sum_{i=1}^{n} \log \mathcal{N}(y_i | \mu(x_i), \sigma^2(x_i)) + \frac{\lambda}{n}\mathrm{KL}[q(\mathbf{W}) \| p(\mathbf{W})]$$

Coefficient to balance regularization (`kl_coeff`)