Lab: Distributional Regression

ACTL3143 & ACTL5111 Deep Learning for Actuaries

CANN

1. Find the coefficients $\boldsymbol{\beta}_{\text{GLM}}$ of the GLM with a link function $g(\cdot)$.
2. Find the weights $\boldsymbol{w}_{\text{CANN}}$ of a neural network $\mathcal{M}_{\text{CANN}} : \mathbb{R}^{d_{\boldsymbol{x}}} \to \mathbb{R}$.
3. Given a new instance $\boldsymbol{x}$, we have

$$\mathbb{E}[Y|\boldsymbol{x}] = g^{-1}\Big(\langle \boldsymbol{\beta}_{\text{GLM}}, \boldsymbol{x} \rangle + \mathcal{M}_{\text{CANN}}(\boldsymbol{x}; \boldsymbol{w}_{\text{CANN}})\Big).$$

MDN

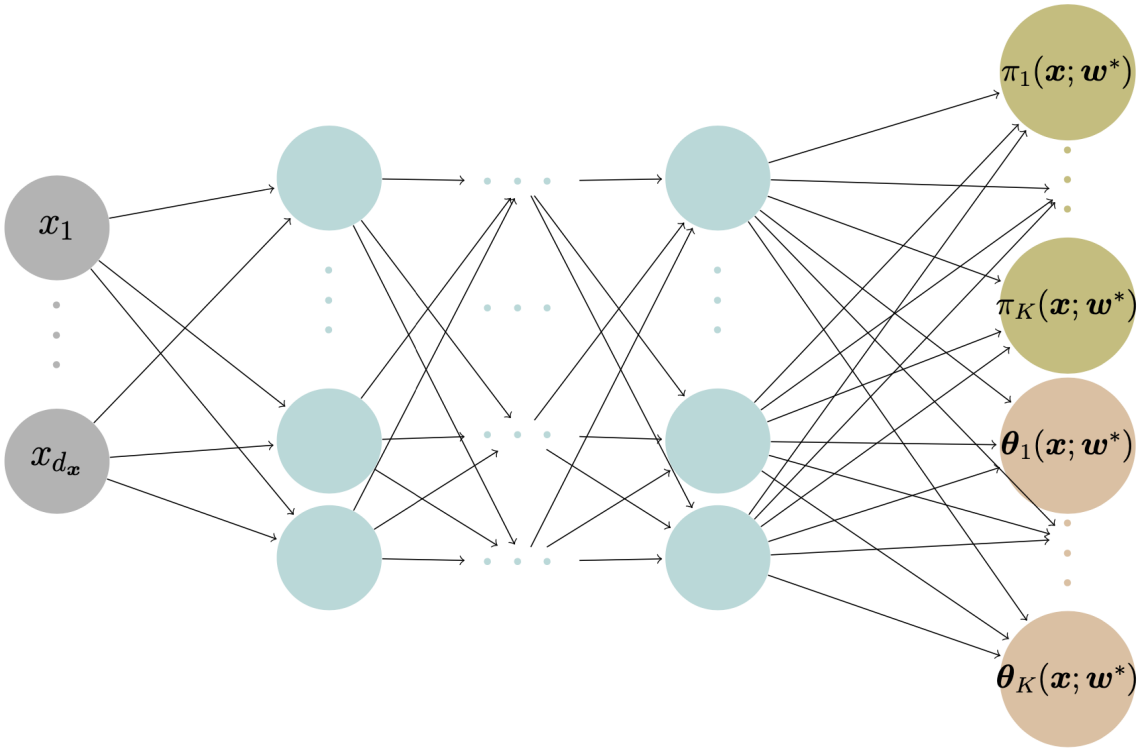

Figure 1: A typical MDN structure.

Exercises

CANN

1. Train a CANN model that predicts the mean as follows:

$$\mathbb{E}[Y|\boldsymbol{x}] = g^{-1}\Big(0.9 \cdot \langle \boldsymbol{\beta}_{\text{GLM}}, \boldsymbol{x}\rangle + 0.1 \cdot \mathcal{M}_{\text{CANN}}(\boldsymbol{x}; \boldsymbol{w}_{\text{CANN}})\Big).$$

   where $g^{-1}(\cdot) = \exp(\cdot)$. Hint: Check slides 20, 23, 24, and 25, and change the following line of code on slide 24.

```
def CANN_negative_log_likelihood(y_true, y_pred):
    ...
    mu = tf.math.exp(CANN_logmu + GLM_logmu)
```

2. Recompute the dispersion parameter using the adjusted model. Hint: use the code from slide 25 and change the following line of code

```
mus = np.exp(np.sum(CANN.predict(X_train), axis = 1))
```

MDN

1. Increase the number of mixture components to 5. You can use the code from slide 33.
2. Change the distributional assumption from gamma to inverse gamma for the mixture density network model. Hint: adjust the following code using this link.

```
mixture_distribution = tfd.MixtureSameFamily(
    mixture_distribution=tfd.Categorical(probs=pis),
    components_distribution=tfd.Gamma(alphas, betas))
```

3. Report the average negative log-likelihood loss (test data) using the new MDN. Hint: slides 36 and 39.

Extension

1. Compute the CRPS for the models trained in Exercise 1.3.1 and Exercise 1.3.2.
2. Build a Mixture Density Network (MDN), where the first component is a gamma distribution, the second component is a log-normal distribution, and the third component is an inverse gamma distribution.

Monte Carlo Dropout

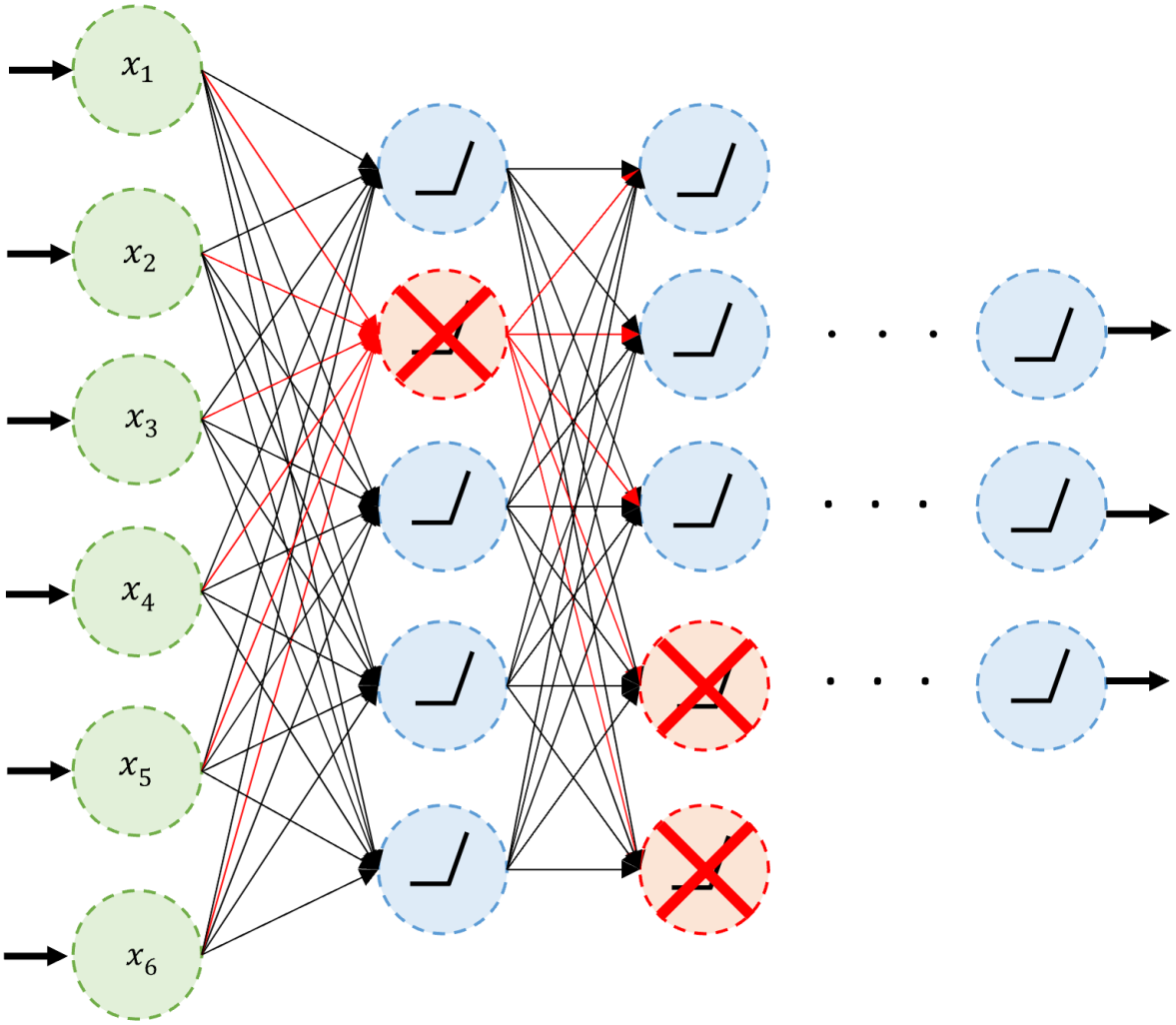

Figure 2: Dropout

For Monte Carlo (MC) dropout, we intentionally leave the dropout on when making predictions.

Deep Ensembles

- Train D neural networks with different random weights initialisations independently in parallel. The trained weights are $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(D)}$.

- Ensemble the outputs when making predictions, i.e., taking the average of the outputs from each individual neural network.

Exercises

Monte Carlo Dropout

1. Construct a neural network `MCDropout_LN` that outputs the parameters of a gamma distribution with the following structure and specification:

   - Use
     ```
     random.seed(1); tf.random.set_seed(1)
     ```
   - `Adam` optimiser with the default learning rate
   - validation split of 0.2 while training
   - two hidden layers with 64 neurons in each layer, and
   - a constant dropout rate of 0.2.

   Hint: the following code can be helpful

```
# Output the paramters of the gamma distribution
outputs = Dense(2, activation = 'softplus')(x)

# Construct the Gamma distribution on the last layer
distributions = tfp.layers.DistributionLambda(
      lambda t: tfd.Gamma(concentration=t[..., 0:1],
                          rate=t[..., 1:2]))(outputs)

# Model
MCDropout_LN = Model(inputs, distributions)

# Loss Function
def gamma_loss(y_true, y_pred):
    return -y_pred.log_prob(y_true)

# Then use the loss function when compiling the model
MCDropout_LN.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                loss=gamma_loss)
```

2. Apply MC dropout 2000 times and store the parameter estimates for the first instance in the test dataset using the model `MCDropout_LN`. Hint: slide 61, and replace

```
predicted_distributions = gamma_bnn(X_test[9:10].values)
```

with

```
predicted_distributions = MCDropout_LN(X_test[:1].values, training = True)
```

3. Calculate the aleatoric and epistemic uncertainty for the instance using equations (**??**) and (**??**). Hint: slide 64.

Deep Ensembles

1. Reuse the code demonstrated in the lecture and calculate the aleatoric and epistemic uncertainty for the first instance in the test dataset using equations (**??**) and (**??**). Hint: slides 66, 67, and 68.

Extension

1. Prove the result on slide 55.
2. Replace the variational distribution with a mixture of Gaussian for the BNN introduced in Exercise 2.5.2.