CS6462

Probabilistic and Explainable AI

Lesson 20

Bayesian Neural Networks

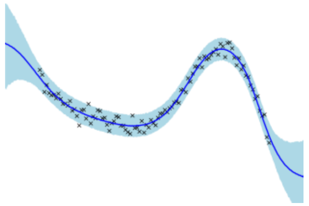
*

Bayesian Linear Regression

by Emil Vassev

March 10, 2025

Optimization in Machine Learning



Definition:

- optimization problem is about maximizing or minimizing a function by systematically choosing input values from a set, so to compute an optimal value of that function
- the process of adjusting hyperparameters in order to minimize the cost (loss) function by using one of the optimization techniques

Minimizing the cost (loss) function:

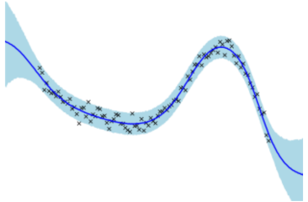
- minimizes the discrepancy between the true value and prediction

Example:

- $\boldsymbol{X}$ and $\boldsymbol{Y}$ are random variables: $\boldsymbol{X = \{x_1, \ldots, x_n\}}$, $\boldsymbol{Y = \{y_1, \ldots, y_n\}}$
- fit a function between $\boldsymbol{X}$ and $\boldsymbol{Y}$, e.g., $\boldsymbol{Y = a_1 + a_2 * X}$ (for any sample in $\boldsymbol{X}$ and $\boldsymbol{Y}$)
- generalized version: $\boldsymbol{y_n = a_1 + a_2 * x_n}$
- cost function: $\boldsymbol{c_n = y_n - a_1 - a_2 * x_n}$

objective: minimize c_n

Cost Function



Specifics:

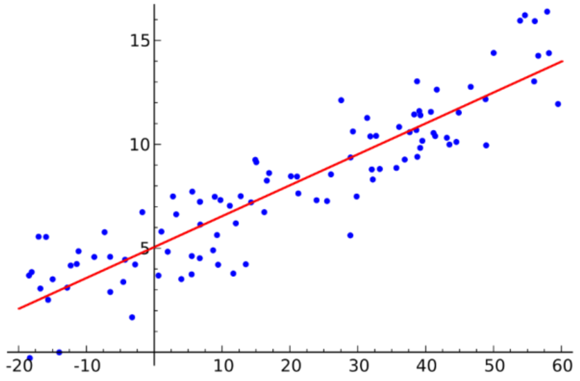
- measures the performance of a machine learning model for any given data
- quantifies the error between predicted values and expected values and presents it in the form of a single real number
- to calculate the Cost function, we make hypothesis with initial hyperparameters
- to reduce the cost function, we modify the hyperparameters by using an optimization algorithm (e.g., gradient descent) over the given data

goal of accurate machine learning models: *minimizing the cost of prediction*

Difference between cost function and loss function:

- cost function is the average error of samples in the data (for the whole training data)
- loss function is the error for individual data points (for one training example)

Cost Function For Linear Regression



Linear Regression:

- data is said to "regress" to a mean
- regression with one independent variable and a linear relationship between the independent and dependent variable

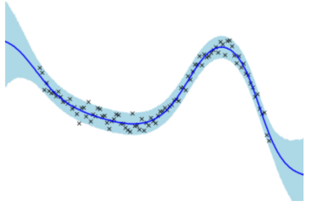
$$y = f(x) = \theta_0 + \theta_1 * x$$

Mathematical definition: Mean Squared Error (MSE)

- *Hypothesis:* $h_0(x) = \theta_0 + \theta_1 x$
- *Parameters:* θ_0, θ_1
- *Cost Function:* $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_0(x^{(i)}) - y^{(i)})^2$
- *Objective:* $\widehat{\theta} = argmin(C_f(\theta))$

m - number of training examples
$x^{(i)}$ - the input vector of the i-th training example
$y^{(i)}$ - class label for the i-th training example
θ – choice of parameters θ_0, θ_1

objective: consider the cost function $C_f(\theta)$ to find values of θ_0, θ_1 that minimize it



Bayesian Neural Linear Model

$$\mathit{Posterior} = \frac{\mathit{Likelyhood} * \mathit{Prior}}{\mathit{Evidence}}$$

Bayesian Neural Networks (revision):

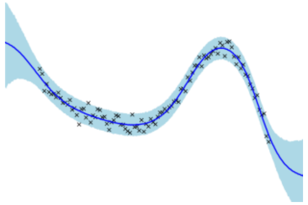
- *objective:*
  - uncover the full posterior distribution over the network weights
  - capture uncertainty and act as a ML regularizer
  - provide a framework for model comparison
- problems:
  - full posterior is intractable for most forms of neural networks
  - requires expensive approximate inference or Markov chain Monte Carlo simulation

Neural Linear Model:

- efficient way to get posterior samples and uncertainty out of a regular neural network
- adds a Bayesian linear regression on top of the last hidden layer of a regular (non-Bayesian, non-probabilistic) neural network
- initially proposed for Bayesian optimization*

*Snoek et al., Scalable Bayesian Optimization Using Deep Neural Networks, 2015

Bayesian Neural Linear Model (cont.)



Training a Neural Linear Model:

- a *Neural Linear Model* - a Bayesian linear regression in a projected feature space

$NLM < \boldsymbol{D}, \phi >$

$\boldsymbol{D} = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{\boldsymbol{i=1}}^{\boldsymbol{n}}, \boldsymbol{x_i} \in \mathbb{R}^{\boldsymbol{k}}, \boldsymbol{X} \in \mathbb{R}^{\boldsymbol{n \times k}}, \boldsymbol{y_i} \in \mathbb{R}, \boldsymbol{Y} \in \mathbb{R}^{\boldsymbol{n}}$ - observed data (training space)

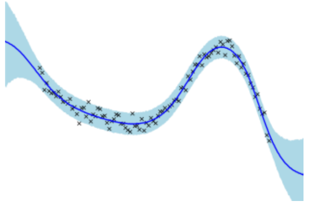
$\phi(\cdot; \theta): \mathbb{R}^{\boldsymbol{k}} \rightarrow \mathbb{R}^{\boldsymbol{m}}$: - a feature space projection with parameters θ and features

$\phi_i = \phi(\boldsymbol{x_i}; \theta)$: individual features

- Bayesian linear regressions have a closed form solution
- *Steps*:
  1) train the neural network to minimize the Mean Squared Error
  2) compute the closed-form solution to the Bayesian linear regression regressing the last hidden layer's activations onto the target variable
     - use variational inference (covered in next lecture) to train the neural network and Bayesian regression together end-to-end

*Snoek et al., Scalable Bayesian Optimization Using Deep Neural Networks, 2015

Bayesian Neural Linear Model with Python

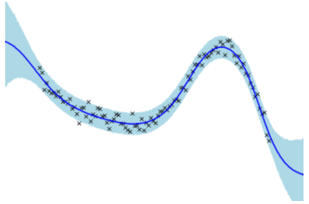


ProbFlow Library:*

- Python package for building probabilistic Bayesian Models with TensorFlow 2.0 and PyTorch
- performs variational inference and evaluates models; inference results
- high-level models for building Bayesian Neural Networks
- low-level parameter and distribution models for building custom-based Bayesian models
- core building blocks of a Bayesian model – parameters, probability distributions, input data
- parameters - define how the independent variables (the features) predict the probability distribution of the dependent variables (the target)

*[ProbFlow on GitHub]

BNLM with ProbFlow (Example)



Neural Linear Model:*

Dense, *DenseNetwork*, *DenseRegression* - use Normal distributions for variational posteriors

probabilistic – Boolean parameter; when = false, ProbFlow does not model any uncertainty, i.e., we create a non-Bayesian neural network

Steps:

1) create a regular non-probabilistic neural network using *DenseNetwork* with probabilistic = False

2) perform a Bayesian linear regression on top of the final hidden layer using the *Dense* class

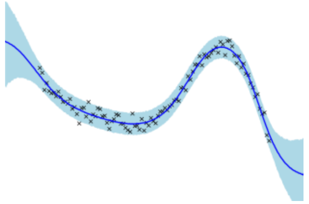
```
import probflow as pf
import probflow.utils.ops as O

class NeuralLinear(pf.ContinuousModel):

    def __init__(self, dims):
        self.net = pf.DenseNetwork(dims, probabilistic=False)
        self.loc = pf.Dense(dims[-1], 1)
        self.std = pf.Dense(dims[-1], 1)

    def __call__(self, x):
        h = O.relu(self.net(x))
        return pf.Normal(self.loc(h), O.softplus(self.std(h)))
```

*[ProbFlow on GitHub](#)



BNLM with ProbFlow (Example – cont.)

Steps (cont.):

3) *instantiate the model*: use a fully-connected sequential architecture, where the first hidden layer has 128 units, the second has 64, and the third has 32

```
model = NeuralLinear([x.shape[1], 256, 128, 64, 32])
```

4) *set callback function: MonitorMetric* to monitor the Mean Absolute Error of the model's predictions while training

```
nlm_mae = pf.MonitorMetric('mae', x_val, y_val)
nlm_lrs = pf.LearningRateScheduler(lambda e: 5e-4 + e * 5e-6)
```

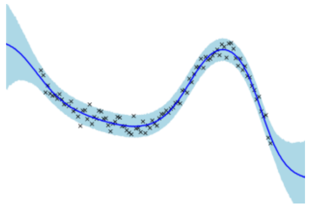
5) *Tune the learning rate:* use the *LeraningRateScheduler* to provide a dynamic learning rate

6) *fit the model*

```
nlm_model.fit(
    x_train,
    y_train,
    batch_size=2048,
    epochs=100,
    callbacks=[nlm_mae, nlm_lrs]
)
```

*[ProbFlow on GitHub](#)

Summary



Bayesian Neural Networks – *Bayesian Linear Regression*

Optimization in Machine Learning

Cost Function

Cost Function for Linear Regression

Bayesian Neural Linear Model

Bayesian Neural Linear Model with Python (ProbFlow)

Next Lesson:

- Bayesian Neural Networks - Posterior Variational Inference

Thank You!

Questions?