Computational Statistics and Data Analysis (MVComp2)

Solutions to exercise 9

Lecturer Tristan Bereau

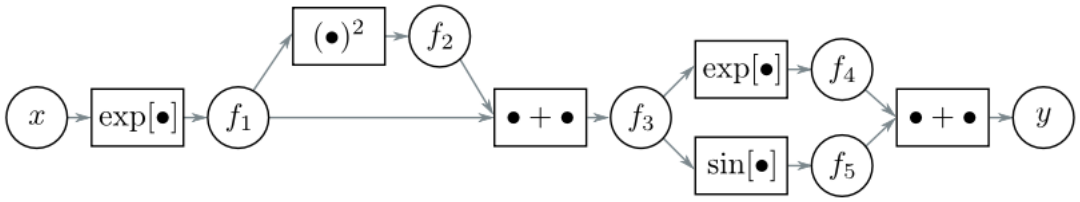
Semester Wi23/24
Due Jan. 11, 2024, 23:59

1 Forward- and reverse-mode differentiation (2 points)

We explore the computation of derivatives on general acyclic computational graphs. Consider the function

$$y = \exp\left[\exp(x) + \exp(x)^2\right] + \sin\left[\exp(x) + \exp(x)^2\right]$$

whose computational graph is depicted in the figure, together with the intermediate functions f_1, f_2, f_3, f_4, f_5.



(a) Compute the derivative $\partial y/\partial x$ by *forward-mode differentiation*. In other words, compute in order

$$\frac{\partial f_1}{\partial x}, \frac{\partial f_2}{\partial x}, \frac{\partial f_3}{\partial x}, \frac{\partial f_4}{\partial x}, \frac{\partial f_5}{\partial x}, \text{ and } \frac{\partial y}{\partial x},$$

using the chain-rule in each case to make use of the derivatives already computed.

(b) Compute the derivative $\partial y/\partial x$ by *reverse-mode differentiation*. In other words, compute in order

$$\frac{\partial y}{\partial f_5}, \frac{\partial y}{\partial f_4}, \frac{\partial y}{\partial f_3}, \frac{\partial y}{\partial f_2}, \frac{\partial y}{\partial f_1}, \text{ and } \frac{\partial y}{\partial x},$$

using the chain-rule in each case to make use of the derivatives already computed.

1.1 Solution

We first express the intermediate functions

$$\begin{aligned} f_1 &= \exp(x) \\ f_2 &= f_1^2 \\ f_3 &= f_1 + f_2 \\ f_4 &= \exp(f_3) \\ f_5 &= \sin(f_3) \\ y &= f_4 + f_5 \end{aligned}$$

(a) The forward-mode derivatives yield

$$\begin{aligned}
\frac{\partial f_1}{\partial x} &= \exp(x) \\
\frac{\partial f_2}{\partial x} &= \frac{\partial f_2}{\partial f_1}\frac{\partial f_1}{\partial x} = 2\exp(2x) \\
\frac{\partial f_3}{\partial x} &= \frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial x} + \frac{\partial f_3}{\partial f_1}\frac{\partial f_1}{\partial x} = 2\exp(2x) + \exp(x) \\
\frac{\partial f_4}{\partial x} &= \frac{\partial f_4}{\partial f_3}\frac{\partial f_3}{\partial x} = \exp\left[\exp(x) + \exp(2x)\right](2\exp(2x) + \exp(x)) \\
\frac{\partial f_5}{\partial x} &= \frac{\partial f_5}{\partial f_3}\frac{\partial f_3}{\partial x} = \cos\left[\exp(x) + \exp(2x)\right](2\exp(2x) + \exp(x)) \\
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial f_4}\frac{\partial f_4}{\partial x} + \frac{\partial y}{\partial f_5}\frac{\partial f_5}{\partial x} \\
&= \exp\left[\exp(x) + \exp(2x)\right](2\exp(2x) + \exp(x)) + \cos\left[\exp(x) + \exp(2x)\right](2\exp(2x) + \exp(x)) \\
&= \left\{\exp\left[\exp(x) + \exp(2x)\right] + \cos\left[\exp(x) + \exp(2x)\right]\right\}(2\exp(2x) + \exp(x))
\end{aligned}$$

(b) The reverse-mode derivatives yield

$$\begin{aligned}
\frac{\partial y}{\partial f_5} &= 1 \\
\frac{\partial y}{\partial f_4} &= 1 \\
\frac{\partial y}{\partial f_3} &= \frac{\partial y}{\partial f_4}\frac{\partial f_4}{\partial f_3} + \frac{\partial y}{\partial f_5}\frac{\partial f_5}{\partial f_3} = \exp(f_3) + \cos(f_3) \\
\frac{\partial y}{\partial f_2} &= \frac{\partial y}{\partial f_3}\frac{\partial f_3}{\partial f_2} = \exp(f_1 + f_2) + \cos(f_1 + f_2) \\
\frac{\partial y}{\partial f_1} &= \frac{\partial y}{\partial f_2}\frac{\partial f_2}{\partial f_1} + \frac{\partial y}{\partial f_3}\frac{\partial f_3}{\partial f_1} = \left[\exp(f_1 + f_2) + \cos(f_1 + f_2)\right](2f_1 + 1) \\
\frac{\partial y}{\partial x} &= \frac{\partial y}{\partial f_1}\frac{\partial f_1}{\partial x} \\
&= \left[\exp(f_1 + f_2) + \cos(f_1 + f_2)\right](2f_1 + 1)\exp(x) \\
&= \exp\left[\exp(x) + \exp(2x)\right](2\exp(2x) + \exp(x)) + \cos\left[\exp(x) + \exp(2x)\right](2\exp(2x) + \exp(x)) \\
&= \left\{\exp\left[\exp(x) + \exp(2x)\right] + \cos\left[\exp(x) + \exp(2x)\right]\right\}(2\exp(2x) + \exp(x))
\end{aligned}$$

2 Second moment of ReLU (2 points)

Consider the continuous random variable X with symmetrical distribution around the mean $E[X] = 0$ and variance $\mathrm{Var}[X] = \sigma^2$. e pass this variable through the ReLU function to obtain the transformed variable, B, such that

$$B(x) = \mathrm{ReLU}[x] = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}.$$

Prove that the second moment around the origin of the transformed variable is $E[B^2] = \sigma^2/2$.

2.1 Solution

Because of the definition of the ReLU function we only consider the positive half of X when calculating $E[B^2]$

$$E[B^2] = \int_0^\infty \mathrm{d}x\, x^2 \ \mathrm{ReLU}(x).$$

Since the distribution of X is symmetric around 0 and $\mathrm{Var}[X] = \sigma^2$, the total variance is equally distributed over the positive and negative halves of the distribution.

Further, recall the expression for the variance

$$\mathrm{Var}[X] = E[X^2] - E[X]^2.$$

Given $E[X] = 0$, the variance simplifies to

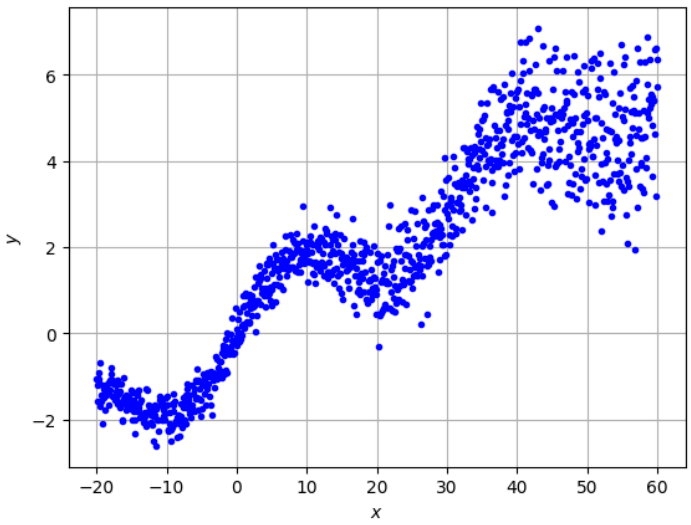
$$\mathrm{Var}[X] = E[X^2].$$

Because of the symmetry of the distribution and that B only represents the positive part of the distribution of X, the second moment of b is half of $\mathrm{Var}[X]$. Therefore,

$$E[B^2] = \frac{\sigma^2}{2}.$$

3 Heteroskedastic regression (6 points)

Please download the following dataset: x_y.csv. It contains 1,000 datapoints with 1-dimensional inputs, $\boldsymbol{x}$, and 1-dimensional outputs, y, as shown in the following figure



(0) Download the dataset, and randomly split it into a training and a test set with ratio (66/33%).

(a) We model the distribution of outputs as

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(y|f_\mu(\boldsymbol{x}), \sigma^2),$$

where $f_\mu(\boldsymbol{x})$ will be a multilayer perceptron (MLP), and we assume that all measurement errors are identical (i.e., homoskedastic regression), $\sigma(\boldsymbol{x}) = \sigma$. Explain why the mean-squared error is a reasonable log-likelihood function.

(b) Build an MLP with the following architecture: 3 fully-connected linear layers with hidden dimensions 32 and 64 (i.e., the dimensionality of your network should yield $1 \to 32 \to 64 \to 1$). Connect them with the ReLU activation function. Optimize the log-likelihood on the training set. Feel free to use any optimizer (e.g., stochastic gradient descent or Adam). Plot the resulting model on the test set together with the reference datapoints. Are you underfitting, overfitting?

(c) One standard way to address the prediction of heteroskedastic data for regression is to predict both the mean and the variance of a Normal distribution: $f_\mu(\boldsymbol{x}) = E[y|\boldsymbol{x}, \boldsymbol{\theta}]$ and $f_{\sigma^2}(\boldsymbol{x}) = \text{Var}[y|\boldsymbol{x}, \boldsymbol{\theta}]$. Derive a log-likelihood function for this heteroskedastic problem.

(d) Modify the MLP to output two values: the mean and the variance. Optimize the model. Plot the mean and variance predicted by your model on the test set together with the reference datapoints. Are you underfitting, overfitting?

Hint: In case you use PyTorch, you may find the following functions useful to reshape tensors and arrays: `numpy.reshape` and `torch.squeeze`.

3.1 Solutions

(0) Read the `csv` file and split into training and test sets

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

df = pd.read_csv("data_09_x_y.csv")
x = df["x"].to_numpy().reshape(-1, 1)
y = df["y"].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
X_train.shape, X_test.shape, y_train.shape
```

```
(torch.Size([670, 1]), torch.Size([330, 1]), torch.Size([670]))
```

(a) For a homoskedastic model, the Normal distribution has a variance that does not depend on the input. Thus the log likelihood reduces to the mean-squared error, as seen in the lecture.

(b) Define the model

```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
```

```
        self.fc1 = nn.Linear(1, 32)  # First layer
        self.fc2 = nn.Linear(32, 64)  # Second layer
        self.fc3 = nn.Linear(64, 1)
        self.relu = nn.ReLU()  # Define ReLU here

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        return self.fc3(x).squeeze()
```

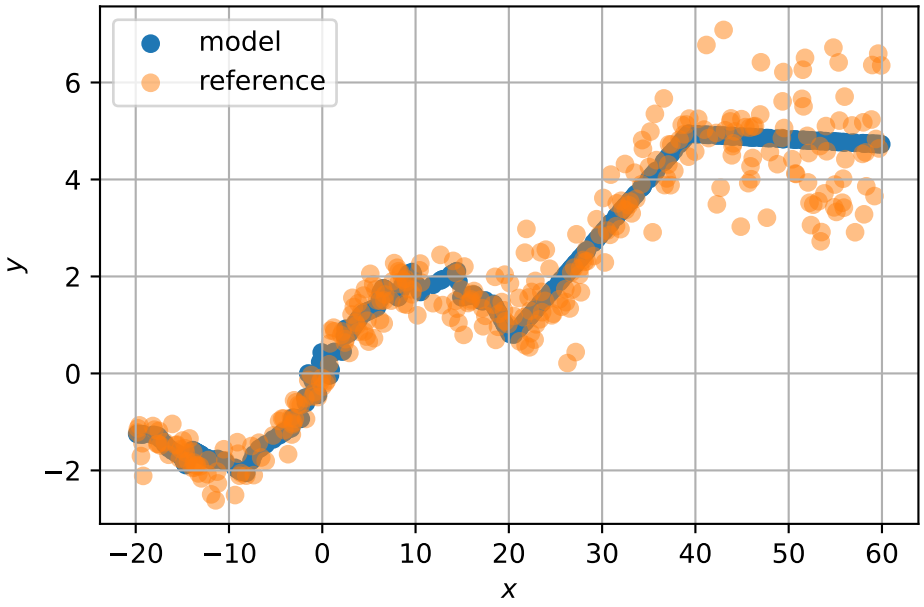
And run the optimization using Adam on 10,000 epochs

```
model = SimpleNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

num_epochs = 20000
train_curve = []
val_curve = []
for epoch in range(num_epochs):
    # Training
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    train_loss = loss.item()
    optimizer.step()

    # Validation
    model.eval()
    with torch.no_grad():  # Disable gradient computation
        outputs = model(X_test)
        val_loss = criterion(outputs, y_test)
    if (epoch + 1) % 2000 == 0:
        train_curve.append(train_loss)
        val_curve.append(val_loss)
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}'
```

```
Epoch [2000/20000], Loss: 0.6639, Val Loss: 0.6671
Epoch [4000/20000], Loss: 0.6634, Val Loss: 0.6661
Epoch [6000/20000], Loss: 0.6637, Val Loss: 0.6739
Epoch [8000/20000], Loss: 0.6634, Val Loss: 0.6732
Epoch [10000/20000], Loss: 0.6619, Val Loss: 0.6688
Epoch [12000/20000], Loss: 0.6611, Val Loss: 0.6748
Epoch [14000/20000], Loss: 0.6612, Val Loss: 0.6673
Epoch [16000/20000], Loss: 0.6766, Val Loss: 0.6957
Epoch [18000/20000], Loss: 0.4244, Val Loss: 0.4301
Epoch [20000/20000], Loss: 0.4238, Val Loss: 0.4368
```

Finally we plot the model and the reference labels on the test set

```
model.eval()  # Set the model to evaluation mode
with torch.no_grad():  # Disable gradient computation
    outputs = model(X_test)
plt.scatter(X_test, outputs, label="model")
plt.scatter(X_test, y_test, label="reference", alpha=0.5)
plt.legend()
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")
plt.grid()
```



Clearly we are underfitting, because the model does not reproduce the scatter of the points.

(c) Consider the heteroskedastic problem. The likelihood yields

$$\mathcal{L}(\boldsymbol{x}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi f_{\sigma^2}(\boldsymbol{x})}} \exp\left[-\frac{1}{2}\frac{(y - f_{\mu}(\boldsymbol{x})^2)}{f_{\sigma^2}(\boldsymbol{x})}\right]$$

$$-\log \mathcal{L}(\boldsymbol{x}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{1}{2}\log f_{\sigma^2}(\boldsymbol{x}) + \frac{(y - f_{\mu}(\boldsymbol{x}))^2}{2 f_{\sigma^2}(\boldsymbol{x})}$$

(d) Modify the neural network and define the new loss function

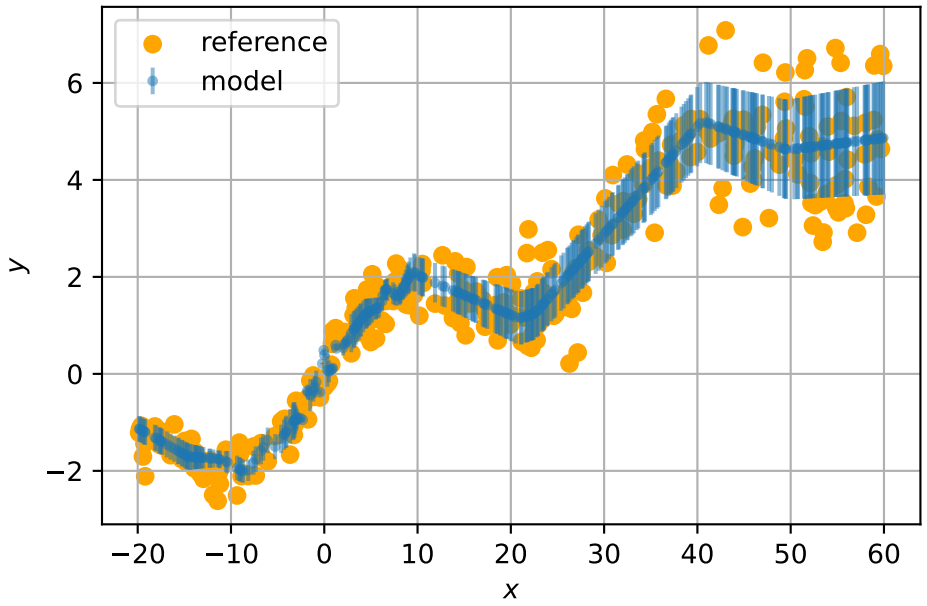
```
class TwoHeadNN(nn.Module):
    def __init__(self):
        super(TwoHeadNN, self).__init__()
        self.fc1 = nn.Linear(1, 32)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(32, 64)
```

```
        self.fc3 = nn.Linear(64, 2)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        output = self.fc3(x)
        mean = output[:, 0]
        variance = torch.exp(output[:, 1])  # Ensuring variance is non-negative
        return mean, variance

model = TwoHeadNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
def custom_loss(mean, variance, y):
    return torch.mean(0.5 * torch.log(variance) + 0.5 * ((y - mean) ** 2) / variance)
```

And optimize

```
num_epochs = 20000
train_curve = []
val_curve = []
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    mean, variance = model(X_train)
    loss = custom_loss(mean, variance, y_train)
    loss.backward()
    train_loss = loss.item()
    optimizer.step()

    # Validation
    model.eval()  # Set the model to evaluation mode
    with torch.no_grad():  # Disable gradient computation
        mean, variance = model(X_test)
        val_loss = custom_loss(mean, variance, y_test)
    if (epoch + 1) % 2000 == 0:
        train_curve.append(train_loss)
        val_curve.append(val_loss)
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}'

model.eval()  # Set the model to evaluation mode
with torch.no_grad():  # Disable gradient computation
    mean, variance = model(X_test)
plt.errorbar(X_test, mean, yerr=np.sqrt(variance), fmt=".", alpha=0.5, label="model")
plt.scatter(X_test, y_test, c="orange", label="reference")
plt.legend()
plt.xlabel(r"$x$")
plt.ylabel(r"$y$")
plt.grid()
```

```
Epoch [2000/20000], Loss: -0.0588, Val Loss: 0.0493
Epoch [4000/20000], Loss: -0.0673, Val Loss: 0.0957
Epoch [6000/20000], Loss: -0.0764, Val Loss: 0.1258
Epoch [8000/20000], Loss: -0.0866, Val Loss: 0.1600
Epoch [10000/20000], Loss: -0.1212, Val Loss: 0.1966
Epoch [12000/20000], Loss: -0.2352, Val Loss: 0.1440
Epoch [14000/20000], Loss: -0.2439, Val Loss: 0.3048
Epoch [16000/20000], Loss: -0.2477, Val Loss: 0.6413
Epoch [18000/20000], Loss: -0.2577, Val Loss: 1.4307
Epoch [20000/20000], Loss: -0.2595, Val Loss: 2.7039
```



Compared to the original neural network, we are underfitting much less, thanks to the spatial modeling of the variance.