Implementasi Mini-batch Gradient Descent

Dibuat sebagai Salah Satu Komponen Pengumpulan Milestone 2 Tugas Besar Pembelajaran Mesin IF3170

Oleh:

Henry Anand Septian Radityo	13521004
Matthew Mahendra	13521007
Hidayatullah Wildan Ghaly Buchary	13521015
Ahmad Nadil	13521024

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

Daftar Isi

Daftar Tabel

BAB 1

Implementasi

1.1 Implementasi Backpropagation

Backpropagation adalah metode pelatihan jaringan saraf yang populer. Metode ini menyesuaikan bobot secara iteratif untuk meminimalkan kesalahan yang dihasilkan oleh jaringan. Dalam kelas ANN, implementasi algoritma backpropagation dilakukan dengan memperhatikan arsitektur jaringan tersebut. Proses iteratif dilakukan dalam urutan terbalik, dengan $j = 0$ merepresentasikan lapisan keluaran (output layer) dan nilai j lainnya merepresentasikan lapisan tersembunyi (hidden layers). Berbagai fungsi aktivasi dapat diimplementasikan, termasuk ReLU, sigmoid, linear, dan softmax.

1.1.1 Fungsi Aktivasi dan Turunannya

Turunan ini digunakan dalam perhitungan gradien yang diperlukan untuk pembaruan bobot.

Turunan Fungsi Aktivasi ReLU

ReLU, atau Rectified Linear Unit, memiliki turunan sederhana:

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{1.1}$$

Turunan Fungsi Aktivasi Sigmoid

Fungsi sigmoid sering digunakan karena sifatnya yang non-linear dan kemampuannya untuk menangani probabilitas:

$$\frac{d}{dx}\sigma(x) = \sigma(x) \times (1 - \sigma(x)) \tag{1.2}$$

Turunan Fungsi Aktivasi Linear

Fungsi linear adalah yang paling sederhana dengan turunan yang konstan:

$$\frac{d}{dx}x = 1 \tag{1.3}$$

Turunan Fungsi Aktivasi Softmax

Softmax biasa digunakan pada lapisan keluaran untuk klasifikasi multikelas:

$$\frac{\partial E_u}{\partial \text{net}_j(x)} = \begin{cases} p_j, & j \neq \text{target} \\ -(1-p_j), & j = \text{target} \end{cases} \tag{1.4}$$

1.1.2 Fungsi Kerugian (Loss Functions)

Fungsi kerugian adalah metrik yang mengukur seberapa baik jaringan saraf melaksanakan tugas prediksi. Nilai kerugian yang rendah menandakan kesesuaian yang lebih tinggi antara prediksi dan nilai aktual.

Fungsi Kerugian untuk ReLU, Sigmoid, dan Linear

Ketika menggunakan aktivasi ReLU, sigmoid, atau linear, kita biasanya mengimplementasikan fungsi kerugian kuadratik, yang didefinisikan sebagai berikut:

$$E = \frac{1}{2} \sum_{k \in \text{output}} (t_k - o_k)^2 \tag{1.5}$$

Fungsi Loss untuk Aktivasi Softmax: Untuk kasus khusus aktivasi Softmax yang sering digunakan pada lapisan keluaran untuk klasifikasi multi-kelas, fungsi kerugian yang digunakan adalah:

$$E = -\log(p_k), \quad k = \text{target} \tag{1.6}$$

1.1.3 Update Bobot

Pembaruan bobot dalam jaringan saraf dilakukan dengan mengikuti aturan update gradien turun, yang dapat dijelaskan dalam dua langkah:

- **Menghitung Delta Bobot:** Perubahan bobot, Δw, dihitung sebagai hasil kali negatif dari gradien dan learning rate η.

$$\Delta w = -\text{gradient} \times \eta \tag{1.7}$$

- **Menghitung Bobot Baru:** Bobot baru, w_{new}, dihitung dengan menambahkan Δw ke bobot lama, w_{old}.

$$w_{\text{new}} = w_{\text{old}} + \Delta w \tag{1.8}$$

1.1.4 Update Bias

Perhitungan bobot bias dilakukan dengan perhitungan yang serupa dengan perhitungan bobot, namun untuk gradiennya, tidak memiliki input, sehingga dihitung menggunakan perhitungan 1.9. Dengan L, nilai loss, b_i bias, dan layer z_i

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial z_i} \cdot \frac{\partial z_i}{\partial b_i} \tag{1.9}$$

1.1.5 Persamaan Gradien

Perhitungan gradien tergantung pada lapisan dimana bobot berada yaitu hidden layer dan output layer.

Gradien pada Hidden Layer

Untuk semua jenis aktivasi, gradien pada lapisan tersembunyi dihitung sebagai berikut:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \text{net}} \cdot \frac{\partial \text{net}}{\partial w} \tag{1.10}$$

Gradien pada Output Layer

Pada lapisan keluaran, perhitungan gradien dibedakan berdasarkan jenis aktivasi yang digunakan. *Untuk Aktivasi Softmax:*

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \text{net}} \cdot \frac{\partial \text{net}}{\partial w} \tag{1.11}$$

Untuk Aktivasi Lainnya: Untuk aktivasi selain Softmax, gradien dihitung menggunakan perantara output lapisan tersebut, Out:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \text{Out}} \cdot \frac{\partial \text{Out}}{\partial \text{net}} \cdot \frac{\partial \text{net}}{\partial w} \tag{1.12}$$

1.2 Implementasi Mini Batch Stochastic Gradient Descent

Mini Batch Stochastic Gradient Descent bertujuan untuk membuat loss menjadi semakin kecil dengan melakukan pemrosesan untuk setiap batch berukuran n data. *Update* bobot w_k dan bias dilakukan pada akhir setiap batch dan untuk setiap data, dihitung Δw_k yang merupakan modifikasi dari 1.7 dan 1.8. Δw_k diinisialisasi 0 terlebih dahulu. Perhitungan 1.13 dilakukan untuk setiap step dan update bobot menggunakan 1.8 dengan Δw menjadi Δw_k.

$$\Delta w_k = \Delta w_k - \text{gradient} \times \eta \tag{1.13}$$

1.3 Parameter Pelatihan

Model memiliki method train yang berfungsi untuk melakukan pelatihan neural network (forward dan backpropagation). Parameter yang digunakan adalah sebagai berikut,

1. epoch: int (jumlah iterasi maksimal)
2. learning_rate: float (nilai learning rate)
3. batch_size: int (ukuran batch)
4. error_threshold: float (error threshold maksimum sebelum stop)

1.4 Contoh Penggunaan Secara Manual

Berikut adalah contoh penggunaan model dengan instansiasi secara manual. Model yang dibangun adalah model dengan 3 layer. Layer 1 memiliki 2 neuron dengan fungsi aktivasi ReLU dan bobotnya. Layer 2 memiliki 2 neuron dengan fungsi aktivasi sigmoid. Layer 3 memiliki 3 neuron dengan fungsi aktivasi sigmoid.

```
modelIris = ANN(
    [
        Layer('relu', 2, np.array([[0.459, 0.459], [0.068, -0.193],
           [0.244, -0.298], [-0.412, -0.22]]), np.array([-0.22, 0.445])
           ),
        Layer('sigmoid', 2, np.array([[-0.059, -0.331], [-0.355,
           -0.195]]), np.array([-0.05, -0.09])),
        Layer('sigmoid', 3, np.array([[-0.391, -0.187, -0.173],
           [-0.221, 0.164, -0.395]]), np.array([0.165, 0.086, 0.461]))
    ]
)

modelIris.fit(X_train.to_numpy(), y_train)

modelIris.train(epoch=100, learning_rate=0.135, error_threshold =
   -100.0, batch_size=32)

modelIris.summarize(2)
```

Listing 1.1: Manual Instantiating

1.5 Contoh Penggunaan dengan Uji Kasus

Berikut contoh penggunaan dari uji kasus yang diberikan pada file linear.json. Digunakan pula method sse untuk penentuan kelulusan test case berupa true or false. Nama file yang digunakan hanya nama sebelum ekstensi .json dan file diletkkan di dalam folder models relatif terhadap tempat menjalankan program.

```
modelLinear = ANN()
modelLinear.load_model("linear")
modelLinear.train(
    epoch=modelLinear.learning_parameters["max_iteration"],
    learning_rate=modelLinear.learning_parameters["learning_rate"],
    batch_size=modelLinear.learning_parameters["batch_size"],
    error_threshold=modelLinear.learning_parameters["error_threshold"],
)
modelLinear.summarize(2)
modelLinear.sse()
```

Listing 1.2: Load Model

BAB 2

Hasil dan Pengujian

2.1 Linear Small lr

Pengujian pada test case pada file linear_small_lr.json memberikan hasil sebagai berikut.

- Stopped by: max iteration
- Current Loss : 0.665
- SSE: $9.600000000000107e-07 \sim 0$
- Error Threshold : 0.0

2.2 Linear Two Iteration

Pengujian pada test case pada file linear_two_iteration.json memberikan hasil sebagai berikut.

- Stopped by: max iteration
- Current Loss : 0.1818
- SSE: $1.6948183510607676e-32 \sim 0$
- Error Threshold : 0.0

2.3 Linear

Pengujian pada test case pada file linear.json memberikan hasil sebagai berikut.

- Stopped by: max iteration
- Current Loss : 0.665
- SSE: $1.9451892438311082e-32 \sim 0$
- Error Threshold : 0.0

2.4 MLP

Pengujian pada test case pada file mlp.json memberikan hasil sebagai berikut.

- Stopped by: max iteration
- Current Loss : 0.677
- SSE: $1.0785207688568521e-32 \sim 0$
- Error Threshold : 0.0

2.5 Relu - B

Pengujian pada test case pada file relu_b.json memberikan hasil sebagai berikut.

- Stopped by: max iteration
- Current Loss : 2.7935
- SSE: $3.851859888774472e-33 \sim 0$
- Error Threshold : 0.0

2.6 Sigmoid

Pengujian pada test case pada file sigmoid.json memberikan hasil sebagai berikut.

- Stopped by: max iteration
- Current Loss : 0.4551
- SSE: $2.0589774528718338e-08 \sim 0$
- Error Threshold : 0.01

2.7 Softmax

Pengujian pada test case pada file softmax.json memberikan hasil sebagai berikut.

- Stopped by: error threshold
- Current Loss : 0.8224
- SSE: $1.950457674231464e-16 \sim 0$
- Error Threshold : 0.05

2.8 Softmax Two Layer

Pengujian pada test case pada file softmax_two_layer.json memberikan hasil sebagai berikut.

- Stopped by: error threshold
- Current Loss : 0.0099
- SSE: $7.922752117108202e-17 \sim 0$
- Error Threshold : 0.01

2.9 Pengujian dengan Dataset Iris

Pengujian dengan menggunakan dataset Iris dilaksanakan melalui kode berikut.

```
modelIris = ANN(
    [
        Layer('relu', 2, np.array([[0.459, 0.459], [0.068, -0.193],
           [0.244, -0.298], [-0.412, -0.22]]), np.array([-0.22, 0.445])
           ),
        Layer('sigmoid', 2, np.array([[-0.059, -0.331], [-0.355,
           -0.195]]), np.array([-0.05, -0.09])),
        Layer('sigmoid', 3, np.array([[-0.391, -0.187, -0.173],
           [-0.221, 0.164, -0.395]]), np.array([0.165, 0.086, 0.461]))
    ]
)

modelIris.fit(X_train.to_numpy(), y_train)

modelIris.train(epoch=200, learning_rate=0.135, error_threshold = 0.01,
    batch_size=32)

modelIris.summarize(2)
```

Listing 2.1: Model Keras ANN Tugas

Kode pembanding sebagai berikut.

```
# Define layers
model = Sequential(
    [
        Input(shape=(4), name="inputs"),
        Dense(2, activation="relu"),
        Dense(2, activation="sigmoid"),
        Dense(3, activation="sigmoid"),
    ]
)

model.layers[0].set_weights([np.array([[0.459, 0.459], [0.068, -0.193],
    [0.244, -0.298], [-0.412, -0.22]]), np.array([-0.22, 0.445])])
model.layers[1].set_weights([np.array([[-0.059, -0.331], [-0.355,
   -0.195]]), np.array([-0.05, -0.09])])
model.layers[2].set_weights([np.array([[-0.391, -0.187, -0.173],
   [-0.221, 0.164, -0.395]]), np.array([0.165, 0.086, 0.461])])

# Compile the model
```

```
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.135,
    name="sgd"), loss="mse", metrics=["accuracy"])

# Train
model.fit(X_train.to_numpy(), y_train, epochs=200, batch_size=32,
    verbose=0)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss}\t|\tTest Accuracy: {test_accuracy}")

# Make predictions
y_pred = model.predict(X_test)
print(np.argmax(y_pred, axis=1))

print(
    classification_report(np.argmax(y_test, axis = 1), np.argmax(y_pred
        , axis=1), target_names=le.classes_)
)
```

Listing 2.2: Model Iris dari Keras

Berdasarkan hasil eksekusi kode di atas, akurasi yang kode kami dapatkan adalah 70% sedangkan pada keras 93.33%. Hal ini dapat disebabkan karena peubah random yang mungkin belum sempat dipelajari oleh penulis dalam pembuatan tugas besar ini. Hal ini terlihat pada nilai loss yang berbeda antara model keras dan model ANN yang dibuat pada tugas ini.

2.10 Perbandingan Hasil Softmax dengan Keras

Berikut merupakan kode untuk membandingkan test case softmax dengan keras.

```
kerasModelSoftmax = keras.Sequential(
    [
        keras.Input(shape=(8), name="inputs"),
        keras.layers.Dense(3, activation="softmax", name="dense_1",
            bias_initializer='ones'),
    ]
)

weights = np.array([[-0.2, 0.8, 0.2],
        [0.3, -0.7, 0.3],
        [0.4, 0.6, -0.4],
        [0.5, 0.5, 0.5],
        [-0.6, 0.4, 0.6],
        [-0.7, -0.3, 0.7],
        [0.8, 0.2, -0.8],
        [0.9, -0.1, 0.0]])
bias = np.array([0.1, 0.9, -0.1])

kerasModelSoftmax.layers[0].set_weights([weights, bias])

optimizer = optimizers.SGD(learning_rate=0.01)
kerasModelSoftmax.compile(optimizer=optimizer, loss='
    categorical_crossentropy')

x = np.array([[-2.4, -2.78, -0.6, 0.37, 2.46, -0.92, 2.76, 2.62],
```

```
      [-1.79, 1.65, -0.77, -1.03, 0.1, 2.12, -2.36, 1.25],
      [1.65, 2.34, 0.27, 2.34, 0.52, 1.37, 1.77, 0.62]])

y = np.array([
             [0, 1, 0],
      [1, 0, 0],
      [0, 0, 1]
        ])
kerasModelSoftmax.fit(x, y, batch_size=1, epochs=10)
kerasModelSoftmax.layers[0].get_weights()
```

Listing 2.3: Keras Model untuk Test Case Softmax

Weights, bias, dan input yang diberikan telah sesuai dengan yang ada di dalam file softmax.json dan didapatkan current loss sebesar 0.8233. Weights yang didapat adalah sebagai berikut.

$$\begin{bmatrix} -0.33595312 & 0.67714775 & 0.45880547 \\ 0.48256832 & -0.85228443 & 0.26971617 \\ 0.3399341 & 0.57242227 & -0.31235605 \\ 0.31376532 & 0.4632895 & 0.72294533 \\ -0.6963112 & 0.4786154 & 0.61769587 \\ -0.5091113 & -0.36354518 & 0.57265645 \\ 0.4190361 & 0.26312298 & -0.48215908 \\ 0.9039546 & -0.01792521 & -0.08602941 \end{bmatrix}$$

Weight bias sebagai berikut.

$$\begin{bmatrix} 0.12673964 \\ 0.9147715 \\ -0.14151111 \end{bmatrix}$$

Hasil eksekusi memberikan weight yang serupa dengan selisih pembulatan float yang dilakukan saat komputasi. Untuk itu, dapat disimpulkan untuk struktur dari model sudah sesuai dengan pembanding model Keras.

Daftar Pustaka

[1] T. P. IF3270, "Perkuliahan pembelajaran mesin," 2024.

Lampiran A

Pembagian Kerja

NIM	Nama	Pekerjaan
13521004	Henry Anand Septian Radityo	Implementasi Mini Batch Stochastic Gradient Descent
13521007	Matthew Mahendra	Implementasi Backpropagation, Stochastic Gradient Descent
13521015	Hidayatullah Wildan Ghaly Buchary	Implementasi Backpropagation
13521024	Ahmad Nadil	Model Architecture

Tabel A.1: Pembagian Kerja Kelompok