Zero to Mastery Learn PyTorch for Deep Learning Course Summary

02. PyTorch Neural Network Classification

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Pada modul ini, kita memahami dasar-dasar masalah klasifikasi dalam machine learning. Klasifikasi melibatkan prediksi apakah suatu elemen masuk ke dalam suatu kategori tertentu atau tidak. Ada tiga jenis utama masalah klasifikasi: binary classification, multi-class classification, dan multi-label classification.

Arsitektur Jaringan Saraf untuk Klasifikasi

Sebelum kita masuk ke penulisan kode, mari kita lihat arsitektur umum dari jaringan saraf untuk masalah klasifikasi. Berikut adalah beberapa hyperparameter dan komponen kunci yang dapat disesuaikan sesuai dengan jenis klasifikasi:

Binary Classification:

- **Input layer shape (in_features):** Jumlah fitur pada data.
- **Hidden layer(s):** Jumlah dan konfigurasi hidden layer (minimal 1, maksimal tidak terbatas).
- **Neurons per hidden layer:** Jumlah neuron dalam setiap hidden layer (problem spesifik, umumnya 10 hingga 512).
- **Output layer shape (out_features):** 1 (satu kelas atau yang lain).
- **Hidden layer activation:** Biasanya ReLU (rectified linear unit) atau fungsi aktivasi lainnya.
- **Output activation:** Sigmoid (torch.sigmoid in PyTorch).
- **Loss function:** Binary crossentropy (torch.nn.BCELoss in PyTorch).
- **Optimizer:** SGD (stochastic gradient descent), Adam, dll.

Multiclass Classification:

- **Input layer shape (in_features):** Sama dengan jumlah fitur pada data.
- **Hidden layer(s):** Sama seperti pada binary classification.
- **Neurons per hidden layer:** Sama seperti pada binary classification.

- **Output layer shape (out_features):** 1 per kelas (misalnya, 3 untuk kelas makanan, orang, atau anjing).
- **Hidden layer activation:** Sama seperti pada binary classification.
- **Output activation:** Softmax (torch.softmax in PyTorch).
- **Loss function:** Cross entropy (torch.nn.CrossEntropyLoss in PyTorch).
- **Optimizer:** Sama seperti pada binary classification.

1. Persiapan Data Klasifikasi

1.1 Membuat dan Menyusun Data Klasifikasi

Data klasifikasi dihasilkan menggunakan metode **make_circles()** dari Scikit-Learn, menghasilkan dua lingkaran dengan titik-titik berwarna berbeda. Data terdiri dari dua fitur (X1 dan X2) dan label (y) yang berisi nilai 0 atau 1.

1.2 Visualisasi Data

Data disajikan dalam DataFrame Pandas dan divisualisasikan dengan scatter plot untuk memahami distribusi dan pola data.

1.3 Penyesuaian Input dan Output

Bentuk data (shape) dieksplorasi untuk memastikan konsistensi. Data diubah menjadi tensor untuk kompatibilitas dengan PyTorch, dan satu sampel data diperiksa untuk memahami strukturnya.

1.4 Konversi Data dan Pembagian

Data diubah menjadi tensor PyTorch, dan dilakukan pembagian data menjadi set pelatihan (80%) dan pengujian (20%) menggunakan train_test_split.

Sekarang, kita memiliki 800 sampel pelatihan dan 200 sampel pengujian yang siap digunakan untuk melatih dan menguji model klasifikasi.

2. Pembangunan Model

2.1 Persiapan Kode Agnostik Perangkat

Kode perangkat-agnostik memungkinkan model berjalan pada CPU atau GPU jika tersedia. Kode ini memastikan kompatibilitas perangkat.

2.2 Konstruksi Model

Model dibangun dengan membuat kelas yang mewarisi nn.Module. Dua lapisan nn.Linear dibuat dalam konstruktor untuk menangani bentuk input dan output data.

2.3 Fungsi Forward dan Instance Model

Fungsi forward() menentukan perhitungan langkah maju model. Model diinstansiasi dan dikirimkan ke perangkat target (CPU atau GPU).

2.4 Pembuatan Model dengan nn.Sequential (Opsional)

Diperkenalkan penggunaan nn.Sequential untuk konstruksi model yang lebih sederhana, namun custom nn.Module sering diperlukan untuk kasus yang lebih kompleks.

2.5 Prediksi dengan Model

Model diuji dengan melakukan prediksi pada data uji. Hasilnya dievaluasi untuk memastikan model memberikan keluaran yang sesuai.

2.6 Penyiapan Fungsi Kerugian dan Optimizer

Fungsi kerugian nn.BCEWithLogitsLoss() dipilih untuk masalah klasifikasi biner. Optimizer SGD dengan laju pembelajaran 0.1 digunakan untuk mengoptimalkan parameter model.

2.7 Fungsi Evaluasi (Opsional)

Fungsi evaluasi accuracy_fn() dibuat untuk mengukur akurasi model selama pelatihan dan evaluasi.

Dengan demikian, model telah berhasil dibangun dan siap untuk melalui proses pelatihan pada data klasifikasi.

3. Pelatihan Model

3.1. Pengubah Output Model ke Label Prediksi

Sebelum memulai langkah-langkah pelatihan, perlu dikonfirmasi bahwa model yang dibangun menghasilkan keluaran yang sesuai dengan label prediksi. Pada langkah ini, output raw model (logit) diubah menjadi probabilitas prediksi dan kemudian dibulatkan untuk mendapatkan label prediksi.

3.2 Pembangunan Loop Pelatihan dan Pengujian

Langkah-langkah pelatihan dan pengujian model dibangun dalam loop. Dalam setiap epoch, model diperbarui menggunakan optimizer dan hasil pelatihan serta pengujian dicetak setiap 10 epoch untuk memantau perkembangan.

3.3 Evaluasi Awal Model

Meskipun model telah melalui proses pelatihan, performanya masih belum memuaskan. Akurasi yang stagnan di sekitar 50% menunjukkan bahwa model saat ini tidak lebih baik daripada pengambilan keputusan acak. Evaluasi lebih lanjut dan penyesuaian mungkin diperlukan untuk meningkatkan kinerja model pada tugas klasifikasi ini.

4. Melakukan Prediksi dan Mengevaluasi Model

4.1 Penilaian Awal Melalui Metrik

Dari metrik yang telah diperoleh, terlihat bahwa model cenderung melakukan prediksi secara acak, sebagaimana ditunjukkan oleh akurasi yang stagnan di sekitar 50%. Namun, untuk mendapatkan pemahaman yang lebih mendalam, perlu dilakukan analisis visual.

4.2 Visualisasi dan Analisis Decision Boundary

Dalam upaya memahami lebih jauh, dilakukan visualisasi menggunakan decision boundary. Decision boundary adalah batas pemisah antara kelas yang dihasilkan oleh model. Plot ini memperlihatkan bahwa model saat ini mencoba memisahkan data dengan garis lurus, yang tidak sesuai dengan distribusi data yang berbentuk lingkaran. Hal ini menjelaskan performa rendah model, karena garis lurus hanya dapat memotong data menjadi dua bagian sejajar, tanpa mampu menangkap pola yang sesuai.

4.3 Identifikasi Underfitting

Dalam terminologi machine learning, kondisi di atas disebut sebagai *underfitting*. Underfitting terjadi ketika model tidak mampu menangkap pola yang ada dalam data, sehingga tidak dapat membuat prediksi yang akurat.

4.4 Perbaikan Model

Untuk meningkatkan performa model, perlu dilakukan penyesuaian agar model dapat menangkap pola yang lebih kompleks, khususnya pada distribusi data yang berbentuk lingkaran. Upaya perbaikan dapat melibatkan penambahan kompleksitas model atau penerapan teknik-teknik khusus seperti kernel tricks pada algoritma Support Vector Machine (SVM). Analisis visual ini memberikan wawasan tambahan yang dapat menjadi dasar untuk langkah-langkah perbaikan selanjutnya.

5. Meningkatkan Model (Dari Perspektif Model)

5.1 Pendekatan Peningkatan Model

Untuk mengatasi masalah *underfitting* pada model kita, kita dapat fokus pada perbaikan model itu sendiri (bukan pada data). Beberapa cara yang dapat diterapkan melibatkan penambahan lapisan atau unit tersembunyi, peningkatan jumlah *epochs*, perubahan fungsi aktivasi, penyesuaian *learning rate*, dan perubahan fungsi loss.

5.2 Penerapan Peningkatan Model

Mari coba memperbaiki model dengan menambahkan lapisan tambahan, meningkatkan jumlah unit tersembunyi, dan melatih untuk lebih lama (epochs=1000).

Kita telah membuat **CircleModelV1** yang memiliki arsitektur sedikit lebih kompleks. Namun, setelah pelatihan lebih lama, model ini masih tidak menunjukkan peningkatan yang signifikan dalam pembelajaran pola dari data.

5.3 Eksplorasi Masalah

Pertanyaannya sekarang, mengapa model kita masih gagal? Apakah model kita tidak dapat memodelkan data lingkaran dengan baik? Untuk menguji ini, kita membuat data linier dan melatih model (model_2) untuk melihat apakah dapat memodelkan data tersebut.

5.4 Penyesuaian Model untuk Data Linier

Model_2, yang dirancang untuk menangani data linier, menunjukkan hasil yang lebih baik daripada model_1 pada data tersebut. Pada kasus ini, model berhasil mengurangi loss selama pelatihan.

6. Puzzel yang Hilang: Non-linearitas

6.1 Membuat Ulang Data Non-linear (Lingkaran Merah dan Biru)

Kita telah melihat bahwa model kita dapat menggambar garis lurus (linear) berkat lapisan linearnya. Tetapi bagaimana jika kita memberinya kapasitas untuk menggambar garis yang tidak lurus (non-linear)?

Mari kita temukan jawabannya.

6.2 Membangun Model dengan Non-linearitas

Kita dapat meningkatkan kapasitas model kita dengan menambahkan fungsi aktivasi non-linear. Sebagai contoh, kita akan menggunakan ReLU (Rectified Linear Unit) di antara lapisan tersembunyi. Fungsi ini dapat membantu model mempelajari pola non-linear dari data.

6.3 Melatih Model dengan Non-linearitas

Setelah menambahkan ReLU, kita melatih model menggunakan data baru yang terdiri dari lingkaran merah dan biru. Kita memantau loss dan akurasi selama pelatihan untuk melihat apakah model mampu memahami pola non-linear.

6.4 Evaluasi Model yang Dilatih dengan Fungsi Aktivasi Non-linear

Setelah pelatihan, kita mengevaluasi model pada data uji dan memeriksa prediksinya. Terakhir, kita membandingkan pengambilan keputusan model yang menggunakan non-linearitas dengan model sebelumnya yang hanya menggunakan linearitas.

7. Menduplikasi Fungsi Aktivasi Non-linear

7.1 Replikasi Fungsi ReLU

Sebelumnya, kita telah melihat bagaimana menambahkan fungsi aktivasi non-linear ke dalam model dapat membantu dalam memodelkan data non-linear. Namun, apa sebenarnya yang dilakukan oleh fungsi aktivasi non-linear tersebut?

Mari kita coba mereplikasi beberapa fungsi aktivasi non-linear dan melihat apa yang mereka lakukan. Pertama, kita akan membuat data kecil menggunakan tensor PyTorch.

7.2 Fungsi ReLU

Fungsi ReLU (Rectified Linear Unit) mengubah nilai negatif menjadi 0 dan membiarkan nilai positif tidak berubah. Kita mencoba mereplikasi fungsi ini menggunakan PyTorch.

7.3 Fungsi Sigmoid

Selanjutnya, kita coba mengimplementasikan fungsi sigmoid yang telah kita gunakan sebelumnya. Fungsi sigmoid digunakan untuk menghasilkan nilai antara 0 dan 1, yang sering diinterpretasikan sebagai probabilitas.

7.4 Visualisasi Hasil

Setelah mereplikasi kedua fungsi tersebut, kita visualisasikan hasilnya. Dari grafik, kita dapat melihat perbedaan antara garis lurus awal dan garis yang telah diubah oleh fungsi aktivasi non-linear.

8. Menggabungkan Semua dalam Model PyTorch Multi-Class

8.1 Membuat Data Multi-Class

Sebelumnya, kita telah bekerja dengan masalah klasifikasi biner. Sekarang, mari terapkan semua konsep ini pada masalah klasifikasi multi-kelas. Perbedaan utama adalah bahwa kita sekarang memiliki lebih dari dua kelas sebagai opsi.

Untuk mulai masalah klasifikasi multi-kelas, kita akan membuat beberapa data multi-kelas. Kita akan menggunakan metode **make_blobs()** dari Scikit-Learn untuk membuat beberapa cluster atau kelompok data.

8.2 Membangun Model Klasifikasi Multi-Class di PyTorch

Kita akan membuat model serupa dengan **model_3**, namun kali ini model harus mampu menangani data multi-kelas. Kita akan membuat kelas **BlobModel** yang mengambil tiga hiperparameter: jumlah fitur masukan (**input_features**), jumlah fitur keluaran yang diinginkan (**output_features**), dan jumlah unit tersembunyi (**hidden_units**).

8.3 Membuat Fungsi Loss dan Optimizer untuk Model PyTorch Multi-Class

Karena kita menghadapi masalah klasifikasi multi-kelas, kita akan menggunakan metode **nn.CrossEntropyLoss()** sebagai fungsi loss. Dan kita akan tetap menggunakan SGD dengan tingkat pembelajaran 0.1 untuk mengoptimalkan parameter model.

8.4 Mendapatkan Probabilitas Prediksi untuk Model PyTorch Multi-Class

Sebelum melatih model, mari lakukan satu kali langkah maju (**forward pass**) dengan model untuk melihat apakah semuanya berfungsi. Hasil output yang kita dapatkan saat ini disebut logits. Untuk mengonversi logits menjadi probabilitas prediksi, kita akan menggunakan fungsi aktivasi softmax.

8.5 Membuat Loop Pelatihan dan Pengujian untuk Model PyTorch Multi-Class

Setelah semua persiapan selesai, kita dapat menulis loop pelatihan dan pengujian untuk meningkatkan dan mengevaluasi model. Kita akan menyesuaikan langkah-langkah untuk mengonversi output model (logits) menjadi probabilitas prediksi dan kemudian menjadi label prediksi.

8.6 Membuat dan Mengevaluasi Prediksi dengan Model PyTorch Multi-Class

Model terlatih sepertinya sudah berkinerja cukup baik. Namun, untuk memastikan, mari membuat beberapa prediksi dan memvisualisasikannya. Kita akan mengonversi logits prediksi model menjadi probabilitas prediksi menggunakan fungsi softmax dan kemudian menjadi label prediksi menggunakan argmax. Selanjutnya, kita akan membandingkan hasilnya dengan label uji dan mengevaluasi akurasi.

Terakhir, kita akan memvisualisasikan hasil prediksi menggunakan fungsi **plot_decision_boundary()**. Ingatlah bahwa data kita ada di GPU, sehingga kita perlu memindahkannya ke CPU untuk digunakan dengan matplotlib, dan fungsi **plot_decision_boundary()** melakukan ini secara otomatis.

9. Lebih Banyak Metrik Evaluasi Klasifikasi

Sejauh ini, kita baru membahas beberapa cara untuk mengevaluasi model klasifikasi (akurasi, loss, dan visualisasi prediksi). Ini adalah beberapa metode umum yang akan Anda temui dan merupakan titik awal yang baik.

Namun, Anda mungkin ingin mengevaluasi model klasifikasi Anda menggunakan lebih banyak metrik seperti yang tercantum di bawah ini:

1. **Akurasi (Accuracy):**
    - Mengukur dari 100 prediksi, berapa banyak yang benar oleh model Anda? Sebagai contoh, akurasi 95% berarti model mendapatkan 95/100 prediksi dengan benar.
2. **Presisi (Precision):**
    - Proporsi positif benar dibandingkan dengan jumlah total sampel. Presisi yang lebih tinggi mengarah ke lebih sedikit positif palsu (model memprediksi 1 ketika seharusnya 0).
3. **Recall:**
    - Proporsi positif benar dibandingkan dengan jumlah total positif benar dan negatif palsu (model memprediksi 0 ketika seharusnya 1). Recall yang lebih tinggi mengarah ke lebih sedikit negatif palsu.
4. **F1-score:**

- Menggabungkan presisi dan recall menjadi satu metrik. Nilai 1 adalah yang terbaik, 0 adalah yang terburuk.

5. **Confusion Matrix:**
    - Membandingkan nilai prediksi dengan nilai sebenarnya secara tabuler. Jika 100% benar, semua nilai dalam matriks akan berada pada garis diagonal dari kiri atas ke kanan bawah.
6. **Classification Report:**
    - Kumpulan beberapa metrik klasifikasi utama seperti presisi, recall, dan f1-score.

Scikit-Learn (perpustakaan pembelajaran mesin yang populer dan kelas dunia) memiliki banyak implementasi metrik-metrik di atas. Jika Anda mencari versi yang mirip dengan PyTorch, coba lihat TorchMetrics, terutama bagian klasifikasi.