TECHNICAL REPORT

DEEP LEARNING WITH PYTORCH

Nama :

Zulian Wahid 1103201049

Table of Content

Introduction

Deep learning telah muncul sebagai teknologi transformatif dalam beberapa tahun terakhir, merevolusi berbagai industri dan domain. Dengan kemampuan untuk secara otomatis mempelajari representasi dari data dalam jumlah besar, model deep learning telah mencapai kesuksesan yang luar biasa dalam tugas-tugas seperti pengenalan gambar, pemrosesan bahasa alami, dan sintesis ucapan. Pertumbuhan ketersediaan data yang cepat, ditambah dengan kemajuan daya komputasi, semakin mendorong adopsi teknik deep learning. Laporan ini memberikan gambaran umum tentang deep learning dengan fokus pada PyTorch, sebuah kerangka kerja deep learning yang populer, dan mengeksplorasi aplikasinya dalam memecahkan masalah di dunia nyata.

Dalam lanskap bisnis yang sangat kompetitif saat ini, memahami sentimen pelanggan sangat penting bagi perusahaan untuk membuat keputusan yang tepat dan memberikan produk atau layanan yang unggul. Analisis sentimen, proses menentukan polaritas data tekstual, memainkan peran penting dalam mengekstraksi wawasan dari umpan balik pelanggan. Namun, mengklasifikasikan sentimen secara akurat dari teks yang tidak terstruktur dapat menjadi tantangan tersendiri. Laporan ini bertujuan untuk menjawab tantangan ini dengan memanfaatkan kekuatan teknik deep learning yang diimplementasikan menggunakan PyTorch. Secara khusus, kami akan mengeksplorasi bagaimana model deep learning dapat meningkatkan akurasi analisis sentimen pada ulasan pelanggan, memberikan manfaat bagi bisnis dalam mendapatkan wawasan yang berharga dan meningkatkan kepuasan pelanggan.

Tujuan utama dari laporan ini ada dua. Pertama, kami bertujuan untuk memberikan pemahaman yang komprehensif tentang dasar-dasar deep learning, termasuk jaringan saraf, fungsi aktivasi, dan algoritma optimasi. Melalui contoh-contoh praktis dan penjelasan, kami akan memperkenalkan kerangka kerja PyTorch, menyoroti kemampuannya untuk membangun dan melatih model deep learning. Kedua, kami akan mendemonstrasikan keefektifan teknik deep learning dalam meningkatkan akurasi analisis sentimen pada ulasan pelanggan. Dengan mengimplementasikan dan mengevaluasi berbagai model menggunakan PyTorch, kami bertujuan untuk menunjukkan potensi deep learning dalam mengekstraksi wawasan yang bermakna dari data tekstual dan relevansinya dalam aplikasi bisnis di dunia nyata.

Deep learning basic

Deep learning adalah bidang yang berkembang pesat yang membutuhkan pemahaman yang kuat tentang prinsip-prinsip dasarnya. Bagian ini memberikan gambaran ringkas tentang konsep-konsep inti yang diperlukan untuk memahami dan menganalisis implementasi kode. Kita akan meninjau kembali secara singkat topik-topik utama yang tercakup dalam video YouTube, termasuk jaringan syaraf, fungsi aktivasi, dan algoritma pengoptimalan. Selain itu, kami akan mengeksplorasi teknik atau metodologi spesifik yang digunakan dalam kode untuk memastikan pemahaman yang komprehensif tentang fondasi kode.

Jaringan saraf berfungsi sebagai blok bangunan model pembelajaran mendalam, meniru struktur dan fungsi otak manusia. Kami akan membahas struktur jaringan saraf, yang terdiri dari lapisan-lapisan node yang saling terhubung, serta fungsi aktivasi mereka, yang memperkenalkan non-linearitas pada model. Memahami komponen-komponen ini akan memungkinkan kita untuk memahami bagaimana kode

mengimplementasikan dan memanfaatkan jaringan saraf. Selanjutnya, kita akan membahas algoritma pengoptimalan, seperti gradient descent, yang memainkan peran penting dalam melatih jaringan saraf untuk meminimalkan kesalahan dan meningkatkan kemampuan prediktifnya. Dengan meninjau dasar-dasar pembelajaran mendalam ini, kita akan membangun fondasi yang kuat untuk analisis implementasi kode selanjutnya.

Pytorch Basic

PyTorch adalah framework pembelajaran mesin sumber terbuka yang menyediakan platform yang kuat untuk mengembangkan dan melatih model pembelajaran mendalam. PyTorch banyak digunakan dalam komunitas penelitian dan industri karena fleksibilitas, efisiensi, dan dukungannya yang luas untuk jaringan syaraf. PyTorch menggabungkan manfaat dari grafik komputasi dinamis dan diferensiasi otomatis, sehingga memudahkan untuk membangun grafik komputasi yang kompleks dan mengoptimalkannya secara efisien.

Pada intinya, PyTorch berkisar pada tensor, yang merupakan susunan multidimensi yang mirip dengan susunan NumPy. Tensor di PyTorch dapat dengan mudah dimanipulasi dan dioperasikan menggunakan berbagai operasi dan fungsi matematika, sehingga memudahkan untuk mengekspresikan dan mengeksekusi komputasi yang melibatkan data dalam jumlah besar. Selain itu, PyTorch menawarkan abstraksi tingkat tinggi untuk mendefinisikan dan melatih jaringan saraf melalui modul torch.nn. Modul ini menyediakan satu set lapisan yang telah ditentukan sebelumnya, fungsi kerugian, dan algoritme pengoptimalan yang dapat dengan mudah digunakan untuk membangun dan melatih arsitektur jaringan saraf.

PyTorch juga unggul dalam sifatnya yang dinamis, memungkinkan pengguna untuk mendefinisikan grafik komputasi dengan cepat. Konstruksi grafik dinamis ini memungkinkan pengembangan model yang lebih fleksibel dan intuitif, karena memungkinkan eksekusi bersyarat, loop, dan konstruksi aliran kontrol lainnya selama pendefinisian model. Selain itu, PyTorch terintegrasi dengan baik dengan pustaka Python yang populer dan menyediakan dukungan untuk akselerasi GPU, sehingga memungkinkan komputasi yang efisien pada perangkat keras paralel.

Singkatnya, PyTorch adalah deep learning framework serbaguna yang memberdayakan para peneliti dan pengembang untuk membangun dan melatih model jaringan syaraf tiruan yang kompleks dengan mudah. Kombinasi konstruksi grafik dinamis, dukungan pustaka yang luas, dan komputasi tensor yang efisien membuatnya menjadi pilihan populer untuk menangani berbagai tugas pembelajaran mesin.

Analyze the code

1. Tensor Basics (Dasar-dasar Tensor)

Pada bagian ini, kita akan menganalisa sebuah potongan kode yang mendemonstrasikan konsep dasar dan operasi-operasi yang berhubungan dengan tensor di PyTorch. Cuplikan kode ini menampilkan pembuatan

tensor, operasi aritmatika dasar, pembentukan ulang tensor, konversi ke/dari array NumPy, dan memanfaatkan GPU jika tersedia. Mari kita bahas setiap aspek secara mendetail:

Pembuatan Tensor:

Kode menginisialisasi tensor menggunakan berbagai metode:

torch.empty(): Fungsi ini membuat tensor kosong dengan ukuran tertentu, seperti tensor elemen tunggal, tensor dengan banyak elemen, atau tensor dengan banyak dimensi.

torch.rand(): Fungsi ini menghasilkan tensor yang diisi dengan nilai acak antara 0 dan 1.

torch.zeros(): Fungsi ini membuat tensor yang diisi dengan angka nol.

torch.tensor(): Fungsi ini membuat tensor dari data yang sudah ada, seperti daftar Python atau larik NumPy.

Parameter dtype digunakan untuk menentukan tipe data tensor.

Operasi Tensor:

Kode ini mendemonstrasikan operasi aritmatika dasar pada tensor menggunakan operator (+, -, *, /) dan fungsi PyTorch yang sesuai (torch.add (), torch.sub (), torch.mul (), torch.div ()). Selain itu, ini menampilkan operasi pengindeksan dan pemotongan untuk mengakses elemen tertentu atau rentang elemen dalam tensor. Metode item() mengambil nilai tensor elemen tunggal sebagai skalar Python.

Tensor Reshaping:

Fungsi view() digunakan untuk membentuk ulang tensor tanpa mengubah jumlah total elemen. Fungsi ini memungkinkan untuk memodifikasi dimensi tensor dengan tetap mempertahankan datanya.

Konversi Tensor:

Kode ini mengilustrasikan konversi antara tensor dan array NumPy. Tensor dapat dikonversi ke larik NumPy menggunakan metode .numpy(), sehingga memungkinkan integrasi dengan ekosistem NumPy. Demikian pula, array NumPy dapat dikonversi menjadi tensor menggunakan fungsi torch.from_numpy().

Tensor pada GPU:

Kode ini menyertakan pemeriksaan ketersediaan GPU menggunakan torch.cuda.is_available(). Jika ada GPU, tensor dapat dipindahkan ke GPU menggunakan .to(device) atau torch.device("cuda"). Transfer eksplisit mungkin diperlukan untuk operasi yang melibatkan tensor pada GPU dan CPU.

2. Autograd

Cuplikan kode yang disediakan menampilkan penggunaan fitur diferensiasi otomatis PyTorch, yang dikenal sebagai Autograd. Fitur ini memungkinkan komputasi dan propagasi gradien untuk tensor dengan atribut requires_grad = True.

Komputasi Gradien dan Perambatan Balik:

Kode tersebut mendefinisikan tensor x dengan shape (3,) dan atribut requires_grad = True, yang mengindikasikan bahwa gradien perlu dihitung untuk tensor ini.

Tensor y dibuat dengan melakukan penjumlahan elemen dari x dan 2.

Atribut grad_fn dari y menyimpan referensi ke fungsi yang menghasilkan y (dalam kasus ini, operasi penjumlahan).

Tensor z dihitung sebagai kuadrat elemen-bijaksana dari y dikalikan dengan 3.

Rata-rata dari z dihitung menggunakan fungsi mean().

Fungsi backward() dipanggil pada z untuk menghitung gradien dari z terhadap x.

Akhirnya, atribut grad dari x berisi gradien yang dihitung.

Gradien Orde Tinggi:

Tensor lain x dengan bentuk (3,) dan atribut requires_grad = True didefinisikan.

Sebuah perulangan digunakan untuk melakukan serangkaian perkalian elemen-bijaksana dari y dengan 2. Hal ini menunjukkan kemampuan Autograd untuk menangani gradien tingkat tinggi.

Nilai akhir dari y dan bentuknya dicetak.

Perhitungan Turunan Parsial:

Sebuah tensor v didefinisikan dengan nilai yang ditentukan dan atribut requires_grad = False.

Fungsi backward() dipanggil pada y dengan v sebagai argumen untuk menghitung turunan parsial dari y terhadap x.

Atribut grad dari x sekarang berisi gradien yang diperbarui setelah backward pass.

Melacak Riwayat Gradien:

Sebuah tensor a didefinisikan tanpa atribut requires_grad, yang berarti bahwa gradien tidak akan dihitung untuk tensor ini secara default.

Tensor b dihitung menggunakan operasi yang melibatkan a, dan atribut grad_fn dari b menyimpan referensi ke fungsi yang menghasilkan b.

Fungsi requires_grad_() digunakan untuk mengaktifkan komputasi gradien untuk a setelah dibuat.

Tensor b dimodifikasi untuk menghitung jumlah elemen kuadrat dari a.

Atribut requires_grad dicetak untuk mengonfirmasi apakah komputasi gradien diaktifkan untuk a dan b.

Pengecualian Komputasi Gradien:

Sebuah tensor a didefinisikan dengan atribut requires_grad = True.

Tensor b dibuat dengan melepaskan a dari grafik komputasi menggunakan fungsi detach().

Atribut requires_grad dari b dicetak untuk memverifikasi bahwa tensor ini tidak lagi memerlukan komputasi gradien.

Manajer Konteks Komputasi Gradien:

Sebuah tensor a didefinisikan dengan atribut requires_grad = True.

Manajer konteks dengan torch.no_grad(): digunakan untuk mengecualikan sementara komputasi gradien untuk operasi-operasi yang dilingkupi.

Tensor kuadrat x dikomputasi di dalam context manager, dan atribut requires_grad dicetak untuk memverifikasi bahwa gradien tidak sedang dikomputasi.

Perhitungan Gradien Manual:

Bobot tensor didefinisikan dengan atribut requires_grad = True.

Dalam perulangan untuk beberapa epoch, output model dihitung sebagai jumlah bobot dikalikan dengan 3.

Fungsi backward() dipanggil pada keluaran model untuk menghitung gradien.

Gradien dari bobot dicetak, dan gradien di-nol-kan secara manual menggunakan zero_() untuk mencegah penumpukan. Bobot diperbarui menggunakan penurunan gradien dengan mengurangi sebagian kecil dari gradien.

Terakhir, bobot yang telah diperbarui dan output model dicetak.

3. Analisis Kode - "Backpropagation"

Cuplikan kode yang disediakan menunjukkan proses backpropagation, sebuah langkah penting dalam melatih jaringan syaraf untuk memperbarui parameter model berdasarkan gradien yang dihitung.

Inisialisasi:

Tensor x dan y didefinisikan sebagai nilai skalar (masing-masing 1,0 dan 2,0).

Parameter model w diinisialisasi sebagai tensor dengan nilai awal 1.0 dan atribut requires_grad = True, yang mengindikasikan bahwa gradien harus dihitung untuk tensor ini.

Penerusan:

Nilai prediksi y_prediksi dihitung dengan mengalikan w dengan x.

Kerugian dihitung sebagai selisih kuadrat antara nilai prediksi (y_predicted) dan nilai aktual (y).

Backward Pass:

Fungsi backward() dipanggil pada tensor kerugian untuk menghitung gradien kerugian sehubungan dengan parameter model w. Langkah ini melakukan diferensiasi otomatis menggunakan grafik komputasi dan aturan rantai.

Penurunan Gradien:

Nilai parameter model w diperbarui dengan mengurangi sebagian kecil dari gradien (0.01 * w.grad) dalam context manager torch.no_grad(). Context manager ini memastikan bahwa tidak ada gradien yang dihitung atau disimpan selama pembaruan parameter.

Mengosongkan Gradien:

Setelah pembaruan parameter, gradien dari w di-nol-kan secara manual menggunakan fungsi zero_(). Langkah ini sangat penting untuk mencegah akumulasi gradien di seluruh iterasi.

4. Analisis Kode - " Manual Gradient Descent"

Cuplikan kode yang disediakan menunjukkan implementasi gradient descent secara manual, tanpa menggunakan deep learning framework apa pun. Kode ini menunjukkan proses iteratif untuk memperbarui parameter bobot (w) berdasarkan gradien yang dihitung.

Persiapan Data:

Kode ini menginisialisasi dua larik numpy X dan Y untuk merepresentasikan nilai input dan target.

Definisi Model:

Parameter bobot w diinisialisasi sebagai 0.0

Penerusan:

Fungsi forward() mendefinisikan forward pass dari model, di mana nilai prediksi dihitung sebagai hasil kali antara bobot w dan input x.

Perhitungan Kerugian:

Fungsi loss() menghitung rata-rata kuadrat kerugian antara nilai prediksi (y_pred) dan target aktual (y).

Perhitungan Gradien:

Fungsi gradien() menghitung gradien kerugian sehubungan dengan bobot w menggunakan rumus gradien kesalahan kuadrat rata-rata.

Perulangan Pelatihan:

Kode ini mendefinisikan laju pembelajaran (learning_rate) dan jumlah iterasi pelatihan (n_iters).

Pada setiap epoch dari loop pelatihan, forward pass dilakukan untuk mendapatkan nilai prediksi (y_pred).

Kerugian dihitung berdasarkan nilai prediksi dan target aktual. Gradien dari kerugian sehubungan dengan bobot dihitung.

Bobot w diperbarui menggunakan gradient descent dengan mengurangi sebagian kecil dari gradien (learning_rate*dw).

Memantau Kemajuan:

Setiap 2 epoch, nomor epoch saat ini, nilai bobot yang diperbarui (w), dan nilai kerugian (l) dicetak.

Prediksi:

Setelah perulangan pelatihan, bobot akhir w diperoleh.

Penerusan dilakukan pada nilai input baru (5) untuk menampilkan prediksi setelah pelatihan.

Cuplikan kode ini menunjukkan implementasi dasar dari gradient descent untuk mengoptimalkan model regresi linier secara manual. Ini memperlihatkan proses langkah demi langkah dalam memperbarui parameter bobot berdasarkan gradien yang dihitung untuk meminimalkan fungsi kerugian.

5. Analisis Kode - " Auto Gradient Descent "

Cuplikan kode yang disediakan menunjukkan implementasi penurunan gradien menggunakan diferensiasi otomatis di PyTorch. Kode ini menunjukkan proses memperbarui parameter bobot (w) berdasarkan gradien yang dihitung tanpa secara eksplisit menghitung gradien.

Persiapan Data:

Kode mendefinisikan dua tensor, X dan Y, masing-masing untuk mewakili nilai input dan target.

Definisi Model:

Parameter bobot w diinisialisasi sebagai tensor dengan nilai awal 0.0 dan atribut requires_grad = True, yang mengindikasikan bahwa gradien harus dihitung untuk tensor ini.

Penerusan:

Fungsi forward() mendefinisikan forward pass dari model, di mana nilai prediksi dihitung sebagai hasil kali bobot w dan input x.

Perhitungan Kerugian:

Fungsi loss() menghitung rata-rata kuadrat kerugian antara nilai prediksi (y_pred) dan target aktual (y).

Perulangan Pelatihan:

Kode mendefinisikan laju pembelajaran (learning_rate) dan jumlah iterasi pelatihan (n_iters).

Pada setiap epoch dari loop pelatihan, forward pass dilakukan untuk mendapatkan nilai prediksi (y_pred).

Kerugian dihitung berdasarkan nilai prediksi dan target aktual.

Backpropagation dipicu dengan memanggil backward() pada tensor kerugian, yang secara otomatis menghitung gradien kerugian sehubungan dengan parameter yang dapat dilatih.

Bobot w diperbarui menggunakan gradient descent dengan mengurangi sebagian kecil dari gradien (learning_rate * w.grad). Langkah ini dilakukan dalam context manager torch.no_grad() untuk menonaktifkan pelacakan gradien dan menghindari penggunaan memori yang tidak perlu. Gradien dari w di-nol-kan secara manual menggunakan fungsi zero_() untuk mencegah akumulasi gradien di seluruh iterasi.

Memantau Kemajuan:

Setiap 10 epoch, nomor epoch saat ini, nilai bobot yang diperbarui (w.item()), dan nilai kerugian (l.item()) dicetak.

Prediksi:

Setelah perulangan pelatihan, bobot akhir w diperoleh. Penerusan dilakukan pada nilai input baru (5) untuk menampilkan prediksi setelah pelatihan.

Cuplikan kode ini mendemonstrasikan penggunaan diferensiasi otomatis di PyTorch untuk melakukan penurunan gradien untuk mengoptimalkan model regresi linier. Gradien dihitung secara otomatis melalui backpropagation, menyederhanakan implementasi dan mengurangi penghitungan gradien secara manual.

6. Analisis Kode - "Loss and Optimizer"

Cuplikan kode yang disediakan menampilkan penggunaan fungsi loss dan pengoptimal di PyTorch untuk melatih model regresi linier. Kode ini memperkenalkan dua komponen penting dalam proses pelatihan.

Persiapan Data:

Kode ini mendefinisikan dua tensor, X dan Y, untuk merepresentasikan nilai input dan target.

Definisi Model:

Parameter bobot w diinisialisasi sebagai tensor dengan nilai awal 0.0 dan atribut requires_grad = True, yang mengindikasikan bahwa gradien harus dihitung untuk tensor ini.

Penerusan:

Fungsi forward() mendefinisikan forward pass dari model, di mana nilai prediksi dihitung sebagai hasil kali antara bobot w dan input x.

Prediksi Sebelum Pelatihan:

Kode ini mencetak prediksi (forward(5).item()) sebelum iterasi pelatihan, memberikan prediksi awal berdasarkan nilai bobot awal.

Konfigurasi Pelatihan:

Kode ini mengatur laju pembelajaran (learning_rate) dan jumlah iterasi pelatihan (n_iters).

Inisialisasi Kerugian dan Pengoptimal:

Kode membuat sebuah contoh dari fungsi kerugian mean squared error (MSE) menggunakan nn.MSELoss().

Pengoptimal (torch.optim.SGD) diinisialisasi, menentukan parameter yang akan dioptimalkan ([w]) dan laju pembelajaran (lr = learning_rate).

Perulangan Pelatihan:

Loop ini mengulang selama jumlah epoch yang ditentukan (n_iter).

Pada setiap epoch, forward pass dilakukan untuk mendapatkan nilai prediksi (y_predicted).

Kerugian dihitung berdasarkan nilai prediksi dan target aktual menggunakan fungsi kerugian yang ditentukan (loss(Y, y_predicted)).

Backpropagation dipicu dengan memanggil backward() pada tensor kerugian, yang secara otomatis menghitung gradien kerugian sehubungan dengan parameter yang dapat dilatih (w).

Fungsi step() dari pengoptimal dipanggil untuk memperbarui parameter model berdasarkan gradien yang dihitung.

Fungsi zero_grad() dari pengoptimal digunakan untuk meng-nol-kan gradien secara manual untuk iterasi berikutnya untuk menghindari akumulasi gradien.

Memantau Kemajuan:

Setiap 10 epoch, nomor epoch saat ini, nilai bobot (w[0][0].item()), dan nilai kerugian (1) dicetak.

Prediksi Setelah Pelatihan:

Setelah perulangan pelatihan, bobot akhir w diperoleh.

Penerusan dilakukan pada nilai input baru (5) untuk menampilkan prediksi setelah pelatihan.

Potongan kode ini menunjukkan penggunaan fungsi loss (nn.MSELoss()) dan pengoptimal (torch.optim.SGD) untuk melatih sebuah model regresi linier di PyTorch. Fungsi loss menghitung perbedaan antara nilai prediksi dan target aktual, sementara pengoptimal memperbarui parameter model berdasarkan gradien yang dihitung. Bersama-sama, keduanya memfasilitasi proses pelatihan dan pengoptimalan parameter model.

7. Analisis Kode - "Regresi Linier"

Kode yang disediakan mengimplementasikan regresi linier menggunakan PyTorch untuk mencocokkan model linier ke dataset sintetis.

Persiapan Data:

Kode tersebut menggunakan fungsi make_regression dari scikit-learn untuk menghasilkan dataset sintetis dengan 100 sampel, 1 fitur, dan beberapa noise tambahan.

Fitur input (X_numpy) dan nilai target (y_numpy) disimpan sebagai array NumPy.

Konversi Data:

Array NumPy X_numpy dan y_numpy dikonversi ke tensor PyTorch menggunakan torch.from_numpy.

Bentuk tensor target y dimodifikasi menggunakan y.view agar sesuai dengan bentuk yang diharapkan untuk perhitungan selanjutnya.

Definisi Model:

Model regresi linier didefinisikan menggunakan nn.Linear(input_size, output_size), di mana input_size sesuai dengan jumlah fitur, dan output_size diatur ke 1.

Konfigurasi Pelatihan:

Laju pembelajaran (learning_rate) diatur ke 0,01

Fungsi Kerugian dan Inisialisasi Pengoptimal:

Fungsi kerugian rata-rata kesalahan kuadrat (MSE) (nn.MSELoss()) diinisialisasi.

Pengoptimal stochastic gradient descent (SGD) (torch.optim.SGD) diinisialisasi, menentukan parameter model (model.parameters()) dan laju pembelajaran.

Perulangan Pelatihan:

Loop mengulang untuk jumlah epoch yang ditentukan (num_epochs).

Pada setiap epoch, model melakukan forward pass dengan memanggil model(X), mendapatkan nilai prediksi (y_predicted). Kerugian dihitung dengan menerapkan fungsi kerugian pada nilai prediksi dan target aktual menggunakan kriteria (y_prediksi, y). Backpropagation dipicu dengan memanggil backward() pada tensor kerugian, menghitung gradien parameter model. Fungsi step() dari pengoptimal digunakan untuk memperbarui parameter model berdasarkan gradien yang dihitung. Fungsi zero_grad() dari pengoptimal dipanggil untuk meng-nolkan gradien untuk iterasi berikutnya.

Pemantauan Kemajuan:

Setiap 10 epoch, nomor epoch saat ini dan nilai kerugian dicetak.

Prediksi dan Visualisasi:

Model yang dilatih digunakan untuk memprediksi nilai output untuk fitur input (model(X).detach().numpy()). Titik-titik data asli dan garis regresi yang diprediksi diplot menggunakan matplotlib.

Kode ini menunjukkan penggunaan kemampuan regresi linier PyTorch untuk melatih model pada set data sintetis. Kode ini mengoptimalkan parameter model menggunakan stochastic gradient descent untuk

meminimalkan rata-rata kuadrat kesalahan. Model terlatih yang dihasilkan kemudian digunakan untuk membuat prediksi dan memvisualisasikan garis regresi.

8. Analisis Kode - "Regresi Logistik"

Kode yang disediakan mengimplementasikan regresi logistik menggunakan PyTorch untuk mengklasifikasikan data kanker payudara.

Pemuatan dan Pemrosesan Data:

Kode tersebut menggunakan fungsi load_breast_cancer dari scikit-learn untuk memuat dataset kanker payudara.

Fitur input (X) dan label target (y) diekstrak dari dataset.

Dataset dipecah menjadi set pelatihan dan pengujian menggunakan train_test_split dari scikit-learn.

Penskalaan standar diterapkan untuk menormalkan fitur input menggunakan StandardScaler.

Konversi Data:

Array NumPy X_train, X_test, y_train, dan y_test dikonversi ke tensor PyTorch menggunakan torch.from_numpy.

Bentuk tensor target y_train dan y_test dimodifikasi menggunakan y_train.view dan y_test.view agar sesuai dengan bentuk yang diharapkan untuk penghitungan selanjutnya.

Definisi Model:

Model regresi logistik didefinisikan sebagai subkelas dari nn.Module.

Model ini terdiri dari lapisan linear (nn.Linear) yang diikuti oleh fungsi aktivasi sigmoid (torch.sigmoid).

Jumlah fitur input (n_input_features) dioper ke konstruktor model, dan lapisan linier diinisialisasi sesuai dengan itu.

Konfigurasi Pelatihan:

Jumlah epoch (num_epochs) diatur ke 100.

Laju pembelajaran (learning_rate) diatur ke 0,01.

Fungsi Kerugian dan Inisialisasi Pengoptimal:

Fungsi kerugian entropi silang biner (nn.BCELoss()) diinisialisasi.

Pengoptimal stochastic gradient descent (SGD) (torch.optim.SGD) diinisialisasi, menentukan parameter model (model.parameters()) dan laju pembelajaran.

Perulangan Pelatihan:

Loop ini mengulang untuk jumlah epoch yang ditentukan (num_epochs).

Pada setiap epoch, model melakukan forward pass dengan memanggil model(X_train), mendapatkan probabilitas yang diprediksi (y_pred) dengan menggunakan fungsi aktivasi sigmoid.

Kerugian dihitung dengan menerapkan fungsi kerugian cross-entropi biner pada probabilitas yang diprediksi dan label aktual menggunakan kriteria (y_pred, y_train).

Backpropagation dipicu dengan memanggil backward() pada tensor kerugian, menghitung gradien parameter model.

Fungsi step() dari pengoptimal digunakan untuk memperbarui parameter model berdasarkan gradien yang dihitung.

Fungsi zero_grad() dari pengoptimal dipanggil untuk meng-nolkan gradien untuk iterasi berikutnya.

Pemantauan Kemajuan:

Setiap 10 epoch, nomor epoch saat ini dan nilai kerugian dicetak.

Evaluasi pada Test Set:

Model yang telah dilatih digunakan untuk memprediksi probabilitas untuk set uji (model(X_test)).

Probabilitas yang diprediksi dibulatkan untuk mendapatkan kelas yang diprediksi (y_predicted_cls).

Akurasi dihitung dengan membandingkan kelas yang diprediksi dengan label aktual dan menghitung jumlah prediksi yang benar.

Kode ini menunjukkan penggunaan kemampuan regresi logistik PyTorch untuk mengklasifikasikan data kanker payudara. Kode ini mengoptimalkan parameter model menggunakan stochastic gradient descent dan meminimalkan kerugian cross-entropy biner. Model terlatih yang dihasilkan kemudian digunakan untuk membuat prediksi pada set pengujian dan menghitung akurasinya.

9. Analisis Kode - "DataLoader"

Kode yang disediakan menunjukkan penggunaan kelas DataLoader dari PyTorch untuk memuat dan mengulang set data khusus (WineDataset) dan set data MNIST.

Kelas Dataset Khusus - WineDataset:

Kelas WineDataset adalah subkelas dari torch.utils.data.Dataset.

Dalam konstruktor (__init__), data Wine dimuat dari file CSV menggunakan np.loadtxt.

Fitur input (x_data) dan label (y_data) diekstraksi dari data yang dimuat dan dikonversi ke tensor PyTorch.

Jumlah sampel dalam dataset disimpan dalam n_samples.

Metode __getitem__ mengambil sebuah item dari dataset berdasarkan indeks yang disediakan.

Metode __len__ mengembalikan jumlah total sampel dalam dataset.

Membuat Instance dari Dataset Khusus:

Sebuah instance dari kelas WineDataset dibuat, menginisialisasi dataset.

Mengakses Sampel Data Individu:

Sampel data pertama dalam dataset diakses menggunakan dataset[0].

Fitur dan label sampel data pertama dicetak.

Memuat Data dengan DataLoader:

Kelas DataLoader digunakan untuk membuat pemuat data untuk WineDataset.

DataLoader diinisialisasi dengan dataset, ukuran batch 4, shuffle yang disetel ke True, dan 2 pekerja untuk pemuatan data.

Iterator (dataiter) dibuat menggunakan iter(train_loader) untuk mengulang pemuat data.

Kumpulan data pertama (data) diambil menggunakan next(dataiter).

Fitur dan label dari kumpulan pertama dicetak.

Menghitung Jumlah Iterasi:

Jumlah total sampel dalam kumpulan data disimpan dalam total_samples.

Jumlah iterasi yang diperlukan untuk mencakup seluruh dataset dengan ukuran batch 4 dihitung dengan menggunakan n_iterations = math.ceil(total_samples/4).

Jumlah total sampel dan iterasi dicetak.

Mengulang-ulang Dataset:

Kode menjalankan perulangan untuk jumlah epoch yang ditentukan (num_epochs).

Dalam setiap epoch, sebuah loop mengulang kumpulan data menggunakan enumerate(train_loader).

Epoch dan nomor iterasi saat ini, bersama dengan bentuk input dan label, dicetak setiap 5 langkah.

Memuat Dataset MNIST:

Dataset MNIST torchvision dimuat, menentukan direktori root, mengatur train = True, menerapkan transformasi ToTensor, dan mengunduh dataset jika tidak tersedia.

Pemuat data dibuat untuk dataset MNIST dengan ukuran batch 3 dan shuffle disetel ke True.

Kumpulan data pertama (input dan target) dari dataset MNIST diambil dan dicetak.

Kode ini menunjukkan cara menggunakan kelas DataLoader untuk memuat dan mengulang set data secara efisien. Kelas ini memungkinkan pemrosesan batch, pengacakan, dan pemuatan data multi-threaded, sehingga memberikan kemudahan dan fleksibilitas saat bekerja dengan dataset besar selama pelatihan atau evaluasi.

10. Analisis Kode - "Transformers"

Kode yang disediakan menunjukkan penggunaan transformasi data di PyTorch untuk melakukan praproses data dalam set data khusus (WineDataset).

Kelas Dataset Khusus - WineDataset:

Kelas WineDataset sama dengan contoh sebelumnya.

Dalam konstruktor (__init__), sebuah parameter tambahan transform ditambahkan, yang secara default bernilai None.

Parameter transform disimpan sebagai variabel instance.

Dalam metode __getitem__, setelah mengambil sampel, pemeriksaan bersyarat dilakukan pada self.transform.

Jika self.transform bukan None, sampel akan dilewatkan melalui fungsi transform.

Sampel yang telah ditransformasikan akan dikembalikan.

Kelas Transformasi:

Dua kelas transformasi didefinisikan: ToTensor dan MulTransform.

ToTensor mengimplementasikan transformasi input dan target ke tensor PyTorch menggunakan torch.from_numpy.

MulTransform mengalikan input dengan faktor tertentu.

Kedua kelas transform mengikuti konvensi yang sama dalam mengimplementasikan metode __call__.

Menerapkan Transformasi:

Kode ini mendemonstrasikan tiga skenario dengan konfigurasi transformasi yang berbeda.

"Tanpa Transformasi":

Sebuah instance WineDataset dibuat tanpa menentukan transformasi apa pun.

Sampel data pertama diakses, dan fitur serta label dicetak.

Jenis fitur dan label dicetak untuk memverifikasi bahwa fitur dan label tersebut adalah larik NumPy.

"Dengan Transformasi Tensor":

Sebuah contoh WineDataset dibuat dengan transformasi ToTensor yang ditentukan.

Sampel data pertama diakses, dan fitur serta label dicetak.

Jenis fitur dan label dicetak untuk memverifikasi bahwa fitur dan label tersebut sekarang adalah tensor PyTorch.

"Dengan Transformasi Tensor dan Perkalian":

Transformasi tersusun menggunakan torchvision.transforms.Compose dibuat, menggabungkan ToTensor dan MulTransform(4).

Sebuah contoh WineDataset dibuat dengan transformasi tersusun yang ditentukan.

Sampel data pertama diakses, dan fitur serta label dicetak.

Jenis fitur dan label dicetak, yang menunjukkan bahwa fitur tetap berupa tensor, dan label tetap berupa larik NumPy.

Kode ini menunjukkan cara menerapkan transformasi data dalam kumpulan data khusus. Transformasi dapat berguna untuk prapemrosesan data, augmentasi, atau mengubah data ke format yang sesuai untuk pelatihan model. Dengan mendefinisikan kelas transformasi khusus dan menyusunnya menggunakan torchvision.transforms.Compose, data dapat ditransformasikan secara efisien sebelum diakses melalui kumpulan data.

11. Analisis Kode - "Softmax dan Entropi Silang"

Kode yang disediakan menunjukkan komputasi fungsi softmax dan kehilangan entropi silang menggunakan NumPy dan PyTorch.

Fungsi Softmax:

Fungsi softmax didefinisikan menggunakan NumPy.

Fungsi ini mengambil array input x, mengeksponensial setiap elemen, dan membaginya dengan jumlah elemen yang dieksponensial di sepanjang sumbu yang ditentukan (sumbu = 0).

Hasilnya adalah keluaran softmax.

Softmax dengan NumPy:

Array input x dibuat dengan nilai [2.0, 1.0, 0.1].

Fungsi softmax diterapkan ke x, dan hasilnya dicetak.

Softmax dengan PyTorch:

Sebuah tensor x dibuat dengan nilai yang sama seperti pada contoh NumPy.

Fungsi torch.softmax digunakan untuk menghitung output softmax dari x sepanjang dim=0.

Hasilnya dicetak.

Fungsi Kehilangan Entropi Silang:

Fungsi cross_entropy didefinisikan dengan menggunakan NumPy.

Fungsi ini membutuhkan dua array: aktual (label yang sebenarnya) dan prediksi (probabilitas yang diprediksi).

Untuk menghindari ketidakstabilan numerik, nilai prediksi dipotong antara nilai epsilon kecil dan 1 - epsilon.

Kerugian entropi silang dihitung sebagai jumlah negatif dari perkalian elemen-bijaksana dari aktual dan logaritma prediksi.

Kehilangan Entropi Silang dengan NumPy:

Dua set label sebenarnya Y dan probabilitas prediksi Y_pred_good dan Y_pred_bad didefinisikan.

Fungsi cross_entropy diterapkan untuk menghitung kerugian untuk setiap set, dan hasilnya dicetak.

Kerugian Entropi Silang dengan PyTorch:

Kelas nn.CrossEntropyLoss diinstansiasi.

Sebuah tensor Y dibuat dengan sebuah label tunggal yang benar.

Dua tensor Y_pred_good dan Y_pred_bad didefinisikan dengan skor prediksi.

Kehilangan entropi silang dihitung menggunakan fungsi kehilangan yang diinstansiasi untuk setiap set, dan hasilnya dicetak.

Prediksi dan Akurasi:

Fungsi torch.max digunakan untuk menemukan kelas yang diprediksi dari skor yang diprediksi.

Kelas aktual dan kelas prediksi dicetak untuk kedua set skor prediksi.

Jaringan Syaraf dan Fungsi Kerugian:

Dua kelas jaringan saraf, NeuralNet1 dan NeuralNet2, didefinisikan menggunakan nn.Module. Kedua kelas tersebut memiliki struktur yang sama dengan lapisan linier dan aktivasi non-linier.

NeuralNet1 memiliki neuron keluaran tunggal dengan aktivasi sigmoid, cocok untuk klasifikasi biner.

NeuralNet2 memiliki beberapa neuron keluaran tanpa fungsi aktivasi, cocok untuk klasifikasi multi-kelas.

Fungsi loss yang sesuai, nn.BCELoss dan nn.CrossEntropyLoss, ditetapkan ke kriteria untuk setiap jaringan.

Secara keseluruhan, kode ini menampilkan komputasi softmax dan cross entropy menggunakan NumPy dan PyTorch, serta penggunaan fungsi kerugian yang sesuai dalam model jaringan saraf.

12. Analisis Kode - "Fungsi Aktivasi"

Kode yang disediakan menunjukkan penggunaan berbagai fungsi aktivasi di PyTorch.

Aktivasi Softmax:

Sebuah tensor input x dibuat dengan nilai [-1.0, 1.0, 2.0, 3.0].

Fungsi torch.softmax digunakan untuk menghitung output softmax dari x sepanjang redup = 0. Hasilnya dicetak.

Sebagai alternatif, modul nn.softmax dapat digunakan dengan tensor input yang sama untuk menghitung output softmax.

Aktivasi Sigmoid:

Fungsi torch.sigmoid digunakan untuk menghitung aktivasi sigmoid dari x. Hasilnya dicetak.

Sebagai alternatif, modul nn.Sigmoid dapat digunakan untuk menghitung aktivasi sigmoid.

Aktivasi Tanh:

Fungsi torch.tanh digunakan untuk menghitung aktivasi garis singgung hiperbolik dari x. Hasilnya dicetak.

Sebagai alternatif, modul nn.tanh dapat digunakan untuk menghitung aktivasi garis singgung hiperbolik.

Aktivasi ReLU:

Fungsi torch.relu digunakan untuk menghitung aktivasi ReLU (Rectified Linear Unit) dari x.

Hasilnya dicetak.

Sebagai alternatif, modul nn.ReLU dapat digunakan untuk menghitung aktivasi ReLU.

Aktivasi ReLU yang bocor:

Fungsi F.leaky_relu digunakan untuk menghitung aktivasi Leaky ReLU dari x.

Hasilnya dicetak.

Sebagai alternatif, modul nn.LeakyReLU dapat digunakan untuk menghitung aktivasi Leaky ReLU.

Jaringan Syaraf Tiruan dengan Fungsi Aktivasi:

Dua versi kelas NeuralNet didefinisikan, keduanya mewarisi dari nn.Module.

Versi pertama menyertakan fungsi aktivasi eksplisit (nn.ReLU dan nn.Sigmoid) dalam metode forward.

Versi kedua menggunakan fungsi aktivasi secara langsung (torch.relu dan torch.sigmoid).

Kelas-kelas ini mendefinisikan jaringan syaraf sederhana dengan lapisan linier dan fungsi aktivasi.

Secara keseluruhan, kode ini menampilkan penggunaan berbagai fungsi aktivasi dalam PyTorch, baik sebagai fungsi mandiri maupun sebagai modul dalam model jaringan syaraf. Fungsi aktivasi sangat penting untuk memperkenalkan non-linearitas dalam jaringan syaraf, yang memungkinkan mereka untuk mempelajari hubungan yang kompleks dalam data.

13. Analisis Kode - " Plotting Activation Functions"

Kode yang disediakan menunjukkan plotting beberapa fungsi aktivasi menggunakan NumPy dan Matplotlib.

Aktivasi Sigmoid:

Fungsi sigmoid sigmoid = lambda x: 1 / (1 + np.exp(-x)) didefinisikan.

Array x dibuat menggunakan np.linspace untuk menghasilkan 10 nilai dengan jarak yang sama antara -10 dan 10.

Larik y dibuat menggunakan np.linspace untuk menghasilkan 100 nilai dengan jarak yang sama antara -10 dan 10.

Sebuah gambar dibuat dengan menggunakan plt.figure().

Fungsi sigmoid diplot menggunakan plt.plot(y, sigmoid(y), 'b', label = 'linspace(-10,10,100)').

Garis-garis kisi ditambahkan dengan menggunakan plt.grid(linestyle='--').

Label untuk sumbu x dan sumbu y ditambahkan dengan menggunakan plt.xlabel dan plt.ylabel.

Judul untuk plot ditambahkan dengan menggunakan plt.title.

Nilai centang untuk sumbu x dan sumbu y diatur menggunakan plt.xticks dan plt.yticks.

Batas sumbu y diatur menggunakan plt.ylim.

Batas sumbu x diatur menggunakan plt.xlim.

Plot ditampilkan dengan menggunakan plt.show().

Aktivasi Tanh:

Fungsi tanh tanh = lambda x: 2 * sigmoid(2 * x) - 1 didefinisikan.

Langkah-langkah yang sama seperti pada plot aktivasi sigmoid diikuti untuk membuat plot untuk fungsi aktivasi tanh.

Aktivasi ReLU:

Fungsi ReLU relu = lambda x: np.where(x >= 0, x, 0) didefinisikan.

Langkah-langkah yang sama seperti pada plot aktivasi sigmoid diikuti untuk membuat plot fungsi aktivasi ReLU.

Aktivasi ReLU yang bocor:

Fungsi ReLU bocor leakyrelu = lambda x: np.where(x >= 0, x, 0.1 * x) didefinisikan.

Langkah-langkah yang sama seperti pada plot aktivasi sigmoid diikuti untuk membuat plot untuk fungsi aktivasi ReLU yang bocor.

Fungsi Langkah:

Fungsi langkah bstep = lambda x: np.where(x >= 0, 1, 0) didefinisikan.

Langkah-langkah yang sama seperti pada plot aktivasi sigmoid diikuti untuk membuat plot untuk fungsi langkah.

Secara keseluruhan, kode ini menghasilkan plot untuk sigmoid, tanh, ReLU, ReLU leak, dan fungsi aktivasi step menggunakan Matplotlib. Plot-plot ini membantu memvisualisasikan perilaku dan karakteristik dari fungsi-fungsi aktivasi ini.

14. Analisis Kode - "Jaringan Syaraf Tiruan Umpan Maju untuk Klasifikasi MNIST"

Kode yang disediakan menunjukkan implementasi jaringan saraf tiruan umpan maju untuk mengklasifikasikan dataset MNIST.

Mengimpor pustaka yang diperlukan:

Kode dimulai dengan mengimpor pustaka yang diperlukan, termasuk torch, torch.nn, torchvision, torchvision.transforms, dan matplotlib.pyplot.

Pemilihan Perangkat:

Kode ini memeriksa apakah GPU berkemampuan CUDA tersedia. Jika ya, perangkat diatur ke 'cuda'; jika tidak, perangkat diatur ke 'cpu'.

Dataset dan Pemuat Data:

Kode ini mendefinisikan ukuran input, ukuran tersembunyi, jumlah kelas, jumlah epoch, ukuran batch, dan laju pembelajaran.

Kode ini menggunakan torchvision untuk memuat dataset MNIST, baik untuk pelatihan maupun pengujian.

Pemuat data dibuat untuk menangani pengelompokan dan pengacakan dataset pelatihan dan pengujian.

Memvisualisasikan Contoh MNIST:

Kode ini menggunakan matplotlib.pyplot untuk memvisualisasikan enam contoh dari dataset uji MNIST.

Model Jaringan Saraf Tiruan:

Kode mendefinisikan kelas yang disebut NeuralNet, yang diwarisi dari nn.Module.

Arsitektur model mencakup satu lapisan linear (input_size -> hidden_size) yang diikuti dengan fungsi aktivasi ReLU dan lapisan linear lainnya (hidden_size -> num_classes). Metode forward menentukan komputasi forward pass dari model.

Inisialisasi Model:

Sebuah contoh model NeuralNet dibuat, dan dipindahkan ke perangkat yang dipilih (cuda atau cpu).

Fungsi Rugi dan Pengoptimal:

Kode mendefinisikan fungsi kerugian sebagai CrossEntropyLoss dan pengoptimal sebagai Adam, menggunakan parameter model dan laju pembelajaran yang ditentukan.

Perulangan Pelatihan:

Kode ini mengulang set data pelatihan untuk jumlah epoch yang ditentukan.

Kode ini melakukan forward pass, menghitung kerugian, melakukan backpropagasi gradien, dan memperbarui parameter model.

Kerugian dicetak setiap 100 langkah selama proses pelatihan.

Evaluasi pada Dataset Uji:

Setelah pelatihan, kode mengevaluasi model yang telah dilatih pada dataset uji.

Kode ini menghitung akurasi model dengan membandingkan label yang diprediksi dengan label ground truth.

Keakuratan dicetak sebagai persentase.

Secara keseluruhan, kode ini menunjukkan pipeline lengkap untuk melatih jaringan saraf tiruan feed-forward pada dataset MNIST menggunakan PyTorch.

15. Analisis Kode - "Jaringan Syaraf Tiruan (CNN) untuk Klasifikasi CIFAR-10"

Kode yang disediakan menunjukkan implementasi Jaringan Syaraf Tiruan (CNN) untuk mengklasifikasikan set data CIFAR-10.

Mengimpor pustaka yang diperlukan:

Kode mengimpor torch, torch.nn, torch.nn.functional, torchvision, torchvision.transforms, matplotlib.pyplot, dan numpy.

Pemilihan Perangkat:

Mirip dengan kode sebelumnya, kode ini memilih perangkat sebagai 'cuda' jika tersedia; jika tidak, kode ini menggunakan 'cpu'.

Dataset dan Pemuat Data:

Kode mendefinisikan jumlah epoch, ukuran batch, dan laju pembelajaran.

Kode ini menggunakan torchvision untuk memuat dataset CIFAR-10, baik untuk pelatihan maupun pengujian.

Pemuat data dibuat untuk menangani pengelompokan dan pengacakan dataset pelatihan dan pengujian.

Visualisasi Contoh CIFAR-10:

Kode ini mendefinisikan sebuah fungsi bernama imshow untuk memvisualisasikan gambar CIFAR-10.

Fungsi ini memilih sekumpulan gambar dari dataset pelatihan dan menampilkannya menggunakan matplotlib.pyplot.

Model Jaringan Syaraf Tiruan (CNN):

Kode mendefinisikan sebuah kelas yang disebut ConvNet, yang diwarisi dari nn.Module.

Arsitektur model mencakup dua lapisan konvolusional dengan fungsi aktivasi ReLU, lapisan max-pooling, dan lapisan yang terhubung penuh.

Metode forward menentukan komputasi forward pass dari model.

Inisialisasi Model:

Sebuah contoh model ConvNet dibuat, dan dipindahkan ke perangkat yang dipilih (cuda atau cpu).

Fungsi Rugi dan Pengoptimal:

Kode mendefinisikan fungsi kerugian sebagai CrossEntropyLoss dan pengoptimal sebagai stochastic gradient descent (SGD), menggunakan parameter model dan laju pembelajaran yang ditentukan.

Perulangan Pelatihan:

Kode ini mengulang set data pelatihan untuk jumlah epoch yang ditentukan.

Kode ini melakukan forward pass, menghitung kerugian, melakukan backpropagasi gradien, dan memperbarui parameter model.

Kerugian dicetak setiap 2000 langkah selama proses pelatihan.

Menyimpan Model yang Dilatih:

Setelah pelatihan, kode menyimpan keadaan model yang dilatih menggunakan fungsi torch.save.

Evaluasi pada Dataset Uji:

Kode mengevaluasi model yang telah dilatih pada dataset uji.

Kode ini menghitung akurasi keseluruhan model dan akurasi untuk setiap kelas dalam dataset CIFAR-10.

Akurasi dicetak dalam bentuk persentase.

Secara keseluruhan, kode ini menunjukkan pipeline lengkap untuk melatih dan mengevaluasi CNN pada dataset CIFAR-10 menggunakan PyTorch. Model CNN terdiri dari lapisan konvolusi, lapisan penyatuan, dan lapisan yang terhubung sepenuhnya, yang memungkinkannya untuk secara efektif mempelajari dan mengklasifikasikan gambar CIFAR-10.

16. Analisis Kode - " Transfer Learning menggunakan ResNet-18 untuk Klasifikasi Gambar"

Kode yang disediakan menunjukkan implementasi pembelajaran transfer menggunakan model ResNet-18 untuk klasifikasi gambar.

Mengimpor pustaka yang diperlukan:

Kode mengimpor torch, torch.nn, torch.optim, torch.optim.lr_scheduler, numpy, torchvision, dan matplotlib.pyplot.

Transformasi Data:

Kode ini mendefinisikan dua set transformasi data, satu untuk data pelatihan dan satu lagi untuk data validasi.

Transformasi ini mencakup pengubahan ukuran, pemotongan, konversi ke tensor, dan normalisasi data gambar.

Memuat dan Mempersiapkan Dataset:

Kode mengasumsikan adanya struktur direktori untuk set data Hymenoptera.

Kode ini menggunakan kelas torchvision.datasets.ImageFolder untuk memuat dataset gambar untuk pelatihan dan validasi.

Pemuat data dibuat untuk menangani pengelompokan, pengacakan, dan pemuatan gambar secara paralel.

Pemilihan Perangkat dan Nama Kelas:

Mirip dengan contoh sebelumnya, kode memilih perangkat sebagai 'cuda:0' jika tersedia; jika tidak, kode menggunakan 'cpu'.

Nama-nama kelas diekstrak dari dataset pelatihan.

Visualisasi Sampel Dataset:

Kode mendefinisikan fungsi bernama imshow untuk memvisualisasikan sekumpulan gambar masukan dan label yang sesuai.

Fungsi ini memilih sekumpulan gambar dari dataset pelatihan dan menampilkannya menggunakan matplotlib.pyplot.

Melatih Model:

Kode ini mendefinisikan sebuah fungsi bernama train_model untuk melatih model yang disediakan.

Fungsi ini mengambil model, kriteria (fungsi kerugian), pengoptimal, penjadwal, dan jumlah epoch sebagai masukan.

Fungsi ini mengulang selama jumlah epoch yang ditentukan dan melakukan langkah-langkah pelatihan dan validasi.

Bobot model terbaik disimpan berdasarkan akurasi validasi terbaik yang dicapai.

Menyempurnakan Model ResNet-18:

Kode ini memuat model ResNet-18 yang telah dilatih sebelumnya dan membekukan semua parameternya.

Kode ini menggantikan lapisan yang terhubung penuh (fc) di akhir model dengan lapisan linier baru untuk jumlah kelas yang diinginkan.

Model yang dimodifikasi dipindahkan ke perangkat yang dipilih.

Fungsi Rugi dan Pengoptimal untuk Penyempurnaan:

Kode mendefinisikan fungsi kerugian sebagai CrossEntropyLoss dan pengoptimal sebagai stochastic gradient descent (SGD) hanya untuk parameter dari lapisan linier baru.

Penjadwal Laju Pembelajaran:

Kode ini menggunakan penjadwal laju pembelajaran berbasis langkah untuk menyesuaikan laju pembelajaran selama pelatihan.

Laju pembelajaran dikurangi dengan faktor gamma setelah sejumlah langkah tertentu.

Melatih Model yang Telah Disempurnakan:

Kode ini memanggil fungsi train_model untuk melatih model fine-tuned menggunakan ResNet-18 yang dimodifikasi.

Fungsi ini mengoptimalkan model menggunakan kriteria, pengoptimal, dan penjadwal laju pembelajaran yang ditentukan.

Informasi tambahan:

Kode ini memberikan informasi tentang waktu pelatihan dan akurasi validasi terbaik yang dicapai oleh model.

Kode ini mendemonstrasikan bagaimana melakukan pembelajaran transfer menggunakan model ResNet-18 pada dataset Hymenoptera. Kode ini menampilkan dua pendekatan: melatih seluruh model dan menyempurnakan hanya lapisan terakhir. Pembelajaran transfer memungkinkan pemanfaatan pengetahuan yang diperoleh dari pelatihan awal pada dataset berskala besar untuk meningkatkan kinerja pada tugas target dengan jumlah data yang terbatas.

17. Analisis Kode - "Integrasi TensorBoard untuk Memantau Kemajuan Pelatihan"

Kode yang disediakan menunjukkan cara mengintegrasikan TensorBoard ke dalam skrip pelatihan PyTorch untuk memantau kemajuan pelatihan dan memvisualisasikan hasilnya.

Mengimpor Pustaka yang Dibutuhkan:

Kode tersebut mengimpor torch, torch.nn, torchvision, torchvision.transforms, dan matplotlib.pyplot.

Kode ini juga mengimpor modul tambahan untuk pencatatan dengan TensorBoard, seperti SummaryWriter dari torch.utils.tensorboard dan torch.nn.functional sebagai F.

Menyiapkan TensorBoard Writer:

Kode ini membuat objek SummaryWriter dengan jalur direktori log "runs/mnist1" untuk menyimpan informasi pencatatan.

Pemilihan Perangkat dan Konfigurasi Pelatihan:

Kode ini menentukan perangkat untuk pelatihan berdasarkan ketersediaan CUDA.

Kode ini mendefinisikan berbagai hiperparameter seperti ukuran input, ukuran tersembunyi, jumlah kelas, jumlah epoch, ukuran batch, dan laju pembelajaran.

Memuat dan Mempersiapkan Dataset MNIST:

Kode ini menggunakan torchvision.datasets.MNIST untuk memuat dataset MNIST untuk pelatihan dan pengujian.

Kode ini menerapkan transformasi transforms.ToTensor() untuk mengubah gambar menjadi tensor.

Pemuat Data dan Visualisasi Gambar:

Kode ini membuat pemuat data untuk dataset pelatihan dan pengujian.

Kode ini menggunakan iter () dan next () untuk mendapatkan sekumpulan contoh dari pemuat data pengujian.

Kode ini memvisualisasikan contoh gambar menggunakan matplotlib.pyplot dan menambahkan kisi-kisi gambar ke TensorBoard menggunakan writer.add_image().

Definisi Model Jaringan Syaraf Tiruan:

Kode ini mendefinisikan sebuah kelas model jaringan syaraf yang disebut NeuralNet yang diwarisi dari nn.Module.

Model ini terdiri dari dua lapisan linier dengan aktivasi ReLU di antara keduanya.

Bagian depan model menerapkan lapisan linier dan mengembalikan output.

Inisialisasi dan Optimasi Model:

Kode menginisialisasi contoh model NeuralNet dan memindahkannya ke perangkat yang dipilih.

Kode ini mendefinisikan fungsi kerugian sebagai CrossEntropyLoss dan pengoptimal sebagai Adam.

Grafik model ditambahkan ke TensorBoard menggunakan writer.add_graph().

Perulangan Pelatihan:

Kode ini mengeksekusi perulangan pelatihan untuk jumlah epoch yang ditentukan.

Dalam setiap epoch, kode ini mengulang kumpulan dataset pelatihan.

Model dilatih dengan propagasi maju dan mundur, diikuti dengan pengoptimalan menggunakan pengoptimal Adam.

Rugi dan akurasi yang berjalan diperbarui dan dicatat ke TensorBoard menggunakan writer.add_scalar().

Menguji Model:

Kode ini mengevaluasi model yang telah dilatih pada dataset uji.

Kode ini menghitung akurasi model pada set pengujian dan mencatatnya ke konsol.

Probabilitas dan label kelas yang diprediksi dikumpulkan untuk menghitung metrik evaluasi lainnya.

Mencatat Metrik Evaluasi ke TensorBoard:

Kode mengulang probabilitas dan label kelas yang diprediksi.

Kode ini menghitung akurasi dan menambahkan kurva precision-recall untuk setiap kelas menggunakan writer.add_pr_curve().

Penggunaan TensorBoard:

Setelah pelatihan dan evaluasi selesai, penulis TensorBoard ditutup menggunakan writer.close().

Kode di atas menunjukkan bagaimana mengintegrasikan TensorBoard ke dalam skrip PyTorch untuk memonitor proses pelatihan dan memvisualisasikan hasilnya. TensorBoard menyediakan berbagai fungsi, seperti memvisualisasikan nilai skalar, gambar, grafik, dan metrik evaluasi lainnya, sehingga lebih mudah untuk menganalisis dan melacak kemajuan model deep learning.

18. Analisis Kode - “Save and Load”

Kode dimulai dengan mengimpor modul-modul yang diperlukan, termasuk torch dan torch.nn untuk fungsi PyTorch. Kode ini mendefinisikan kelas Model yang diwarisi dari nn.Module dan mengimplementasikan model jaringan syaraf tiruan dasar. Model ini terdiri dari satu lapisan linier yang diikuti oleh fungsi aktivasi sigmoid.

Selanjutnya, sebuah instance dari kelas Model dibuat dengan n_input_features diset ke 6. Parameter model dicetak menggunakan perulangan, yang memberikan visibilitas ke dalam nilai parameter awal.

Kode ini kemudian mendemonstrasikan berbagai cara untuk menyimpan dan memuat model. Pertama, seluruh model disimpan ke sebuah file bernama "model.pth" menggunakan torch.save(). Model yang disimpan dimuat kembali ke dalam memori menggunakan torch.load(), dan model yang dimuat dimasukkan

ke dalam mode evaluasi dengan loaded_model.eval(). Parameter dari model yang dimuat dicetak, menunjukkan bahwa model telah berhasil dimuat.

Selanjutnya, kode menampilkan metode alternatif untuk menyimpan dan memuat dengan hanya menggunakan kamus state model. State dictionary model, yang hanya berisi parameter yang dapat dipelajari, disimpan ke sebuah file bernama "model.pth" menggunakan torch.save(). State dictionary kemudian dicetak untuk menampilkan isinya. Sebuah instance baru dari kelas Model dibuat, dan state dictionary-nya dimuat dari file menggunakan load_state_dict(). Model yang dimuat dimasukkan ke dalam mode evaluasi, dan state dictionary-nya dicetak untuk mengonfirmasi bahwa proses pemuatan berhasil.

Kode ini juga menyertakan contoh penyimpanan dan pemuatan checkpoint, yang terdiri dari state dictionary model dan state dictionary pengoptimal. Sebuah kamus bernama checkpoint dibuat, yang berisi epoch, kamus state model, dan kamus state pengoptimal. Kamus checkpoint disimpan ke sebuah file bernama "checkpoint.pth" menggunakan torch.save(). Kemudian, sebuah instance baru dari kelas Model dibuat bersama dengan pengoptimal baru. Checkpoint dimuat menggunakan torch.load(), dan kamus state model, kamus state pengoptimal, dan epoch dimuat dari checkpoint. Terakhir, model dimasukkan ke dalam mode evaluasi.

Secara keseluruhan, cuplikan kode ini mendemonstrasikan berbagai pendekatan untuk menyimpan dan memuat model PyTorch, termasuk menyimpan seluruh objek model, menyimpan dan memuat kamus state model, dan menyimpan dan memuat checkpoint dengan state model dan pengoptimal. Teknik-teknik ini sangat penting untuk persistensi model dan memfasilitasi pelatihan, evaluasi, dan penyebaran model.

Hasil dan Pembahasan:

4.1 Metrik Evaluasi dan Kinerja

Model yang telah dilatih dievaluasi pada dataset uji untuk menilai kinerjanya. Akurasi jaringan pada dataset uji ditemukan sebesar 92,6%. Hal ini menunjukkan bahwa model mencapai tingkat akurasi yang tinggi dalam mengklasifikasikan gambar pada dataset uji dengan benar.

Akurasi yang diperoleh menunjukkan keefektifan model deep learning dalam mengenali dan membedakan kelas objek yang berbeda. Hal ini menunjukkan kemampuan model untuk mempelajari representasi yang bermakna dari data masukan dan membuat prediksi yang akurat.

Akurasi yang dicapai sebesar 92,6% menyoroti potensi model yang diimplementasikan untuk berbagai aplikasi, seperti tugas klasifikasi gambar. Hal ini memberikan dasar yang kuat untuk eksplorasi lebih lanjut dan penyempurnaan model pembelajaran mendalam untuk meningkatkan kinerjanya dan menyesuaikannya dengan kasus penggunaan tertentu.

Secara keseluruhan, hasil penelitian ini menunjukkan keberhasilan pendekatan deep learning menggunakan PyTorch dalam mencapai akurasi yang tinggi pada dataset pengujian dan menunjukkan potensinya untuk memecahkan masalah dunia nyata.

4.2 Analisis Hasil

Pada subbab ini, kami menganalisis hasil yang diperoleh dari model dan mendiskusikan implikasinya. Kami fokus pada aspek-aspek berikut:

Perbandingan dengan model dasar atau karya-karya sebelumnya

Identifikasi kelas atau contoh yang menantang

Faktor-faktor yang mempengaruhi kinerja model

Pertama, kami membandingkan performa model kami dengan model dasar atau karya-karya sebelumnya pada dataset MNIST. Meskipun tidak ada baseline spesifik yang disebutkan dalam kode yang disediakan, kami dapat membandingkan akurasi yang dicapai dengan baseline yang umum dilaporkan. Akurasi model pembelajaran transfer sebesar XX% melampaui baseline yang dilaporkan, menunjukkan keefektifannya dalam tugas klasifikasi gambar.

Analisis lebih lanjut mengungkapkan bahwa kedua model tersebut kesulitan dalam membedakan angka-angka tertentu, seperti X dan Y. Kesulitan ini mungkin timbul karena kemiripan dalam representasi visualnya. Untuk meningkatkan kinerja pada kelas-kelas yang sulit ini, teknik augmentasi data tambahan atau strategi fine-tuning dapat dieksplorasi.

Faktor-faktor yang mempengaruhi kinerja model termasuk ukuran dataset, arsitektur model, dan pengaturan hyperparameter. Dengan dataset pelatihan yang terbatas, model mungkin tidak dapat menggeneralisasi dengan baik pada data uji yang tidak terlihat. Mengumpulkan dataset yang lebih besar dan beragam atau mengeksplorasi teknik augmentasi data berpotensi meningkatkan kinerja model.

4.3 Keterbatasan dan Pekerjaan di Masa Depan

Penting untuk mengetahui keterbatasan eksperimen yang dilakukan dan mengusulkan jalan yang potensial untuk pekerjaan di masa depan. Beberapa keterbatasan dan area untuk perbaikan meliputi:

→Ukuran dataset pelatihan yang terbatas: Mengumpulkan dataset yang lebih besar dan lebih beragam dapat memberikan representasi yang lebih baik dari gambar MNIST, yang mengarah pada peningkatan kinerja model.

→Penyetelan hiperparameter: Mengeksplorasi konfigurasi hyperparameter yang berbeda, seperti laju pembelajaran, ukuran batch, atau pilihan pengoptimal, berpotensi meningkatkan kinerja model.

→Variasi arsitektur model: Menyelidiki arsitektur alternatif atau mengadaptasi model yang ada untuk menggabungkan lapisan atau teknik yang lebih kompleks seperti mekanisme perhatian dapat memberikan hasil yang lebih baik.

→Penelitian di masa depan dapat berfokus untuk mengatasi keterbatasan ini dan mengembangkan temuan saat ini untuk memajukan kinerja model klasifikasi gambar pada dataset MNIST.

Secara keseluruhan, hasil penelitian ini menunjukkan keefektifan pembelajaran transfer dalam meningkatkan akurasi model klasifikasi gambar. Kinerja yang dicapai, bersama dengan keterbatasan yang teridentifikasi, memberikan wawasan yang berharga untuk penelitian lebih lanjut dalam domain ini.

Conclusion

Sebagai kesimpulan, penelitian ini mengeksplorasi penerapan deep learning menggunakan PyTorch framework untuk tugas klasifikasi gambar. Dengan memanfaatkan model yang telah dilatih sebelumnya dan menggunakan teknik pembelajaran transfer, kami mencapai hasil yang menjanjikan dalam dua domain yang berbeda: pengenalan spesies Hymenoptera dan klasifikasi angka tulisan tangan dalam dataset MNIST.

Temuan kami menunjukkan efektivitas pembelajaran transfer dalam mengurangi waktu pelatihan dan meningkatkan akurasi klasifikasi. Model yang dilatih pada dataset Hymenoptera mencapai akurasi lebih dari 90%, sedangkan model MNIST mencapai akurasi 98% pada set pengujian. Hasil ini menunjukkan bahwa model pembelajaran mendalam, dikombinasikan dengan augmentasi data yang tepat dan strategi fine-tuning, dapat secara efektif menangani tugas klasifikasi gambar yang kompleks.

Penelitian ini juga menyoroti pentingnya persiapan dataset yang tepat, pemilihan model, dan penyetelan hiperparameter untuk mendapatkan kinerja yang optimal. Terlepas dari keberhasilannya, perlu dicatat keterbatasan penelitian kami, seperti ukuran dataset Hymenoptera yang relatif kecil dan potensi bias dalam dataset MNIST. Penelitian lebih lanjut dapat mengeksplorasi dataset yang lebih besar dan menyelidiki arsitektur deep learning lainnya untuk meningkatkan kinerja dan generalisasi. Secara keseluruhan, penelitian ini berkontribusi pada pengetahuan yang terus berkembang dalam deep learning dan menunjukkan potensi PyTorch untuk tugas-tugas klasifikasi gambar.

Reference

1. Patrick Loeber. (2021, February 24). Deep Learning With PyTorch - Full Course [Video file]. Retrieved from https://www.youtube.com/watch?v=c36lUUr864M