TECHNICAL REPORT : DEEP LEARNING WITH PYTORCH

Dinda Rahma 1103204051

I. Pendahuluan

Deep learning adalah cabang dari pembelajaran mesin (machine learning) yang bertujuan untuk melatih model yang dapat mempelajari representasi yang mendalam (hierarchical) dari data. Model deep learning yang kompleks, seperti jaringan saraf tiruan (neural networks), telah terbukti sangat sukses dalam berbagai tugas seperti pengenalan gambar, pemrosesan bahasa alami, dan prediksi.

PyTorch adalah sebuah framework deep learning yang populer dan kuat yang dikembangkan oleh Facebook AI Research. PyTorch menawarkan fleksibilitas dan ekspresivitas tinggi dalam mengembangkan model deep learning. Hal ini membuatnya menjadi salah satu pilihan utama bagi para peneliti dan praktisi yang ingin mengimplementasikan dan menguji konsep-konsep baru dalam deep learning.

PyTorch menyediakan berbagai fitur yang mempermudah implementasi model deep learning, seperti tensor operations yang efisien, kemampuan autograd untuk menghitung gradien secara otomatis, dan alat-alat untuk memuat, menyimpan, dan menganalisis model. PyTorch juga menyediakan API yang intuitif dan dokumentasi yang kaya, yang memudahkan pengguna untuk memahami dan mengimplementasikan berbagai konsep dalam deep learning.

Laporan teknis ini bertujuan untuk memberikan gambaran tentang konsep dan fungsionalitas utama mengenai dasar-dasar tensor, autograd, backpropagation, gradient descent, alur latihan, regresi linear, regresi logistik, dataset dan Dataloader, transformasi dataset, softmax dan crossentropy, fungsi aktivasi, jaringan feed forward, jaringan saraf konvolusi (CNN), transfer learning, tensorboard, dan menyimpan/memuat model.

II. Dasar-dasar Tensor

Tensor adalah objek serupa dengan array atau matriks yang digunakan untuk menyimpan dan memanipulasi data numerik. Tensor dapat memiliki berbagai dimensi, seperti vektor (1D), matriks (2D), atau tensor dengan dimensi yang lebih tinggi.

Dimulai dengan membuat tensor kosong dengan menggunakan fungsi torch.empty(). Misalnya, membuat tensor skalar, vektor, matriks, dan tensor tiga dimensi. Selanjutnya, fungsi torch.rand() untuk mengisi tensor dengan angka acak antara 0 dan 1. Kemudian fungsi torch.zeros() dan torch.ones() untuk membuat tensor dengan nilai 0 atau 1.

Untuk memeriksa ukuran tensor dengan menggunakan metode .size() dan melihat tipe data tensor dengan menggunakan atribut .dtype. Selain itu, kita dapat mengubah tipe data tensor dengan menyediakan argumen dtype saat membuat tensor baru.

Selain membuat tensor dari awal, pada bagian ini juga menunjukkan cara membuat tensor dari data yang sudah ada menggunakan fungsi torch.tensor(). Bagian ini juga menjelaskan tentang argumen requires_grad yang digunakan untuk menandai tensor sebagai variabel yang perlu dihitung gradiennya saat melaksanakan langkah-langkah optimisasi.

Tensor juga dapat melakukan operasi seperti penjumlahan, pengurangan, perkalian, dan pembagian elemen-wise. Pada bagian ini juga menunjukkan bagaimana melakukan slicing pada tensor untuk mengakses subset elemennya.

III. Autograd

Autograd merupakan salah satu fitur yang sangat penting dalam deep learning, karena memungkinkan kita untuk dengan mudah menghitung gradien dari suatu fungsi atau model neural network.

Pada codingan yang telah dicontohkan, kita dapat melihat penerapan autograd dalam PyTorch. Pertama, kita perlu mengaktifkan perhitungan gradien pada tensor yang ingin kita lacak. Hal ini dilakukan dengan menambahkan argumen requires_grad=True saat membuat tensor. Sebagai contoh, saya menggunakan perintah x = torch.randn(3, requires_grad=True) untuk membuat tensor x dengan ukuran 3 dan mengaktifkan perhitungan gradien.

Kemudian, kita dapat melakukan berbagai operasi pada tensor yang memiliki requires_grad=True. Misalnya, pada contoh coding tersebut, saya melakukan operasi penambahan, pengurangan, perkalian, dan pembagian antara tensor x dan tensor lainnya. Setiap operasi yang dilakukan pada tensor akan secara otomatis terrekam oleh autograd, dan tensor baru yang dihasilkan akan memiliki atribut grad_fn yang merujuk pada fungsi yang menciptakan tensor tersebut.

Selanjutnya, ketika kita ingin menghitung gradien dari suatu fungsi atau model yang menggunakan tensor dengan requires_grad=True, kita dapat menggunakan metode .backward() pada tensor output yang ingin kita hitung gradiennya. Dalam contoh coding di atas, saya menggunakan perintah z.backward() untuk menghitung gradien dari tensor z.

Hasil dari perhitungan gradien akan disimpan dalam atribut .grad dari tensor yang memiliki requires_grad=True. Dalam contoh coding di atas, saya mencetak gradien dari tensor x dengan menggunakan perintah print(x.grad). Hal ini memungkinkan kita untuk mengakses gradien dan menggunakannya untuk melakukan optimisasi atau perhitungan lainnya.

Selain itu, autograd juga menyediakan cara untuk menghentikan tensor dari melacak riwayat atau perhitungan gradien. Hal ini berguna saat kita ingin mengupdate parameter model tanpa mempengaruhi perhitungan gradien. Dalam contoh coding di atas, saya menunjukkan beberapa cara untuk menghentikan tensor dari melacak riwayat, seperti menggunakan metode .requires_grad_(False), .detach(), atau with torch.no_grad():.

IV. Backpropagation

Backpropagation adalah salah satu teknik penting dalam deep learning yang memungkinkan kita untuk secara efisien menghitung gradien dari parameter-parameter model neural network.

Pada coding yang telah dicontohkan, kita dapat melihat implementasi backpropagation dalam PyTorch. Pertama, kita mendefinisikan beberapa tensor, yaitu x, y, dan w. Tensor x dan y adalah input dan target yang digunakan dalam perhitungan, sedangkan tensor w merupakan parameter yang ingin kita optimalkan. Kita juga mengaktifkan perhitungan gradien pada tensor w dengan mengatur argumen requires_grad=True saat membuat tensor tersebut.

Selanjutnya, kita melakukan forward pass untuk menghitung prediksi y_predicted dengan menggunakan rumus y_predicted = w * x. Kemudian, kita menghitung loss dengan menggunakan rumus (y_predicted - y)^2. Loss ini merupakan ukuran seberapa baik prediksi kita dengan target yang sebenarnya.

Setelah itu, kita melakukan backward pass dengan memanggil metode .backward() pada tensor loss. Hal ini akan menghitung gradien dari loss terhadap tensor w, sehingga kita dapat mengetahui seberapa besar pengaruh perubahan pada w terhadap perubahan loss.

Hasil dari perhitungan gradien akan tersimpan dalam atribut .grad dari tensor w. Dalam contoh coding di atas, kita mencetak gradien dengan menggunakan perintah print(w.grad). Gradien ini akan digunakan untuk memperbarui nilai w agar loss semakin kecil, sehingga model kita dapat melakukan prediksi yang lebih akurat.

Selanjutnya, kita melakukan update pada nilai w dengan menggunakan perintah w -= 0.01 * w.grad. Perlu diperhatikan bahwa operasi ini dilakukan di luar grafik komputasi, sehingga tidak akan terjadi perhitungan gradien baru. Hal ini penting karena kita hanya ingin mengubah nilai w berdasarkan gradien yang telah dihitung sebelumnya.

Terakhir, kita mengatur nilai gradien w menjadi nol dengan menggunakan perintah w.grad.zero_(). Hal ini dilakukan agar gradien yang dihitung pada iterasi sebelumnya tidak mempengaruhi perhitungan gradien pada iterasi selanjutnya.

V. Gradient Descent

Pada coding yang telah dicontohkan, menggambarkan implementasi Gradient Descent dalam sebuah model regresi linear sederhana. Tujuan utama dari optimasi Gradient Descent adalah untuk menemukan nilai optimal dari parameter-model (dalam hal ini, nilai w) yang dapat meminimalkan fungsi loss.

Proses optimasi dimulai dengan mendefinisikan dataset input (X) dan target (Y) yang akan digunakan dalam pelatihan model. Selanjutnya, kita mendefinisikan parameter-model w dan mengaktifkan perhitungan gradien pada w dengan mengatur argumen requires_grad=True.

Selanjutnya, kita mendefinisikan fungsi forward(x) yang menerima input x dan menghasilkan output model w * x. Fungsi loss(y, y_pred) digunakan untuk menghitung nilai loss antara prediksi y_pred dan target y, dalam hal ini menggunakan Mean Squared Error (MSE).

Pada tahap pelatihan, kita melakukan iterasi sebanyak n_iters (100 kali pada contoh ini). Pada setiap iterasi, kita melakukan langkah-langkah seperti Melakukan forward pass dengan menggunakan input X pada model untuk mendapatkan prediksi y_pred, menghitung loss antara prediksi y_pred dan target Y dengan menggunakan fungsi loss(), melakukan backward pass dengan memanggil metode .backward() pada loss. Hal ini akan menghitung gradien loss terhadap parameter-model w, mengupdate nilai parameter-model w menggunakan metode Gradient Descent. Pada contoh ini, kita mengurangi nilai w dengan hasil perkalian antara learning rate (learning_rate) dan gradien w.grad. Operasi ini dilakukan di luar grafik komputasi dengan menggunakan perintah with torch.no_grad(), sehingga tidak akan terjadi perhitungan gradien baru, mengatur nilai gradien w.grad menjadi nol menggunakan w.grad.zero_(). Hal ini dilakukan agar gradien yang dihitung pada iterasi sebelumnya tidak mempengaruhi perhitungan gradien pada iterasi selanjutnya.
Selama iterasi pelatihan, kita mencetak informasi tentang nilai w dan loss pada setiap iterasi tertentu untuk memantau proses optimisasi.

Setelah pelatihan selesai, kita dapat menggunakan model yang telah dioptimasi untuk melakukan prediksi baru. Dalam contoh ini, kita mencetak prediksi model untuk input x=5 dengan menggunakan perintah forward(5).item().

VI. Training Pipeline

Tahap-tahap dalam Training Pipeline Model Deep Learning:

1. Desain Model:

   Pada tahap ini, kita mendesain arsitektur model deep learning yang akan digunakan. Dalam contoh ini, kita menggunakan model linear dengan menggunakan modul nn.Linear dari PyTorch. Model ini memiliki input_size yang sesuai dengan jumlah fitur pada data (n_features) dan output_size yang sesuai dengan jumlah fitur pada target (n_features).

2. Konstruksi Loss dan Optimizer:

   Pada tahap ini, kita mendefinisikan fungsi loss yang akan digunakan untuk mengukur sejauh mana prediksi model kita berbeda dari target yang sebenarnya. Dalam contoh ini, kita menggunakan Mean Squared Error (MSE) loss dengan modul nn.MSELoss dari PyTorch.

   Selain itu, kita juga mendefinisikan optimizer yang akan digunakan untuk mengoptimalkan model. Dalam contoh ini, kita menggunakan Stochastic Gradient Descent (SGD) sebagai optimizer dengan learning rate yang ditentukan.

3. Training Loop:

   Pada tahap ini, kita melakukan iterasi sebanyak n_iters (100 kali pada contoh ini) untuk melatih model.

Pada setiap iterasi, kita melakukan langkah-langkah berikut:

- Melakukan forward pass dengan memasukkan data training (X) ke dalam model untuk mendapatkan prediksi (y_predicted).
- Menghitung loss antara prediksi (y_predicted) dan target (Y) menggunakan fungsi loss yang telah didefinisikan.
- Melakukan backward pass dengan memanggil metode .backward() pada loss. Hal ini akan menghitung gradien loss terhadap parameter-model.
- Mengupdate nilai parameter-model menggunakan optimizer dengan memanggil metode .step(). Hal ini akan mengupdate nilai parameter-model berdasarkan gradien yang dihitung pada backward pass.
- Mengatur nilai gradien menjadi nol menggunakan metode .zero_grad() pada optimizer. Hal ini dilakukan untuk menghapus gradien yang dihitung pada iterasi sebelumnya sehingga tidak mempengaruhi perhitungan gradien pada iterasi selanjutnya.
- Pada setiap iterasi yang kelipatan 10, kita mencetak informasi tentang nilai parameter-model (w) dan loss saat ini untuk memantau proses pelatihan.

4. Prediksi setelah Pelatihan:

   Setelah pelatihan selesai, kita dapat menggunakan model yang telah dilatih untuk melakukan prediksi baru. Dalam contoh ini, kita mencetak prediksi model untuk input X_test dengan menggunakan perintah model(X_test).item().

VII. Linear Regression

Regresi linear adalah metode statistik yang digunakan untuk memodelkan hubungan antara variabel input (X) dan variabel output (y) dengan asumsi hubungan tersebut dapat didekati dengan persamaan linear. Tujuan regresi linear adalah untuk menemukan garis lurus terbaik yang dapat memprediksi nilai output berdasarkan nilai input.

Berikut adalah tahapan implementasi regresi linear menggunakan PyTorch:

1. Persiapan Data:

   Pada tahap ini, kita mempersiapkan data untuk regresi linear. Dalam contoh ini, kita menggunakan dataset sintetis yang dihasilkan menggunakan fungsi make_regression dari modul sklearn.datasets. Dataset ini terdiri dari 100 sampel dengan 1 fitur (n_features) dan memiliki tingkat kebisingan (noise). Kemudian, data tersebut diubah menjadi Tensor PyTorch menggunakan torch.from_numpy.
2. Model:

   Pada tahap ini, kita mendefinisikan model regresi linear. Dalam contoh ini, kita menggunakan modul nn.Linear dari PyTorch untuk membuat model dengan input_size yang sesuai dengan jumlah fitur pada data (n_features) dan output_size sebesar 1 (karena kita ingin melakukan prediksi nilai tunggal).

3. Loss dan Optimizer:

   Pada tahap ini, kita mendefinisikan fungsi loss yang akan digunakan untuk mengukur kesalahan prediksi model. Dalam contoh ini, kita menggunakan Mean Squared Error (MSE) loss dengan modul nn.MSELoss dari PyTorch.

   Selain itu, kita juga mendefinisikan optimizer yang akan digunakan untuk mengoptimalkan model. Dalam contoh ini, kita menggunakan Stochastic Gradient Descent (SGD) sebagai optimizer dengan learning rate yang ditentukan.

4. Training Loop:

   Pada tahap ini, kita melakukan iterasi sebanyak num_epochs (100 kali pada contoh ini) untuk melatih model.

   Pada setiap iterasi, kita melakukan langkah-langkah berikut:

   - Melakukan forward pass dengan memasukkan data training (X) ke dalam model untuk mendapatkan prediksi (y_predicted).
   - Menghitung loss antara prediksi (y_predicted) dan target (y) menggunakan fungsi loss yang telah didefinisikan.
   - Melakukan backward pass dengan memanggil metode .backward() pada loss. Hal ini akan menghitung gradien loss terhadap parameter-model.
   - Mengupdate nilai parameter-model menggunakan optimizer dengan memanggil metode .step(). Hal ini akan mengupdate nilai parameter-model berdasarkan gradien yang dihitung pada backward pass.
   - Mengatur nilai gradien menjadi nol menggunakan metode .zero_grad() pada optimizer. Hal ini dilakukan untuk menghapus gradien yang dihitung pada iterasi sebelumnya sehingga tidak mempengaruhi perhitungan gradien pada iterasi selanjutnya.
   - Pada setiap iterasi yang kelipatan 10, kita mencetak informasi tentang loss saat ini untuk memantau proses pelatihan.

5. Visualisasi Hasil:

   Setelah pelatihan selesai, kita melakukan visualisasi hasil prediksi model. Pada contoh ini, kita menggunakan matplotlib untuk membuat scatter plot dari data asli (titik merah) dan garis prediksi yang dihasilkan oleh model (garis biru).

   Dengan mengikuti langkah-langkah di atas, kita dapat mengimplementasikan regresi linear menggunakan PyTorch. Hasilnya adalah model yang dapat memprediksi nilai output berdasarkan nilai input dengan menggunakan persamaan garis lurus terbaik yang ditemukan selama proses pelatihan.

Dalam contoh ini, hasil visualisasi menunjukkan bahwa model regresi linear telah mempelajari pola data dengan baik dan dapat melakukan prediksi yang cukup akurat.

VIII. Logistic Regression

Regresi logistik adalah metode statistik yang digunakan untuk memodelkan hubungan antara variabel input (X) dan variabel output (y) dengan asumsi hubungan tersebut dapat didekati dengan fungsi logistik (sigmoid). Tujuan regresi logistik adalah untuk memprediksi probabilitas kelas tertentu berdasarkan variabel input. Dimana tahapan implementasi regresi logistik menggunakan PyTorch sebagai berikut:

1. Persiapan Data:
    - Pada tahap ini, kita mempersiapkan data untuk regresi logistik. Dalam contoh ini, kita menggunakan dataset Breast Cancer yang tersedia dalam modul sklearn.datasets.
    - Dataset ini terdiri dari fitur-fitur yang menggambarkan tumor payudara dan label yang menunjukkan apakah tumor tersebut jinak atau ganas.
    - Data dibagi menjadi data training dan data test menggunakan train_test_split dari modul sklearn.model_selection.
    - Selanjutnya, kita melakukan penskalaan fitur menggunakan StandardScaler dari modul sklearn.preprocessing.
    - Setelah itu, data dikonversi menjadi Tensor PyTorch menggunakan torch.from_numpy.
2. Model:
    - Pada tahap ini, kita mendefinisikan model regresi logistik. Dalam contoh ini, kita membuat kelas Model yang merupakan turunan dari kelas nn.Module dari PyTorch.
    - Model ini terdiri dari lapisan linear (nn.Linear) yang menghasilkan output berdasarkan jumlah fitur pada data.
    - Output dari lapisan linear diproses menggunakan fungsi aktivasi sigmoid untuk menghasilkan probabilitas kelas.
3. Loss dan Optimizer:
    - Pada tahap ini, kita mendefinisikan fungsi loss yang akan digunakan untuk mengukur kesalahan prediksi model. Dalam contoh ini, kita menggunakan Binary Cross Entropy (BCE) Loss dengan nn.BCELoss dari PyTorch, karena kita melakukan klasifikasi biner.
    - Selain itu, kita juga mendefinisikan optimizer yang akan digunakan untuk mengoptimalkan model. Dalam contoh ini, kita menggunakan Stochastic Gradient Descent (SGD) sebagai optimizer dengan learning rate yang ditentukan.
4. Training Loop:
    - Pada tahap ini, kita melakukan iterasi sebanyak num_epochs (100 kali pada contoh ini) untuk melatih model.
    - Pada setiap iterasi, kita melakukan langkah-langkah berikut:
        - Melakukan forward pass dengan memasukkan data training (X_train) ke dalam model untuk mendapatkan prediksi probabilitas kelas (y_pred).

- Menghitung loss antara prediksi (y_pred) dan target (y_train) menggunakan fungsi loss yang telah didefinisikan.
- Melakukan backward pass dengan memanggil metode .backward() pada loss. Hal ini akan menghitung gradien loss terhadap parameter-model.
- Mengupdate nilai parameter-model menggunakan optimizer dengan memanggil metode .step(). Hal ini akan mengupdate nilai parameter-model berdasarkan gradien yang dihitung pada langkah sebelumnya.
- Mengatur gradien parameter-model menjadi nol dengan memanggil metode .zero_grad(). Hal ini diperlukan untuk mengosongkan gradien sebelum langkah backward pass berikutnya.
- Pada setiap epoch ke-10, mencetak nilai loss saat ini untuk melihat kemajuan pelatihan.

5. Evaluasi:
   - Setelah pelatihan selesai, kita melakukan evaluasi model pada data test (X_test dan y_test).
   - Kita menggunakan model yang telah dilatih untuk memprediksi probabilitas kelas pada data test (y_predicted).
   - Probabilitas kelas diubah menjadi label kelas biner dengan membulatkan nilai prediksi (y_predicted_cls).
   - Akurasi model dihitung dengan membandingkan prediksi dengan label sebenarnya (y_test).

Dalam contoh codingan ini, hasil evaluasi menunjukkan akurasi model pada data test. Ini memberikan gambaran tentang sejauh mana model yang telah dilatih dapat memprediksi kelas dengan benar.

IX. Dataset dan Dataloader

Penggunaan Dataset dan Dataloader dalam PyTorch sangat mempermudah pengolahan data dalam batch dan memungkinkan pelatihan model yang efisien. Dataset digunakan untuk mengorganisir dan mengakses data, sedangkan Dataloader mengelola pembagian data menjadi batch-batch kecil yang dapat digunakan untuk proses pelatihan. Ini sangat berguna saat dataset berukuran besar dan tidak dapat dimuat secara keseluruhan ke dalam memori pada saat yang bersamaan.

Pada coding yang telah dicontohkan menggunakan Custom Dataset (WineDataset) dimana Kelas WineDataset adalah turunan dari kelas Dataset dalam PyTorch. Pada fungsi __init__, dataset Wine diinisialisasi. Data dibaca menggunakan numpy dari file 'wine.csv' dengan delimiter koma. Fitur-fitur disimpan dalam x_data dengan tipe data Tensor PyTorch, sedangkan label kelas disimpan dalam y_data. Fungsi __getitem__ mengembalikan pasangan fitur dan label dari indeks yang diminta. Fungsi __len__ mengembalikan jumlah total sampel dalam dataset.

Dataloader digunakan untuk memuat dataset dan melakukan komputasi batch.

Dalam contoh ini, dataset Wine dimuat menggunakan Dataloader dengan batch_size=4, shuffle=True (mengacak urutan sampel), dan num_workers=2 (jumlah proses bekerja paralel untuk memuat data secara lebih cepat). Dataloader mengatur dataset dalam batch-batch kecil yang dapat digunakan untuk pelatihan model. Iterator dataiter digunakan untuk mengakses batch pertama dari Dataloader. Setiap batch yang diperoleh dari Dataloader berisi fitur dan label yang sesuai. Selanjutnya pada loop pelatihan sederhana selama beberapa epoch. Jumlah total sampel dalam dataset dihitung, dan jumlah iterasi (n_iterations) dihitung sebagai pembagian atas dari total sampel dengan ukuran batch (dalam hal ini, 4). Dalam setiap iterasi, Dataloader memberikan batch fitur dan label ke variabel inputs dan labels. Proses pelatihan model sebenarnya tidak dilakukan dalam contoh ini, hanya mencetak informasi tentang setiap iterasi.

PyTorch menyediakan modul torchvision.datasets yang berisi beberapa dataset terkenal seperti MNIST, Fashion-MNIST, CIFAR10, COCO, dll. Contoh ini menggunakan dataset MNIST sebagai contoh. Dataset MNIST dimuat menggunakan torchvision.datasets.MNIST dengan argumen root (direktori penyimpanan), train (untuk data pelatihan), transform (transformasi yang diterapkan pada setiap sampel), dan download (jika True, dataset akan diunduh jika belum ada). Dataloader digunakan untuk memuat dataset MNIST dengan batch_size=3 dan shuffle=True. Dataiter digunakan untuk mengakses batch pertama dari Dataloader. Setiap batch yang diperoleh dari Dataloader berisi citra digit dan labelnya.

X. Dataset Transforms

Transformasi dataset digunakan untuk mempersiapkan data sebelum melatih model. Dalam contoh tersebut, transformasi diterapkan pada dataset "WineDataset" yang berisi data angka dan label. Transformasi dataset dapat dilakukan pada berbagai tipe data, termasuk PIL images, tensors, ndarrays, atau tipe data kustom, saat membuat objek dataset. Dalam contoh ini, transformasi diterapkan pada data tensor. Transformasi yang digunakan dalam contoh tersebut adalah:

- ToTensor: Mengonversi ndarrays menjadi Tensors. Transformasi ini digunakan untuk mengubah input dan target dari WineDataset menjadi Tensors.
- MulTransform: Mengalikan input dengan faktor tertentu. Transformasi ini mengalikan input dengan faktor 4.

Kelas WineDataset merupakan turunan dari kelas Dataset pada PyTorch. Saat objek WineDataset dibuat, data diambil dari file "wine.csv" dan disimpan dalam bentuk ndarrays. Transformasi yang diberikan pada objek dataset diatur menggunakan parameter "transform". Dalam metode getitem(self, index), transformasi diterapkan pada setiap sampel (sample) dataset sebelum mengembalikannya. Dalam contoh ini, transformasi diterapkan jika parameter "transform" tidak None. Jika transformasi diberikan, metode call() pada transformasi tersebut akan dipanggil untuk memodifikasi sampel sebelum mengembalikannya.

Dalam contoh tersebut, transformasi ToTensor() digunakan untuk mengonversi ndarrays menjadi Tensors. Sebagai hasilnya, features (input) dan labels (target) pada objek dataset menjadi Tensors setelah transformasi. Selain itu, dapat pula menggabungkan beberapa transformasi menggunakan kelas Compose dari torchvision.transforms. Dalam contoh tersebut, transformasi Compose([ToTensor(), MulTransform(4)]) diterapkan pada objek dataset. Sehingga, setelah transformasi, input akan dikalikan dengan faktor 4.

Dengan menggunakan transformasi dataset, kita dapat mempersiapkan data dengan cara yang diinginkan sebelum melatih model. Transformasi ini membantu dalam pra-pemrosesan data seperti normalisasi, augmentasi data, pengubahan format data, dan transformasi kustom lainnya yang diperlukan untuk melatih model dengan efektivitas yang lebih baik.

XI. Softmax dan Crossentropy

Pada coding yang dicontohkan, terdapat dua fungsi yang relevan dalam konteks klasifikasi multikelas, yaitu fungsi softmax dan fungsi loss cross entropy.

1. Fungsi Softmax:

   Fungsi softmax digunakan untuk menghasilkan probabilitas dari output model klasifikasi multikelas. Fungsi ini mengonversi nilai output (logits) menjadi probabilitas yang dinormalisasi antara 0 dan 1. Hasil dari softmax adalah distribusi probabilitas yang menggambarkan tingkat keyakinan model terhadap setiap kelas.

   Dalam contoh tersebut, terdapat implementasi fungsi softmax menggunakan NumPy dan PyTorch. Fungsi softmax numpy diterapkan pada array `x`, sedangkan fungsi softmax PyTorch diterapkan pada tensor `x`. Kedua implementasi menghasilkan distribusi probabilitas yang sama untuk setiap kelas.

2. Fungsi Loss Cross Entropy:

   Fungsi loss cross entropy digunakan untuk mengukur kinerja model klasifikasi multikelas. Loss ini mengukur sejauh mana probabilitas prediksi model menyimpang dari label aktual yang diberikan. Semakin besar perbedaan antara probabilitas prediksi dan label aktual, semakin tinggi nilai loss cross entropy.

   Dalam contoh tersebut, terdapat implementasi fungsi loss cross entropy menggunakan NumPy dan PyTorch. Fungsi cross_entropy numpy menerima argumen `actual` (label aktual) dan `predicted` (probabilitas prediksi). Fungsi ini melakukan perhitungan loss dengan menghitung nilai log dari probabilitas prediksi aktual dan mengalikan dengan label aktual. Kemudian, loss dihitung dengan menjumlahkan hasilnya.

   Pada implementasi PyTorch, digunakan kelas `nn.CrossEntropyLoss()`. Fungsi ini menggabungkan fungsi log softmax (nn.LogSoftmax) dan fungsi

loss negatif log likelihood (nn.NLLLoss). Fungsi cross entropy loss PyTorch ini langsung menerima input logits (belum melalui softmax) dan label aktual. PyTorch secara otomatis menerapkan softmax pada input dan menghitung loss cross entropy.

Dalam konteks klasifikasi multikelas, fungsi softmax dan fungsi loss cross entropy digunakan secara bersamaan. Fungsi softmax menghasilkan distribusi probabilitas untuk setiap kelas, sedangkan fungsi loss cross entropy menghitung loss berdasarkan probabilitas prediksi dan label aktual. Loss tersebut digunakan untuk mengoptimasi model dalam proses pelatihan sehingga dapat mempelajari hubungan antara fitur input dan label kelas yang benar.

XII. Activation Functions

Pada coding yang dicontohkan, terdapat beberapa jenis Activation Functions yang digunakan dalam model neural network, yaitu:

1. Softmax:

   Activation function softmax digunakan untuk menghasilkan probabilitas yang dinormalisasi dari output model. Softmax mengonversi nilai input menjadi distribusi probabilitas antara 0 dan 1, dengan memastikan bahwa jumlah probabilitas dari semua kelas adalah 1. Activation function softmax digunakan dalam kasus klasifikasi multikelas.

2. Sigmoid:

   Activation function sigmoid juga dikenal sebagai logistic function. Sigmoid menghasilkan output antara 0 dan 1, dan digunakan untuk masalah klasifikasi biner atau dalam layer output saat menghitung probabilitas dari kelas positif.

3. Tanh:

   Activation function tanh (hyperbolic tangent) menghasilkan output antara -1 dan 1. Tanh non-linear dan simetris terhadap titik 0. Fungsi ini digunakan dalam layer tersembunyi jaringan saraf yang lebih dalam.

4. ReLU (Rectified Linear Unit):

   Activation function ReLU mengabaikan nilai negatif dan mengaktifkan neuron dengan nilai positif. Fungsi ini menghasilkan output 0 untuk nilai input yang negatif dan langsung mengalirkan nilai input yang positif. ReLU umumnya digunakan dalam layer tersembunyi jaringan saraf yang lebih dalam karena efisiensinya dalam perhitungan.

5. Leaky ReLU:

   Activation function Leaky ReLU adalah variasi dari ReLU yang memiliki gradien kecil untuk nilai negatif. Leaky ReLU mengatasi

masalah "dying ReLU" di mana neuron dengan input negatif yang besar bisa "mati" dan berhenti berkontribusi pada pembelajaran. Leaky ReLU mengizinkan gradien kecil untuk input negatif, sehingga memperbaiki masalah tersebut.

Kedua implementasi model neural network dalam contoh tersebut menggunakan beberapa jenis activation functions di atas. Activation functions ini memberikan kemampuan non-linearitas pada model, yang sangat penting dalam mempelajari hubungan yang kompleks antara fitur input dan output yang diinginkan.

XIII. Feed Forward Net

Pada coding yang telah dicontohkan, terdapat implementasi dari sebuah feedforward neural network untuk melakukan klasifikasi gambar menggunakan dataset MNIST. Berikut penjelasan mengenai feedforward neural network yang digunakan dan contoh pengimplementasiannya:

1. Feedforward Neural Network:
    - Arsitektur model: Model terdiri dari sebuah layer input, satu layer tersembunyi, dan sebuah layer output. Layer input memiliki ukuran sesuai dengan dimensi input gambar (28x28 piksel). Layer tersembunyi memiliki ukuran 500 neuron atau unit tersembunyi. Layer output memiliki ukuran 10 neuron, sesuai dengan jumlah kelas dalam dataset MNIST (0-9).
    - Activation function: Layer tersembunyi menggunakan fungsi aktivasi ReLU (Rectified Linear Unit), sedangkan layer output tidak menggunakan fungsi aktivasi atau softmax. Fungsi aktivasi ReLU memberikan non-linearitas pada model dan membantu dalam mempelajari fitur-fitur yang kompleks dari data.
    - Loss function: Menggunakan CrossEntropyLoss sebagai fungsi loss untuk membandingkan output yang dihasilkan oleh model dengan label yang sebenarnya dalam kasus klasifikasi multikelas.
    - Optimizer: Menggunakan algoritma optimasi Adam untuk mengoptimalkan parameter-model berdasarkan loss yang dihitung.
2. Pengimplementasian pada coding:
    - Dataset MNIST diunduh dan dibagi menjadi dataset pelatihan dan pengujian menggunakan `torchvision.datasets.MNIST`.
    - Data loader dibuat menggunakan `torch.utils.data.DataLoader` untuk memuat data dalam bentuk batch.
    - Model feedforward neural network didefinisikan dalam kelas `NeuralNet` yang merupakan turunan dari `nn.Module`. Arsitektur model dan fungsi forward diimplementasikan di dalam kelas ini.
    - Model, fungsi loss (CrossEntropyLoss), dan optimizer (Adam) diinisialisasi.
    - Selanjutnya, model dilatih pada data pelatihan menggunakan perulangan nested. Pada setiap iterasi, gambar dan label diambil dari

loader, diubah bentuknya, dilakukan forward pass melalui model, dihitung loss, dilakukan backward pass, dan optimizer digunakan untuk memperbarui parameter-model.
- Setelah pelatihan selesai, model diuji pada data pengujian. Prediksi dihasilkan untuk setiap gambar dalam loader pengujian, kemudian akurasi model dihitung dengan membandingkan prediksi dengan label yang sebenarnya.

Dengan menggunakan feedforward neural network, model dilatih untuk mengenali angka pada dataset MNIST dan mencapai akurasi pada data pengujian.

XIV. CNN (Convolutional Neural Network)

Convolutional Neural Network (CNN) menggunakan PyTorch untuk melakukan klasifikasi gambar pada dataset CIFAR-10. Berikut penjelasan mengenai CNN yang digunakan dan contoh pengimplementasiannya:

1. Convolutional Neural Network (CNN):
   - Arsitektur model: Model terdiri dari beberapa layer konvolusi (Conv2d) yang diikuti oleh layer pooling (MaxPool2d) untuk mengekstraksi fitur-fitur visual dari gambar. Setelah itu, hasilnya diteruskan ke beberapa layer linear (Linear) untuk klasifikasi.
   - Activation function: Fungsi aktivasi ReLU (Rectified Linear Unit) digunakan setelah setiap operasi konvolusi dan linear untuk memberikan non-linearitas pada model.
   - Loss function: Menggunakan CrossEntropyLoss sebagai fungsi loss untuk membandingkan output yang dihasilkan oleh model dengan label yang sebenarnya dalam kasus klasifikasi multikelas.
   - Optimizer: Menggunakan algoritma optimasi SGD (Stochastic Gradient Descent) untuk mengoptimalkan parameter-model berdasarkan loss yang dihitung.
2. Pengimplementasian pada coding:
   - Pertama, dataset CIFAR-10 diunduh dan dilakukan transformasi pada gambar-gambar dalam dataset menggunakan `transforms.Compose`.
   - Data loader dibuat menggunakan `torch.utils.data.DataLoader` untuk memuat data dalam bentuk batch.
   - Fungsi `imshow` digunakan untuk menampilkan beberapa contoh gambar dari dataset pelatihan.
   - Model CNN didefinisikan dalam kelas `ConvNet` yang merupakan turunan dari `nn.Module`. Arsitektur model dan fungsi forward diimplementasikan di dalam kelas ini.
     Model, fungsi loss (CrossEntropyLoss), dan optimizer (SGD) diinisialisasi.

- Selanjutnya, model dilatih pada data pelatihan menggunakan perulangan nested. Pada setiap iterasi, gambar dan label diambil dari loader, dilakukan forward pass melalui model, dihitung loss, dilakukan backward pass, dan optimizer digunakan untuk memperbarui parameter-model.
- Setelah pelatihan selesai, model diuji pada data pengujian. Prediksi dihasilkan untuk setiap gambar dalam loader pengujian, kemudian akurasi model secara keseluruhan dan akurasi per kelas dihitung.

Dengan menggunakan Convolutional Neural Network, model dilatih untuk mengklasifikasikan gambar-gambar CIFAR-10 dan mencapai akurasi pada data pengujian.

XV. Transfer Learning

Transfer learning adalah teknik yang populer dalam deep learning yang melibatkan penggunaan model yang telah dilatih sebelumnya pada dataset besar sebagai titik awal untuk tugas-tugas klasifikasi yang berbeda. Pada coding di atas, terdapat implementasi transfer learning menggunakan PyTorch.

Pertama, kita menggunakan model pre-trained ResNet-18 yang telah dilatih pada dataset ImageNet. Model ini memiliki kemampuan untuk mengenali berbagai objek dalam 1.000 kategori yang berbeda. Namun, kita ingin menggunakannya untuk tugas klasifikasi biner antara gambar hymenoptera dan non-hymenoptera. Oleh karena itu, kita mengganti lapisan fully connected (fc) terakhir dengan lapisan baru yang memiliki output size 2.

Selanjutnya, kita membagi dataset menjadi dua bagian, yaitu data pelatihan (train) dan data validasi (val). Transformasi data diterapkan pada kedua bagian tersebut, seperti crop acak, flip horizontal, resize, dan normalisasi. Hal ini bertujuan untuk meningkatkan kemampuan model dalam mempelajari pola-pola yang umum pada dataset.

Kemudian, kita melatih model menggunakan pendekatan fine-tuning, di mana seluruh model (termasuk lapisan fc terakhir) diperbarui selama pelatihan. Fungsi loss yang digunakan adalah CrossEntropyLoss, dan optimizer yang digunakan adalah SGD. Selama pelatihan, kita menggunakan learning rate scheduler (StepLR) yang mengurangi learning rate secara bertahap setiap beberapa epoch untuk meningkatkan stabilitas pelatihan.

Selain itu, kita juga menerapkan pendekatan ConvNet as fixed feature extractor. Pada pendekatan ini, semua parameter kecuali lapisan fc terakhir dibekukan. Hal ini

berguna ketika kita memiliki dataset target yang relatif kecil. Dalam pendekatan ini, hanya lapisan fc terakhir yang diperbarui selama pelatihan.

Selama proses pelatihan, model dievaluasi pada data validasi untuk memonitor performa model. Model terbaik (dengan akurasi validasi tertinggi) disimpan, dan pada akhirnya, model dengan akurasi terbaik tersebut diload kembali. Dengan menggunakan transfer learning, kita dapat memanfaatkan pengetahuan yang telah dipelajari oleh model yang telah dilatih pada dataset besar seperti ImageNet, dan menerapkannya pada tugas klasifikasi yang spesifik seperti klasifikasi hymenoptera. Dengan demikian, kita dapat menghasilkan model yang baik bahkan dengan dataset terbatas.

XVI. Tensorboard

TensorBoard adalah alat visualisasi yang disediakan oleh TensorFlow yang juga dapat digunakan dengan framework PyTorch. Hal ini memungkinkan kita untuk memantau dan menganalisis pelatihan dan evaluasi model secara interaktif.

Pertama, kita mengimpor modul yang diperlukan untuk TensorBoard, yaitu `SummaryWriter` dari `torch.utils.tensorboard` dan beberapa modul tambahan seperti `sys` dan `torch.nn.functional as F`.

Selanjutnya, kita membuat objek `SummaryWriter` dengan menentukan direktori penyimpanan log yang akan digunakan oleh TensorBoard. Dalam contoh ini, direktori "runs/mnist1" digunakan. Objek `SummaryWriter` ini akan digunakan untuk mencatat informasi dan kejadian selama pelatihan dan evaluasi model.
Pada bagian ini, beberapa fitur TensorBoard digunakan untuk memvisualisasikan data:

1. Visualisasi Gambar:
   - Kita menggunakan `make_grid` dari torchvision untuk membuat grid gambar dari beberapa contoh data.
   - Grid gambar tersebut ditambahkan ke TensorBoard menggunakan `add_image` dengan nama "mnist_images".

2. Visualisasi Grafik Model:
   - Kita menggunakan `add_graph` pada objek `SummaryWriter` untuk memvisualisasikan grafik model yang digunakan.
     Grafik model ditentukan dengan memasukkan model (`model`) dan input data contoh (`example_data`) yang telah direshape ke bentuk yang sesuai.

3. Visualisasi Loss dan Akurasi:

- Pada setiap iterasi pelatihan, kita mencatat nilai loss dan akurasi pada TensorBoard menggunakan `add_scalar` dengan nama "training loss" dan "accuracy".
- Nilai loss rata-rata dari setiap 100 iterasi juga dicatat.

4. Visualisasi Precision-Recall Curve:
   - Pada tahap evaluasi model, kita mencatat prediksi dan label untuk setiap kelas pada TensorBoard.
   - Precision-Recall Curve (kurva presisi-recall) ditambahkan ke TensorBoard menggunakan `add_pr_curve` untuk setiap kelas dengan menyediakan label, prediksi, dan langkah global sebagai argumen.

Setelah selesai mencatat informasi pada TensorBoard, kita menutup objek `SummaryWriter` dengan menggunakan `writer.close()`. Dalam contoh di atas, penutupan `SummaryWriter` dilakukan setelah mencatat Precision-Recall Curve.

Dengan menggunakan TensorBoard, kita dapat memantau dan menganalisis berbagai aspek pelatihan dan evaluasi model secara visual. Hal ini membantu kita dalam memahami dan memperbaiki performa model serta memperoleh wawasan yang lebih dalam tentang proses pembelajaran yang terjadi selama pelatihan.

XVII. Save Load

Pada coding yang telah dicontohkan, terdapat implementasi metode save dan load model pada PyTorch. Terdapat 3 metode yang dapat digunakan untuk menyimpan dan memuat model:

1. Metode Pertama: Menyimpan dan Memuat Model Lengkap
   - Kita dapat menggunakan `torch.save` untuk menyimpan seluruh model.
   - Contohnya adalah dengan menyimpan model menggunakan `torch.save(model, PATH)`.
   - Untuk memuat model, kita menggunakan `torch.load(PATH)` dan kemudian memanggil `model.eval()` untuk menetapkan model dalam mode evaluasi.
   - Dengan menggunakan metode ini, kita dapat langsung menggunakan model tanpa perlu membuat objek model baru.

2. Metode Kedua: Menyimpan dan Memuat Hanya state_dict Model
   - Disarankan untuk hanya menyimpan dan memuat state_dict dari model.
   - Untuk menyimpan state_dict, kita menggunakan `torch.save(model.state_dict(), PATH)`.

- Saat memuat model, kita harus membuat objek model baru terlebih dahulu dan kemudian memuat state_dict menggunakan `model.load_state_dict(torch.load(PATH))`.
- Setelah memuat state_dict, kita harus memanggil `model.eval()` untuk menetapkan model dalam mode evaluasi.

3. Metode Ketiga: Menyimpan dan Memuat Checkpoint
   - Kita juga dapat menyimpan dan memuat checkpoint yang mencakup state_dict model serta status optimizer.
   - Sebelum menyimpan, kita perlu membuat dictionary yang berisi informasi checkpoint seperti epoch, model_state, dan optim_state.
   - Dictionary tersebut kemudian dapat disimpan menggunakan `torch.save(checkpoint, FILE)`.
   - Saat memuat checkpoint, kita perlu membuat objek model dan optimizer baru, dan kemudian memuat state_dict dan optim_state menggunakan `model.load_state_dict(checkpoint['model_state'])` dan `optimizer.load_state_dict(checkpoint['optim_state'])`.
   - Dalam contoh ini, kita juga memuat nilai epoch untuk melanjutkan pelatihan dari checkpoint yang disimpan.

Selain itu, terdapat juga beberapa catatan penting saat menggunakan save dan load model pada PyTorch:

- Saat menyimpan dan memuat model yang dilatih pada GPU, kita perlu memperhatikan perangkat yang digunakan.
- Jika menyimpan pada GPU dan memuat pada CPU atau sebaliknya, kita perlu menggunakan `map_location` dalam `torch.load` untuk menentukan perangkat yang digunakan.
- Setelah memuat model ke GPU, kita perlu menggunakan `model.to(torch.device('cuda'))` untuk mengubah parameter tensor model menjadi tensor CUDA.
- Jangan lupa memanggil `model.eval()` untuk mengaktifkan dropout dan batch normalization layers saat menjalankan inferensi.
- Jika ingin melanjutkan pelatihan, panggil `model.train()` untuk mengaktifkan mode pelatihan.

XVIII. Kesimpulan

Setelah melakukan percobaan dan mengetahui bagaimana pemrosesan data dalam PyTorch. Pada contoh tersebut menggunakan PyTorch untuk melakukan analisis klasifikasi menggunakan regresi logistik pada dataset breast cancer. Kami membagi data menjadi data latih dan data uji, dan menormalkan fitur-fiturnya. Model regresi logistik kami terdiri dari satu layer linear dan fungsi sigmoid. Selama pelatihan, kami menggunakan binary cross entropy loss dan stochastic gradient descent optimizer

untuk memperbarui parameter-model berdasarkan gradien loss. Setelah pelatihan, kami mengevaluasi model menggunakan data uji dan menghitung akurasi prediksi. Kami juga menggunakan TensorBoard untuk visualisasi data dan pelatihan model, serta mengimplementasikan metode save dan load model menggunakan torch.save dan torch.load. Penting untuk memperhatikan penanganan perangkat saat menyimpan dan memuat model, serta mengaktifkan mode evaluasi dan pelatihan pada model.

XIX. Referensi

Ahmad, S., & Ahmed, S. (2021). PyTorch: An open source machine learning framework. Diakses pada 23 Juni 2023, dari

https://pytorch.org/

PyTorch. (2021). TorchVision: Transforms. Diakses pada 23 Juni 2023, dari https://pytorch.org/vision/stable/transforms.html

Brownlee, J. (2021). How to Prepare Your Data When Working with Image Data in Python. Machine Learning Mastery. Diakses pada 23 Juni 2023, dari https://machinelearningmastery.com/how-to-prepare-your-data-when-working-with-image-data-in-python/

Brownlee, J. (2021). How to Use Data Preparation for Deep Learning (Normalize, Standardize, and Rescale Input Data). Machine Learning Mastery. Diakses pada 23 Juni 2023, dari https://machinelearningmastery.com/how-to-use-data-preparation-for-deep-learning-with-python/

PyTorch. (2021). Dataset and DataLoader. Diakses pada 23 Juni 2023, dari https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

PyTorch. (2021). nn.Module. Diakses pada 23 Juni 2023, dari https://pytorch.org/docs/stable/generated/torch.nn.Module.html

PyTorch. (2021). Linear Layers. Diakses pada 23 Juni 2023, dari https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

Scikit-learn. (2021). sklearn.datasets.make_regression. Diakses pada 23 Juni 2023, dari https://scikitlearn.org/stable/modules/generated/sklearn.datasets.make_regression.html