TECHNICAL REPORT

DEEP LEARNING WITH PYTORCH

TUGAS UNTUK MEMENUHI MATA KULIAH

MACHINE LEARNING

Oleh:

Muhammad Tharreq An Nahl

1103204040

PROGRAM STUDI SI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

BANDUNG

2023

A. Pengertian Deep Learning

Deep learning adalah salah satu cabang yang paling menjanjikan dalam bidang kecerdasan buatan (artificial intelligence) yang telah mengalami perkembangan pesat selama beberapa dekade terakhir. Sejarah deep learning dimulai pada tahun 1940-an dengan penemuan awal jaringan saraf tiruan. Namun, perkembangan yang signifikan terjadi pada tahun 1980-an dan 1990-an ketika teknik pelatihan jaringan saraf multi-lapis menggunakan algoritma backpropagation ditemukan. Sejak saat itu, kemajuan dalam pemrosesan komputasi dan ketersediaan data yang melimpah telah memicu minat dan investasi yang besar dalam deep learning, dan ini telah membuka jalan bagi aplikasi yang luas di berbagai bidang seperti pengenalan gambar, pengenalan suara, dan analisis data kompleks.

Salah satu keunggulan utama deep learning adalah kemampuannya untuk belajar secara end-to-end. Dalam deep learning, jaringan saraf tiruan terdiri dari banyak lapisan yang saling terhubung, dan setiap lapisan bertanggung jawab untuk memproses informasi secara bertahap. Lapisan-lapisan ini bekerja secara hierarkis untuk mempelajari representasi yang semakin abstrak dan kompleks dari data mentah. Dalam proses ini, deep learning mampu secara otomatis mengekstraksi fitur-fitur yang relevan tanpa memerlukan pemrosesan manual yang rumit. Hal ini menjadikan deep learning sangat efektif dalam memecahkan masalah yang kompleks dan menemukan pola yang tersembunyi dalam data.

Keberhasilan deep learning dalam beberapa aplikasi utama, seperti pengenalan wajah, terjemahan mesin, dan pengenalan suara, telah mengubah banyak aspek kehidupan kita. Peningkatan kemampuan komputasi dan perangkat keras yang lebih kuat telah memungkinkan pelatihan model deep learning yang lebih besar dan lebih kompleks. Selain itu, ketersediaan data yang besar dan bervariasi juga telah menjadi faktor penting dalam kesuksesan deep learning. Dengan semakin banyaknya organisasi dan peneliti yang menerapkan deep learning, diharapkan akan terus ada inovasi dan perkembangan lebih lanjut dalam bidang ini, membawa manfaat yang lebih besar bagi masyarakat di masa depan.

B. Analisis Kode - TENSOR BASIC

Dari hasil kodingan yang telah dibuat, dapat dijelaskan bahwa dari awal kita akan menggunakan pustaka PyTorch untuk memperkenalkan konsep dasar dalam penggunaan tensor. PyTorch menyediakan berbagai fungsi untuk inisialisasi tensor, seperti mengisi dengan bilangan acak atau nol, serta operasi tensor seperti penjumlahan, pengurangan, perkalian, dan pembagian. Slicing digunakan untuk mengakses elemen atau subset dari tensor. Terdapat juga fungsi untuk mengubah dimensi tensor menggunakan metode view(), dan kemampuan konversi antara tensor PyTorch dan array NumPy. Selain itu, kodingan juga menunjukkan bagaimana memindahkan tensor ke GPU jika tersedia. Dengan memahami konsep-konsep dasar ini, kita dapat memanipulasi dan mengolah data dengan menggunakan tensor PyTorch dalam implementasi deep learning.

C. Analisis Kode - AUTOGRAD

Dari hasil kodingan yang telah dibuat, hal yang dilakukan adalah dengan menggunakan pustaka PyTorch untuk menggambarkan konsep autograd yang menyediakan diferensiasi otomatis dalam operasi pada Tensor. Saat requires_grad diatur sebagai True pada suatu Tensor, PyTorch akan melacak semua operasi yang dilakukan pada Tensor tersebut. Dalam kodingan, kita membuat Tensor x dengan nilai acak dan menghitung y dengan menambahkan 2 ke setiap elemen x. Kemudian, kita melakukan serangkaian operasi pada y, seperti perkalian dan perhitungan rata-rata. Dengan memanggil fungsi backward() pada z, kita dapat melakukan backpropagation untuk menghitung gradien dz/dx, dan gradien tersebut akan disimpan dalam atribut .grad dari Tensor x.

Selanjutnya, kodingan juga menunjukkan penggunaan autograd pada Tensor non-scalar. Misalnya, kita mengalikan Tensor x dengan 2 sebanyak 10 kali. Kemudian, kita menghitung gradien menggunakan tensor v sebagai argumen pada fungsi backward(). Ini diperlukan karena Tensor y memiliki bentuk yang sama dengan v, dan kita membutuhkan produk vektor-Jacobian saat menghitung gradien.

Selain itu, kodingan juga menggambarkan cara untuk menghentikan pelacakan riwayat pada suatu Tensor. Ada beberapa cara untuk melakukannya, seperti mengatur requires_grad=False pada Tensor, menggunakan fungsi detach() untuk mendapatkan Tensor baru tanpa perhitungan gradien, atau melingkunginya dengan torch.no_grad(). Hal

ini berguna saat kita ingin memperbarui parameter model selama pelatihan tanpa melibatkan perhitungan gradien.

Terakhir, kodingan juga menunjukkan penggunaan backward() dalam pengoptimalan model. Dalam contoh tersebut, kita menggunakan loop training sederhana dan setiap langkah optimasi diikuti dengan mengatur gradien ke nol menggunakan metode zero_(). Hal ini penting untuk memastikan bahwa gradien tidak terakumulasi antar langkah optimasi.

D. Analisis Kode - BACKPROPAGATION

Kodingan yang telah dibuat menggambarkan implementasi sederhana dari alur forward pass, backward pass, dan optimisasi pada sebuah model menggunakan PyTorch. Pada awalnya, kita mendefinisikan dua tensor, x dan y, yang mewakili input dan target output. Selanjutnya, kita mendefinisikan parameter yang ingin dioptimasi, yaitu tensor w dengan requires_grad=True.

Kemudian, kita melakukan forward pass dengan mengalikan x dengan w untuk menghasilkan y_predicted, yang merupakan perkiraan output dari model. Selanjutnya, kita menghitung loss, yaitu selisih kuadrat antara y_predicted dan y, yang merupakan indikator sejauh mana perkiraan model mendekati target.

Setelah melakukan forward pass dan menghitung loss, kita melakukan backward pass dengan memanggil fungsi backward() pada loss. Hal ini akan menghitung gradien loss terhadap semua tensor yang memiliki requires_grad=True, termasuk tensor w. Gradien ini akan disimpan dalam atribut .grad dari w.

Selanjutnya, kita dapat melakukan optimisasi dengan memperbarui nilai w berdasarkan gradien yang telah dihitung. Dalam kodingan tersebut, kita menggunakan metode torch.no_grad() untuk memastikan bahwa operasi pembaruan bobot tidak masuk ke dalam komputasi grafis. Kita mengurangi w dengan 0,01 kali gradien w, dan kemudian mengatur gradien w ke nol menggunakan metode zero_().

Setelah langkah optimisasi, kita dapat melanjutkan dengan forward pass dan backward pass berikutnya untuk melatih dan memperbarui model. Proses ini akan diulang hingga model mencapai hasil yang diinginkan atau memenuhi kondisi berhenti yang ditentukan.

E. Analisis Kode - 1 GRADIENT DESCENT MANUALLY

Kodingan yang telah dibuat merupakan implementasi sederhana dari algoritma gradient descent pada regresi linear menggunakan NumPy. Tujuannya adalah untuk mempelajari bagaimana setiap langkah dalam proses optimisasi dapat dihitung secara manual.

Pertama, kita mendefinisikan data input (X) dan target output (Y). Kemudian, kita mendefinisikan parameter w yang akan dioptimasi. Dalam kasus ini, kita ingin mencari nilai w yang dapat memprediksi output (Y) berdasarkan input (X) dengan persamaan linear sederhana f = w * x.

Selanjutnya, kita mendefinisikan beberapa fungsi. Fungsi forward(x) digunakan untuk memprediksi output (y_pred) berdasarkan nilai w saat ini dan input x. Fungsi loss(y, y_pred) menghitung mean squared error (MSE) antara target output (y) dan prediksi output (y_pred). Fungsi gradient(x, y, y_pred) menghitung gradien fungsi loss terhadap parameter w.

Selanjutnya, kita melakukan iterasi sebanyak n_iters untuk melatih model. Pada setiap iterasi, kita melakukan forward pass untuk memperoleh prediksi output (y_pred), menghitung loss (l), dan menghitung gradien (dw) menggunakan fungsi gradient. Kemudian, kita memperbarui nilai w dengan mengurangi learning_rate dikalikan dengan gradien dw.

Selama proses pelatihan, kita mencetak nilai w dan loss setiap dua iterasi untuk melihat perkembangan pelatihan. Setelah pelatihan selesai, kita mencetak prediksi output (y_pred) untuk input x=5.

Metode ini membantu memahami secara intuitif bagaimana algoritma gradient descent bekerja dengan menghitung setiap langkah secara manual, mulai dari prediksi output hingga pembaruan parameter w.

F. Analisis Kode - 2 GRADIENT DESCENT AUTO

Kodingan yang telah dibuat adalah implementasi algoritma gradient descent menggunakan PyTorch dengan fitur autograd. Fitur ini memungkinkan kita untuk

menghitung gradien secara otomatis tanpa perlu menghitungnya secara manual seperti pada kodingan sebelumnya. Pertama, kita mendefinisikan data input (X) dan target output (Y) menggunakan tensor dari PyTorch. Kemudian, kita mendefinisikan parameter w yang akan dioptimasi. Dalam kasus ini, kita ingin mencari nilai w yang dapat memprediksi output (Y) berdasarkan input (X) dengan persamaan linear sederhana f = w * x.

Selanjutnya, kita mendefinisikan fungsi forward(x) untuk memprediksi output (y_pred) berdasarkan nilai w saat ini dan input x. Fungsi loss(y, y_pred) menghitung mean squared error (MSE) antara target output (y) dan prediksi output (y_pred).

Selanjutnya, kita melakukan iterasi sebanyak n_iters untuk melatih model. Pada setiap iterasi, kita melakukan forward pass untuk memperoleh prediksi output (y_pred), menghitung loss (l), dan menghitung gradien menggunakan fungsi backward() pada objek loss. PyTorch secara otomatis menghitung gradien terhadap parameter w karena telah kita set requires_grad=True pada w sebelumnya.

Setelah menghitung gradien, kita memperbarui nilai w menggunakan metode gradient descent dengan mengurangi learning_rate dikalikan dengan gradien w. Dalam kodingan ini, kita menggunakan blok "with torch.no_grad()" untuk memastikan bahwa operasi pembaruan parameter w tidak terdaftar dalam komputasi graf sehingga tidak mempengaruhi perhitungan gradien.

Setelah pembaruan parameter w, kita menggunakan w.grad.zero_() untuk mengatur ulang gradien menjadi 0 sebelum iterasi berikutnya. Hal ini penting karena PyTorch akan mengakumulasi gradien dari iterasi sebelumnya jika tidak diatur ulang.

Selama proses pelatihan, kita mencetak nilai w dan loss setiap 10 iterasi untuk memantau perkembangan pelatihan. Setelah pelatihan selesai, kita mencetak prediksi output (y_pred) untuk input x=5.

Dengan menggunakan fitur autograd dalam PyTorch, kita dapat dengan mudah menghitung gradien secara otomatis, sehingga mempermudah proses optimisasi dan pelatihan model.

G. Analisis Kode - 1 LOSS AND OPTIMIZER

Kodingan yang telah dibuat merupakan implementasi linear regression menggunakan PyTorch dengan pendefinisian loss function dan optimizer yang lebih terstruktur.

Pertama, kita mendefinisikan data input (X) dan target output (Y) menggunakan tensor dari PyTorch. Kemudian, kita mendefinisikan parameter w yang akan dioptimasi, dengan requires_grad=True agar gradien dapat dihitung secara otomatis.

Selanjutnya, kita mendefinisikan fungsi forward(x) yang mengimplementasikan persamaan linear regression f = w * x. Fungsi ini digunakan untuk memperoleh prediksi output (y_predicted) berdasarkan nilai w saat ini dan input x.

Selanjutnya, kita mencetak prediksi output sebelum pelatihan dilakukan. Langkah berikutnya adalah mendefinisikan loss function dan optimizer. Dalam kodingan ini, kita menggunakan Mean Squared Error (MSE) sebagai loss function dengan menggunakan nn.MSELoss().

Selanjutnya, kita mendefinisikan optimizer yang akan digunakan untuk mengoptimasi parameter w. Dalam kodingan ini, kita menggunakan stochastic gradient descent (SGD) dengan menggunakan torch.optim.SGD(). Optimizer ini akan mengoptimasi parameter w dengan learning rate yang telah ditentukan.

Selanjutnya, kita masuk ke dalam loop pelatihan. Pada setiap iterasi, kita melakukan forward pass untuk memperoleh prediksi output (y_predicted). Kemudian, kita menghitung loss dengan membandingkan prediksi output dengan target output menggunakan loss function yang telah didefinisikan sebelumnya. Setelah menghitung loss, kita melakukan backward pass dengan memanggil metode backward() pada objek loss. Hal ini menghitung gradien terhadap parameter w secara otomatis menggunakan fitur autograd dari PyTorch.

Setelah menghitung gradien, kita memperbarui nilai parameter w menggunakan metode step() pada optimizer. Optimizer ini akan melakukan pembaruan parameter w berdasarkan gradien yang telah dihitung sebelumnya, dengan mempertimbangkan learning rate yang telah ditentukan.

Setelah memperbarui parameter w, kita mengatur ulang gradien menjadi 0 menggunakan metode zero_grad() pada optimizer. Hal ini penting karena PyTorch akan mengakumulasi gradien dari iterasi sebelumnya jika tidak diatur ulang. Selama proses pelatihan, kita mencetak nilai parameter w dan loss setiap 10 iterasi untuk memantau perkembangan pelatihan.

Setelah pelatihan selesai, kita mencetak prediksi output (y_predicted) untuk input x=5 dengan menggunakan nilai parameter w yang telah dioptimasi. Dengan menggunakan loss function dan optimizer yang telah didefinisikan, kita dapat mengoptimasi parameter w dengan lebih terstruktur dan mudah dalam implementasi linear regression.

H. Analisis Kode - 2 MODEL LOSS AND OPTIMIZER

Kodingan yang telah dibuat merupakan implementasi linear regression menggunakan PyTorch dengan pendefinisian model, loss function, dan optimizer yang lebih kompleks. Pertama, kita mendefinisikan data input (X) dan target output (Y) menggunakan tensor dari PyTorch. Dalam kodingan ini, X memiliki dimensi [4, 1], yang berarti terdapat 4 sampel dengan 1 fitur. Kami juga membuat X_test sebagai sampel uji dengan nilai x=5.

Selanjutnya, kita mendefinisikan model untuk linear regression. Dalam kodingan ini, kita menggunakan nn.Linear() yang merupakan model built-in dari PyTorch. Model ini memiliki satu lapisan linier dengan input_size dan output_size yang sama. Kita juga bisa mendefinisikan model secara kustom dengan membuat kelas turunan dari nn.Module dan mendefinisikan lapisan-lapisan yang berbeda. Namun, dalam kodingan ini, kita menggunakan model linear bawaan untuk kesederhanaan. Kemudian, kita mencetak prediksi output sebelum pelatihan dilakukan dengan memanggil model(X_test).

Selanjutnya, kita mendefinisikan loss function dan optimizer. Dalam kodingan ini, kita menggunakan Mean Squared Error (MSE) sebagai loss function dengan nn.MSELoss(). Kemudian, kita mendefinisikan optimizer yang akan digunakan untuk mengoptimasi parameter model. Dalam kodingan ini, kita menggunakan stochastic gradient descent (SGD) dengan menggunakan torch.optim.SGD(). Optimizer ini akan mengoptimasi parameter model berdasarkan gradien yang dihitung selama pelatihan.

Selanjutnya, kita masuk ke dalam loop pelatihan. Pada setiap iterasi, kita melakukan forward pass dengan memasukkan input X ke model untuk memperoleh prediksi output (y_predicted). Kemudian, kita menghitung loss dengan membandingkan prediksi output dengan target output menggunakan loss function yang telah didefinisikan sebelumnya. Setelah menghitung loss, kita melakukan backward pass dengan memanggil metode backward() pada objek loss. Hal ini menghitung gradien terhadap parameter model secara otomatis menggunakan fitur autograd dari PyTorch. Setelah menghitung gradien, kita memperbarui nilai parameter model menggunakan metode step() pada optimizer.

Optimizer ini akan melakukan pembaruan parameter model berdasarkan gradien yang telah dihitung sebelumnya, dengan mempertimbangkan learning rate yang telah ditentukan. Setelah memperbarui parameter model, kita mengatur ulang gradien menjadi 0 menggunakan metode zero_grad() pada optimizer. Hal ini penting karena PyTorch akan mengakumulasi gradien dari iterasi sebelumnya jika tidak diatur ulang.

Selama proses pelatihan, kita mencetak nilai parameter w dan loss setiap 10 iterasi untuk memantau perkembangan pelatihan. Setelah pelatihan selesai, kita mencetak prediksi output (y_predicted) untuk input X_test dengan menggunakan model yang telah dioptimasi.

Dengan menggunakan model, loss function, dan optimizer yang telah didefinisikan dengan lebih kompleks, kita dapat mengoptimasi parameter model secara lebih fleksibel dan dapat digunakan untuk tugas-tugas yang lebih kompleks dalam machine learning.

I. Analisis Kode - LINEAR REGRESSION

Kodingan yang telah dibuat merupakan implementasi dari regresi linear menggunakan PyTorch. Regresi linear adalah metode pembelajaran mesin yang digunakan untuk memodelkan hubungan linier antara variabel input (X) dan variabel target (y). Pertama, kita mempersiapkan data dengan menggunakan fungsi make_regression dari modul sklearn.datasets. Data yang dibuat terdiri dari 100 sampel dengan 1 fitur, dan kita juga menambahkan noise untuk menambah variasi dalam data. Data tersebut kemudian diubah menjadi tensor float menggunakan torch.from_numpy().

Selanjutnya, kita mendefinisikan model linear dengan menggunakan nn.Linear. Model ini memiliki satu lapisan linier dengan input_size yang sesuai dengan jumlah fitur dan output_size sebesar 1, karena kita ingin memprediksi nilai tunggal (y) berdasarkan satu fitur (X). Kemudian, kita mendefinisikan loss function dan optimizer. Dalam kodingan ini, kita menggunakan Mean Squared Error (MSE) sebagai loss function dengan nn.MSELoss(). Optimizer yang digunakan adalah stochastic gradient descent (SGD) dengan torch.optim.SGD().

Setelah itu, kita memasuki loop pelatihan dengan jumlah iterasi sebanyak num_epochs. Pada setiap iterasi, kita melakukan forward pass dengan memasukkan input X ke model untuk memperoleh prediksi output (y_predicted). Selanjutnya, kita

menghitung loss antara prediksi (y_predicted) dan target (y) menggunakan loss function yang telah didefinisikan sebelumnya.

Setelah menghitung loss, kita melakukan backward pass dengan memanggil metode backward() pada objek loss. Hal ini menghitung gradien terhadap parameter model secara otomatis menggunakan fitur autograd dari PyTorch. Kemudian, kita melakukan pembaruan parameter model menggunakan metode step() pada optimizer. Sebelum melakukan iterasi berikutnya, kita mengatur ulang gradien menjadi 0 menggunakan metode zero_grad() pada optimizer. Hal ini penting karena PyTorch akan mengakumulasi gradien dari iterasi sebelumnya jika tidak diatur ulang. Selama proses pelatihan, kita mencetak loss setiap 10 iterasi untuk memantau perkembangan pelatihan.

Setelah pelatihan selesai, kita mendetach nilai prediksi (predicted) dari tensor PyTorch dan mengubahnya menjadi array numpy menggunakan metode detach().numpy(). Kemudian, kita memvisualisasikan data asli (X_numpy, y_numpy) sebagai titik merah dan garis biru yang merepresentasikan prediksi model.

Dengan menggunakan kodingan tersebut, kita dapat melatih model regresi linear untuk melakukan prediksi berdasarkan input X dan menghasilkan visualisasi yang menunjukkan sejauh mana model kita cocok dengan data asli.

J. Analisis Kode - LOGISTIC REGRESSION

Kodingan yang telah dibuat merupakan implementasi regresi logistik menggunakan PyTorch untuk masalah klasifikasi. Regresi logistik adalah metode pembelajaran mesin yang digunakan untuk memodelkan probabilitas kelas target yang biner berdasarkan fitur input. Pertama, kita mempersiapkan data menggunakan dataset "Breast Cancer" dari modul sklearn.datasets. Data tersebut terdiri dari fitur-fitur (X) dan target (y). Fitur-fitur tersebut kemudian dibagi menjadi data pelatihan (X_train, y_train) dan data uji (X_test, y_test) menggunakan fungsi train_test_split. Selanjutnya, kita melakukan penskalaan fitur-fitur menggunakan StandardScaler agar memiliki skala yang serupa. Setelah data dipersiapkan, kita mendefinisikan model regresi logistik. Model ini terdiri dari satu lapisan linier (nn.Linear) yang diikuti oleh fungsi aktivasi sigmoid (torch.sigmoid). Fungsi sigmoid digunakan untuk menghasilkan probabilitas kelas target yang biner.

Selanjutnya, kita mendefinisikan loss function dan optimizer. Dalam kodingan ini, kita menggunakan Binary Cross Entropy (BCE) Loss (nn.BCELoss) sebagai loss function karena kita memiliki masalah klasifikasi biner. Optimizer yang digunakan adalah stochastic gradient descent (SGD) dengan torch.optim.SGD.

Selanjutnya, kita memasuki loop pelatihan dengan jumlah iterasi sebanyak num_epochs. Pada setiap iterasi, kita melakukan forward pass dengan memasukkan input X_train ke model untuk memperoleh prediksi probabilitas kelas (y_pred). Selanjutnya, kita menghitung loss antara prediksi (y_pred) dan target (y_train) menggunakan loss function yang telah didefinisikan sebelumnya. Setelah menghitung loss, kita melakukan backward pass dengan memanggil metode backward() pada objek loss. Hal ini menghitung gradien terhadap parameter model secara otomatis menggunakan fitur autograd dari PyTorch. Kemudian, kita melakukan pembaruan parameter model menggunakan metode step() pada optimizer. Sebelum melakukan iterasi berikutnya, kita mengatur ulang gradien menjadi 0 menggunakan metode zero_grad() pada optimizer. Selama proses pelatihan, kita mencetak loss setiap 10 iterasi untuk memantau perkembangan pelatihan.

Setelah pelatihan selesai, kita menggunakan model yang telah dilatih untuk melakukan prediksi pada data uji (X_test). Dengan menggunakan metode no_grad(), kita memastikan bahwa tidak ada perhitungan gradien yang terjadi selama tahap prediksi. Kita mengambil prediksi probabilitas kelas (y_predicted) dan membulatkannya menjadi kelas biner (y_predicted_cls) dengan menggunakan metode round(). Terakhir, kita menghitung akurasi klasifikasi dengan membandingkan prediksi (y_predicted_cls) dengan target (y_test).

Dengan menggunakan kodingan tersebut, kita dapat melatih model regresi logistik untuk melakukan klasifikasi biner pada dataset "Breast Cancer" dan memperoleh akurasi klasifikasi sebagai metrik evaluasi.

K. Analisis Kode - DATALOADER

Kodingan yang telah dibuat merupakan implementasi penggunaan DataLoader dalam PyTorch untuk mempermudah proses pelatihan model dengan membagi dataset menjadi batch-batch kecil. DataLoader adalah sebuah kelas yang memungkinkan kita untuk memuat data dalam bentuk batch, melakukan pengacakan (shuffle), dan menggunakan

multiple subprocesses untuk mempercepat proses pemuatan data. Pertama, kita mendefinisikan sebuah kelas dataset khusus yang merupakan subclass dari torch.utils.data.Dataset. Kelas ini digunakan untuk mengatur data yang akan digunakan dalam pelatihan. Pada kodingan ini, kita mengimplementasikan kelas WineDataset yang membaca data dari file CSV menggunakan numpy dan kemudian mengubahnya menjadi tensor menggunakan torch. Kelas WineDataset memiliki tiga method yang perlu diimplementasikan yaitu init, getitem, dan len. Method getitem digunakan untuk mengambil sampel data berdasarkan indeks tertentu, sedangkan method len digunakan untuk mengembalikan jumlah total sampel dalam dataset.

Setelah dataset dibuat, kita menggunakan DataLoader untuk memuat dataset tersebut. DataLoader memungkinkan kita untuk memuat dataset dengan membaginya menjadi batch-batch kecil. Pada kodingan ini, kita membuat sebuah DataLoader train_loader yang menggunakan dataset yang telah kita buat sebelumnya. Kita menentukan batch_size (jumlah sampel dalam satu batch), shuffle (pengacakan data), dan num_workers (jumlah subprocess untuk memuat data).

Selanjutnya, kita dapat menggunakan DataLoader sebagai iterator untuk mengakses batch-batch data dalam proses pelatihan. Kodingan tersebut menunjukkan cara mengambil sampel pertama dari DataLoader menggunakan iterasi. Kita juga dapat melihat bentuk features dan labels pada sampel tersebut. Setelah itu, kita dapat melakukan training loop dengan menggunakan DataLoader. Kita dapat melakukan iterasi melalui DataLoader untuk memperoleh batch-batch data pada setiap iterasi. Pada kodingan ini, kita melakukan iterasi sebanyak num_epochs (jumlah epoch) dan n_iterations (jumlah iterasi dalam satu epoch). Pada setiap iterasi, kita dapat menjalankan proses pelatihan model. Pada contoh tersebut, setiap 5 iterasi, kita mencetak informasi mengenai epoch, step, serta bentuk input dan label dari batch yang sedang diproses. Selain itu, kodingan tersebut juga menunjukkan penggunaan DataLoader dengan dataset MNIST dari torchvision.datasets. Dataset tersebut dapat diunduh dan dimuat menggunakan DataLoader dengan konfigurasi yang sesuai. Contoh tersebut menunjukkan cara mengambil sampel pertama dari DataLoader MNIST dan melihat bentuk input dan target dari sampel tersebut.

Dengan menggunakan DataLoader, kita dapat membagi dataset menjadi batch-batch kecil dan melakukan proses pelatihan dengan lebih efisien. DataLoader menyederhanakan

proses pengaturan dan pemuatan data dalam pelatihan model, sehingga memudahkan kita dalam mengimplementasikan algoritma machine learning dengan PyTorch.

L. Analisis Kode - TRANSFORMER

Kodingan yang telah dibuat merupakan contoh implementasi transformasi (transforms) dalam PyTorch. Transformasi digunakan untuk mengubah atau memodifikasi data dalam dataset saat dibuat. Transformasi dapat diterapkan pada gambar PIL, tensor, ndarray, atau data kustom selama pembuatan dataset.

Pada kodingan tersebut, terdapat beberapa contoh transformasi yang dapat dilakukan. Pertama, kita dapat melihat daftar lengkap transformasi bawaan yang disediakan oleh PyTorch di halaman dokumentasi yang diberikan. Transformasi tersebut mencakup berbagai operasi seperti pemotongan (crop), konversi ke grayscale, penambahan padding, transformasi affine acak, pemutaran acak, dan lain-lain.

Transformasi dapat diterapkan pada gambar, tensor, atau ndarrays. Sebagai contoh, pada gambar, kita dapat menggunakan transformasi seperti CenterCrop, RandomCrop, atau Resize untuk memodifikasi ukuran atau memotong bagian tertentu dari gambar. Pada tensor, kita dapat menggunakan transformasi seperti Normalize atau RandomErasing. Selain itu, terdapat juga transformasi konversi seperti ToPILImage dan ToTensor untuk mengubah antara tensor, ndarrays, atau PILImage.

Selain transformasi bawaan, kita juga dapat membuat transformasi kustom dengan mengimplementasikan kelas sendiri. Pada contoh kodingan tersebut, terdapat dua transformasi kustom yaitu ToTensor dan MulTransform. ToTensor mengkonversi ndarray menjadi tensor, sedangkan MulTransform mengalikan input dengan suatu faktor. Untuk menggabungkan beberapa transformasi, kita dapat menggunakan kelas Compose yang disediakan oleh torchvision.transforms. Dengan Compose, kita dapat menggabungkan transformasi dalam urutan tertentu. Sebagai contoh, pada kodingan tersebut, kita membuat transformasi composed yang terdiri dari transformasi ToTensor dan MulTransform dengan faktor 4. Transformasi tersebut akan diterapkan berurutan pada dataset.

Pada saat membuat dataset WineDataset, kita dapat menyertakan transformasi yang ingin digunakan dengan menyediakan parameter transform. Ketika mendapatkan sampel

dari dataset menggunakan metode getitem, transformasi akan diterapkan pada sampel tersebut jika transformasi telah diberikan.

Dengan menggunakan transformasi, kita dapat dengan mudah melakukan modifikasi atau konversi pada data saat memuat dataset. Transformasi memungkinkan kita untuk mempersiapkan data secara efisien sebelum digunakan dalam pelatihan atau pengujian model.

M. Analisis Kode - SOFTMAX AND CROSSENTROPY

Kodingan yang telah dibuat merupakan contoh implementasi fungsi softmax dan perhitungan loss cross entropy dalam Python menggunakan numpy dan PyTorch. Pertama, kita melihat implementasi fungsi softmax menggunakan numpy. Fungsi softmax mengambil input dalam bentuk array dan mengaplikasikan fungsi eksponensial pada setiap elemen, kemudian melakukan normalisasi dengan membagi hasilnya dengan jumlah eksponensial dari semua elemen input. Fungsi softmax menghasilkan probabilitas yang menyquash (memampatkan) output antara 0 dan 1, dengan asumsi bahwa probabilitas semua kelas yang mungkin akan memiliki jumlah total 1. Pada contoh kodingan tersebut, kita melihat penggunaan fungsi softmax pada array input dan menghasilkan output probabilitas.

Selanjutnya, kita melihat implementasi perhitungan loss cross entropy menggunakan numpy. Loss cross entropy atau log loss digunakan untuk mengukur kinerja model klasifikasi yang menghasilkan probabilitas sebagai output antara 0 dan 1. Loss ini meningkat saat probabilitas prediksi divergen dari label aktual. Pada contoh kodingan tersebut, kita melihat penggunaan fungsi cross_entropy yang mengambil argumen aktual (label sebenarnya) dan prediksi (probabilitas yang diprediksi oleh model). Fungsi cross_entropy menghitung loss dengan membandingkan probabilitas prediksi dengan label aktual menggunakan rumus -Σ(y * log(y_hat)), di mana y adalah label aktual dan y_hat adalah probabilitas prediksi. Hasil loss akan lebih tinggi jika prediksi semakin jauh dari label aktual.

Selanjutnya, kita melihat implementasi softmax dan loss cross entropy menggunakan PyTorch. PyTorch menyediakan fungsi softmax dalam modul torch dengan menggunakan torch.softmax. Kita juga melihat penggunaan fungsi CrossEntropyLoss dalam modul torch.nn. Fungsi CrossEntropyLoss pada PyTorch menerapkan softmax secara otomatis

pada input dan menghitung loss cross entropy. Pada contoh kodingan tersebut, kita melihat penggunaan fungsi CrossEntropyLoss dengan input logit (skor yang belum di-normalisasi sebelum softmax) dan label aktual untuk menghitung loss.

Selain itu, contoh kodingan juga menunjukkan penggunaan loss cross entropy pada kasus multikelas dan kasus biner. Untuk kasus multikelas, output dari model adalah logits (skor yang belum di-normalisasi sebelum softmax) dari masing-masing kelas, dan label aktual yang digunakan dalam perhitungan loss adalah dalam bentuk integer yang mewakili kelas yang benar. Sedangkan untuk kasus biner, output dari model adalah probabilitas tunggal antara 0 dan 1, dan label aktual yang digunakan dalam perhitungan loss adalah dalam bentuk biner (0 atau 1).

Dalam kodingan tersebut, juga ditunjukkan bagaimana membuat model menggunakan PyTorch dengan menggunakan modul nn.Module. Contoh model neural network digunakan untuk ilustrasi, termasuk model untuk masalah biner dan multikelas. Kemudian, loss yang sesuai (BCELoss untuk kasus biner dan CrossEntropyLoss untuk kasus multikelas) digunakan untuk menghitung loss pada output model.

N. Analisis Kode - ACTIVATION FUNCTION

Kodingan yang telah dibuat menunjukkan contoh penggunaan berbagai fungsi aktivasi pada implementasi neural network menggunakan PyTorch. Pada bagian awal kodingan, kita melihat penggunaan beberapa fungsi aktivasi secara langsung pada tensor input x. Fungsi-fungsi aktivasi yang digunakan termasuk softmax, sigmoid, tanh, relu, dan leaky relu. Fungsi softmax digunakan untuk menghasilkan probabilitas pada output tensor x dengan memampatkan nilai-nilai input ke dalam rentang antara 0 dan 1 serta memastikan bahwa total probabilitas dari semua kelas adalah 1. Fungsi sigmoid digunakan untuk menghasilkan nilai antara 0 dan 1 yang dapat diinterpretasikan sebagai probabilitas atau sebagai aktivasi neuron. Fungsi tanh menghasilkan nilai antara -1 dan 1 dan sering digunakan pada hidden layer dalam neural network. Fungsi relu (rectified linear unit) menghasilkan output x untuk nilai x yang positif dan 0 untuk nilai x yang negatif, sehingga dapat mempelajari representasi linear dan non-linear yang kompleks. Fungsi leaky relu juga memiliki sifat yang serupa dengan relu, tetapi dengan penambahan kemiringan kecil pada sisi negatif, sehingga mengatasi masalah "dying ReLU" di mana neuron dengan output negatif bisa mati selama pelatihan.

Selanjutnya, dalam kodingan tersebut, ditunjukkan dua opsi untuk mengimplementasikan fungsi aktivasi dalam model neural network menggunakan PyTorch. Opsi pertama adalah dengan membuat modul nn.Module yang berisi definisi fungsi aktivasi sebagai atribut dari model. Contoh ini menunjukkan penggunaan nn.Linear, nn.ReLU, dan nn.Sigmoid sebagai modul-modul dalam model. Opsi kedua adalah dengan menggunakan fungsi aktivasi secara langsung dalam metode forward model. Contoh ini menunjukkan penggunaan fungsi-fungsi aktivasi seperti torch.relu dan torch.sigmoid pada tahap-tahap yang relevan dalam metode forward. Kedua opsi ini mencapai tujuan yang sama, yaitu menerapkan fungsi aktivasi pada output dari layer-layer dalam model neural network.

Dengan menggunakan fungsi-fungsi aktivasi ini, model neural network dapat mempelajari representasi dan pola yang lebih kompleks dan non-linear dari data input. Fungsi-fungsi aktivasi ini memainkan peran penting dalam memberikan sifat non-linear pada model neural network dan meningkatkan kemampuannya untuk memodelkan hubungan yang kompleks antara fitur-fitur input dan label output yang diinginkan.

O. Analisis Kode - PLOT ACTIVATIONS

Kodingan yang telah dibuat menunjukkan bagaimana kita dapat memvisualisasikan beberapa fungsi aktivasi yang umum digunakan dalam jaringan saraf (neural network) menggunakan NumPy dan Matplotlib. Pada awal kodingan, beberapa fungsi aktivasi yang akan divisualisasikan telah didefinisikan. Fungsi-fungsi aktivasi yang digunakan termasuk sigmoid, tanh, ReLU, leaky ReLU, dan binary step. Setiap fungsi aktivasi diimplementasikan sebagai lambda function yang menerima input x dan menghasilkan output sesuai dengan aturan fungsi tersebut.

Selanjutnya, kita menentukan rentang nilai x yang akan divisualisasikan, yaitu dari -10 hingga 10. Rentang ini dibagi menjadi 10 nilai untuk keperluan plotting dan 100 nilai untuk menghasilkan kurva yang lebih halus pada visualisasi. Kemudian, menggunakan Matplotlib, kita membuat empat subplot terpisah untuk setiap fungsi aktivasi. Untuk setiap subplot, kita memplot hasil fungsi aktivasi pada rentang nilai y yang telah ditentukan sebelumnya. Penambahan label, grid, sumbu x dan y, serta judul plot dilakukan untuk meningkatkan kejelasan visualisasi.

Selain itu, batasan dan penandaan sumbu x dan y juga ditentukan untuk memberikan konteks pada visualisasi. Rentang batasan pada sumbu y berbeda-beda tergantung pada fungsi aktivasi yang ditampilkan. Terakhir, menggunakan plt.show(), kita menampilkan keseluruhan visualisasi dari keempat fungsi aktivasi.

Kodingan tersebut memberikan visualisasi grafik yang menggambarkan karakteristik dan bentuk dari setiap fungsi aktivasi. Dengan memvisualisasikan fungsi-fungsi aktivasi ini, kita dapat memahami cara kerja dan efek dari masing-masing fungsi pada transformasi nilai input menjadi output dalam konteks neural network.

P. Analisis Kode - FEEDFORWARD

Kodingan yang telah dibuat mengimplementasikan pelatihan dan pengujian sebuah model jaringan saraf (neural network) menggunakan dataset MNIST. Pertama, kodingan mendefinisikan beberapa konfigurasi dan hyperparameter yang akan digunakan dalam proses pelatihan. Ini termasuk konfigurasi perangkat (device) yang digunakan (CPU atau CUDA jika tersedia), ukuran input layer (input_size), ukuran hidden layer (hidden_size), jumlah kelas output (num_classes), jumlah epoch (num_epochs), ukuran batch (batch_size), dan learning rate (learning_rate).

Selanjutnya, kodingan mengunduh dataset MNIST dan memuatnya ke dalam data loader untuk pelatihan dan pengujian. Dataset MNIST berisi gambar angka tulisan tangan dari 0 hingga 9. Gambar-gambar ini kemudian ditampilkan menggunakan Matplotlib dalam bentuk subplot untuk memvisualisasikan beberapa contoh gambar dari dataset.

Setelah itu, sebuah kelas NeuralNet didefinisikan sebagai turunan dari nn.Module. Kelas ini mewakili model jaringan saraf yang akan digunakan dalam pelatihan. Model ini memiliki dua lapisan linear (fully connected layer) yaitu l1 dan l2, dengan fungsi aktivasi ReLU di antara keduanya. Kelas ini juga memiliki metode forward untuk menjalankan proses feedforward pada model.

Selanjutnya, sebuah objek model NeuralNet dibuat dengan menggunakan ukuran input layer, hidden layer, dan jumlah kelas output yang telah ditentukan sebelumnya. Model ini juga dipindahkan ke perangkat yang ditentukan (CPU atau CUDA).

Selanjutnya, fungsi loss yang digunakan adalah CrossEntropyLoss, dan optimizer yang digunakan adalah Adam optimizer dengan learning rate yang telah ditentukan. Ini

digunakan untuk menghitung loss dan melakukan backpropagation serta optimisasi pada setiap iterasi pelatihan.

Selanjutnya, model dilatih menggunakan data loader untuk beberapa epoch. Pada setiap iterasi, gambar-gambar dan label-label diambil dari data loader, dan dilakukan feedforward pada model untuk menghasilkan output. Loss dihitung dengan membandingkan output dengan label menggunakan fungsi loss yang telah ditentukan. Kemudian, dilakukan backpropagation dan optimisasi dengan memanggil optimizer.zero_grad(), loss.backward(), dan optimizer.step().

Selama pelatihan, informasi seperti epoch, step, dan loss di-print untuk melacak kemajuan pelatihan. Setelah pelatihan selesai, model diuji pada data pengujian. Dalam fase pengujian, perhitungan gradien tidak diperlukan, sehingga dilakukan dengan menggunakan torch.no_grad(). Model diterapkan pada setiap gambar pengujian, dan label hasil prediksi dibandingkan dengan label asli. Akurasi model dihitung dengan membandingkan jumlah prediksi yang benar dengan jumlah total sampel pengujian. Akurasi model kemudian dicetak sebagai hasil akhir dari pengujian.

Dengan demikian, kodingan "FEEDFORWARD" tersebut melibatkan proses pelatihan dan pengujian sebuah model jaringan saraf dengan menggunakan dataset MNIST untuk mengklasifikasikan gambar angka tulisan tangan.

Q. Analisis Kode - CNN

Kodingan "CNN" di atas merupakan implementasi dari Convolutional Neural Network (CNN) menggunakan dataset CIFAR-10. Pertama, kodingan menentukan beberapa konfigurasi dan hyperparameter yang akan digunakan dalam pelatihan. Ini meliputi jumlah epoch (num_epochs), ukuran batch (batch_size), dan learning rate (learning_rate). Selanjutnya, dataset CIFAR-10 diunduh dan dimuat menggunakan torchvision. Dataset ini berisi 60.000 gambar berukuran 32x32 piksel dalam 10 kelas yang berbeda. Transformasi transform digunakan untuk mengubah gambar-gambar tersebut menjadi tensor dan melakukan normalisasi.

Selanjutnya, kodingan membuat data loader untuk pelatihan dan pengujian dengan menggunakan train_loader dan test_loader. Kemudian, fungsi imshow() didefinisikan untuk menampilkan beberapa contoh gambar dari dataset pelatihan. Setelah itu, sebuah

kelas ConvNet didefinisikan sebagai turunan dari nn.Module. Kelas ini mewakili model CNN yang akan digunakan dalam pelatihan. Model ini terdiri dari beberapa lapisan konvolusi (convolutional layer) dan lapisan linear (fully connected layer). Di antara lapisan-lapisan tersebut, fungsi aktivasi ReLU diterapkan. Kelas ini juga memiliki metode forward untuk menjalankan proses feedforward pada model.

Selanjutnya, objek model ConvNet dibuat dan dipindahkan ke perangkat yang ditentukan (CPU atau CUDA). Loss function yang digunakan adalah CrossEntropyLoss, dan optimizer yang digunakan adalah SGD optimizer dengan learning rate yang telah ditentukan sebelumnya. Ini digunakan untuk menghitung loss dan melakukan backpropagation serta optimisasi pada setiap iterasi pelatihan.

Selama pelatihan, model dilatih menggunakan data loader selama beberapa epoch. Pada setiap iterasi, gambar-gambar dan label-label diambil dari data loader, dan dilakukan feedforward pada model untuk menghasilkan output. Loss dihitung dengan membandingkan output dengan label menggunakan fungsi loss yang telah ditentukan sebelumnya. Kemudian, dilakukan backpropagation dan optimisasi dengan memanggil optimizer.zero_grad(), loss.backward(), dan optimizer.step().

Selama pelatihan, informasi seperti epoch, step, dan loss di-print untuk melacak kemajuan pelatihan. Setelah pelatihan selesai, model disimpan ke dalam file "./cnn.pth" menggunakan torch.save(). Selanjutnya, model diuji pada data pengujian. Dalam fase pengujian, perhitungan gradien tidak diperlukan, sehingga dilakukan dengan menggunakan torch.no_grad(). Model diterapkan pada setiap gambar pengujian, dan label hasil prediksi dibandingkan dengan label asli. Akurasi model dihitung dengan membandingkan jumlah prediksi yang benar dengan jumlah total sampel pengujian.

Akurasi model secara keseluruhan dan akurasi untuk setiap kelas dihitung dan dicetak sebagai hasil akhir dari pengujian. Dengan demikian, kodingan "CNN" tersebut melibatkan proses pelatihan dan pengujian sebuah model CNN menggunakan dataset CIFAR-10 untuk mengklasifikasikan gambar-gambar dalam 10 kelas yang berbeda.

R. Analisis Kode - TRANSFER LEARNING

Pada kodingan yang telah dibuat, dilakukan penggunaan teknik transfer learning untuk mengatasi permasalahan dalam mengembangkan model jaringan saraf konvolusi

(CNN) dari awal. Transfer learning adalah pendekatan yang memanfaatkan pengetahuan yang telah dipelajari oleh model yang sudah terlatih sebelumnya pada tugas yang serupa atau terkait. Pendekatan ini dapat membantu dalam situasi ketika dataset yang tersedia terbatas atau ketika ingin mempercepat proses pelatihan model.

Pada kodingan tersebut, langkah pertama adalah menyiapkan dataset untuk pelatihan dan validasi. Dataset yang digunakan adalah Hymenoptera, yang berisi gambar dari dua kelas serangga: semut dan lebah. Kemudian dilakukan normalisasi dan augmentasi data menggunakan transformasi data seperti RandomResizedCrop, RandomHorizontalFlip, Resize, dan Normalize. Data dipecah menjadi dua bagian: data pelatihan dan data validasi.

Setelah itu, dilakukan inisialisasi model menggunakan arsitektur ResNet-18 yang telah dilatih sebelumnya pada dataset ImageNet. Model ini diubah untuk memprediksi dua kelas dengan mengganti lapisan terakhir menjadi lapisan linier dengan keluaran 2. Model tersebut diatur untuk menggunakan perangkat keras (GPU jika tersedia) dan fungsi kerugian CrossEntropyLoss.

Kemudian, dilakukan pelatihan model dengan fungsi train_model(). Model dilatih dalam beberapa epoch. Pada setiap epoch, dilakukan iterasi pada data pelatihan dan validasi. Pada setiap iterasi, input dilewatkan melalui model dan prediksi dibandingkan dengan label yang sebenarnya. Kemudian, dilakukan perhitungan fungsi kerugian dan pembaruan parameter model menggunakan optimasi stokastik gradien (SGD). Model dievaluasi menggunakan akurasi dan fungsi kerugian pada setiap fase. Model terbaik (dengan akurasi validasi tertinggi) disimpan.

Selanjutnya, dilakukan finetuning model. Model ResNet-18 diinisialisasi kembali, tetapi kali ini semua parameter kecuali lapisan terakhir dibekukan (requires_grad = False). Lapisan terakhir diganti dengan lapisan linier baru untuk memprediksi dua kelas. Hanya parameter lapisan terakhir yang dioptimalkan saat melatih model menggunakan SGD. Proses pelatihan berlangsung dalam beberapa epoch dengan jadwal penurunan tingkat pembelajaran menggunakan StepLR.

Dengan menggunakan teknik transfer learning, kodingan tersebut memungkinkan pengembangan model CNN yang efektif dengan menggunakan pengetahuan yang telah dipelajari oleh model yang lebih besar dan terlatih sebelumnya. Hal ini membantu dalam

meningkatkan kinerja model dan mengurangi waktu dan sumber daya yang dibutuhkan untuk pelatihan dari awal.

S. Analisis Kode - TENSORBOARD

Kodingan yang telah dibuat merupakan penggunaan TensorBoard dalam pelatihan dan evaluasi model jaringan saraf menggunakan library PyTorch. TensorBoard adalah alat visualisasi yang kuat yang digunakan untuk memahami, memantau, dan menganalisis pelatihan model. Pertama, kode menginisialisasi objek SummaryWriter yang bertanggung jawab untuk mencatat data pelatihan dan pengujian yang akan divisualisasikan menggunakan TensorBoard. Objek SummaryWriter ini menyimpan log yang akan digunakan untuk membangun visualisasi seperti grafik, kurva kehilangan (loss), atau kurva akurasi.

Selanjutnya, sejumlah visualisasi ditambahkan menggunakan TensorBoard. Pertama, gambar-gambar contoh dari dataset MNIST ditampilkan sebagai grid menggunakan make_grid, dan kemudian gambar tersebut ditambahkan ke TensorBoard menggunakan add_image. Selanjutnya, grafik arsitektur model ditambahkan menggunakan add_graph, yang mengambil model dan input untuk menampilkan grafik arsitektur model dalam TensorBoard.

Selama pelatihan model, beberapa nilai seperti loss dan akurasi dihitung. Nilai-nilai ini kemudian ditambahkan ke TensorBoard menggunakan add_scalar, yang mencatat nilai skalar seperti loss dan akurasi dari setiap langkah pelatihan. Dalam kasus ini, add_scalar digunakan untuk mencatat loss pelatihan dan akurasi dari setiap 100 langkah pelatihan.

Setelah pelatihan, model dievaluasi menggunakan data pengujian. Hasil evaluasi seperti akurasi model dan kurva presisi-dan-recall (precision-recall curve) ditambahkan ke TensorBoard menggunakan add_pr_curve. Ini memungkinkan kita untuk memvisualisasikan performa model dalam memprediksi setiap kelas dengan presisi dan recall yang berbeda.

Setelah penambahan semua visualisasi, objek SummaryWriter ditutup menggunakan close(), dan data yang telah dicatat akan disimpan dan dapat dilihat dalam TensorBoard.

Dengan menggunakan TensorBoard, kita dapat memantau dan menganalisis pelatihan dan evaluasi model secara interaktif. Visualisasi seperti grafik, kurva kehilangan, akurasi, dan presisi-dan-recall curve memberikan wawasan yang berguna dalam memahami performa dan perilaku model kita.

T. Analisis Kode - SAVE LOAD

Kodingan yang telah dibuat menjelaskan tentang metode-metode yang digunakan untuk menyimpan (save) dan memuat (load) model jaringan saraf menggunakan PyTorch. Terdapat dua cara utama untuk menyimpan model, yaitu:

Cara yang lebih sederhana adalah dengan menyimpan keseluruhan model. Dalam cara ini, seluruh model beserta parameter-parameter dan struktur arsitektur disimpan ke dalam file. Untuk menyimpan model, kita menggunakan torch.save(model, PATH). Kemudian, untuk memuat kembali model, kita menggunakan model = torch.load(PATH). Model yang dimuat akan berada dalam mode evaluasi dengan menggunakan model.eval().

Cara yang lebih disarankan adalah dengan menyimpan hanya state_dict dari model. State_dict adalah sebuah dictionary Python yang berisi parameter-parameter dari setiap lapisan (layer) dalam model. Untuk menyimpan state_dict, kita menggunakan torch.save(model.state_dict(), PATH). Ketika memuat kembali model, kita perlu membuat kembali objek model dengan menggunakan model = Model(*args, **kwargs), dan kemudian memuat state_dict menggunakan model.load_state_dict(torch.load(PATH)). Seperti pada cara sebelumnya, model yang dimuat juga akan berada dalam mode evaluasi.

Selain itu, kodingan juga menunjukkan bagaimana menyimpan dan memuat "checkpoint" selama pelatihan model. Checkpoint merupakan titik pemulihan yang menyimpan informasi seperti epoch terakhir, state_dict model, dan state_dict optimizer. Checkpoint sangat berguna ketika kita ingin melanjutkan pelatihan dari titik yang tertunda. Untuk menyimpan checkpoint, kita membuat dictionary yang berisi informasi yang relevan dan menggunakan torch.save untuk menyimpannya. Kemudian, kita dapat memuat checkpoint menggunakan torch.load, memuat state_dict model dan state_dict optimizer, serta mengatur model dalam mode evaluasi atau mode pelatihan.

Selain itu, kodingan juga menunjukkan beberapa cara untuk menyimpan dan memuat model ketika digunakan GPU atau CPU. Misalnya, jika model dilatih menggunakan GPU

dan ingin memuatnya pada CPU, kita dapat menggunakan argumen map_location dalam torch.load untuk memastikan model dimuat pada CPU. Sebaliknya, jika model dilatih menggunakan GPU dan ingin memuatnya kembali pada GPU, kita perlu memastikan model dan data masukan dikirimkan ke perangkat GPU yang sama menggunakan model.to(device).

Dengan menggunakan metode-metode ini, kita dapat menyimpan model yang dilatih dan memuatnya kembali untuk penggunaan berikutnya, baik itu untuk melakukan inferensi atau melanjutkan pelatihan dari titik terakhir.