TECHNICAL REPORT MACHINE LEARNING

"PyTorch for Deep Learning"

Disusun oleh :

Arfiq Rimeldo

(1103202102)

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO UNIVERSITAS

TELKOM

2023

Pendahuluan

PyTorch adalah sebuah framework machine learning berbasis Python yang sangat populer dan sering digunakan dalam pengembangan dan penelitian di bidang kecerdasan buatan. PyTorch menyediakan alat-alat yang kuat untuk membangun, melatih, dan menerapkan berbagai model jaringan saraf, terutama model jaringan saraf berbasis konvolusi (CNN) dan model jaringan saraf berulang (RNN).

Tensor Basis

Pada kode yang diberikan, kita dapat melihat beberapa contoh penggunaan tensor dalam PyTorch. Tensor dalam PyTorch adalah objek utama yang digunakan untuk melakukan operasi dan komputasi dalam framework ini. Kode tersebut memperlihatkan berbagai cara untuk membuat tensor dengan dimensi yang berbeda, baik itu tensor skalar, vektor, matriks, maupun tensor dengan dimensi yang lebih tinggi. Selain itu, kita juga dapat melakukan operasi dasar pada tensor seperti penjumlahan, pengurangan, perkalian, dan pembagian. Kode tersebut juga menunjukkan cara untuk memanipulasi tensor melalui slicing dan reshape menggunakan metode view(). Selain itu, kita juga melihat bagaimana kita dapat mengkonversi tensor PyTorch menjadi array NumPy dan sebaliknya. Terakhir, terdapat contoh penggunaan GPU untuk mempercepat komputasi dengan memindahkan tensor ke GPU jika tersedia

Autograd

Pada kode yang diberikan, kita dapat melihat penggunaan modul **torch.autograd** dalam PyTorch, yang menyediakan kemampuan diferensiasi otomatis untuk operasi pada tensor. Ketika kita mengatur **requires_grad=True** pada tensor, hal ini memungkinkan pelacakan operasi yang dilakukan pada tensor tersebut, sehingga kita dapat melakukan backpropagation dan menghitung gradien secara otomatis. Kode tersebut memperlihatkan contoh penggunaan autograd, di mana kita membuat tensor **x** dengan **requires_grad=True** dan melakukan beberapa operasi pada tensor tersebut. Kemudian, dengan memanggil **backward()** pada tensor keluaran **z**, kita dapat menghitung gradien tensor input **x** terhadap tensor output **z**. Kode juga menunjukkan cara menghentikan pelacakan histori pada tensor dengan menggunakan metode **.requires_grad_(False)**, **.detach()**, atau **with torch.no_grad():**. Selain itu, dalam contoh terakhir, kita melihat bagaimana gradien dapat diakumulasikan pada setiap langkah optimisasi dan perlu direset dengan **.zero_()** agar tidak terjadi perubahan yang tidak diinginkan.

Backpropagation

Pada kode yang diberikan, kita dapat melihat implementasi backpropagation dalam PyTorch untuk mengoptimasi parameter. Kode tersebut menggunakan pendekatan sederhana dengan satu variabel input (**x**), satu variabel target (**y**), dan satu parameter yang ingin dioptimasi (**w**). Dalam langkah forward pass, kita mengalikan **x** dengan **w** untuk mendapatkan prediksi **y_predicted**. Selanjutnya, kita menghitung loss dengan mengkuadratkan selisih antara **y_predicted** dan **y**. Kemudian, dengan memanggil **loss.backward()**, kita dapat menghitung gradien loss terhadap parameter **w** menggunakan backpropagation. Gradien tersebut dapat diakses melalui atribut **.grad** dari tensor **w**. Setelah itu, kita dapat memperbarui nilai **w** dengan meminimalkan loss melalui operasi **w -= 0.01 * w.grad** yang dilakukan di luar grafik komputasi menggunakan **torch.no_grad()**. Terakhir, kita perlu mengatur ulang gradien dengan **w.grad.zero_()** sebelum melanjutkan forward dan backward pass pada langkah-langkah optimisasi berikutnya. Dengan melakukan iterasi forward pass, backward pass, dan update parameter, kita dapat mengoptimasi nilai parameter **w** berdasarkan loss yang dihasilkan dan mencapai hasil yang lebih baik seiring dengan iterasi yang berlangsung.

Gradient Descent Manually

Pada kode yang diberikan kita melihat implementasi algoritma Gradient Descent secara manual untuk melakukan Linear Regression. Pertama, kita mendefinisikan fungsi forward yang menghitung prediksi model dengan memperoleh hasil perkalian parameter **w** dengan input **x**. Selanjutnya, kita menggunakan Mean Squared Error (MSE) sebagai fungsi loss untuk mengukur sejauh mana prediksi model berbeda dari nilai target **y**. Dalam langkah backpropagation, kita menghitung gradien loss terhadap parameter **w** secara manual dengan menggunakan turunan parsial dari MSE. Kemudian, kita memperbarui nilai parameter **w** dengan mengurangi gradien yang telah dikalikan dengan learning rate. Iterasi ini dilakukan selama beberapa epoch dengan menggunakan loop for. Pada setiap iterasi, kita mencetak nilai parameter **w** dan loss yang terkini. Akhirnya, kita melakukan prediksi menggunakan model yang telah diupdate pada nilai input 5 dan mencetak hasilnya setelah pelatihan selesai

Gradient Descent Auto

Pada kode yang diberikan merupakan implementasi gradient descent menggunakan PyTorch dengan bantuan fitur autograd. Gradient descent adalah algoritma optimisasi yang digunakan untuk meminimalkan fungsi loss dengan mengupdate parameter secara iteratif. Dalam kode ini, kita menggunakan linear regression sebagai contoh, di mana kita mencoba mempelajari parameter **w** yang menghubungkan input **X** dengan target **Y**. Dengan menggunakan autograd, kita dapat menghitung gradien loss terhadap parameter **w** secara otomatis, sehingga tidak perlu menghitung gradien secara manual. Selama proses training, kita mengupdate nilai **w** dengan mengurangi learning rate dikali gradien yang telah dihitung. Dengan melakukan iterasi sejumlah epoch yang ditentukan, kita dapat mengamati perubahan nilai **w** dan loss pada setiap iterasi, dan akhirnya mendapatkan prediksi yang lebih baik setelah pelatihan selesai.

Loss and Optimizer

Pada kode yang diberikan itu menggunakan loss dan optimizer untuk melatih model linear regression. Pertama,mendefinisikan objek loss menggunakan nn.MSELoss() yang merupakan fungsi kerugian Mean Squared Error (MSE). Fungsi ini mengukur kesalahan antara prediksi model dan target output. Selanjutnya, mendefinisikan optimizer menggunakan torch.optim.SGD([w], lr=learning_rate), di mana w adalah parameter yang akan dioptimasi dan learning_rate adalah laju pembelajaran yang menentukan seberapa besar perubahan yang dilakukan pada setiap langkah optimisasi. SGD (Stochastic Gradient Descent) adalah algoritme optimisasi yang digunakan di sini

Model Loss and Optimizer

Pada kode yang diberikan itu adalah implementasi linear regression dengan model yang dibangun menggunakan modul nn.Linear. Pertama, mendefinisikan data input (X) dan target output (Y) sebagai tensor PyTorch. Jumlah sampel dan fitur dihitung dengan X.shape dan dicetak. Kemudian, kita mendefinisikan data input untuk pengujian (X_test) sebagai tensor PyTorch. Selanjutnya,mendefinisikan ukuran input dan output model. Model linear regression dibangun dengan menggunakan nn.Linear(input_size, output_size), di mana input_size dan output_size adalah ukuran dimensi input dan output. Model tersebut adalah objek dari kelas nn.Linear yang sudah tersedia di PyTorch. Setelah itu, dicetak prediksi sebelum pelatihan menggunakan model(X_test). Selanjutnya,

mendefinisikan fungsi kerugian (loss function) menggunakan nn.MSELoss(), dan optimizer menggunakan SGD dengan memanggil torch.optim.SGD(model.parameters(), lr=learning_rate).

Linear Regression

Pada kode yang diberikan itu adalah implementasi regresi linier untuk memprediksi data yang dibuat secara acak. Pertama, kita menggunakan datasets.make_regression dari modul sklearn untuk membuat data acak dengan jumlah sampel (n_samples) sebanyak 100, jumlah fitur (n_features) sebanyak 1, tingkat kebisingan (noise) sebesar 20, dan seed (random_state) sebesar 4. Data tersebut kemudian disimpan dalam variabel X_numpy (fitur) dan y_numpy (target). Selanjutnya, data tersebut dikonversi menjadi tensor PyTorch menggunakan torch.from_numpy dan ditentukan tipe datanya sebagai float32. Selain itu, bentuk tensor target y diubah menggunakan y.view(y.shape[0], 1) untuk memastikan memiliki dimensi yang sesuai dengan output model. Kemudian, diinisialisasi ukuran input (input_size) sesuai dengan jumlah fitur dan ukuran output (output_size) sebesar 1. Model regresi linier dibangun menggunakan nn.Linear(input_size, output_size). Selanjutnya, ditentukan learning_rate untuk optimasi dan kriteria (criterion) yang merupakan fungsi kerugian Mean Squared Error (MSE). Optimizer yang digunakan adalah SGD (Stochastic Gradient Descent) dengan memanggil torch.optim.SGD(model.parameters(), lr=learning_rate)

Logistic Regression

Pada kode yang diberikan, kita menggunakan PyTorch untuk melakukan klasifikasi pada dataset kanker payudara. Kami memanfaatkan dataset kanker payudara yang tersedia dalam library scikit-learn dan membaginya menjadi data latih dan data uji. Setelah itu, kami melakukan normalisasi data menggunakan StandardScaler dan mengubahnya menjadi tensor PyTorch. Model yang digunakan dalam kode tersebut dirancang menggunakan modul nn, dan kami menggunakan stochastic gradient descent untuk mengoptimasi model tersebut. Setelah proses pelatihan, kami mencetak akurasi model pada data uji untuk mengevaluasi performa model yang telah dilatih

Data Loader

Pada kode yang diberikan tersebut menggunakan PyTorch untuk memuat dan memproses dataset wine menggunakan modul Dataset dan

DataLoader. Kode tersebut mendefinisikan kelas WineDataset yang membaca data wine.csv dan mengubahnya menjadi tensor PyTorch. Selanjutnya, kode tersebut memanfaatkan DataLoader untuk membagi data menjadi batch-batch kecil dan memuat data secara paralel. Di dalam kode tersebut, terdapat juga cetakan yang menampilkan fitur dan label dari data pertama serta batch pertama.

Transformers

Pada kode yang diberikan tersebut merupakan implementasi dataset kustom menggunakan kelas turunan dari `torch.utils.data.Dataset` pada PyTorch. Dataset ini digunakan untuk membaca dan memuat data dari file CSV yang berisi informasi terkait klasifikasi jenis-jenis anggur. Kelas `WineDataset` memiliki metode `__init__` untuk menginisialisasi atribut `x_data` dan `y_data` yang merepresentasikan atribut dan label dari dataset, serta atribut `transform` yang digunakan untuk melakukan transformasi pada dataset. Metode `__getitem__` digunakan untuk mengakses elemen dataset berdasarkan indeks, mengambil sampel (input dan target), dan menerapkan transformasi jika diberikan. Metode `__len__` mengembalikan jumlah sampel dalam dataset.

Selain itu, terdapat kelas transformasi seperti `ToTensor` dan `MulTransform`. `ToTensor` mengkonversi sampel menjadi tensor PyTorch, sementara `MulTransform` mengalikan input dengan faktor tertentu.Pada bagian utama kode, dilakukan pengujian dataset dengan dan tanpa transformasi. Pertama, dataset dibuat tanpa transformasi (transform=None) dan kemudian elemen pertama dataset diakses. Hasilnya, fitur dan label dalam bentuk numpy array. Selanjutnya, dataset diuji dengan transformasi tensor (transform=ToTensor()) dan transformasi tensor dengan perkalian (transform=composed). Hasilnya, fitur dan label dalam bentuk tensor PyTorch dan telah mengalami transformasi sesuai yang ditentukan.

Softmax and Crossentropy

Pada kode yang diberikan itu adalah contoh implementasi softmax dan fungsi loss cross-entropy menggunakan NumPy dan PyTorch. Softmax digunakan untuk menghitung probabilitas kelas, sementara fungsi loss cross-entropy digunakan untuk mengukur perbedaan antara distribusi probabilitas yang dihasilkan oleh model dan label yang sebenarnya. Kode tersebut juga menunjukkan penggunaan fungsi loss cross-entropy pada PyTorch untuk klasifikasi biner dan multikelas. Selain itu, kode tersebut

juga mengilustrasikan penggunaan DataLoader pada PyTorch untuk memuat data dalam batch dan melakukan transformasi data menggunakan PyTorch Transform. Terakhir, kode tersebut juga menunjukkan contoh pembuatan model MLP menggunakan PyTorch.

Activation Function

Pada kode yang diberikan tersebut merupakan contoh penggunaan beberapa fungsi aktivasi pada PyTorch, seperti softmax, sigmoid, tanh, ReLU, dan Leaky ReLU. Fungsi-fungsi aktivasi ini digunakan untuk memberikan aktivasi atau non-linearitas pada neuron dalam jaringan saraf. Selain itu, kode tersebut juga menunjukkan contoh pembuatan model MLP (Multilayer Perceptron) menggunakan PyTorch.Selanjutnya, kode tersebut juga menggambarkan penggunaan fungsi loss BCELoss (Binary Cross Entropy Loss) dan CrossEntropyLoss pada PyTorch. Fungsi-fungsi loss ini digunakan untuk mengukur perbedaan antara nilai prediksi dan nilai yang sebenarnya pada tugas klasifikasi biner dan multi-class. Fungsi-fungsi loss ini membantu dalam menentukan seberapa baik model kita dalam memprediksi kelas yang benar.

Plot Activations

Pada kode yang diberikan tersebut memuat definisi dan plot beberapa fungsi aktivasi yang sering digunakan dalam jaringan saraf. Bagian awal kode mengenai fungsi sigmoid, yang merupakan fungsi aktivasi populer dalam jaringan saraf. Fungsi sigmoid memiliki bentuk kurva yang menyerupai huruf "s". Selanjutnya, kode tersebut mendefinisikan dan memplot fungsi aktivasi seperti TanH, ReLU, Leaky ReLU, dan fungsi tangga. Fungsi-fungsi aktivasi ini digunakan untuk mengaktifkan neuron dalam jaringan saraf, memberikan non-linearitas pada output. Selain itu, dalam kode tersebut juga disebutkan tentang fungsi sigmoid biner, yang merupakan variasi dari fungsi sigmoid yang menghasilkan output biner..

Feedforward

Pada kode yang diberikan tersebut adalah implementasi neural network menggunakan PyTorch untuk melakukan klasifikasi pada dataset MNIST. Pertama, beberapa library seperti PyTorch, Torchvision, dan Matplotlib diimpor. Kemudian, parameter-parameter seperti ukuran input, ukuran hidden layer, dan jumlah kelas diinisialisasi.Selanjutnya, dataset MNIST diambil dan dilakukan preprocessing, seperti transformasi ke tensor dan normalisasi data. Setelah itu, model neural network dibuat menggunakan

kelas NeuralNet. Model tersebut terdiri dari dua layer linear dan satu layer ReLU.

Proses training dilakukan pada model menggunakan Cross Entropy Loss sebagai fungsi loss dan optimizer Adam untuk memperbarui parameter model. Setelah proses training selesai, model dievaluasi menggunakan data uji, dan akurasi dari model tersebut dihitung.Ini adalah contoh penggunaan PyTorch untuk melatih neural network pada dataset MNIST untuk tugas klasifikasi.

CNN

Pada kode yang diberikan tersebut merupakan implementasi Convolutional Neural Network (CNN) menggunakan PyTorch untuk melakukan klasifikasi gambar. CNN merupakan jenis neural network yang khusus dirancang untuk memproses data gambar. Kode tersebut terdiri dari beberapa bagian yang mencakup pengaturan device yang digunakan, hyperparameter seperti jumlah epoch, batch size, dan learning rate, pemrosesan dataset CIFAR-10, pembuatan model CNN, pengaturan loss function dan optimizer, pelatihan model, serta penghitungan akurasi pada data uji.

Selain itu, kode tersebut juga menggunakan library matplotlib untuk menampilkan beberapa gambar dari dataset CIFAR-10. Implementasi CNN dalam kode tersebut menggunakan pustaka PyTorch, yang merupakan pustaka deep learning berbasis tensor yang dioptimalkan untuk Python dan Torch.Kode tersebut memberikan contoh penggunaan PyTorch dalam membangun dan melatih CNN untuk tugas klasifikasi gambar pada dataset CIFAR-10.

Transfer Learning

Pada kode yang diberikan tersebut adalah implementasi pelatihan model jaringan saraf tiruan menggunakan kerangka kerja PyTorch untuk klasifikasi gambar pada dataset "hymenoptera_data". Pada kode tersebut, dilakukan konfigurasi transformasi data seperti pemangkasan acak, pembalikan horizontal, konversi gambar menjadi tensor, dan normalisasi. Kemudian, dataset gambar diatur menggunakan ImageFolder, dan dataloader digunakan untuk memuat data dalam batch. Selain itu, dilakukan pengaturan ukuran dataset dan definisi kelas-kelas yang ada dalam dataset.Perangkat yang digunakan (cuda atau cpu) ditentukan, dan fungsi imshow digunakan untuk menampilkan gambar dalam tensor.

Pelatihan model dilakukan menggunakan fungsi train_model. Dalam fungsi ini, model dilatih menggunakan metode mini-batch gradient descent. Pada setiap epoch, model dievaluasi pada data pelatihan dan data validasi, dihitung loss dan akurasi, serta dilakukan pembaharuan bobot model menggunakan optimizer. Selama pelatihan, penjadwalan learning rate dilakukan menggunakan lr_scheduler.

Terakhir, fungsi train_model dipanggil untuk dua skenario. Pertama, dengan menggunakan model resnet18 yang telah diinisialisasi dengan bobot pre-trained pada dataset ImageNet. Kemudian, dilakukan pelatihan pada model dengan mengubah layer fully connected pada model tersebut.Kode tersebut menunjukkan contoh penggunaan PyTorch dalam membangun dan melatih model jaringan saraf tiruan untuk tugas klasifikasi gambar pada dataset "hymenoptera_data".

Tensor Board

Pada kode yang diberikan tersebut merupakan implementasi pelatihan jaringan saraf tiruan menggunakan PyTorch untuk pengenalan angka pada dataset MNIST. Pada awal kode, dilakukan import module yang diperlukan dan pengaturan perangkat yang digunakan (GPU atau CPU). Kemudian, diatur beberapa parameter seperti ukuran input, ukuran tersembunyi, jumlah kelas, jumlah epoch, ukuran batch, dan learning rate.Dataset MNIST dibagi menjadi dataset pelatihan dan dataset pengujian, kemudian dimuat menggunakan torchvision.datasets.MNIST dan dibagi menjadi batch menggunakan torch.utils.data.DataLoader. Contoh data dari dataset pengujian ditampilkan menggunakan matplotlib.pyplot dan ditambahkan ke tensorboard menggunakan SummaryWriter.

Selanjutnya, didefinisikan model jaringan saraf tiruan menggunakan kelas NeuralNet. Model ini memiliki tiga lapisan: lapisan linier pertama, fungsi aktivasi ReLU, dan lapisan linier kedua. Fungsi loss (CrossEntropyLoss) dan optimizer (Adam) juga ditentukan untuk melatih model. Model dan data pelatihan dijalankan melalui beberapa epoch, dan pada setiap langkah, dilakukan perhitungan loss, pembaruan bobot, dan penghitungan akurasi. Hasil pelatihan juga ditambahkan ke tensorboard.

Setelah pelatihan selesai, dilakukan evaluasi akurasi pada dataset pengujian. Hasil prediksi dan label digunakan untuk menghitung akurasi dan menghasilkan kurva presisi-recall (PR curve) untuk setiap kelas menggunakan fungsi softmax dan SummaryWriter.Kode tersebut

menunjukkan contoh penggunaan PyTorch dalam melatih jaringan saraf tiruan untuk tugas pengenalan angka pada dataset MNIST.

Safe Load

Pada kode yang diberikan tersebut merupakan contoh implementasi PyTorch untuk membuat, menyimpan, dan memuat model serta parameter terkait. Di dalam kode tersebut, terdapat definisi model linear sederhana, penyimpanan model ke file, pengambilan model dari file, penyimpanan state_dict model, pengambilan state_dict model yang telah disimpan, pengaturan optimizer, pembuatan checkpoint, penyimpanan checkpoint ke file, pengambilan checkpoint dari file, pengaturan parameter model dan optimizer dari checkpoint, serta evaluasi model. Kode tersebut memberikan contoh praktis dalam penggunaan PyTorch untuk mengelola dan menggunakan model secara efisien.

Kesimpulan

PyTorch adalah sebuah framework machine learning berbasis Python yang populer dan sering digunakan dalam pengembangan dan penelitian kecerdasan buatan. PyTorch menyediakan alat-alat kuat untuk membangun, melatih, dan menerapkan model jaringan saraf, termasuk model berbasis konvolusi dan berulang. Kode yang diberikan mengilustrasikan penggunaan tensor dalam PyTorch, kemampuan autograd untuk diferensiasi otomatis, implementasi backpropagation untuk mengoptimasi parameter, penggunaan algoritma gradient descent, dan penggunaan loss function dan optimizer. Selain itu, kode juga mencakup implementasi regresi linier, logistik, dan contoh penggunaan dataset, transformasi, fungsi aktivasi, dan plot aktivasi.