Tugas Ujian Akhir Semester

Disusun Untuk Memenuhi Mata Kuliah Machine Learning

Dosen Pengampu :

Risman Adnan, Ph.D.

Disusun Oleh :

Puteri Andini Rosmadila

1103204014

PROGRAM STUDI S1 TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

BANDUNG

1. Deep Learning

Deep Learning adalah suatu teknik dalam bidang machine learning yang menggunakan neural network dengan banyak layer untuk memproses dan menganalisa data. Deep learning memungkinkan mesin untuk mempelajari representasi data secara hierarkis dan otomatis dari fitur yang sederhana hingga kompleks. Dengan begitu, deep learning dapat digunakan untuk mengatasi masalah kompleks seperti pengenalan suara dan gambar, deteksi objek, dan pemrosesan bahasa alami.Dalam Deep learning, data diproses melalui serangkaian layer yang terdiri dari node-node atau artificial neuron. Setiap layer akan mempelajari representasi data yang semakin kompleks. Layer pertama akan mempelajari representasi fitur-fitur sederhana seperti edge dan garis. Sedangkan layer terakhir akan mempelajari representasi fitur-fitur yang kompleks dan abstrak.

Deep Learning bekerja melalui jaringan syaraf yang mencoba meniru cara kerja otak manusia dengan menggabungkan input data, bobot, dan bias untuk mengenali, mengklasifikasikan, dan menggambarkan objek dalam data secara akurat. Jaringan syaraf Deep Learning terdiri dari beberapa layer node yang terhubung satu sama lain, dimana setiap layer membangun pada layer sebelumnya untuk memperbaiki dan mengoptimalkan prediksi atau kategorisasi. Proses komputasi melalui jaringan ini disebut propagasi maju (forward propagation), dimana layer input dan output disebut sebagai layer terlihat (visible layer), yang memproses data dan melakukan prediksi atau klasifikasi akhir.

2. PyTorch

PyTorch merupakan salah satu framework populer dalam pemrograman machine learning dan deep learning. PyTorch dikembangkan oleh Facebook AI Research dan menyediakan alat yang kuat dan fleksibel untuk mengimplementasikan dan melatih model-machine learning. PyTorch menyediakan berbagai komponen yang memfasilitasi proses machine learning. Salah satu komponen utamanya adalah Tensor, yaitu struktur data yang mirip dengan array multidimensi yang dapat digunakan untuk menyimpan dan memanipulasi data numerik. PyTorch juga menyediakan modul untuk membuat arsitektur model, fungsi-fungsi aktivasi, fungsi-fungsi kerugian (loss functions), dan optimizers yang digunakan untuk mengoptimasi model.

Salah satu fitur kunci PyTorch adalah kemampuannya dalam menghitung gradien secara otomatis menggunakan fitur yang disebut autograd. Autograd memungkinkan PyTorch untuk melacak dan menghitung gradien dari operasi yang dilakukan pada Tensor. Ini sangat berguna dalam proses pelatihan model, di mana gradien digunakan untuk mengoptimasi

parameter model. PyTorch juga mendukung akselerasi GPU, yang memungkinkan pengguna untuk melatih model dengan lebih cepat menggunakan unit pemrosesan grafis (GPU) yang kuat. Dengan memanfaatkan GPU, PyTorch dapat meningkatkan kinerja komputasi paralel dan mengurangi waktu pelatihan model.

3. Tensor Basic

Tensor adalah struktur data utama dalam PyTorch yang digunakan untuk melakukan komputasi. Tensors mirip dengan array multidimensi dan dapat memiliki berbagai dimensi, seperti scalar (tensor 0 dimensi), vector (tensor 1 dimensi), matrix (tensor 2 dimensi), dan tensor dengan dimensi yang lebih tinggi. Pada code yang saya masukkan,terdapat pembuatan tensor dengan menggunakan fungsi-fungsi seperti ‘ torch.empty() ‘, ‘ torch.rand() ’, ‘ torch.zeros()’, dan ‘ torch.tensor() ‘. PyTorch memiliki dukungan built-in untuk perhitungan diferensiasi otomatis. Hal ini sangat membantu dalam pemodelan dan pelatihan model deep learning. Saat menggunakan PyTorch, bisa menandai tensor dengan atribut ‘ requires_grad=True ‘. yang mana hal ini memberi tahu PyTorch untuk menghitung gradien (turunan) tensor tersebut terhadap suatu variabel, sehingga dapat digunakan dalam proses optimisasi model.

Dalam PyTorch, juga dapat mengubah bentuk tensor menggunakan metode ‘ view() ‘. Metode ini memungkinkan untuk mengubah dimensi tensor dengan tetap mempertahankan jumlah elemen yang sama. Kita dapat menentukan dimensi yang diinginkan atau menggunakan nilai -1 untuk dimensi yang ingin diinferensi secara otomatis. Selain itu, PyTorch memiliki integrasi yang baik dengan NumPy. Kita dapat dengan mudah mengonversi tensor menjadi array NumPy menggunakan metode ‘ .numpy() ’, dan sebaliknya, membuat tensor dari array NumPy menggunakan ‘ torch.from_numpy() ’. Perlu diingat bahwa jika tensor dan array NumPy berada di perangkat yang sama (CPU atau GPU), perubahan yang dilakukan pada salah satu akan mempengaruhi yang lain.

4. Autograd

Dalam kode yang saya masukkan menggunakan pustaka ‘ torch.autograd ’ dalam PyTorch yang memberikan dukungan untuk perhitungan diferensiasi otomatis. Dalam konteks deep learning, diferensiasi otomatis sangat penting karena memungkinkan untuk menghitung gradien (turunan) dari suatu fungsi atau model terhadap parameter yang ingin dioptimalkan. Salah satu fitur penting dari ‘ torch.autograd ’ adalah kemampuannya untuk melacak operasi yang dilakukan pada tensor dan menghasilkan graf komputasi

(computational graph). Graf komputasi ini merepresentasikan urutan operasi yang dilakukan pada tensor, dan setiap tensor memiliki atribut ‘grad_fn’ yang merujuk ke fungsi yang telah membuat tensor tersebut. Ini memungkinkan PyTorch untuk melakukan backpropagation secara otomatis dan menghitung gradien melalui rantai operasi menggunakan aturan rantai (chain rule).

Dalam kode tersebut, melakukan serangkaian operasi pada tensor ‘y‘, seperti perkalian dengan dirinya sendiri dan pengaliannya dengan konstanta 3. Kemudian, untuk menghitung rata-rata dari tensor ‘ z ‘. Setelah itu, memanggil metode ‘ backward() ’ pada ‘ z ‘ untuk melakukan backpropagation dan menghitung gradien dari ‘ z ‘ terhadap ‘ x ‘. Gradien ini kemudian dapat diakses melalui atribut ‘ x ‘ .grad, yang memberikan turunan parsial dari ‘ z ‘ terhadap ‘ x’.

Dengan menggunakan ‘ torch.autograd ’, bisa dengan mudah menghitung gradien secara otomatis tanpa perlu mengimplementasikan perhitungan gradien secara manual. Ini sangat membantu dalam proses pelatihan model deep learning, bertujuan untuk mengoptimalkan parameter model dengan menggunakan metode seperti stochastic gradient descent (SGD) atau algoritma optimasi lainnya. Dengan adanya fitur diferensiasi otomatis ini, kita dapat dengan mudah memperbarui parameter model berdasarkan gradien yang dihitung secara otomatis.

5. Backpropagation

Backpropagation memungkinkan kita untuk menghitung gradien dari fungsi kesalahan terhadap parameter-model, sehingga kita dapat memperbarui parameter-model tersebut menggunakan metode optimisasi seperti stochastic gradient descent (SGD).

Pertama, menginisialisasi tensor ‘x’ dan ‘y’ sebagai input dengan nilai 1.0 dan 2.0. Kemudian, mendefinisikan parameter ‘w’ dengan nilai awal 1.0 dan ‘requires_grad=True’ untuk mengindikasikan bahwa kita ingin mengoptimalkan parameter tersebut. Selanjutnya, melakukan proses forward pass dengan menghitung nilai prediksi ‘y_predicte’d menggunakan parameter ‘w’ dan input ‘x’. Untuk menghitung fungsi kesalahan dengan mengkuadratkan selisih antara nilai prediksi dan target ‘y’.

Setelah itu, melakukan proses backward pass dengan memanggil ‘loss.backward()’. Ini adalah langkah kunci dalam backpropagation yang menghitung gradien dari fungsi kesalahan terhadap semua tensor yang memiliki ‘requires_grad=True’. Dalam kasus ini, menghitung gradien ‘dLoss/dw’ dan menyimpannya dalam atribut grad dari ‘w’.

Setelah menghitung gradien, kita dapat memperbarui parameter ‘w’ menggunakan metode optimisasi seperti SGD. Dalam kode tersebut, menggunakan metode gradient descent dengan mengurangi sejumlah kecil (learning rate) dikalikan dengan gradien ‘w.grad’ dari nilai ‘w’. Penting untuk dicatat bahwa operasi pembaruan parameter ini tidak termasuk dalam graf komputasi yang dilacak oleh PyTorch, sehingga kita menggunakan ‘torch.no_grad()’ untuk memastikan operasi ini tidak mempengaruhi perhitungan gradien pada iterasi berikutnya.

Terakhir, sebelum melakukan iterasi selanjutnya dari proses forward dan backward, kita mengatur ulang gradien menjadi nol menggunakan ‘w.grad.zero_()’. Hal ini penting untuk memastikan bahwa gradien dikosongkan sehingga tidak terakumulasi dari iterasi sebelumnya.

6. Gradient Descent Manual

Pada kode, mengimplementasikan gradient descent secara manual untuk melakukan optimasi dalam regresi linear tanpa menggunakan library PyTorch. Gradient descent adalah metode optimasi yang digunakan untuk mencari nilai minimum atau maksimum dalam sebuah fungsi dengan memanfaatkan informasi gradien dari fungsi tersebut. Dalam kasus regresi linear ini, ingin menemukan nilai parameter ‘w’ yang meminimalkan fungsi loss (Mean Squared Error). Untuk itu, kita menggunakan pendekatan gradient descent untuk memperbarui nilai ‘w’ secara iteratif.

Proses yang dilakukan pada kode tersebut pertama menyiapkan data input ‘X’ dan target output ‘Y’ yang akan digunakan dalam regresi linear. Kemudian mendefinisikan parameter awal ‘w’ dengan nilai 0.0. Lalu mendefinisikan tiga fungsi yang diperlukan dalam regresi linear untuk melakukan prediksi, menghitung loss, dan menghitung gradien fungsi loss terhadap parameter ‘w’ yaitu ‘forward(x)’, ‘loss(y, y_pred)’, dan ‘gradient(x, y, y_pred)’yang kemudian dilakukan proses training dengan melakukan iterasi sebanyak jumlah epoch yang sudah ditentukan. Setelah proses training selesai, dapat melakukan prediksi pada nilai baru dengan memanggil ‘forward(x)’ menggunakan nilai ‘x’ yang diinginkan.

7. Gradient Descent Otomatis

Pada kode, menggunakan PyTorch untuk mengimplementasikan algoritma Gradient Descent secara otomatis dengan memanfaatkan fitur autograd. Autograd adalah modul

dalam PyTorch yang menyediakan perhitungan diferensiasi otomatis untuk komputasi tensor.

Proses yang dilakukan pada kode tersebut pertama menyiapkan data input ‘X’ dan target output ‘Y’ dalam bentuk tensor menggunakan ‘torch.tensor’. Kemudian mendefinisikan parameter awal ‘w’ dengan nilai 0.0 yang diberi atribut ‘requires_grad=True’ dengan tujuan bisa melacak operasi-operasi yang melibatkan parameter ini untuk perhitungan diferensiasi otomatis. Lalu mendefinisikan fungsi ‘forward(x)’ yang mengimplementasikan model regresi linear dengan menggunakan parameter ‘w’. Fungsi ini akan melakukan operasi perkalian antara ‘w’ dan ‘x’. Selanjutnya, mendefinisikan fungsi ‘loss(y, y_pred)’ yang menghitung Mean Squared Error (MSE) antara target ‘y’ dan prediksi ‘y_pred’. Kemudian dilakukan proses training dengan melakukan iterasi sebanyak jumlah epoch yang sudah ditentukan. Pada setiap beberapa epoch, mencetak informasi seperti nilai ‘w’ dan loss saat ini. Setelah proses training selesai, dapat melakukan prediksi pada nilai baru dengan memanggil ‘forward(x)’ menggunakan nilai ‘x’ yang diinginkan.

8. Loss dan Optimizer

Loss function digunakan untuk mengukur sejauh mana prediksi model mendekati nilai yang sebenarnya pada target yang diberikan. Pada kali ini, menggunakan Mean Squared Error (MSE) loss function yang merupakan metrik umum yang digunakan dalam masalah regresi.

Dalam implementasinya, PyTorch menyediakan modul ‘torch.nn’ yang menyediakan berbagai fungsi loss yang dapat digunakan dengan mudah. Dalam kasus ini, menggunakan fungsi ‘nn.MSELoss()’ untuk menghitung MSE loss antara prediksi ‘y_predicted’ dan target ‘Y’. MSE loss menghitung selisih kuadrat antara prediksi dan target, dan kemudian mengambil rata-rata dari seluruh data.

Hasil dari perhitungan loss function adalah sebuah nilai loss yang merepresentasikan seberapa baik model kita dalam mempelajari pola pada data pelatihan. Semakin kecil nilai loss, semakin baik model kita dalam melakukan prediksi yang mendekati target yang sebenarnya. Loss function menjadi acuan penting dalam proses pelatihan, di mana kita berusaha untuk meminimalkan nilai loss dengan memperbarui parameter model.

Optimizer digunakan untuk mengoptimasi parameter-parameter model agar loss yang dihasilkan semakin kecil. Pada kasus ini, menggunakan algoritma Stochastic Gradient Descent (SGD) sebagai optimizer yang populer dan sederhana. Pada kode menggunakan

modul ‘torch.optim’ untuk mendefinisikan optimizer. Dalam contoh ini, kita menggunakan ‘torch.optim.SGD’ untuk mendefinisikan SGD optimizer. Kita mengatur learning rate (‘learning_rate’) yang dapat disesuaikan dan parameter ‘w’ sebagai parameter yang akan dioptimasi.

Setelah menghitung gradien loss terhadap parameter-parameter model menggunakan algoritma backpropagation, optimizer akan melakukan update parameter berdasarkan gradien tersebut. Dalam kasus ini, menggunakan metode ‘optimizer.step()’ untuk memperbarui parameter ‘w’ berdasarkan gradien yang dihitung sebelumnya.

9. Model, Loss dan Optimizer

Langkah Pertama yang dilakukan pada kasus ini yaitu dengan mendefiniskan model yang akan digunakan. Pada kode digunakan modul ‘nn.Linear’ dari PyTorch untuk mendefinisikan model regresi linear. Model ini memiliki input_size dan output_size yang sesuai dengan jumlah fitur pada data masukan dan keluaran.

Kemudian untuk konstruksi loss dan optimizer mendefinisikan terlebih dahulu fungsi loss yang akan digunakan untuk mengukur sejauh mana prediksi model mendekati nilai target. Dalam contoh ini, kita menggunakan Mean Squared Error (MSE) loss function dari modul ‘nn.MSELoss’. Selain itu, kita juga mendefinisikan optimizer yang akan digunakan untuk mengoptimasi parameter model. Dalam kasus ini menggunakan optimizer SGD (Stochastic Gradient Descent) dari ‘modul torch.optim’.

Dan pada Training Loop, dilakukan beberapa langkah. Pertama, melakukan forward pass dengan memasukkan data masukan ke dalam model untuk menghasilkan prediksi. Selanjutnya, menghitung loss antara prediksi dan nilai target menggunakan fungsi loss yang telah ditentukan sebelumnya. Setelah itu, melakukan backward pass untuk menghitung gradien loss terhadap parameter-parameter model. Gradien ini akan digunakan oleh optimizer untuk melakukan update parameter. Terakhir, mengatur gradien menjadi nol dan melakukan langkah optimasi dengan memanggil optimizer.step(). Langkah-langkah ini diulang sebanyak n_iters kali.

Pada akhir training loop ini bisa menggunakan model yang telah dilatih untuk melakukan prediksi pada data baru dengan memanggil ‘model(X_test)’. Dalam kasus ini, mencoba memprediksi nilai f(5) dengan memasukkan nilai 5 ke dalam model yang telah dilatih.

10. Regresi Linier

Pada codingan, dalam melakukan pelatihan model regresi linear menggunakan PyTorch untuk memprediksi nilai target (y) berdasarkan fitur input (X). Pertama, data input dan target disiapkan menggunakan ‘datasets.make_regression’ dari scikit-learn, lalu diubah menjadi Tensor menggunakan ‘torch.from_numpy’. Model yang digunakan adalah regresi linear dengan satu fitur input dan satu output. Model tersebut didefinisikan menggunakan ‘nn.Linear’ dengan mengatur ukuran input dan output.

Selanjutnya, mendefinisikan loss function yang digunakan sebagai Mean Squared Error (MSE) melalui ‘nn.MSELoss’. Untuk melakukan optimasi parameter model, kita menggunakan optimizer Stochastic Gradient Descent (SGD) yang diimplementasikan menggunakan ‘torch.optim.SGD’. Learning rate juga ditentukan.

Pada Training Loop, melakukan perulangan sejumlah epoch yang telah ditentukan. Pada setiap epoch, dilakukan forward pass dengan memasukkan input ke dalam model untuk mendapatkan prediksi (y_predicted). Kemudian, kita menghitung loss antara prediksi dan target menggunakan loss function yang telah ditentukan. Selanjutnya, dilakukan backward pass dengan memanggil ‘loss.backward()’ untuk menghitung gradien loss terhadap parameter model. Gradien tersebut digunakan untuk melakukan update parameter model menggunakan optimizer dengan memanggil ‘optimizer.step()’. Sebelum melanjutkan ke iterasi berikutnya, gradien diatur menjadi nol menggunakan ‘optimizer.zero_grad()’. Setelah proses training loop selesai, hasil prediksi model diambil menggunakan ‘model(X).detach().numpy()’. Data asli dan hasil prediksi kemudian diplot menggunakan matplotlib untuk memvisualisasikan perbandingannya.

11. Regresi Logistik

Regresi logistik adalah sebuah pendekatan statistik yang digunakan untuk memprediksi probabilitas terjadinya suatu kejadian biner berdasarkan fitur-fitur input. Dalam kasus ini, menggunakan regresi logistik untuk melakukan klasifikasi binary pada dataset Breast Cancer.

Pertama, mempersiapkan data dengan memuat dataset Breast Cancer menggunakan ‘datasets.load_breast_cancer()’ dari scikit-learn. Selanjutnya, data tersebut dibagi menjadi data pelatihan (X_train, y_train) dan data uji (X_test, y_test) menggunakan ‘train_test_split’ dari scikit-learn. Kemudian, data pelatihan tersebut diubah menjadi Tensor menggunakan ‘torch.from_numpy’ dan disesuaikan dimensinya dengan format yang dibutuhkan oleh model.

Model yang digunakan adalah regresi logistik, yang didefinisikan dengan kelas Model yang merupakan subclass dari ‘nn.Module’. Model ini terdiri dari lapisan linear (‘nn.Linear’) yang menghubungkan fitur-fitur input ke keluaran dengan menggunakan fungsi aktivasi sigmoid. Sigmoid digunakan untuk menghasilkan prediksi dalam bentuk probabilitas yang berkisar antara 0 dan 1.

Dalam training loop, melakukan perulangan sebanyak jumlah epoch yang ditentukan. Pada setiap iterasi, dilakukan forward pass dengan memasukkan data pelatihan (X_train) ke model untuk menghasilkan prediksi (y_pred). Kemudian, menghitung loss antara prediksi dan target menggunakan BCELoss. Setelah itu, dilakukan backward pass dengan memanggil ‘loss.backward()’ untuk menghitung gradien loss terhadap parameter model. Gradien tersebut digunakan untuk melakukan update parameter model menggunakan optimizer dengan memanggil ‘optimizer.step()’. Sebelum melanjutkan ke iterasi berikutnya, gradien diatur menjadi nol menggunakan ‘optimizer.zero_grad()’.

12. DataLoader

PyTorch digunakan untuk memproses dataset menggunakan DataLoader. DataLoader merupakan salah satu fitur penting dalam PyTorch yang membantu dalam mengelola dan memuat data secara efisien dalam bentuk batch-batch kecil saat proses pelatihan model. Dalam machine learning, terutama dalam deep learning, seringkali memiliki dataset yang sangat besar. Memuat seluruh dataset sekaligus ke dalam memori dan memprosesnya secara simultan dapat menjadi tidak efisien dan memakan banyak sumber daya. Oleh karena itu, dataset dibagi menjadi batch-batch kecil yang lebih mudah diolah. ‘DataLoader’ memungkinkan kita untuk melakukan ini dengan mudah.

Pertama, menggunakan ‘WineDataset’ yang merupakan custom dataset yang kita definisikan sendiri. Di dalam kelas ‘WineDataset’, kita mengimplementasikan metode ‘__getitem__ dan __len__’ yang memungkinkan kita untuk mengakses sampel dataset berdasarkan indeks dan mendapatkan jumlah total sampel dalam dataset. Kemudian, kita menggunakan ‘DataLoader’ untuk memuat dataset ‘WineDataset’ dengan konfigurasi yang diinginkan seperti ‘batch_size’ dan ‘shuffle’. ‘DataLoader’ akan mengambil alih tugas membagi dataset menjadi batch-batch kecil dan memuatnya secara otomatis.

Selanjutnya, menggunakan ‘DataLoader’ untuk memuat dataset MNIST dari torchvision. Dalam hal ini, menggunakan dataset yang sudah tersedia dan dapat diunduh secara otomatis. Yang dilakukan hanya perlu menyediakan konfigurasi seperti ‘root’ untuk menentukan lokasi penyimpanan dataset, ‘train’ untuk menentukan apakah dataset yang

dimuat adalah dataset pelatihan, dan ‘transform’ untuk mengubah data menjadi bentuk yang sesuai untuk diproses oleh PyTorch.

13. Transformers

Pada codingan, menggunakan modul torchvision.transforms untuk menerapkan transformasi data pada PyTorch. Transformasi data merupakan proses mengubah atau memodifikasi data sebelum data tersebut dimasukkan ke dalam model atau saat proses pelatihan. Transformasi data ini berguna dalam berbagai hal seperti pra-pemrosesan data, augmentasi data, dan persiapan data sebelum dimasukkan ke dalam model.

Transformasi data dapat diterapkan pada berbagai jenis data, seperti gambar dalam format PIL, tensor, ndarray, atau data kustom. Modul torchvision.transforms menyediakan berbagai transformasi bawaan yang dapat digunakan, seperti perubahan ukuran gambar, rotasi, flipping, cropping, normalisasi, konversi tipe data, dan banyak lagi. Selain itu, kita juga dapat membuat transformasi kustom sesuai kebutuhan.

Dalam kode di atas, kita mendefinisikan kelas WineDataset yang merupakan dataset kustom. Pada konstruktor WineDataset, terdapat opsi untuk menerapkan transformasi data dengan menggunakan parameter transform. Transformasi data ini diterapkan pada setiap sampel saat metode getitem dipanggil. Jika transformasi ditentukan, sampel akan melalui proses transformasi sebelum dikembalikan.

Selanjutnya, kita mendefinisikan beberapa transformasi kustom seperti ToTensor dan MulTransform. Kelas ToTensor mengkonversi ndarray menjadi tensor, sedangkan Kelas MulTransform mengalikan input dengan suatu faktor. Dengan adanya transformasi kustom ini, kita dapat dengan mudah mengubah tipe data, melakukan augmentasi data, atau menerapkan operasi khusus pada setiap sampel.

14. Softmax dan Cross Entropy

Softmax adalah fungsi aktivasi yang umum digunakan dalam klasifikasi multi kelas. Fungsi ini memetakan output dari model ke dalam distribusi probabilitas dengan menjalankan fungsi eksponensial pada setiap elemen dan kemudian melakukan normalisasi dengan membagi hasilnya dengan jumlah eksponensial dari semua elemen. Softmax membantu dalam menginterpretasikan output model sebagai probabilitas kelas, di mana setiap elemen dalam output mewakili probabilitas model untuk setiap kelas yang mungkin.

Cross Entropy adalah sebuah metrik yang digunakan untuk mengukur perbedaan antara probabilitas prediksi model dan label aktual. Pada tugas klasifikasi, Cross Entropy Loss digunakan untuk mengestimasi seberapa baik model mengklasifikasikan data dengan membandingkan probabilitas prediksi dengan label yang benar. Dalam implementasi kode, mengimplementasikan fungsi 'cross_entropy' menggunakan NumPy dan juga menggunakan kelas 'nn.CrossEntropyLoss' dari PyTorch yang secara otomatis menggabungkan fungsi softmax dan negative log likelihood loss (NLLLoss). Cross Entropy Loss sangat penting dalam pelatihan model karena itu merupakan kriteria yang digunakan untuk mengoptimalkan parameter model melalui algoritma backpropagation.

Dalam perhitungan diferensiasi otomatis, Softmax dan Cross Entropy berperan penting. Softmax memastikan bahwa output model adalah probabilitas yang valid dan sesuai dengan distribusi probabilitas. Kemudian, Cross Entropy Loss digunakan untuk mengukur perbedaan antara prediksi model dan label aktual, yang menjadi dasar untuk mengoptimalkan model melalui algoritma backpropagation. Dengan menggunakan implementasi Softmax dan Cross Entropy dari PyTorch, dapat memanfaatkan kemampuan diferensiasi otomatis yang kuat dari library ini untuk menghitung gradien loss terhadap parameter model secara otomatis, sehingga memudahkan pelatihan model dan peningkatan kinerjanya.

15. Activation Function

Pada codingan mencakup implementasi beberapa fungsi aktivasi yang umum digunakan dalam jaringan saraf, yaitu softmax, sigmoid, tanh, relu, dan leaky relu, menggunakan PyTorch. Fungsi aktivasi ini berperan penting dalam memperkenalkan non-linearitas ke dalam jaringan saraf, yang memungkinkan model untuk mempelajari hubungan yang kompleks antara input dan output.

Softmax merupakan fungsi aktivasi yang umum digunakan pada lapisan output dari jaringan saraf dalam masalah klasifikasi multikelas. Fungsi ini menghasilkan distribusi probabilitas dari output model, di mana setiap elemen dalam output mewakili probabilitas model untuk setiap kelas yang mungkin. Dalam implementasi kode di atas, kita menggunakan fungsi softmax dari PyTorch baik melalui pemanggilan langsung 'torch.softmax' maupun dengan menggunakan kelas 'nn.Softmax'. Kedua pendekatan tersebut menghasilkan output yang sama.

Sigmoid merupakan fungsi aktivasi yang umum digunakan dalam lapisan tersembunyi atau lapisan output dalam masalah klasifikasi biner. Fungsi ini memetakan input ke dalam

rentang antara 0 dan 1, yang dapat diinterpretasikan sebagai probabilitas kelas positif. Dalam kode di atas, kita menggunakan fungsi sigmoid dari PyTorch baik melalui pemanggilan langsung ‘torch.sigmoid’ maupun dengan menggunakan kelas ‘nn.Sigmoid’. Kedua pendekatan tersebut menghasilkan output yang sama.

Tanh (tangens hiperbolik) merupakan fungsi aktivasi yang sering digunakan dalam jaringan saraf. Fungsi ini memetakan input ke dalam rentang antara -1 dan 1, sehingga mempertahankan sifat non-linear tetapi juga dapat menghasilkan nilai negatif. Dalam kode di atas, kita menggunakan fungsi tanh dari PyTorch baik melalui pemanggilan langsung ‘torch.tanh’ maupun dengan menggunakan kelas ‘nn.Tanh’. Kedua pendekatan tersebut menghasilkan output yang sama.

Relu (Rectified Linear Unit) merupakan fungsi aktivasi yang umum digunakan dalam lapisan tersembunyi dalam jaringan saraf. Fungsi ini memetakan input negatif menjadi 0 dan mempertahankan nilai input positif. Dalam implementasi kode di atas, kita menggunakan fungsi relu dari PyTorch baik melalui pemanggilan langsung ‘torch.relu’ maupun dengan menggunakan kelas ‘nn.ReLU’. Kedua pendekatan tersebut menghasilkan output yang sama.

Leaky Relu merupakan variasi dari fungsi relu yang mengatasi masalah "dying ReLU" yang dapat terjadi ketika neuron menjadi tidak responsif terhadap input negatif. Fungsi ini memetakan input negatif menjadi nilai yang sangat kecil (misalnya, dengan faktor 0,01) dan mempertahankan nilai input positif. Dalam kode di atas, kita menggunakan fungsi leaky relu dari PyTorch baik melalui pemanggilan langsung ‘F.leaky_relu’ maupun dengan menggunakan kelas ‘nn.LeakyReLU. Kedua pendekatan tersebut menghasilkan output yang sama.

16. Plot Activation

Pada kodingan adalah implementasi dari beberapa fungsi aktivasi yang umum digunakan dalam jaringan saraf menggunakan NumPy dan Matplotlib. Fungsi aktivasi yang diimplementasikan meliputi sigmoid, tangens hiperbolik (tanh), Rectified Linear Unit (ReLU), Leaky ReLU, dan fungsi Binary Step.

Fungsi sigmoid diimplementasikan menggunakan persamaan matematika sigmoid(x) = 1 / (1 + exp(-x)). Fungsi ini memetakan input ke dalam rentang antara 0 dan 1. Pada grafik yang dihasilkan, rentang input dari -10 hingga 10 diuji dan fungsi sigmoid menghasilkan kurva yang menunjukkan pertumbuhan eksponensial dan saturasi ke nilai 1.

Fungsi tangens hiperbolik (tanh) diimplementasikan menggunakan persamaan matematika tanh(x) = 2 * sigmoid(2x) - 1. Fungsi ini memetakan input ke dalam rentang antara -1 dan 1. Kurva yang dihasilkan oleh fungsi tanh menunjukkan sifat simetris dan saturasi ke nilai -1 dan 1.

Fungsi ReLU (Rectified Linear Unit) diimplementasikan menggunakan persamaan matematika relu(x) = max(0, x). Fungsi ini memetakan input negatif menjadi 0 dan mempertahankan nilai input positif. Pada grafik yang dihasilkan, fungsi ReLU menghasilkan kurva yang linier di bagian positif dan datar di bagian negatif.

Fungsi Leaky ReLU diimplementasikan menggunakan persamaan matematika leakyrelu(x) = max(0.1x, x). Fungsi ini serupa dengan ReLU, namun memiliki kemiringan kecil pada bagian negatif (0.1x) untuk mengatasi masalah "dying ReLU" yang dapat terjadi ketika neuron menjadi tidak responsif terhadap input negatif. Pada grafik yang dihasilkan, fungsi Leaky ReLU menghasilkan kurva yang linier di bagian positif dan memiliki kemiringan kecil di bagian negatif.

Fungsi Binary Step diimplementasikan menggunakan persamaan matematika bstep(x) = 1 if x >= 0, else 0. Fungsi ini memetakan input ke dalam dua nilai diskret, yaitu 1 atau 0. Pada grafik yang dihasilkan, fungsi Binary Step menunjukkan ambang batas yang terjadi pada nilai 0.

17. Feed Forward

Pada codingan mengimplementasi sebuah jaringan saraf menggunakan PyTorch untuk melakukan klasifikasi digit menggunakan dataset MNIST. Pada awalnya, mengatur hyperparameter seperti ukuran input, ukuran lapisan tersembunyi, jumlah kelas, jumlah epoch, ukuran batch, dan learning rate. Kemudian, memuat dataset MNIST dan mengubahnya menjadi tensor menggunakan torchvision.transforms.ToTensor(). Dataset ini terdiri dari gambar-gambar digit tulisan tangan dan label yang sesuai.

Selanjutnya, membangun model jaringan saraf dengan satu lapisan tersembunyi menggunakan fungsi aktivasi ReLU. Lapisan output tidak menggunakan fungsi aktivasi atau softmax. Model ini akan dilatih menggunakan CrossEntropyLoss sebagai fungsi loss dan optimizer Adam untuk mengoptimasi parameter-model.

Setelah mengatur model dan fungsi loss, melakukan pelatihan dengan melakukan loop pada setiap epoch dan setiap batch dalam dataset pelatihan. Pada setiap iterasi, gambar dan label dikirimkan ke perangkat yang sesuai, dan proses forward pass dilakukan untuk menghasilkan prediksi. Loss dihitung berdasarkan perbandingan prediksi dengan label

yang sebenarnya, dan kemudian dilakukan proses backward pass untuk menghitung gradien dan mengoptimasi parameter-model menggunakan optimizer.