# TECHNICAL REPORT

# DEEP LEARNING WITH PYTORCH


Disusun Oleh:

Lutfiana Erniasari

1103204029

**Program Studi Teknik Komputer**

**Fakultas Teknik Elektro**

**Universitas Telkom**

**2023**

## 1. Pendahuluan

Dalam beberapa dekade terakhir, Machine Learning telah menjadi bidang yang sangat penting dalam pengembangan teknologi. Dalam era di mana data semakin melimpah, kemampuan untuk menggali wawasan dan informasi berharga dari data tersebut menjadi krusial. Machine Learning memberikan algoritma dan metode yang memungkinkan komputer untuk belajar dari data dan menghasilkan prediksi atau keputusan yang cerdas tanpa harus secara eksplisit diprogram. Salah satu pendekatan dalam Machine Learning yang telah merevolusi cara kita memahami dan memanfaatkan data adalah Deep Learning.

Deep Learning, sebagai sub-bidang dalam Machine Learning, telah mencapai kemajuan yang luar biasa dalam beberapa tahun terakhir. Konsep Deep Learning didasarkan pada struktur jaringan saraf tiruan yang terinspirasi oleh cara kerja otak manusia. Dengan menggunakan jaringan saraf tiruan yang mendalam dan kompleks, Deep Learning mampu mempelajari representasi yang semakin abstrak dan kompleks dari data. Ini memberikan kemampuan untuk mengatasi tantangan pemrosesan data yang sangat kompleks dan menghasilkan pemahaman yang lebih dalam tentang pola, fitur, dan hubungan dalam data yang dianalisis.

Dengan kombinasi Machine Learning dan Deep Learning, kita dapat menjawab pertanyaan-pertanyaan yang rumit, mengungkap pola tersembunyi, dan menghasilkan wawasan yang bernilai dari data yang semakin kompleks. Laporan ini akan memberikan pemahaman yang komprehensif tentang Deep Learning dalam konteks Machine Learning, serta mengilustrasikan potensi dan aplikasi nyata melalui studi kasus dan contoh penggunaan yang relevan.

## 2. Dasar Deep Learning

Deep Learning telah menjadi bagian yang sangat kuat dalam bidang machine learning yang telah mengubah paradigma dengan memungkinkan komputer untuk belajar dan membuat keputusan cerdas dari jumlah data yang sangat besar. Pada dasarnya, deep learning menggunakan struktur jaringan saraf yang rumit yang terinspirasi oleh struktur dan fungsi otak manusia. Pendekatan ini memungkinkan model deep learning untuk mempelajari representasi hierarkis dari data, mengekstrak fitur-fitur yang rumit dan abstrak yang sulit ditangkap oleh algoritma machine learning tradisional.

Selanjutnya, kami akan membahas proses pelatihan model deep learning. Proses ini melibatkan optimisasi parameter model melalui teknik yang disebut backpropagation, di mana gradien dihitung dan digunakan untuk memperbarui bobot jaringan. Kami juga akan menjelaskan algoritma optimisasi populer seperti stochastic gradient descent, serta teknik untuk menghindari overfitting, seperti regularisasi dan dropout. Selain itu, kami akan menjelajahi berbagai jenis arsitektur deep learning, termasuk convolutional neural networks (CNN) untuk data gambar dan video. Memahami kelebihan dan keterbatasan dari masing-masing arsitektur ini sangat penting dalam memilih model yang sesuai untuk tugas-tugas spesifik.

Deep learning telah mencapai pencapaian yang luar biasa dalam tugas-tugas seperti deteksi objek, klasifikasi gambar, terjemahan bahasa, dan sintesis suara. Dengan memahami prinsip-prinsip dasar deep learning, para peneliti, insinyur, dan praktisi di bidang machine learning dapat mengoptimalkan kekuatan teknologi ini. Laporan ini bertujuan untuk memberikan pemahaman yang kokoh tentang deep learning, sehingga individu dapat memanfaatkan kemampuannya dan berkontribusi dalam kemajuan kecerdasan buatan dan solusi berbasis data di berbagai bidang.

## 3. Dasar Pytorch

PyTorch, sebagai salah satu framework Deep Learning yang terkenal, telah menarik perhatian luas di bidang Machine Learning karena fleksibilitas, kemampuan komputasinya, dan antarmuka pemrograman yang user-friendly. Dibangun dengan bahasa pemrograman Python, PyTorch memberikan pengalaman yang lancar dan mudah dipahami dalam pengembangan dan implementasi model Deep Learning.

Konsep dasar PyTorch menjadi dasar penting untuk memahami dan menerapkan model Deep Learning. Dimulaidengan memperkenalkan konsep Tensor, yang merupakan struktur data utama dalam PyTorch. Tensor adalah array multidimensi yang digunakan untuk menyimpan dan memanipulasi data numerik yang diperlukan dalam pelatihan dan inferensi model. Selanjutnya, adafitur unik PyTorch dalam komputasi grafis. PyTorch mengadopsi pendekatan "define-by-run", di mana pengguna dapat secara dinamis mendefinisikan aliran komputasi saat model dieksekusi. Fleksibilitas ini memberikan kebebasan kepada pengembang untuk melakukan eksperimen dan iterasi yang lebih efektif selama pelatihan model.

Selain itu, juga ada autograd yang merupakan komponen penting dalam PyTorch. Autograd memungkinkan perhitungan diferensiasi otomatis dari operasi yang

didefinisikan pada Tensor, yang sangat berguna dalam menghitung gradien selama pelatihan model Deep Learning. Perhitungan gradien yang otomatis ini memainkan peran penting dalam memperbarui parameter model melalui proses optimisasi.

## 4. Analisa Code

a. Tensor

Pada awal program, terdapat inisialisasi tensor x dan y dengan menggunakan fungsi torch.tensor(). Tensor x berisi nilai [1, 2, 3, 4], sedangkan tensor y berisi nilai [2, 4, 6, 8]. Kedua tensor ini menggunakan tipe data float32, yang sering digunakan dalam komputasi numerik. Selanjutnya, kita melihat inisialisasi parameter model dengan tensor w. Tensor w memiliki nilai awal 0.0 dan menggunakan tipe data float32. Perhatikan juga bahwa kita menentukan requires_grad=True, yang berarti kita ingin menghitung gradien terhadap tensor ini selama proses pelatihan.

Setelah itu, kita mendefinisikan dua fungsi: forward(x) dan loss(y, y_predicted). Fungsi forward mengalikan tensor input x dengan parameter w untuk melakukan prediksi. Sedangkan fungsi loss menghitung mean squared error (MSE) antara nilai yang diprediksi (y_predicted) dan nilai yang sebenarnya (y). Selanjutnya, program mencetak hasil prediksi model sebelum dilakukan pelatihan dengan input 5 menggunakan fungsi forward(5). Pada tahap pelatihan, kita mendefinisikan hyperparameter seperti learning_rate dan n_iterations. Learning_rate menentukan seberapa besar langkah pembaharuan parameter, sedangkan n_iterations menentukan berapa kali kita akan melatih model.

Dalam loop pelatihan, program melakukan langkah-langkah berikut secara berulang:

- Melakukan prediksi y_pred dengan menggunakan fungsi forward dan input x.
- Menghitung loss l menggunakan fungsi loss dengan nilai sebenarnya y dan nilai prediksi y_pred.
- Melakukan backward propagation dengan memanggil l.backward() untuk menghitung gradien loss terhadap parameter w.
- Memperbarui nilai w dengan menggunakan metode gradient descent: w = w - learning_rate * w.grad.

- Mengatur gradien w menjadi nol dengan w.grad.zero_() untuk mencegah akumulasi gradien dari iterasi sebelumnya.

Selama proses pelatihan, program mencetak nilai w dan loss pada setiap 10 epoch untuk melihat perkembangan pelatihan. Terakhir, program mencetak hasil prediksi model setelah dilakukan pelatihan dengan input 5 menggunakan fungsi forward(5). Program ini memberikan contoh bagaimana tensor digunakan dalam konteks deep learning. Tensor x dan y digunakan sebagai data input dan target, sementara tensor w digunakan sebagai parameter yang diperbarui selama pelatihan. Dengan melakukan iterasi pelatihan dan memperbarui parameter w berdasarkan gradien loss, kita dapat mengoptimalkan model untuk melakukan prediksi yang semakin mendekati nilai yang sebenarnya.

b. Autograd

Baris program di atas merupakan inisialisasi tensor weights dengan menggunakan fungsi torch.ones(). Tensor weights memiliki ukuran 4 dan requires_grad=True, yang berarti kita ingin menghitung gradien terhadap tensor ini selama proses pelatihan. Tensor weights diinisialisasi dengan nilai 1 untuk setiap elemennya. Hal ini berarti tensor ini awalnya memiliki bobot yang seragam untuk digunakan dalam suatu model atau algoritma. Dalam konteks deep learning, tensor weights sering kali digunakan sebagai parameter yang akan diperbarui selama proses pelatihan model.

Dengan requires_grad=True, PyTorch akan secara otomatis melacak operasi yang dilakukan pada tensor weights dan menghitung gradien terhadapnya saat melakukan backward propagation. Ini memungkinkan kita untuk melakukan optimisasi model menggunakan algoritma seperti gradient descent atau algoritma pelatihan lainnya.

Dalam program lebih lanjut, kita dapat memanipulasi dan menggunakan tensor weights dalam berbagai operasi dan komputasi yang terkait dengan deep learning, seperti mengalikan dengan input, memperbarui nilai berdasarkan gradien, atau menghitung loss. Tensor weights akan berperan penting dalam mengoptimalkan model dan meningkatkan performa prediksi.

c. Backpropagation

Program tersebut menggambarkan langkah-langkah untuk melakukan perhitungan prediksi menggunakan tensor dan menghitung gradien melalui proses backward pass menggunakan PyTorch. Pada awal program, tensor x dan y diinisialisasi dengan nilai 1.0 dan 2.0 masing-masing. Tensor ini digunakan sebagai

input dan target untuk prediksi. Selanjutnya, terdapat inisialisasi tensor w dengan nilai 1.0 dan requires_grad=True. Dengan requires_grad=True, PyTorch akan melacak operasi yang melibatkan tensor w dan menghitung gradien terhadapnya selama proses pelatihan. Kemudian, dilakukan perhitungan prediksi y_hat dengan mengalikan tensor w dengan tensor x. Prediksi ini kemudian digunakan untuk menghitung loss, yang merupakan selisih kuadrat antara nilai prediksi y_hat dan nilai target y.

Program mencetak nilai loss yang dihasilkan. Pada tahap ini, loss belum digunakan untuk mengoptimasi model, hanya sebagai contoh perhitungan loss. Selanjutnya, dilakukan backward pass dengan memanggil loss.backward(). Ini akan menghitung gradien loss terhadap tensor yang memiliki requires_grad=True, dalam hal ini tensor w. Program mencetak nilai gradien w menggunakan w.grad. Nilai ini menunjukkan perubahan yang harus dilakukan terhadap tensor w untuk mengoptimalkan loss. Gradien ini diperoleh melalui perhitungan gradien menggunakan metode autograd dari PyTorch. Dengan menggunakan gradien ini, kita dapat memperbarui parameter model, seperti tensor w, menggunakan algoritma optimisasi seperti gradient descent untuk meningkatkan performa prediksi model.

d. Gradient Descent

- Dengan Numpy

  Pada awal program, terdapat inisialisasi array NumPy x dan y dengan nilai [1, 2, 3, 4] dan [2, 4, 6, 8] secara berturut-turut. Array ini digunakan sebagai input dan target untuk prediksi. Selanjutnya, terdapat inisialisasi nilai awal w dengan 0.0. Nilai ini merepresentasikan parameter yang akan diperbarui selama proses pelatihan.

  Kemudian, terdapat definisi fungsi forward(x) yang mengembalikan hasil perkalian w dengan x. Fungsi ini digunakan untuk melakukan prediksi. Selanjutnya, terdapat definisi fungsi loss(y, y_predicted) yang menghitung nilai loss, yaitu selisih kuadrat antara nilai prediksi y_predicted dan nilai target y. Terdapat juga definisi fungsi gradient(x, y, y_predicted) yang menghitung gradien dari loss terhadap parameter w. Fungsi ini menggunakan metode dot product untuk melakukan perhitungan.

  Program mencetak prediksi sebelum pelatihan dengan memanggil fungsi forward(5). Ini memberikan prediksi f(5) dengan menggunakan parameter w saat ini. Selanjutnya, dilakukan proses pelatihan dengan menggunakan

algoritma gradient descent. Terdapat variabel learning_rate yang menentukan seberapa besar langkah yang diambil saat mengoptimalkan parameter. Variabel n_iterations menentukan jumlah iterasi pelatihan.

Dalam setiap iterasi, dilakukan perhitungan prediksi y_pred dengan memanggil fungsi forward(x). Selanjutnya, dihitung loss l dengan memanggil fungsi loss(y, y_pred). Gradien dw dihitung dengan memanggil fungsi gradient(x, y, y_pred). Kemudian, nilai parameter w diperbarui dengan mengurangi learning_rate dikali dw. Program mencetak nilai parameter w dan loss pada setiap iterasi pelatihan.

Setelah proses pelatihan selesai, program mencetak prediksi setelah pelatihan dengan memanggil fungsi forward(5). Ini memberikan prediksi f(5) dengan menggunakan parameter w yang telah diperbarui setelah pelatihan.

- Dengan Torch Tensor

  Pada awal program, terdapat inisialisasi tensor x dan y dengan nilai [1, 2, 3, 4] dan [2, 4, 6, 8] secara berturut-turut. Tensor ini digunakan sebagai input dan target untuk prediksi.

  Selanjutnya, terdapat inisialisasi nilai awal tensor w dengan 0.0. Nilai ini merepresentasikan parameter yang akan diperbarui selama proses pelatihan. Dengan requires_grad=True, PyTorch akan melacak operasi yang melibatkan tensor w dan menghitung gradien terhadapnya selama proses pelatihan. Kemudian, terdapat definisi fungsi forward(x) yang mengembalikan hasil perkalian w dengan x. Fungsi ini digunakan untuk melakukan prediksi. Selanjutnya, terdapat definisi fungsi loss(y, y_predicted) yang menghitung nilai loss, yaitu selisih kuadrat antara nilai prediksi y_predicted dan nilai target y. Fungsi ini menggunakan mean() untuk menghitung rata-rata dari seluruh elemen dalam tensor.

  Program mencetak prediksi sebelum pelatihan dengan memanggil fungsi forward(5). Ini memberikan prediksi f(5) dengan menggunakan parameter w saat ini. Selanjutnya, dilakukan proses pelatihan dengan menggunakan algoritma gradient descent. Terdapat variabel learning_rate yang menentukan seberapa besar langkah yang diambil saat mengoptimalkan parameter. Variabel n_iterations menentukan jumlah iterasi pelatihan.

Dalam setiap iterasi, dilakukan perhitungan prediksi y_pred dengan memanggil fungsi forward(x). Selanjutnya, dihitung loss l dengan memanggil fungsi loss(y, y_pred). Selanjutnya, dilakukan backward pass dengan memanggil l.backward(). Ini akan menghitung gradien loss terhadap tensor yang memiliki requires_grad=True, dalam hal ini tensor w.

Setelah backward pass, gradien yang dihitung disimpan di w.grad. Kemudian, dilakukan pembaruan parameter w dengan mengurangi learning_rate dikali w.grad. Pembaruan ini dilakukan dengan menggunakan torch.no_grad() agar tidak ada perhitungan gradien yang terjadi saat pembaruan parameter.

Setelah pembaruan, gradien w direset menjadi nol menggunakan w.grad.zero_(). Hal ini diperlukan agar gradien tidak terakumulasi setiap iterasi. Program mencetak nilai parameter w dan loss pada setiap iterasi pelatihan, hanya saat epoch % 10 == 0 untuk mengurangi jumlah output yang ditampilkan. Setelah proses pelatihan selesai, program mencetak prediksi setelah pelatihan dengan memanggil fungsi forward(5). Ini memberikan prediksi f(5) dengan menggunakan parameter w yang telah diperbarui setelah pelatihan.

e. Training Pipeline

Program di atas merupakan contoh implementasi linear regression menggunakan PyTorch. Linear regression digunakan untuk memodelkan hubungan linier antara variabel input x dan variabel target y.

Pada awal program, terdapat inisialisasi tensor x dan y dengan nilai [[1], [2], [3], [4]] dan [[2], [4], [6], [8]] secara berturut-turut. Tensor ini digunakan sebagai input dan target untuk pelatihan model. Selanjutnya, terdapat inisialisasi tensor x_test dengan nilai [5]. Tensor ini digunakan sebagai input untuk melakukan prediksi setelah pelatihan. Kemudian, dilakukan penghitungan jumlah sampel (n_samples) dan jumlah fitur (n_features) dari tensor x menggunakan x.shape. Nilai ini digunakan untuk menginisialisasi input_size dan output_size pada model.

Selanjutnya, didefinisikan kelas LinearRegression yang merupakan turunan dari nn.Module. Kelas ini memiliki satu layer linear (nn.Linear) yang menghubungkan input_dim dengan output_dim. Model LinearRegression diinisialisasi dengan input_size dan output_size yang telah ditentukan sebelumnya. Program mencetak prediksi sebelum pelatihan dengan memanggil model(x_test).item(). Ini memberikan prediksi f(5) menggunakan model sebelum dilakukan pelatihan.

Selanjutnya, dilakukan proses pelatihan. Terdapat variabel learning_rate yang menentukan seberapa besar langkah yang diambil saat mengoptimalkan parameter model. Variabel n_iterations menentukan jumlah iterasi pelatihan.

Dalam setiap iterasi, dilakukan perhitungan prediksi y_pred dengan memanggil model(x). Selanjutnya, dihitung loss l dengan memanggil fungsi nn.MSELoss() yang menghitung mean squared error antara y_pred dan y. Selanjutnya, dilakukan backward pass dengan memanggil l.backward(). Ini akan menghitung gradien loss terhadap parameter model. Selanjutnya, dilakukan pembaruan parameter model menggunakan optimizer. Pembaruan ini dilakukan dengan memanggil optimizer.step() yang secara otomatis mengoptimalkan parameter berdasarkan gradien yang telah dihitung sebelumnya. Setelah pembaruan, gradien parameter direset menjadi nol menggunakan optimizer.zero_grad(). Hal ini diperlukan agar gradien tidak terakumulasi setiap iterasi.

Program mencetak nilai parameter w dan loss pada setiap iterasi pelatihan, hanya saat epoch % 10 == 0 untuk mengurangi jumlah output yang ditampilkan. Setelah proses pelatihan selesai, program mencetak prediksi setelah pelatihan dengan memanggil model(x_test).item(). Ini memberikan prediksi f(5) menggunakan model setelah dilakukan pelatihan.

f. Linear Regression

Program di atas merupakan contoh implementasi regresi linear menggunakan PyTorch dengan menggunakan dataset yang dibuat secara acak menggunakan `make_regression` dari `sklearn.datasets`.

Pada awal program, dilakukan pembuatan dataset dengan menggunakan `make_regression` dengan jumlah sampel (n_samples) sebanyak 100, jumlah fitur (n_features) sebanyak 1, tingkat kebisingan (noise) sebesar 20, dan random_state=1. Dataset ini terdiri dari pasangan nilai x dan y. Selanjutnya, dilakukan konversi dataset menjadi tensor menggunakan `torch.from_numpy`. Variabel x dan y diinisialisasi dengan nilai tensor yang sesuai. Selain itu, bentuk tensor y diubah menjadi (n_samples, 1) menggunakan `y = y.view(y.shape[0], 1)`.

Program mencetak jumlah sampel (n_samples) dan jumlah fitur (n_features) dari tensor x menggunakan x.shape. Selanjutnya, dilakukan inisialisasi model dengan menggunakan nn.Linear. Model ini memiliki satu layer linear yang menghubungkan input_size (jumlah fitur) dengan output_size (1). Selanjutnya, dilakukan inisialisasi optimizer dengan menggunakan torch.optim.SGD. Optimizer ini digunakan untuk

mengoptimalkan parameter model selama proses pelatihan dengan learning_rate sebesar 0.01.

Program juga melakukan inisialisasi criterion (loss function) dengan menggunakan nn.MSELoss. Criterion ini digunakan untuk menghitung mean squared error antara prediksi model (y_pred) dan target (y). Selanjutnya, dilakukan proses pelatihan dengan melakukan iterasi sebanyak num_epochs (100 kali). Pada setiap epoch, dilakukan forward pass dengan memanggil model(x) untuk mendapatkan prediksi y_pred. Selanjutnya, dihitung loss dengan memanggil criterion(y_pred, y) untuk menghitung error antara prediksi dan target. Setelah itu, dilakukan backward pass dengan memanggil loss.backward(). Ini akan menghitung gradien loss terhadap parameter model.

Selanjutnya, dilakukan pembaruan parameter model dengan memanggil optimizer.step(). Hal ini akan mengoptimalkan parameter berdasarkan gradien yang telah dihitung sebelumnya. Gradien parameter direset menjadi nol menggunakan optimizer.zero_grad(). Hal ini diperlukan agar gradien tidak terakumulasi setiap iterasi. Pada setiap epoch yang kelipatan 10, program mencetak nilai loss.

Setelah proses pelatihan selesai, program melakukan plot data asli (x_np, y_np) dengan titik merah (ro) dan plot hasil prediksi model dengan garis biru (b). Dengan demikian, program ini mengilustrasikan proses pelatihan regresi linear menggunakan PyTorch dengan dataset acak.

g. Logistic Regression

Program di atas merupakan contoh implementasi regresi logistik menggunakan PyTorch pada dataset Breast Cancer Wisconsin (Diagnostic) dari `sklearn.datasets`. Pada awal program, dataset Breast Cancer dimuat menggunakan `datasets.load_breast_cancer()`. Data x dan label y diambil dari dataset tersebut.

Selanjutnya, dilakukan pemisahan dataset menjadi data latih (x_train, y_train) dan data uji (x_test, y_test) menggunakan `train_test_split` dari `sklearn.model_selection`. Data uji sebesar 20% dari total dataset dengan random_state=42. Selanjutnya, data diukur menggunakan skala standar (StandardScaler) menggunakan `StandardScaler` dari `sklearn.preprocessing`. Data latih dan data uji diubah menjadi tipe tensor menggunakan `torch.from_numpy` dan diubah menjadi tipe data float32.

Selanjutnya, bentuk tensor y_train dan y_test diubah menjadi (n_samples, 1) menggunakan `y_train = y_train.view(y_train.shape[0], 1)` dan `y_test =

y_test.view(y_test.shape[0], 1)`. Program mendefinisikan model regresi logistik dengan menggunakan kelas `LogisticRegression` yang merupakan turunan dari `nn.Module`. Model ini memiliki satu layer linear yang menghubungkan input_size (jumlah fitur) dengan output_size (1) dengan menggunakan fungsi aktivasi sigmoid.

Selanjutnya, dilakukan inisialisasi optimizer dengan menggunakan `torch.optim.SGD`. Optimizer ini digunakan untuk mengoptimalkan parameter model selama proses pelatihan dengan learning_rate sebesar 0.01. Program juga melakukan inisialisasi criterion (loss function) dengan menggunakan `nn.BCELoss`. Criterion ini digunakan untuk menghitung binary cross-entropy loss antara prediksi model (y_pred) dan target (y_train).

Selanjutnya, dilakukan proses pelatihan dengan melakukan iterasi sebanyak num_epochs (100 kali). Pada setiap epoch, dilakukan forward pass dengan memanggil model(x_train) untuk mendapatkan prediksi y_pred. Selanjutnya, dihitung loss dengan memanggil criterion(y_pred, y_train) untuk menghitung error antara prediksi dan target. Setelah itu, dilakukan backward pass dengan memanggil loss.backward(). Ini akan menghitung gradien loss terhadap parameter model. Selanjutnya, dilakukan pembaruan parameter model dengan memanggil optimizer.step(). Hal ini akan mengoptimalkan parameter berdasarkan gradien yang telah dihitung sebelumnya. Gradien parameter direset menjadi nol menggunakan optimizer.zero_grad(). Hal ini diperlukan agar gradien tidak terakumulasi setiap iterasi. Pada setiap epoch yang kelipatan 10, program mencetak nilai loss.

Setelah proses pelatihan selesai, program melakukan evaluasi pada data uji. Prediksi pada data uji (y_pred) diambil dengan menggunakan model dan diubah menjadi nilai biner dengan menggunakan fungsi `round()`. Akurasi dihitung dengan membandingkan prediksi biner dengan label y_test. Akurasi ini mencerminkan sejauh mana model dapat memprediksi kelas dengan benar.

Dengan demikian, program ini mengilustrasikan proses pelatihan dan evaluasi regresi logistik menggunakan PyTorch pada dataset Breast Cancer.

h. Dataset dan Dataloader

Program di atas merupakan contoh implementasi dataset kustom (`WineDataset`) dan penggunaan `DataLoader` untuk memuat data dalam batch selama proses pelatihan. Pada awal program, kelas `WineDataset` didefinisikan sebagai turunan dari kelas `Dataset` dari PyTorch. Dalam metode `__init__`, data dari file 'wine.csv' dimuat menggunakan `np.loadtxt`. Kolom pertama (label) dipisahkan dan disimpan

dalam `self.y`, sedangkan kolom kedua hingga terakhir (fitur) disimpan dalam `self.x`. Jumlah sampel (`self.n_samples`) dihitung berdasarkan ukuran dataset.

Metode `__getitem__` mengembalikan fitur dan label untuk indeks tertentu dalam dataset. Metode `__len__` mengembalikan jumlah total sampel dalam dataset. Selanjutnya, objek dataset dibuat menggunakan `WineDataset()`. Ini akan memuat dataset dari file 'wine.csv' dan mengorganisir data menjadi pasangan fitur-label.

Objek dataset digunakan untuk membuat `DataLoader`. DataLoader digunakan untuk memuat data dalam batch selama proses pelatihan. Pada contoh ini, batch_size diatur menjadi 4, artinya setiap batch akan berisi 4 sampel. Parameter shuffle diatur sebagai True untuk mengacak urutan data dalam setiap epoch. Parameter num_workers diatur sebagai 2 untuk menggunakan 2 proses paralel untuk memuat data.

Setelah DataLoader dibuat, program melakukan iterasi sebanyak num_epochs (2 kali) untuk melakukan proses pelatihan. Dalam setiap epoch, program melakukan iterasi sebanyak n_iterasions untuk memuat data dalam batch menggunakan DataLoader. Dalam setiap iterasi, batch data (inputs) dan label (labels) diambil. Pada contoh ini, setiap 5 iterasi, program mencetak pesan yang menunjukkan nomor epoch, total epoch, nomor iterasi, total iterasi, dan nilai inputs. Dengan menggunakan DataLoader, program dapat dengan mudah memuat data dalam batch selama proses pelatihan. Hal ini berguna ketika jumlah data sangat besar sehingga tidak dapat dimuat secara keseluruhan ke dalam memori.

i. Data Transform

Program di atas merupakan contoh penggunaan transformasi data pada dataset kustom (`WineDataset`) menggunakan kelas transformasi `ToTensor` dan `MultiTransform`. Kelas `WineDataset` memiliki parameter opsional `transform` yang digunakan untuk menerapkan transformasi pada setiap pasangan fitur-label dalam dataset. Dalam metode `__getitem__`, setelah mengambil sampel, transformasi diterapkan pada sampel menggunakan `self.transform` jika ada.

Kelas `ToTensor` adalah transformasi yang mengubah data dari NumPy array menjadi tensor PyTorch menggunakan `torch.from_numpy`. Ini diterapkan pada fitur dan label dalam metode `__call__`. Kelas `MultiTransform` adalah transformasi kustom yang mengalikan fitur dengan faktor tertentu. Ini diterapkan pada fitur dalam metode `__call__`.

Pada contoh pertama, `WineDataset` diinisialisasi dengan `transform=None`, yang berarti tidak ada transformasi yang diterapkan. Setelah mengambil data pertama dari dataset, fitur dan label dicetak dan tipe datanya diperiksa. Pada contoh kedua, transformasi didefinisikan menggunakan `torchvision.transforms.Compose`. Transformasi `ToTensor` dan `MultiTransform` dikomposisikan menjadi satu transformasi. Kemudian, `WineDataset` diinisialisasi dengan `transform=composed`. Setelah mengambil data pertama dari dataset, fitur dan label dicetak dan tipe datanya diperiksa. Kali ini, fitur akan mengalami transformasi `MultiTransform` yang mengalikan fitur dengan faktor 2.

Dengan menggunakan transformasi, dataset dapat dengan mudah dimodifikasi sesuai dengan kebutuhan, seperti mengubah tipe data, menerapkan augmentasi data, atau melakukan transformasi kustom lainnya sebelum digunakan dalam proses pelatihan.

j. Softmax

Program di atas menggambarkan implementasi fungsi softmax menggunakan NumPy dan PyTorch. Pada fungsi softmax menggunakan NumPy, nilai input x akan dieksponensialkan menggunakan `np.exp(x)`. Kemudian, hasil eksponensial tersebut akan dinormalisasi dengan membaginya dengan jumlah eksponensial dari semua elemen x menggunakan `np.sum(np.exp(x), axis=0)`. Akhirnya, hasil normalisasi tersebut akan menjadi output dari fungsi softmax.

Pada implementasi menggunakan NumPy, nilai input x adalah array NumPy `[2.0, 1.0, 0.1]`. Setelah menjalankan fungsi softmax numpy, hasil softmax akan dicetak. Pada implementasi menggunakan PyTorch, nilai input x didefinisikan sebagai tensor PyTorch `[2.0, 1.0, 0.1]`. Kemudian, fungsi `torch.softmax` digunakan dengan parameter `dim=0` untuk melakukan softmax pada dimensi pertama (dimensi kolom). Hasil softmax akan dicetak.

Kedua implementasi menghasilkan output yang sama yaitu probabilitas yang dinormalisasi secara sesuai dengan fungsi softmax.

k. Cross-Entropy

Program di atas menggambarkan perhitungan loss menggunakan cross-entropy loss dengan NumPy dan PyTorch. Pada implementasi menggunakan NumPy, fungsi `cross_entropy` mengambil argumen `actual` dan `predicted`. Loss dihitung dengan rumus `-np.sum(actual * np.log(predicted))`, di mana `actual` adalah nilai target

yang sebenarnya dan `predicted` adalah nilai prediksi. Kemudian, dua contoh loss dihitung menggunakan fungsi tersebut dan dicetak.

Pada implementasi menggunakan PyTorch, kita menggunakan kelas `nn.CrossEntropyLoss()` sebagai fungsi loss. Kemudian, kita menyiapkan nilai target `y` dan nilai prediksi `y_pred_good` dan `y_pred_bad` menggunakan tensor PyTorch. Loss dihitung dengan memanggil `loss(y_pred, y)`. Kembali, dua contoh loss dihitung dan dicetak. Selain itu, kita menggunakan fungsi `torch.max()` untuk mendapatkan prediksi kelas dari nilai prediksi. Hasil prediksi dari `y_pred_good` dan `y_pred_bad` dicetak. Kedua implementasi memberikan hasil yang sama yaitu loss dari perhitungan cross-entropy dan prediksi kelas yang diperoleh dari nilai prediksi.

l. Activision Function

Kedua opsi tersebut merupakan implementasi dari model neural network dengan dua layer, menggunakan modul `nn.Module` dari PyTorch. Opsi pertama menggunakan modul ReLU (`nn.ReLU()`) sebagai fungsi aktivasi pada layer pertama dan modul Sigmoid (`nn.Sigmoid()`) pada layer kedua. Dalam metode `forward()`, output dari layer pertama dijalankan melalui ReLU, kemudian output dari layer kedua dijalankan melalui Sigmoid.

Opsi kedua menggunakan fungsi ReLU (`torch.relu()`) secara langsung pada layer pertama dan fungsi Sigmoid (`torch.sigmoid()`) pada layer kedua. Fungsi aktivasi diaplikasikan langsung pada output layer. Kedua opsi tersebut memperlihatkan cara yang berbeda dalam menggunakan fungsi aktivasi ReLU dan Sigmoid. Pemilihan opsi yang tepat tergantung pada kebutuhan dan kompleksitas model yang akan dibangun.

m. Feedforward

Pada bagian awal, kita mengatur konfigurasi perangkat dengan menggunakan `torch.device` untuk memilih perangkat CUDA jika tersedia, jika tidak, menggunakan perangkat CPU. Selanjutnya, kita mendefinisikan hyperparameter yang akan digunakan dalam model. `input_size` adalah ukuran input (jumlah fitur) yang akan diberikan ke model, `hidden_size` adalah ukuran lapisan tersembunyi (hidden layer) dalam model, `num_classes` adalah jumlah kelas yang ada dalam dataset MNIST, `num_epochs` adalah jumlah epoch yang akan dilakukan selama pelatihan, `batch_size` adalah ukuran batch yang akan digunakan dalam pelatihan,

dan `learning_rate` adalah laju pembelajaran (learning rate) yang akan digunakan dalam optimisasi model.

Selanjutnya, kita mempersiapkan dataset MNIST dengan menggunakan `torchvision.datasets.MNIST`. Kita membagi dataset menjadi data pelatihan (`train_dataset`) dan data pengujian (`test_dataset`). Kemudian, kita menggunakan `torch.utils.data.DataLoader` untuk memuat dataset dengan ukuran batch yang ditentukan (`batch_size`) dan mengatur mode pengacakan (`shuffle`) pada data pelatihan.

Setelah itu, kita mencetak informasi tentang dimensi sampel dan label dari data pelatihan untuk memastikan bahwa pemrosesan data dilakukan dengan benar. Selanjutnya, kita menggunakan perulangan untuk mencetak 6 gambar dari dataset MNIST sebagai contoh.

Selanjutnya, kita membuat objek model `NeuralNet` dengan menginisialisasi objek tersebut menggunakan hyperparameter yang telah ditentukan. Kemudian, kita memindahkan model ke perangkat CUDA jika tersedia. Kemudian, kita mendefinisikan fungsi loss yang akan digunakan dalam pelatihan, yaitu `nn.CrossEntropyLoss()`. Selanjutnya, kita mendefinisikan optimizer yang akan digunakan untuk mengoptimasi parameter model, yaitu `torch.optim.Adam` dengan laju pembelajaran (`learning_rate`) yang telah ditentukan.

Setelah itu, kita melakukan pelatihan model dengan melakukan iterasi pada setiap epoch dan batch data dari `train_loader`. Pada setiap iterasi, kita memindahkan data gambar ke perangkat CUDA (jika tersedia) dan mengirimkannya ke model untuk menghitung output. Selanjutnya, kita menghitung loss menggunakan fungsi loss yang telah ditentukan sebelumnya. Setelah menghitung loss, kita melakukan langkah optimisasi dengan menghapus gradien sebelumnya menggunakan `optimizer.zero_grad()`, melakukan backpropagation dengan memanggil `loss.backward()` untuk menghitung gradien, dan melakukan update parameter model dengan memanggil `optimizer.step()`.

Selama pelatihan, kita mencetak informasi loss pada setiap 100 langkah dengan menggunakan pernyataan kondisional `(i+1) % 100 == 0`. Setelah selesai pelatihan, kita melakukan evaluasi model pada dataset pengujian (`test_loader`). Kita menghitung jumlah prediksi yang benar dan jumlah total sampel yang ada. Dari perhitungan tersebut, kita menghitung akurasi dan mencetaknya sebagai hasil akhir.

n. Convolutional Neural Networks (CNN)

Pertama-tama, dilakukan konfigurasi perangkat dengan menggunakan `torch.device` untuk menentukan apakah perangkat CUDA (GPU) tersedia atau tidak. Jika CUDA tersedia, perangkat ditetapkan sebagai "cuda", jika tidak, perangkat ditetapkan sebagai "cpu". Selanjutnya, ditentukan beberapa parameter hiper seperti `num_epochs` (jumlah epoch), `batch_size` (ukuran batch), dan `learning_rate` (tingkat pembelajaran).

Selanjutnya, dilakukan pengaturan transformasi data untuk dataset CIFAR-10 menggunakan `transforms.Compose`. Transformasi ini mencakup konversi ke tensor dan normalisasi. Dataset pelatihan dan dataset pengujian CIFAR-10 diunduh dan dimuat menggunakan `torchvision.datasets.CIFAR10`, dengan menyertakan transformasi yang telah ditentukan sebelumnya. Data loader dibuat untuk dataset pelatihan dan pengujian menggunakan `torch.utils.data.DataLoader`, dengan mengatur ukuran batch dan pengacakan data.

Selanjutnya, didefinisikan daftar kelas yang digunakan dalam dataset CIFAR-10. Model ConvNet dibuat menggunakan kelas `ConvNet`, kemudian model tersebut dipindahkan ke perangkat yang ditentukan sebelumnya menggunakan `.to(device)`. Kriteria loss untuk pelatihan model ditentukan sebagai `nn.CrossEntropyLoss()`. Optimizer yang digunakan adalah Stochastic Gradient Descent (SGD), yang didefinisikan dengan `torch.optim.SGD`, dengan mengambil parameter-model dan learning rate.

Selanjutnya, dilakukan pelatihan model dengan iterasi melalui jumlah epoch yang telah ditentukan. Setiap langkah dalam satu epoch, data gambar dan label diperoleh dari data loader. Gambar dan label dipindahkan ke perangkat yang ditentukan. Output dari model diperoleh dengan memasukkan gambar ke dalam model. Loss dihitung dengan membandingkan output dengan label menggunakan kriteria loss yang telah ditentukan. Gradien dihitung dan parameter-model diperbarui menggunakan optimizer.

Selama pelatihan, setiap 2000 langkah, dicetak pesan yang menampilkan informasi tentang epoch, langkah saat ini, jumlah total langkah, dan loss yang diperoleh. Setelah pelatihan selesai, dilakukan evaluasi model pada data pengujian. Prediksi dilakukan pada data gambar dan akurasi keseluruhan dihitung dengan membandingkan prediksi dengan label yang benar. Selain itu, akurasi untuk setiap

kelas juga dihitung. Akurasi keseluruhan dan akurasi untuk setiap kelas dicetak sebagai output.

o. CNN

Dalam kode yang diberikan, digunakan `SummaryWriter` dari `torch.utils.tensorboard` untuk membuat objek penulis TensorBoard dengan nama "runs/ants-bees". Kemudian, didefinisikan variabel `device` yang akan menggunakan perangkat CUDA jika tersedia, jika tidak, akan menggunakan perangkat CPU. Selanjutnya, didefinisikan rata-rata (`mean`) dan standar deviasi (`std`) untuk normalisasi gambar.

Digunakan `data_transforms` untuk mendefinisikan transformasi data untuk set pelatihan dan set validasi. Transformasi ini mencakup pemotongan acak, pembalikan horizontal, konversi ke tensor, dan normalisasi menggunakan rata-rata dan standar deviasi yang telah ditentukan sebelumnya. Direktori data `'ants-bees-dataset'` ditentukan, serta daftar set yang terdiri dari `'train'` dan `'val'`. Kemudian, `torchvision.datasets.ImageFolder` digunakan untuk membuat dataset gambar dari direktori dengan menggunakan transformasi yang sesuai.

`torch.utils.data.DataLoader` digunakan untuk membuat loader data dengan batch_size 4, pengacakan, dan jumlah pekerja 0 (tanpa penggunaan multiprocessing) untuk setiap set dalam `dataloaders`. Selanjutnya, ukuran dataset untuk setiap set dalam `dataset_sizes` dihitung dan daftar kelas diperoleh menggunakan `image_datasets['train'].classes`.

Digunakan `iter(dataloaders['train'])` untuk membuat iterator dari objek `dataloaders['train']`. Kemudian, `next(examples)` digunakan untuk mengambil contoh data dan target pertama dari iterator tersebut. Dilakukan loop `for` untuk menampilkan 4 gambar contoh dalam bentuk subplot menggunakan `plt.subplot(2, 2, i+1)`. `plt.imshow` digunakan untuk menampilkan gambar dengan menggunakan cmap 'gray' untuk memastikan tampilan dalam skala abu-abu.

Selanjutnya, `torchvision.utils.make_grid` digunakan untuk membuat tampilan grid dari contoh_data dengan ukuran batch_size (4 dalam hal ini). Kemudian, gambar grid tersebut ditambahkan ke tensorboard menggunakan `writer.add_image('ants-bees_images', img_grid)`.Terakhir, penulisan tensorboard ditutup dengan `writer.close()`.

p. Transfer Learning

Berikut adalah sebuah fungsi `train_model` yang akan melatih model menggunakan pendekatan pembelajaran yang ditentukan. Fungsi ini akan mengevaluasi dan mengoptimalkan model untuk beberapa epoch dengan menggunakan kriteria tertentu. Fungsi ini menerima parameter sebagai berikut:

- `model`: model yang akan dilatih
- `criterion`: kriteria yang digunakan untuk menghitung loss
- `optimizer`: optimizer yang digunakan untuk mengoptimalkan model
- `scheduler`: scheduler yang digunakan untuk mengatur laju pembelajaran
- `num_epochs`: jumlah epoch yang akan dilatih (default: 20)

Pada awalnya, fungsi ini akan menyimpan waktu mulai (`since`), salinan terbaik dari bobot model (`best_model_wts`), dan akurasi terbaik yang telah dicapai (`best_acc`). Selanjutnya, dilakukan loop untuk setiap epoch. Di dalam loop ini, dilakukan loop lagi untuk setiap fase, yaitu `'train'` dan `'val'`. Model akan diatur ke mode pelatihan atau evaluasi sesuai dengan fase yang sedang berjalan. Dalam setiap fase, dilakukan loop untuk setiap batch dalam loader data sesuai dengan fase yang sedang berjalan. Input dan label akan dipindahkan ke perangkat yang ditentukan (`device`). Dengan menggunakan `torch.set_grad_enabled`, dinyalakan atau dimatikan perhitungan gradien tergantung pada fase yang sedang berjalan. Output dari model dihitung dan prediksi dilakukan dengan menggunakan `torch.max`. Kemudian, loss dihitung menggunakan `criterion`.

Jika fase adalah `'train'`, optimasi dilakukan dengan melakukan pembatalan gradien (`optimizer.zero_grad()`), melakukan backpropagation (`loss.backward()`), dan melakukan langkah optimasi (`optimizer.step()`). Statistik seperti running_loss dan running_corrects diupdate untuk setiap batch. Setelah loop batch selesai, dilakukan penghitungan loss dan akurasi rata-rata untuk epoch saat ini. Informasi ini dicetak dan juga ditambahkan ke tensorboard menggunakan `writer.add_scalar`.

Jika fase adalah `'val'` dan akurasi epoch saat ini lebih baik dari `best_acc`, maka `best_acc` diperbarui dan bobot model terbaik disalin. Setelah loop epoch selesai, penulisan tensorboard ditutup dengan `writer.close()`. Waktu yang dibutuhkan untuk pelatihan dicetak, serta akurasi terbaik yang dicapai. Terakhir, bobot model yang memberikan akurasi terbaik dimuat ke dalam model, dan model tersebut dikembalikan.

q. CNN ResNet18 Model

Dalam potongan kode di atas, kita menggunakan arsitektur ResNet-18 yang telah dilatih sebelumnya sebagai model awal (`models.resnet18(pretrained=True)`). Kemudian, kita mengatur `requires_grad` menjadi `False` untuk semua parameter dalam model tersebut untuk memastikan bahwa parameter-parameter ini tidak akan diperbarui selama pelatihan. Selanjutnya, kita mendapatkan jumlah fitur masukan (`num_features`) dari lapisan fully connected terakhir dalam model (`model.fc.in_features`). Kita kemudian mengganti lapisan fully connected terakhir dengan `nn.Linear(num_features, 2)` untuk menyesuaikan output dengan jumlah kelas yang ingin kita prediksi (dalam contoh ini, 2 kelas: 'ants' dan 'bees').

Selanjutnya, model dipindahkan ke perangkat yang ditentukan (`device`) menggunakan metode `.to(device)`. Kita mendefinisikan kriteria loss sebagai `nn.CrossEntropyLoss()` dan menggunakan optimizer SGD dengan laju pembelajaran 0.001 (`optim.SGD(model.parameters(), lr=0.001)`). Selanjutnya, kita mendefinisikan scheduler dengan `lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)`. Scheduler ini akan mengurangi laju pembelajaran (`gamma`) sebesar faktor 0.1 setiap 7 epoch (`step_size`).

Terakhir, kita memanggil fungsi `train_model` dengan model, kriteria, optimizer, dan scheduler yang telah ditentukan, serta jumlah epoch yang diinginkan (20 dalam contoh ini). Hasil model yang dilatih kemudian ditugaskan kembali ke variabel `model`.

r. Tensorboard + Save Model + Load Mode

TensorBoard adalah alat visualisasi yang disediakan oleh TensorFlow untuk membantu dalam memahami, memantau, dan menganalisis model dan data. Ini menyediakan antarmuka berbasis web yang memungkinkan melihat grafik komputasi, histogram, statistik, dan lainnya yang terkait dengan model.

Untuk menggunakan TensorBoard dengan PyTorch, dapat menggunakan paket `tensorboard` yang disertakan dengan PyTorch. Dapat mengimpor `SummaryWriter` dari `torch.utils.tensorboard` dan membuat objek `SummaryWriter` dengan menentukan direktori log yang akan digunakan untuk menyimpan catatan TensorBoard. Misalnya, `writer = SummaryWriter("logs")`.

Dapat menggunakan metode `add_scalar()` pada objek `SummaryWriter` untuk mencatat skalar seperti loss atau akurasi selama pelatihan. Misalnya, `writer.add_scalar('training loss', loss, step)` akan mencatat loss pelatihan pada langkah tertentu.

Untuk mengamati visualisasi yang dihasilkan oleh TensorBoard, dapat menjalankan TensorBoard dengan menggunakan perintah `%tensorboard --logdir logs`, di mana `'logs'` adalah direktori log yang telah ditentukan sebelumnya. Setelah menjalankan perintah ini, TensorBoard akan berjalan di notebook dan dapat mengakses antarmuka web TensorBoard untuk melihat visualisasi.

Selain itu, dapat menggunakan perintah `torch.save()` untuk menyimpan model PyTorch ke file. Misalnya, dengan menggunakan perintah `torch.save(model, FILE)`, dapat menyimpan model ke file dengan nama `'model.pth'`. Kemudian, dapat memuat model tersebut dari file yang disimpan menggunakan perintah `model = torch.load(FILE)`. Setelah memuat model, dapat mengaturnya ke mode evaluasi dengan menggunakan metode `.eval()` agar siap digunakan untuk melakukan prediksi pada data baru.

Dengan demikian, dapat menggunakan TensorBoard untuk memvisualisasikan pelatihan model, menyimpan model ke file, dan memuat model dari file untuk digunakan dalam inferensi atau prediksi pada data baru.

**5. Analisa Hasil**

Dalam beberapa program di atas, kita melihat penggunaan dataset yang berbeda untuk melatih model-model machine learning. Pada program pertama, digunakan dataset MNIST yang berisi gambar digit tulisan tangan. Model neural network sederhana digunakan untuk mengklasifikasikan gambar-gambar tersebut. Program kedua menggunakan dataset CIFAR-10 yang berisi gambar-gambar objek dalam 10 kelas yang berbeda. Model Convolutional Neural Network (CNN) digunakan dalam program ini. Pada program ketiga, dataset yang digunakan adalah ants-bees yang berisi gambar semut dan lebah. Dalam kasus ini, model yang digunakan adalah model transfer learning dengan arsitektur ResNet-18 yang telah dilatih pada dataset ImageNet.

Seluruh program melibatkan langkah-langkah umum dalam pelatihan model, seperti membagi data menjadi batch menggunakan DataLoaders, menggunakan optimizer dan criterion untuk mengoptimalkan model, dan melakukan iterasi pada setiap batch untuk melakukan forward pass, backward pass, dan optimisasi parameter. Selain itu, program-program ini juga menggunakan metode pengukuran kinerja seperti perhitungan loss dan akurasi.

Dalam proses pelatihan, TensorBoard digunakan untuk memvisualisasikan metrik seperti loss dan akurasi. Ini memungkinkan kita untuk memantau dan menganalisis kinerja model selama pelatihan. Di akhir pelatihan, model yang telah dilatih dapat

disimpan dalam file untuk digunakan dalam tahap inferensi atau prediksi pada data baru.

Secara keseluruhan, program-program ini memberikan gambaran tentang langkah-langkah yang diperlukan dalam pelatihan model machine learning dan deep learning. Dari mempersiapkan data, merancang model, melatih model dengan optimisasi, hingga menganalisis kinerja model menggunakan TensorBoard, semua langkah ini merupakan bagian penting dari pengembangan dan penerapan model dalam berbagai tugas machine learning.

**Referensi:**

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