NAMA: RAI BAROKAH UTARI

NIM: 1103200066

Technical Report Deep Learning

Pada UAS kali ini saya mengambil topik technical report deep learning yang merupakan kerangka kerja yang populer untuk pemrosesan data dan pembelajaran mendalam. Deep learning adalah salah satu cabang dari pembelajaran mesin yang menggunakan arsitektur jaringan syaraf tiruan untuk mempelajari pola dan fitur yang kompleks dari data. Adapun tujuan dari pengambilan topik tersebut yaitu untuk memberikan pemahaman yang lebih luas khususnya bagi saya untuk memperdalam topik yang saya ambil.

A. TENSOR

Dalam pengembangan aplikasi yang berbasis deep learning, PyTorch menjadi salah satu kerangka kerja yang populer. PyTorch menggunakan struktur data yang disebut tensor sebagai fondasi utama untuk melakukan pemrosesan dan pembelajaran dalam konteks deep learning. Tensor dalam PyTorch dapat merepresentasikan data dalam bentuk multidimensional, yang memiliki berbagai dimensi seperti skalar, vektor, matriks, atau tensor dengan dimensi yang lebih tinggi. Dengan menggunakan tensor, PyTorch memungkinkan manipulasi data yang efisien dan operasi matematika yang kompleks dalam pemodelan dan pelatihan jaringan saraf.

Adapun yang dapat saya jelaskan mengenai kodingan bagian tensor yaitu:

- Import library: Dua library yang diimpor adalah `torch` dan `numpy` yang digunakan untuk operasi tensor dan manipulasi array.
- Inisialisasi tensor kosong:
  - torch.empty(size)` digunakan untuk membuat tensor kosong dengan ukuran yang ditentukan.
  - torch.empty(shape)` menghasilkan tensor kosong dengan bentuk yang ditentukan. Misalnya, `torch.empty((2, 3))` menghasilkan tensor berbentuk (2, 3).
- Inisialisasi tensor acak:
  - torch.rand(size)` digunakan untuk membuat tensor acak dengan nilai dalam rentang [0, 1) dan ukuran yang ditentukan. Misalnya, `torch.rand((5, 3))` menghasilkan tensor berukuran (5, 3) dengan nilai acak.
  - torch.randn(size)` menghasilkan tensor dengan elemen acak yang diambil dari distribusi normal standar.
- Inisialisasi tensor nol:
  - `torch.zeros(size)` digunakan untuk membuat tensor berisi nol dengan ukuran yang ditentukan.
- Menampilkan ukuran dan tipe data tensor:
  - `tensor.size()` digunakan untuk mengambil ukuran tensor.
  - `tensor.dtype` digunakan untuk mengambil tipe data tensor.
- Konversi antara NumPy array dan tensor PyTorch:
  - `torch.from_numpy(numpy_array)` digunakan untuk mengonversi array NumPy menjadi tensor PyTorch.
  - `tensor.numpy()` digunakan untuk mengonversi tensor PyTorch menjadi array NumPy.
- Operasi tensor dasar:
  - `torch.add(tensor1, tensor2)` digunakan untuk menjumlahkan dua tensor secara elemen-wise.
  - `torch.sub(tensor1, tensor2)` digunakan untuk mengurangkan dua tensor secara elemen-wise.
  - `torch.mul(tensor1, tensor2)` digunakan untuk mengalikan dua tensor secara elemen-wise.
  - `torch.div(tensor1, tensor2)` digunakan untuk membagi dua tensor secara elemen-wise.

- Indeks dan slicing tensor:
  - `tensor[:, index]` digunakan untuk mengambil kolom dengan indeks tertentu dari tensor.
  - `tensor[index, :]` digunakan untuk mengambil baris dengan indeks tertentu dari tensor.
  - `tensor[index1, index2]` digunakan untuk mengakses elemen pada posisi tertentu dalam tensor.
- Operasi pada GPU (jika tersedia):
  - `torch.cuda.is_available()` digunakan untuk memeriksa ketersediaan GPU.
  - `tensor.to(device)` digunakan untuk memindahkan tensor ke perangkat (GPU atau CPU) yang ditentukan.
  - `tensor.to("cpu")` digunakan untuk memindahkan tensor dari GPU ke CPU.

B. Auto Grad

Pertama, terdapat penggunaan tensor dalam PyTorch dengan menambahkan fitur `requires_grad=True`. Fitur ini memungkinkan penghitungan gradien tensor terhadap operasi yang dilakukan pada tensor tersebut. Misalnya, pada contoh pertama, tensor `x` memiliki fitur `requires_grad=True`, sehingga saat tensor `y` dibentuk dengan operasi `x + 2`, PyTorch secara otomatis melacak operasi tersebut dan memungkinkan perhitungan gradien terhadap `x`. Hal ini berguna dalam proses pelatihan model deep learning.

Pada bagian kedua kode, terdapat contoh perhitungan gradien dengan memanfaatkan operasi lanjutan pada tensor. Setelah tensor `y` dibentuk dari tensor `x`, dilakukan operasi pengkuadratan dan perkalian dengan 3 pada tensor `z`. Kemudian, dilakukan perhitungan mean dari tensor `z`. Dengan memanggil metode `backward()` pada tensor `z`, gradien dari tensor `z` terhadap `x` dapat dihitung menggunakan algoritma backpropagation. Hasil gradien ini tersimpan dalam `x.grad`, yang menunjukkan perubahan yang dibutuhkan pada tensor `x` untuk meminimalkan nilai dari `z`.

Pada bagian terakhir kode, diperlihatkan contoh penggunaan metode `detach()` dan penggunaan `torch.no_grad()`. Metode `detach()` digunakan untuk memutuskan koneksi antara tensor dengan sejarah perhitungan gradiennya. Misalnya, pada contoh ketiga, tensor `b` dihasilkan dengan menggunakan operasi yang melibatkan tensor `a`, namun tensor `b` tidak memiliki sejarah gradien yang terkait dengan `a`. Hal ini berguna ketika ingin menghentikan pelacakan gradien pada bagian tertentu dari model atau saat melakukan evaluasi model di mana gradien tidak diperlukan. Metode `torch.no_grad()` digunakan untuk menonaktifkan perhitungan gradien pada blok kode tertentu untuk efisiensi komputasi, seperti yang terlihat pada contoh keempat.

C. Back Propagation

- Perhitungan Prediksi: Nilai `y_predicted` dihitung dengan mengalikan variabel `w` dengan `x`. Ini mewakili prediksi model yang dihasilkan berdasarkan parameter `w`.
- Perhitungan Loss: Loss function yang digunakan adalah mean squared error (MSE). Loss dihitung dengan menghitung selisih kuadrat antara `y_predicted` dan `y`. Hasil loss tersebut ditampilkan menggunakan perintah `print(loss)`.
- Perhitungan Gradien: Langkah ini melibatkan perhitungan gradien terhadap variabel yang memiliki atribut `requires_grad=True`. Pada kode ini, gradien dihitung terhadap variabel `w` dengan memanggil `loss.backward()`. Hasil gradien tersebut dapat diakses menggunakan `w.grad`.

D. Gradientdescent Manually

- Fungsi Forward: Dalam fungsi `forward(x)`, prediksi `y_pred` dihitung dengan mengalikan nilai `x` dengan parameter `w`. Fungsi ini mewakili tahap perhitungan output model berdasarkan input dan parameter yang ada.
- Fungsi Loss: Fungsi `loss(y, y_pred)` menghitung mean squared error (MSE) antara nilai target `y` dan prediksi `y_pred`. MSE menggambarkan seberapa jauh prediksi model dari nilai target yang sebenarnya. Fungsi ini mengembalikan nilai loss yang dihitung.
- Perhitungan Gradien: Fungsi `gradient(x, y, y_pred)` digunakan untuk menghitung gradien dari loss function terhadap parameter `w`. Gradien ini digunakan dalam pembaruan parameter selama pelatihan untuk memperbaiki model. Gradien dihitung menggunakan rumus yang sesuai dengan mean squared error.
- Pelatihan Model: Dalam loop pelatihan (`for epoch in range(n_iters)`), langkah-langkah pelatihan dilakukan. Prediksi `y_pred` dihitung menggunakan fungsi forward, loss `l` dihitung menggunakan fungsi loss, dan gradien `dw` dihitung menggunakan fungsi gradient. Parameter `w` diperbarui dengan mengurangi learning rate (`learning_rate`) dikali dengan gradien `dw`. Pada setiap epoch yang merupakan kelipatan 2, nilai parameter `w` dan loss `l` dicetak untuk melihat perkembangan pelatihan.
- Prediksi Setelah Pelatihan: Setelah loop pelatihan selesai, dilakukan prediksi untuk input baru (`x = 5`) menggunakan fungsi forward dengan parameter yang telah diperbarui. Hasil prediksi ini dicetak sebagai "Prediction after training: f(5) = {nilai_prediksi}".
- Pembaruan Parameter: Dalam langkah ini, dilakukan pembaruan parameter model dengan menggunakan gradien turun (gradient descent). Dalam contoh ini, dilakukan pembaruan pada variabel `w` dengan mengurangi nilai gradien yang telah dihitung sebelumnya dikalikan dengan learning rate (`0.01`). Pembaruan parameter dilakukan dalam blok `torch.no_grad()` untuk memastikan bahwa operasi ini tidak dihitung dalam perhitungan gradien. Selanjutnya, gradien direset ke nol menggunakan `w.grad.zero_()` agar gradien tidak terakumulasi dalam iterasi berikutnya.

E. Gradientdescent Auto

- Cetak Nilai: Pada setiap epoch yang merupakan kelipatan 10, nilai parameter `w` dan loss `l` dicetak untuk melihat perkembangan pelatihan.
- Prediksi Setelah Pelatihan: Setelah loop pelatihan selesai, dilakukan prediksi untuk input baru (`x = 5`) menggunakan fungsi forward dengan parameter yang telah diperbarui. Hasil prediksi ini dicetak sebagai "Prediction after training: f(5) = {nilai_prediksi}".

F. Loss and Optimizer

- Cetak Nilai Awal: Cetak nilai prediksi sebelum pelatihan dilakukan, yaitu "Prediction before training: f(5) = {nilai_prediksi}".
- Pelatihan Model: Dalam loop pelatihan (`for epoch in range(n_iters)`), langkah-langkah pelatihan dilakukan. Prediksi `y_predicted` dihitung menggunakan fungsi forward, dan loss `l` dihitung menggunakan fungsi `nn.MSELoss()`. Optimizer yang digunakan adalah stochastic gradient descent (SGD) dengan `torch.optim.SGD`, dengan parameter `w` sebagai parameter yang akan dioptimasi dan `learning_rate` sebagai learning rate.
- Pembaruan Parameter: Dilakukan langkah-langkah untuk memperbarui parameter `w`. Pertama, gradien direset dengan `optimizer.zero_grad()`. Selanjutnya, gradien dihitung

menggunakan `l.backward()` dan optimizer melakukan pembaruan parameter menggunakan `optimizer.step()`.

- Cetak Nilai: Pada setiap epoch yang merupakan kelipatan 10, nilai parameter `w` dan loss `l` dicetak untuk melihat perkembangan pelatihan.
- Prediksi Setelah Pelatihan: Setelah loop pelatihan selesai, dilakukan prediksi untuk input baru (`x = 5`) menggunakan fungsi forward dengan parameter yang telah diperbarui. Hasil prediksi ini dicetak sebagai "Prediction after training: f(5) = {nilai_prediksi}".

G. Model Loss and Optimizer

- Data Training: Data training (`X` dan `Y`) terdiri dari 4 sampel dengan 1 fitur. Data `X` berisi input, yaitu [1, 2, 3, 4], dan data `Y` berisi output yang diharapkan, yaitu [2, 4, 6, 8].
- Model: Model linear regression didefinisikan menggunakan `nn.Linear` dari PyTorch. Model ini memiliki satu input dan satu output.
- Prediksi Sebelum Pelatihan: Prediksi `f(5)` sebelum pelatihan dilakukan dengan memasukkan nilai 5 ke dalam model (`model(X_test)`). Hasil prediksi ini dicetak sebagai "Prediction before training: f(5) = {nilai_prediksi}".
- Loss dan Optimizer: Loss function yang digunakan adalah mean squared error (MSE) yang diimplementasikan dengan `nn.MSELoss()`. Optimizer yang digunakan adalah stochastic gradient descent (SGD) dengan `torch.optim.SGD`, dengan parameter `model.parameters()` sebagai parameter yang akan dioptimasi dan `learning_rate` sebagai learning rate.
- Loop Pelatihan: Dalam loop pelatihan (`for epoch in range(n_iters)`), langkah-langkah pelatihan dilakukan. Prediksi `y_predicted` dihitung menggunakan model, dan loss `l` dihitung menggunakan fungsi `nn.MSELoss()`. Gradien dihitung dengan memanggil `l.backward()` dan model diperbarui menggunakan `optimizer.step()`.
- Prediksi Setelah Pelatihan: Setelah loop pelatihan selesai, dilakukan prediksi untuk input baru (`X_test`) menggunakan model yang telah dilatih. Hasil prediksi ini dicetak sebagai "Prediction after training: f(5) = {nilai_prediksi}".

H. Linear Regression

- Cetak Loss: Pada setiap epoch yang merupakan kelipatan 10, nilai loss dicetak sebagai "epoch: {epoch+1}, loss = {loss.item():.4f}".
- Plot: Setelah pelatihan selesai, dilakukan plotting hasil prediksi model. Rentang input (`X_plot`) ditentukan menggunakan `torch.linspace` untuk membuat 100 titik pada rentang nilai X. Prediksi model (`predicted`) diambil menggunakan `model(X_plot).detach().numpy()`. Data asli ditampilkan sebagai titik merah (`ro`), sedangkan prediksi model ditampilkan sebagai garis biru (`b`).

Kodingan ini mengilustrasikan penggunaan PyTorch untuk melakukan pelatihan regresi non-linear dan memvisualisasikan hasilnya menggunakan matplotlib.

I. Logistic Regression

- Loop Pelatihan: Dalam loop pelatihan (`for epoch in range(num_epochs)`), langkah-langkah pelatihan dilakukan. Prediksi `y_pred` dihitung menggunakan model, dan loss `loss` dihitung menggunakan fungsi `nn.BCELoss`. Gradien dihitung dengan memanggil `loss.backward()` dan model diperbarui menggunakan `optimizer.step()`.
- Cetak Loss: Pada setiap epoch yang merupakan kelipatan 10, nilai loss dicetak sebagai "epoch: {epoch+1}, loss = {loss.item():.4f}".
- Evaluasi: Setelah pelatihan selesai, dilakukan evaluasi pada data uji. Prediksi `y_predicted` diperoleh dengan memanggil model pada `X_test`, kemudian nilai prediksi diambil sebagai

nilai kelas menggunakan `torch.round`. Akurasi dihitung dengan membandingkan prediksi dengan label aktual `y_test`.

J. DATALOADER

- Definisi Dataset:
  - Dalam kelas `WineDataset`, dataset dari file CSV "wine.csv" dimuat menggunakan `np.loadtxt` dan disimpan dalam atribut `x_data` dan `y_data`.
  - Metode `__getitem__` digunakan untuk mengakses sampel tertentu dengan indeks, dan metode `__len__` mengembalikan jumlah total sampel dalam dataset.
- Membuat Dataset:
  - Dataset dibuat dengan membuat objek dari kelas `WineDataset`.
  - Sampel pertama dalam dataset diambil dan dipecah menjadi fitur (`features`) dan label (`labels`), kemudian dicetak.
- DataLoader:
  - DataLoader digunakan untuk memuat dataset secara efisien.
  - Parameter `batch_size` menentukan ukuran batch yang akan digunakan.
  - `shuffle=True` digunakan untuk mengacak data dalam setiap epoch.
  - `num_workers=2` mengizinkan pemrosesan paralel untuk mempercepat pembebanan data (jika terjadi kesalahan, num_workers harus diatur menjadi 0).
- Training Loop:
  - Sebuah loop pelatihan dummy digunakan untuk mengilustrasikan penggunaan DataLoader.
  - Jumlah total sampel dan jumlah iterasi dihitung berdasarkan ukuran batch.
  - Dalam loop, setiap batch diproses dan dicetak informasi terkait.
- Menggunakan torchvision.datasets:
  - Kode juga menunjukkan penggunaan dataset dari `torchvision.datasets`, yaitu MNIST.
  - MNIST dataset diunduh dan dimuat menggunakan `torchvision.datasets.MNIST`.
  - DataLoader `train_loader` dibuat dengan dataset MNIST.

K. Transformers

- Import Library:
  - torch adalah pustaka utama dalam PyTorch yang digunakan untuk komputasi tensor dan operasi terkait.
  - torchvision adalah pustaka yang menyediakan dataset, model arsitektur, dan fungsi transformasi untuk visi komputer.
  - torch.utils.data.Dataset adalah kelas dasar untuk membuat dataset kustom dalam PyTorch.
  - numpy adalah pustaka populer untuk komputasi numerik dalam Python.
  - PIL(Python Imaging Library) adalah pustaka untuk memanipulasi gambar.
- Kelas WineDataset:
  - Kelas WineDataset adalah subclass dari `torch.utils.data.Dataset`.
  - Metode __init__memuat data dari file "wine.csv" menggunakan `np.loadtxt` dan menyimpan fitur (`x_data`) dan label (`y_data`).
  - Metode __getitem__mengembalikan sampel tertentu dari dataset.
  - Jika ada transformasi yang diberikan, metode `__getitem__` akan menerapkan transformasi tersebut pada sampel sebelum mengembalikannya.
  - Metode __len__mengembalikan jumlah total sampel dalam dataset.
- Kelas Transformasi:
  - Kelas ToTensor adalah transformasi khusus yang mengubah sampel menjadi tensor menggunakan `torch.from_numpy`.

- Penggunaan Dataset dan Transformasi:
  - Mencetak hasil tanpa transformasi: Dataset `WineDataset` dibuat tanpa transformasi. Sampel pertama diambil dan dicetak fitur dan labelnya.
  - Penggunaan transformasi Tensor: Dataset `WineDataset` dibuat dengan transformasi `ToTensor()`. Sampel pertama diambil dan dicetak fitur dan labelnya, yang sudah berupa tensor.
  Penggunaan transformasi Tensor dan Multiplication: Transformasi `ToTensor()` dan `MulTransform(4)` digabungkan dengan menggunakan `torchvision.transforms.Compose`. Dataset `WineDataset` dibuat dengan transformasi tersebut. Sampel pertama diambil dan dicetak fitur dan labelnya setelah transformasi tensor dan perkalian.

L. Sofmax dan Cross Entropy

Softmax adalah fungsi yang digunakan untuk mengonversi output dari sebuah model ke dalam distribusi probabilitas. Fungsi softmax mengambil input dalam bentuk vektor dan menghasilkan probabilitas yang berada dalam rentang 0 hingga 1.Output dari softmax dapat diinterpretasikan sebagai probabilitas kelas yang dihasilkan oleh model.

Cross Entropy adalah sebuah metrik yang digunakan untuk mengukur perbedaan (divergensi) antara distribusi probabilitas yang dihasilkan oleh model dan distribusi probabilitas yang sebenarnya (label kelas yang benar). Cross Entropy biasanya digunakan dalam tugas klasifikasi multikelas.Fungsi cross entropy mengambil input berupa dua distribusi probabilitas: probabilitas prediksi yang dihasilkan oleh model dan probabilitas yang sebenarnya (one-hot encoded label).

Softmax dan Cross Entropy sering digunakan bersama dalam tugas klasifikasi multikelas. Softmax digunakan untuk menghasilkan probabilitas output, sedangkan Cross Entropy digunakan sebagai fungsi loss untuk melatih model sehingga prediksi model mendekati label yang benar.

M. Plot Activations

- orch.tensor: Fungsi ini digunakan untuk membuat tensor baru dari data yang ada. Di contoh kode, digunakan untuk membuat tensor `x` dengan nilai [-1.0, 1.0, 2.0, 3.0].
- torch.softmax: Fungsi ini menghitung softmax dari input tensor di sepanjang dimensi yang ditentukan. Outputnya adalah tensor dengan probabilitas yang dihasilkan dari fungsi softmax.
- nn.Softmax: Kelas ini adalah implementasi softmax dalam bentuk modul dalam modul `nn` dari PyTorch. Ini memberikan lebih banyak fleksibilitas dalam penggunaannya, misalnya dapat ditambahkan sebagai bagian dari model neural network.
- torch.sigmoid: Fungsi ini menghitung sigmoid dari input tensor. Sigmoid menghasilkan output dalam rentang 0 hingga 1 dan sering digunakan dalam masalah biner.
- nn.Sigmoid: Kelas ini adalah implementasi sigmoid dalam bentuk modul dalam modul `nn` dari PyTorch.
- torch.tanh: Fungsi ini menghitung tangen hiperbolik dari input tensor. Tangen hiperbolik menghasilkan output dalam rentang -1 hingga 1 dan sering digunakan dalam jaringan saraf.
- nn.Tanh: Kelas ini adalah implementasi tangen hiperbolik dalam bentuk modul dalam modul `nn` dari PyTorch.
- torch.relu: Fungsi ini menghitung Rectified Linear Unit (ReLU) dari input tensor. ReLU menghasilkan output nol untuk nilai negatif dan nilainya sendiri untuk nilai positif.
- nn.ReLU: Kelas ini adalah implementasi ReLU dalam bentuk modul dalam modul `nn` dari PyTorch.
- F.leaky_relu: Fungsi ini menghitung Leaky ReLU dari input tensor. Leaky ReLU memiliki kelebihan dibandingkan ReLU karena memperkenalkan gradien non-nol untuk nilai negatif yang kecil.

N. Feed & Forward

- Hyperparameter:
- Definisikan hyperparameter seperti ukuran input, ukuran lapisan tersembunyi, jumlah kelas, jumlah epoch, ukuran batch, dan learning rate.
- Persiapan Data:
  - Unduh dataset MNIST dan siapkan dataset pelatihan dan pengujian.
  - Gunakan `torchvision.transforms.ToTensor()` untuk mengubah gambar menjadi tensor.
  - Buat `DataLoader` untuk memuat data dalam batch.
- Visualisasi Contoh Data:
  -Tampilkan contoh data dari dataset pengujian menggunakan `matplotlib.pyplot.imshow()`
- Definisi Model:
  - Buat kelas `NeuralNet` yang mewarisi `nn.Module`.
  - Dalam kelas ini, definisikan lapisan-lapisan linear (`nn.Linear`) dan fungsi aktivasi ReLU (`nn.ReLU`).
  - Metode `forward` menggambarkan aliran masukan ke keluaran melalui lapisan-lapisan model.
- Inisialisasi Model dan Optimizer:
  - Buat objek model `NeuralNet` dan pindahkan ke perangkat yang ditentukan.
- Pengujian Model:
  - Dalam fase pengujian, matikan perhitungan gradien (`torch.no_grad()`) karena tidak memerlukan perhitungan gradien.
  - Hitung jumlah prediksi yang benar dan total sampel untuk menghitung akurasi model.

O. CNN

- Hyperparameter:
- Persiapan Data:
  - Transformasi data menggunakan `transforms.Compose` untuk mengubah gambar menjadi tensor dan melakukan normalisasi.
  - Unduh dataset CIFAR-10 dan siapkan dataset pelatihan dan pengujian.
  - Buat `DataLoader` untuk memuat data dalam batch.
- Visualisasi Contoh Data:
  - Buat fungsi `imshow` untuk menampilkan gambar dalam matplotlib.
  - Tampilkan beberapa contoh gambar dari dataset pelatihan menggunakan `imshow`.
- Definisi Model:
  - Buat kelas `ConvNet` yang mewarisi `nn.Module`.
  - Dalam kelas ini, definisikan lapisan-lapisan konvolusi (`nn.Conv2d`), lapisan pooling (`nn.MaxPool2d`), dan lapisan linear (`nn.Linear`).
  - Metode `forward` menggambarkan aliran masukan ke keluaran melalui lapisan-lapisan model.
- Inisialisasi Model dan Optimizer:
  - Buat objek model `ConvNet` dan pindahkan ke perangkat yang ditentukan.
  - Tentukan fungsi kerugian (`nn.CrossEntropyLoss`) dan optimizer (`torch.optim.SGD`) untuk melatih model.
- Pelatihan Model:
  - Loop melalui setiap epoch dan setiap batch dalam dataset pelatihan.
- Simpan Model yang Dilatih:

  -Simpan model yang dilatih ke dalam file menggunakan `torch.save`.

- Pengujian Model:
  - Dalam fase pengujian, matikan perhitungan gradien (`torch.no_grad()`) karena tidak memerlukan perhitungan gradien.

- Hitung jumlah prediksi yang benar dan total sampel untuk menghitung akurasi model.

P. Transfer Lisensing

- Library dan Package: Terdapat beberapa library dan package yang diimpor pada awal kode, antara lain:
  - torch` dan `torch.nn`: Library utama untuk komputasi tensor dan operasi neural network pada PyTorch.
  - torch.optim`: Modul yang menyediakan berbagai algoritma optimasi untuk melatih model.
  - torchvision`: Package yang menyediakan dataset, model, dan transformasi yang umum digunakan dalam visi komputer.
  - numpy`: Package untuk operasi numerik pada Python.
  - matplotlib.pyplot`: Package untuk membuat visualisasi plot dan grafik.
  - time` dan `os`: Package untuk mengelola waktu dan sistem operasi.
  - copy` dan `zipfile`: Package untuk melakukan operasi penyalinan objek dan pengarsipan file.
- Ekstraksi Dataset: Dataset dalam format zip diekstraksi menggunakan `zipfile.ZipFile`. Path dari file zip dan tempat ekstraksi dapat disesuaikan. Setelah diekstraksi, path direktori dataset ditentukan dalam variabel `data_dir`.
- Transformasi Data: Data pada dataset diubah menggunakan transformasi yang ditentukan dalam `data_transforms`. Transformasi ini diterapkan untuk data pelatihan (`'train'`) dan data validasi (`'val'`).
- Pembuatan DataLoader: `torch.utils.data.DataLoader` digunakan untuk memuat data dalam batch. Data loader dibuat untuk data pelatihan dan data validasi dengan ukuran batch yang ditentukan.
- Definisi Model: Model yang digunakan adalah `models.resnet18` yang telah dilatih sebelumnya (`pretrained=True`). Kemudian, fully connected layer terakhir diganti dengan `nn.Linear` untuk memodifikasi output sesuai dengan jumlah kelas dalam dataset. Model dimuat ke perangkat yang sesuai.
- Fine-tuning Model: Setelah pelatihan, model ConvNet diubah menjadi fixed feature extractor. Bagian awal model yang telah dilatih dibekukan (`requires_grad=False`), sehingga hanya layer terakhir (fully connected layer) yang dioptimasi. Proses pelatihan dilakukan kembali pada model yang telah dimodifikasi ini.

Q. Tensorboard

- Memuat dataset MNIST:
  - Menggunakan `torchvision.datasets.MNIST` untuk memuat dataset MNIST dan menerapkan transformasi `transforms.ToTensor()` untuk mengubah gambar menjadi tensor.
  - Membagi dataset menjadi data pelatihan dan pengujian.
  - Membuat objek `torch.utils.data.DataLoader` untuk memuat data dalam batch.
- Menampilkan contoh gambar dari dataset:
  -Menggunakan `matplotlib.pyplot.imshow` untuk menampilkan beberapa contoh gambar dalam grid.
- Membuat objek `SummaryWriter` untuk TensorBoard dan menambahkan grid gambar ke TensorBoard menggunakan `writer.add_image`.
- Membangun model neural network:
  - Membuat kelas `NeuralNet` yang merupakan turunan dari `nn.Module`.
  - Model ini memiliki lapisan-lapisan linear dan fungsi aktivasi ReLU.
- Menginisialisasi model, loss, dan optimizer:
  - Membuat objek model `NeuralNet` dan memindahkannya ke perangkat yang ditentukan.
- Melatih model:
  - Menggunakan loop `for` untuk melatih model dalam beberapa epoch.

- Menggunakan loop `for` lagi untuk melatih model dalam batch menggunakan data pelatihan.
- Melakukan proses forward, backward, dan optimisasi.
- Mencatat loss dan akurasi pelatihan ke TensorBoard menggunakan `writer.add_scalar`.

- Menguji model:
  - Menggunakan loop `for` untuk melakukan pengujian model menggunakan data pengujian.
  - Menghitung akurasi pengujian dan mencatatnya ke TensorBoard menggunakan `writer.add_pr_curve`.
  - Menutup objek `SummaryWriter` untuk mengakhiri sesi TensorBoard.

R. Save & Load Modules Metode

- Menyimpan dan Memuat Seluruh Model Metode ini melibatkan penyimpanan dan pemulihan seluruh model beserta semua parameter dan strukturnya. kita dapat menyimpan model dengan menggunakan torch.save(model, PATH) dan memuatnya dengan model = torch.load(PATH). Setelah memuat model, kita dapat menggunakannya untuk inferensi dengan memanggil model.eval().
- Menyimpan dan Memuat State Dictionary Metode ini lebih disarankan karena hanya menyimpan state dictionary dari model. kita dapat menyimpan state dictionary dengan kita perlu membuat model baru terlebih dahulu dengan menggunakan struktur yang sama, dan kemudian memuat state dictionary ke model tersebut. Setelah memuat state dictionary, kita juga perlu memanggil model.eval() untuk inferensi.
- Menyimpan dan Memuat Checkpoint Metode ini berguna saat kita ingin menyimpan lebih dari sekadar state dictionary model. Misalnya, kita ingin menyimpan juga optimizer dan informasi lainnya. kita dapat membuat checkpoint yang berisi informasi seperti epoch, state dictionary model, dan state dictionary optimizer, dan menyimpannya dengan torch.save(checkpoint, FILE).

KESIMPULAN

Tensor adalah struktur data fundamental dalam PyTorch, digunakan untuk mewakili data multidimensional. Kita dapat membuat tensor dengan fungsi `torch.empty()`, `torch.rand()`, `torch.zeros()`, dan `torch.ones()`. Operasi tensor seperti penambahan, pengurangan, perkalian, dan pembagian elemen demi elemen juga tersedia. Autograd adalah fitur PyTorch yang menyediakan diferensiasi otomatis. Saat kita mendefinisikan tensor dengan `requires_grad=True`, PyTorch akan melacak operasi pada tensor tersebut. Ini memungkinkan kita untuk melakukan backpropagation dan menghitung gradien dengan mudah menggunakan metode `.backward()`.

Backpropagation digunakan dalam deep learning untuk menghitung gradien loss terhadap parameter model. PyTorch secara otomatis mengelola backpropagation melalui autograd. Dengan pembaruan parameter menggunakan optimisasi seperti gradient descent, kita dapat mengoptimalkan model untuk menghasilkan prediksi yang lebih baik. Selain itu, laporan ini juga membahas penggunaan tensor dalam integrasi dengan NumPy untuk konversi data, serta kemampuan PyTorch untuk memanfaatkan komputasi GPU untuk kecepatan pemrosesan yang lebih tinggi