TECHNICAL REPORT UTS MACHINE LEARNING

Deep Learning with PyTorch

Diajukan untuk memenuhi tugas pengganti Ujian Tengah Semester (UAS) pada mata kuliah Machine Learning

Disusun oleh :

Nurrafi Bagus Pratama - 1103180026

PROGRAM STUDI TEKNIK KOMPUTER
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
2023

I. Pendahuluan

Deep learning telah merevolusi bidang kecerdasan buatan, memungkinkan mesin untuk belajar dan membuat keputusan cerdas dari data yang sangat besar. Sebagai kerangka kerja yang kuat untuk deep learning, PyTorch menyediakan platform yang fleksibel dan efisien untuk membangun dan melatih neural network.

Di sepanjang laporan ini, kita akan mempelajari berbagai topik, mulai dari dasar-dasar tensor dan autograd, yang menjadi fondasi PyTorch. Kemudian kita akan mengeksplorasi backpropagation, gradient descent, dan pipeline pelatihan, memahami bagaimana model belajar dari data dan meningkatkan kinerjanya.

Laporan ini akan membahas teknik-teknik penting seperti regresi linier dan regresi logistik, mendemonstrasikan cara menyelesaikan masalah regresi dan klasifikasi. Kami juga akan mempelajari topik-topik yang lebih lanjut, termasuk convolutional neural network (CNN) untuk analisis gambar dan transfer learning untuk meningkatkan model yang telah dilatih sebelumnya.

Selain itu, kita akan membahas pemuatan dan prapemrosesan data, fungsi aktivasi, dan konsep penyimpanan dan pemuatan model untuk penerapan. Di sepanjang jalan, contoh kode dan wawasan praktis akan diberikan untuk memfasilitasi pemahaman dan penerapan.

Pada akhir laporan ini, pembaca akan mendapatkan pemahaman yang kuat tentang prinsip-prinsip pembelajaran mendalam, memperoleh pengalaman langsung dalam mengimplementasikan berbagai model, dan mengembangkan keterampilan untuk menangani beragam tugas pembelajaran mesin menggunakan PyTorch.

II. Perjelasan per Chapter Video

1. Installation

2. Tensor Basics

Bab ini berfokus pada dasar - dasar tensor di PyTorch. Tensor adalah struktur data inti yang digunakan di PyTorch untuk merepresentasikan dan memanipulasi data. Seperti array pada umumnya, tensor bisa memiliki satu, dua, tiga, atau lebih dimensi.

2.1. Fungsi – Fungsi Tensor

Kode pada bagian Tensor Basics dimulai dengan syntax untuk menginisialisasi tensor dengan berbagai dimensi. Beberapa fungsi dasar yang didemonstrasikan pada code yang berkaitan dengan tensor adalah:

- torch.empty(size): Digunakan untuk membuat tensor yang tidak diinisialisasi
- torch.rand(size): Menghasilkan tensor yang diisi dengan angka acak antara 0 dan 1.
- torch.zeros(size): Menginisialisasi tensor yang berukuran tertentu dengan nilai 0.
- torch.ones(size): Menginisialisasi tensor yang berukuran tertentu dengan nilai 1.

Terdapat pula meode untuk memeriksa ukuran tensor. Metode size() mengembalikan dimensi tensor, dan atribut dtype memberikan informasi tentang tipe data tensor, seperti torch.float32 atau torch.float64.

2.2. Operasi Tensor

Pada contoh kode, ditunjukkan bahwa operasi elemen, seperti penjumlahan, pengurangan, perkalian, dan pembagian dapat dilakukan terhadap tensor:

- Penjumlahan: torch.add(x, y)
- Pengurangan: torch.sub(x, y)
- Pembagian: torch.div(x, y)
- Perkalian: torch.mul(x, y)

Selain operasi elemen, ditunjukkan juga tentang pemotongan (slicing) tensor. Dengan ini kita dapat mengakses elemen atau himpunan bagian tertentu dari sebuah tensor. Contohnya dengan menunjukkan cara mengambil baris, kolom, dan elemen individual menggunakan pengindeksan. Metode item() juga digunakan untuk mengekstraksi nilai sebenarnya dari satu elemen dari sebuah tensor.

Kode berikutnya mencakup pembentukan kembali tensor menggunakan metode view(). Bagian ini menjelaskan bagaimana dimensi tensor dapat diatur ulang dengan menentukan ukuran yang diinginkan atau dengan menggunakan -1 sebagai placeholder, yang memungkinkan PyTorch untuk secara otomatis menentukan ukuran yang diperlukan.

Kemampuan penting lainnya dari PyTorch adalah integrasi dengan library NumPy. Tensor dapat dikonversi menjadi array NumPy menggunakan metode .numpy() dan sebaliknya menggunakan torch.from_numpy().

3. Autograd

3.1. Penjelasan Autograd

Paket autograd dalam PyTorch menyediakan diferensiasi otomatis untuk semua operasi yang dilakukan pada tensor. Ini memungkinkan kita untuk menghitung gradien secara otomatis, yang penting untuk melatih model deep learning menggunakan teknik seperti backpropagation. Konsep utama di balik autograd adalah melacak operasi yang dilakukan pada tensor dan membangun grafik komputasi, yang memungkinkan perhitungan gradien yang efisien terhadap tensor.

3.2. Penjelasan Kode

Pada kode yang diberikan, kita dapat melihat penggunaan autograd untuk menghitung gradien. Penjelasan kode adalah sebagai berikut:

- Membuat Tensor dengan requires_grad=True
  Baris x = torch.randn(3, requires_grad=True) membuat tensor x dengan bentuk (3,) dengan nilai acak.
- Mengatur requires_grad=True
  memungkinkan pelacakan operasi pada x untuk perhitungan gradien.

- Melakukan Operasi dan Melacak Fungsi Grad

  Baris y = x + 2 melakukan operasi penjumlahan elemen-wise antara x dan nilai skalar 2, menghasilkan tensor y. Karena y diperoleh melalui operasi, ia memiliki atribut grad_fn yang merujuk pada fungsi yang membuat tensor tersebut.

Operasi Lanjutan dan Perhitungan Gradien

Kode dilanjutkan dengan operasi lain pada y, seperti perkalian dan perhitungan rata-rata (z = y * y * 3 dan z = z.mean()).

Untuk menghitung gradien, kita memanggil z.backward(), yang memicu proses backpropagation dan menghitung gradien z terhadap tensor yang memiliki requires_grad=True (dalam hal ini, x).

Akhirnya, kita mencetak x.grad untuk mengakses gradien yang dihitung dari x (dz/dx).

3.3. Konsep Autograd

Sebelum memahami autograd, penting untuk memahami konsep gradien dan backpropagation. Gradien merepresentasikan turunan parsial suatu fungsi terhadap inputnya (tensor), menunjukkan laju perubahan fungsi terhadap setiap input. Backpropagation adalah proses mempropagasi gradien ini ke belakang melalui grafik komputasi untuk menghitung gradien semua parameter dalam model deep learning secara efisien.

Autograd dalam PyTorch mengotomatiskan perhitungan gradien dengan membangun grafik komputasi secara dinamis dan melakukan diferensiasi otomatis dengan mode balik yang efisien. Memungkinkan kita untuk fokus pada mendefinisikan proses forward pada model kita sementara PyTorch menangani perhitungan gradien. Dengan memanggil fungsi backward() pada tensor output yang bersifat skalar, kita dapat menghitung gradien dari semua tensor yang terlibat dalam grafik komputasi dengan requires_grad=True.

Memahami autograd sangat penting untuk melatih neural network karena menyederhanakan proses perhitungan gradien, memungkinkan kita untuk mengoptimalkan parameter model menggunakan algoritma optimisasi berbasis gradien.

4. Backpropagation

4.1. Penjelasan Backpropagation

Backpropagation adalah algoritma penting dalam pelatihan model deep learning yang memungkinkan perhitungan efisien dari gradien model terhadap bobotnya. Dalam backpropagation, gradien dari fungsi kerugian terhadap bobot model dihitung dan digunakan untuk memperbarui bobot tersebut menggunakan algoritma optimisasi berbasis gradien, seperti gradien turun (gradient descent).

Konsep utama di balik backpropagation adalah penyebaran mundur gradien melalui neural network untuk menghitung kontribusi relatif setiap

bobot terhadap kesalahan prediksi.

4.2. Penjelasan Kode

a. Inisialisasi Tensor dan Bobot
   - Kode x = torch.tensor(1.0) dan y = torch.tensor(2.0) membuat tensor x dan y dengan nilai skalar.
   - Kode w = torch.tensor(1.0, requires_grad=True) membuat tensor w dengan nilai skalar dan mengatur requires_grad=True untuk mengindikasikan bahwa kita ingin mengoptimalkan bobot ini.

b. Langkah-Langkah Forward Pass
   - Kode y_predicted = w * x melakukan perkalian elemen-wise antara w dan x, menghasilkan tensor y_predicted yang merupakan prediksi model.
   - Kode loss = (y_predicted - y)**2 menghitung fungsi kerugian berdasarkan selisih kuadrat antara prediksi y_predicted dan nilai target y.
   - Cetak loss untuk melihat nilai fungsi kerugian.

c. Langkah-Langkah Backward Pass
   - Kode loss.backward() memicu proses backpropagation dan menghitung gradien fungsi kerugian terhadap tensor yang memiliki requires_grad=True (dalam hal ini, w).
   - Cetak w.grad untuk melihat nilai gradien yang dihitung dari w (dLoss/dw).

d. Pembaruan Bobot
   - Kode with torch.no_grad() digunakan untuk mengelompokkan langkah pembaruan bobot agar tidak termasuk dalam grafik komputasi dan tidak mempengaruhi perhitungan gradien.
   - Dalam blok tersebut, kita mengurangi bobot w dengan hasil perkalian antara gradien dan suatu nilai (misalnya, 0.01) sebagai langkah optimisasi.
   - Setelah pembaruan bobot, kita menggunakan w.grad.zero_() untuk menghapus gradien yang telah terakumulasi sebelumnya sebelum melanjutkan langkah-langkah berikutnya pada proses optimisasi.

5. Gradient Descent

5.1. Penjelasan Gradient Descent

Dalam regresi linear, tujuan kita adalah mencari bobot (w) yang paling baik menyesuaikan data input (X) dengan target (Y). Untuk mencapai tujuan ini, kita menggunakan metode gradient descent, yang melibatkan langkah-langkah berikut:

a. Inisialisasi bobot (w) dengan nilai awal.
b. Melakukan prediksi dengan memasukkan data input (X) melalui model.
c. Menghitung nilai fungsi kerugian antara prediksi (y_pred) dan target (Y).
d. Menghitung gradien fungsi kerugian terhadap bobot (w).
e. Memperbarui bobot (w) dengan mengurangi learning rate

(learning_rate) dikali gradien (dw).

f. Mengulangi langkah 2-5 dengan iterasi (epoch) yang cukup.
g. Setelah pelatihan selesai, kita dapat menggunakan model yang terlatih untuk melakukan prediksi pada data baru.

5.2. Penjelasan Kode

Dalam kode yang diberikan, dapt kita lihat implementasi manual dari metode gradient descent digunakan untuk menyelesaikan masalah regresi linear sederhana.

a. Persiapan Data
   - Kode X = np.array([1, 2, 3, 4], dtype=np.float32) dan Y = np.array([2, 4, 6, 8], dtype=np.float32) digunakan untuk mendefinisikan data input (X) dan target (Y) dalam bentuk array NumPy.

b. Persiapan Model dan Fungsi Kerugian:
   - Kode w = 0.0 menginisialisasi bobot (w) dengan nilai awal 0.
   - Fungsi forward(x) mengimplementasikan model regresi linear sederhana dengan mengalikan bobot (w) dengan input (x).
   - Fungsi loss(y, y_pred) menghitung fungsi kerugian Mean Squared Error (MSE) antara nilai target (y) dan prediksi (y_pred).
   - Fungsi gradient(x, y, y_pred) menghitung gradien fungsi kerugian MSE terhadap bobot (w) dengan menggunakan rumus derivatif parsial.

c. Training dengan Gradient Descent:
   - Kode learning_rate = 0.01 dan n_iters = 20 mengatur laju pembelajaran (learning rate) dan jumlah iterasi (epoch) untuk pelatihan.
   - Melalui loop for epoch in range(n_iters), kita melakukan iterasi sebanyak n_iters kali untuk melatih model.
   - Di dalam setiap iterasi:
     - Melakukan prediksi dengan memanggil forward(X) untuk mendapatkan prediksi (y_pred).
     - Menghitung nilai fungsi kerugian dengan memanggil loss(Y, y_pred) dan menyimpannya dalam variabel l.
     - Menghitung gradien dengan memanggil gradient(X, Y, y_pred) dan menyimpannya dalam variabel dw.
     - Memperbarui bobot (w) dengan menggunakan rumus gradient descent: w -= learning_rate * dw.
     - Selama pelatihan, cetak bobot (w) dan nilai fungsi kerugian (l) setiap 2 epoch.

d. Prediksi Setelah Pelatihan:
   - Setelah selesai pelatihan, kita mencetak prediksi dari model untuk input 5 dengan memanggil forward(5).

6. Training Pipeline

6.1. Penjelasan Training Pipeline

Kode pertama adalah contoh pelatihan regresi linear sederhana menggunakan PyTorch dengan mengoptimasi parameter langsung menggunakan tensor w. Kode ini mengikuti langkah-langkah yang telah dijelaskan sebelumnya.

Kode kedua adalah contoh pelatihan regresi linear menggunakan model linier yang diimplementasikan menggunakan kelas nn.Linear. Model ini menghasilkan output dengan melakukan forward pass pada tensor X. Kode juga menunjukkan bagaimana mengakses parameter bobot dengan menggunakan model.parameters() dan mencetak nilai bobot dengan w[0][0].item(). Kode ini juga mengikuti langkah-langkah yang sama dengan kode sebelumnya untuk melatih model dan mengoptimasi parameter.

7. Linear Regression

7.1. Penjelasan Kode

Berdasarkan kode yang terdapat pada video, didemonstrasikan regresi linier menggunakan PyTorch yang mencakup hal – hal berikut:

a. Persiapan Data

Hasilkan data regresi sintetis menggunakan fungsi make_regression dari library sklearn. Ubah array NumPy menjadi tensor PyTorch.

b. Definisi Model

Tentukan model regresi linier menggunakan modul nn.Linear. Model ini menggunakan ukuran input (n_features) dan ukuran output (output_size) sebagai parameter.

c. Kerugian dan Pengoptimalisasi

Tentukan fungsi kerugian (Mean Squared Error, nn.MSELoss) dan pengoptimal (Stochastic Gradient Descent, torch.optim.SGD). Parameter model diteruskan ke pengoptimal untuk dioptimalkan.

d. Perulangan Pelatihan

Lakukan pengulangan selama jumlah epoch yang ditentukan. Lakukan forward pass untuk mendapatkan nilai prediksi, hitung kerugian, lakukan backward pass untuk menghitung gradien, dan perbarui parameter model menggunakan pengoptimal. Nolkan gradien sebelum langkah berikutnya.

e. Visualisasi

Plot titik data asli dengan warna merah (X_numpy, y_numpy) dan garis regresi yang diprediksi (X_numpy, prediksi) menggunakan Matplotlib.

8. Logistic Regression

8.1. Penjelasan Kode

a. Persiapan Data

Muat dataset kanker payudara dari sklearn.datasets. Pisahkan data ke dalam set pelatihan dan pengujian menggunakan train_test_split. Standarisasi fitur menggunakan StandardScaler dari sklearn.preprocessing.

Ubah larik NumPy menjadi tensor PyTorch dan bentuk ulang tensor target.

b. Definisi Model

Tentukan model regresi logistik dengan membuat kelas Model yang diwarisi dari nn.Module. Dalam konstruktor, inisialisasi lapisan linear (nn.Linear) dengan ukuran input yang sesuai (n_input_features) dan ukuran output (1). Dalam metode forward, terapkan fungsi sigmoid pada keluaran lapisan linier dan kembalikan nilai prediksi.

c. Kerugian dan Pengoptimalisasi

Tetapkan jumlah epoch (num_epochs) dan laju pembelajaran (learning_rate). Tetapkan fungsi kerugian sebagai entropi silang biner (nn.BCELoss). Inisialisasi pengoptimal sebagai Stochastic Gradient Descent (torch.optim.SGD) dan masukkan parameter model dan laju pembelajaran.

d. Perulangan Pelatihan

Lakukan iterasi selama jumlah epoch yang ditentukan. Lakukan forward pass untuk mendapatkan nilai prediksi, hitung loss menggunakan binary cross-entropy loss, lakukan backward pass untuk menghitung gradien, dan perbarui parameter model menggunakan pengoptimal. Nolkan gradien sebelum langkah berikutnya.

e. Evaluasi

Setelah pelatihan, evaluasi kinerja model pada set pengujian. Lakukan forward pass pada set pengujian dan bulatkan nilai prediksi ke bilangan bulat terdekat (0 atau 1). Hitung akurasi dengan membandingkan prediksi yang dibulatkan dengan label yang sebenarnya.

Kode ini melatih model regresi logistik pada dataset kanker payudara dan mencetak kerugian pada setiap epoch. Kemudian mengevaluasi akurasi model pada set pengujian. Tujuannya adalah untuk mengklasifikasikan sampel kanker payudara menjadi ganas (1) atau jinak (0) berdasarkan fitur-fitur yang disediakan.

9. Dataset and Dataloader

9.1. Penjelasan Dataset dan Dataloader

Kelas torch.utils.data.DataLoader dan dataset kustom di PyTorch menyediakan cara yang mudah untuk memuat dan mengulang set data yang besar selama pelatihan. DataLoader menangani pemuatan data dalam batch dan juga dapat melakukan pengacakan dan pemuatan data paralel menggunakan beberapa pekerja.

Dataloader menyediakan cara yang efisien untuk menangani kumpulan data besar yang mungkin tidak muat dalam memori sekaligus. Dengan membagi data ke dalam kelompok yang lebih kecil, kita dapat memprosesnya secara berulang selama pelatihan dan memanfaatkan paralelisme. Dataloader mengabstraksikan kompleksitas pemuatan dan

pengelolaan data, sehingga memungkinkan integrasi yang mulus ke dalam loop pelatihan.

9.2. Penjelasan Kode

a. Dataset Kustom
   - Kelas WineDataset mewarisi dari kelas torch.utils.data.Dataset dan menimpa tiga metode:__init__,__getitem__, dan__len__.
   - Dalam metode __init__, dataset anggur dimuat menggunakan NumPy (np.loadtxt). Fitur dan label disimpan sebagai tensor PyTorch (self.x_data dan self.y_data).
   - Metode__getitem__memungkinkan pengindeksan dataset untuk mengambil sampel. Metode ini mengembalikan sebuah tupel fitur dan label untuk indeks yang ditentukan.
   - Metode__len__mengembalikan jumlah total sampel dalam dataset.

b. Membuat Dataset
   - Sebuah instance dari kelas WineDataset dibuat, yang mewakili dataset anggur.
   - Mengakses sampel tertentu dari dataset menggunakan pengindeksan (dataset[0]) akan mengembalikan tupel fitur dan label.

c. Inisialisasi Dataloader:
   - Dataloader diinisialisasi menggunakan kelas DataLoader.
   - Instance dataset (kumpulan data) diberikan sebagai argumen.
   - Argumen tambahan seperti batch_size, shuffle, dan num_workers dapat ditentukan.
   - Dalam kode ini, batch_size = 4 berarti empat sampel akan dimuat dalam setiap iterasi, shuffle = True mengacak data sebelum setiap epoch, dan num_workers = 2 memungkinkan pemuatan data secara paralel menggunakan dua subproses.

d. Iterasi ulang Dataloader
   - Dataloader dapat diperlakukan sebagai iterator, dan iter(train_loader) membuat sebuah iterator.
   - next(dataiter) mengembalikan kumpulan sampel berikutnya dari pemuat data.
   - Di dalam kode, sekumpulan fitur dan label secara acak diambil menggunakan next(dataiter), dan bentuknya dicetak.

e. Perulangan Pelatihan
   - Sebuah loop pelatihan tiruan disediakan untuk tujuan demonstrasi.
   - Jumlah epoch dan iterasi dihitung berdasarkan ukuran dataset dan ukuran batch.
   - Loop pelatihan mengulangi pemuat data dan mencetak informasi tentang epoch, langkah, dan bentuk input dan label saat ini.

f. Menggunakan Dataset Bawaan

- Kode ini juga mendemonstrasikan penggunaan dataset bawaan (torchvision.datasets.MNIST) yang tersedia di modul torchvision.datasets.
- Dataset MNIST dimuat dengan transformasi yang ditentukan (torchvision.transforms.ToTensor()) dan parameter lainnya.
- Pemuat data diinisialisasi dengan dataset MNIST, dan sekumpulan input dan target secara acak diambil dan dicetak.

10. Dataset Transforms

10.1. Penjelasan Dataset Transforms

Transformasi dataset di PyTorch mengacu pada operasi atau modifikasi yang diterapkan pada sampel dalam dataset. Transformasi dapat digunakan untuk melakukan praproses data, menambah dataset, atau mengonversi data ke format tertentu sebelum dimasukkan ke dalam model selama pelatihan atau evaluasi. PyTorch menyediakan berbagai macam transformasi bawaan yang dapat diterapkan pada gambar, tensor, larik, atau data khusus.

10.2. Penjelasan Kode

a. Dataset Khusus dengan Transformasi:
   - Kelas WineDataset telah didefinisikan, yang mewakili dataset anggur.
   - Kelas ini memiliki parameter transformasi opsional yang memungkinkan penerapan transformasi ke dataset.
   - Dalam metode_init_, dataset wine dimuat menggunakan NumPy (np.loadtxt), dan fitur serta label disimpan sebagai larik (self.x_data dan self.y_data).
   - Metode_getitem____mengambil sampel dari dataset dan menerapkan transformasi yang ditentukan jika ada.
   - Metode_len____mengembalikan jumlah total sampel dalam kumpulan data.

b. Transformasi Khusus
   - Dua transformasi kustom diimplementasikan sebagai kelas yang terpisah: ToTensor dan MulTransform.
   - Transformasi ToTensor mengubah ndarrays menjadi tensor PyTorch menggunakan torch.from_numpy.
   - Transformasi MulTransform mengalikan data input dengan faktor tertentu.

c. Menerapkan Transformasi ke Dataset:
   - Contoh transformasi kustom dapat diteruskan ke konstruktor WineDataset untuk menerapkan transformasi yang diinginkan ke dataset.
   - Dalam kode, kombinasi transformasi yang berbeda didemonstrasikan, termasuk tidak ada transformasi, transformasi tensor, dan kombinasi transformasi tensor dan transformasi perkalian.

d. Keluaran dan Hasil

- Kode mencetak jenis dan nilai fitur dan label untuk setiap kasus.
- Keluarannya menunjukkan efek transformasi pada sampel dataset.

11. Softmax dan Cross Entropy

11.1. Penjelasan Softmax dan Cross Entropy

Fungsi Softmax

Fungsi softmax biasanya digunakan dalam pembelajaran mesin untuk mengubah skor mentah atau logit menjadi probabilitas. Fungsi ini mengambil vektor input dan mengubahnya sedemikian rupa sehingga setiap elemen berada dalam rentang [0, 1] dan jumlah semua elemen adalah 1. Fungsi softmax didefinisikan sebagai berikut:

softmax(x_i) = exp(x_i) / sum(exp(x_j)) untuk semua j

Fungsi softmax memungkinkan kita untuk menginterpretasikan output dari sebuah model sebagai distribusi probabilitas dari beberapa kelas. Fungsi ini sering digunakan sebagai fungsi aktivasi akhir dalam masalah klasifikasi multikelas.

Cross Entropy Loss

Cross entropy loss, atau log loss, adalah fungsi kerugian yang banyak digunakan dalam pembelajaran mesin untuk mengevaluasi kinerja model klasifikasi. Fungsi ini mengukur ketidaksamaan antara distribusi probabilitas yang diprediksi dan distribusi label yang sebenarnya. Kehilangan entropi silang didefinisikan sebagai berikut:

cross_entropy(aktual, prediksi) = -jumlah(aktual * log(prediksi))

- Parameter aktual mewakili distribusi probabilitas yang sebenarnya (sering kali dikodekan satu-panas).
- Parameter predicted mewakili distribusi probabilitas yang diprediksi (biasanya diperoleh dari fungsi softmax).

Kerugian entropi silang menghukum model ketika probabilitas yang diprediksi menyimpang dari label yang sebenarnya. Hal ini mendorong model untuk memberikan probabilitas yang lebih tinggi ke kelas yang benar.

11.2. Penjelasan Kode

- Kode ini pertama-tama mendemonstrasikan implementasi softmax menggunakan NumPy dan PyTorch. Kode ini menerapkan fungsi softmax ke sebuah vektor input sampel dan mencetak probabilitas yang dihasilkan.
- Selanjutnya, kode tersebut mengimplementasikan fungsi cross-entropy menggunakan NumPy. Kode ini menghitung kerugian antara dua set probabilitas yang diprediksi (Y_pred_good dan Y_pred_bad) dan distribusi yang sebenarnya (Y).
- Kode ini kemudian menampilkan implementasi PyTorch untuk cross-entropy menggunakan kelas nn.CrossEntropyLoss. Kode ini membuat sebuah instance dari kelas ini dan menghitung kerugian antara skor prediksi (Y_pred_good dan Y_pred_bad) dan label

yang sebenarnya (Y).

- Kode ini menunjukkan bagaimana cara mendapatkan label kelas yang diprediksi dengan menemukan indeks nilai maksimum dalam skor yang diprediksi menggunakan fungsi torch.max.
- Selain itu, kode tersebut mengilustrasikan cara menghitung kerugian cross-entropy untuk sekumpulan sampel, di mana target (Y) dan skor yang diprediksi (Y_pred_good dan Y_pred_bad) disediakan untuk beberapa sampel.
- Terakhir, kode ini menyajikan dua contoh pendefinisian model jaringan syaraf, satu untuk klasifikasi biner (NeuralNet1) dan satu lagi untuk klasifikasi multikelas (NeuralNet2). Untuk klasifikasi biner, fungsi aktivasi sigmoid diterapkan pada lapisan keluaran, sedangkan untuk klasifikasi multikelas, lapisan keluaran memberikan skor mentah tanpa aktivasi softmax.

Fungsi kerugian nn.BCELoss (Binary Cross Entropy) digunakan untuk model klasifikasi biner, sedangkan fungsi kerugian nn.CrossEntropyLoss, yang secara internal menerapkan softmax, digunakan untuk model klasifikasi multikelas.

12. Activation Functions

12.1. Penjelasan Activation Functions

Fungsi aktivasi adalah komponen penting dari neural network. Fungsi aktivasi memperkenalkan non-linearitas pada model, sehingga memungkinkan model untuk mempelajari pola-pola yang kompleks dan membuat prediksi pada berbagai macam input. Fungsi aktivasi diterapkan pada output setiap neuron dalam lapisan neural network, mengubahnya menjadi representasi yang lebih bermakna.

Pilihan fungsi aktivasi berdampak pada dinamika pembelajaran jaringan, kemampuannya untuk mendekati fungsi yang kompleks, dan rentang output neuron. Beberapa fungsi aktivasi populer yang biasa digunakan dalam neural network, termasuk softmax, sigmoid, tanh, ReLU, dan leaky ReLU.

12.2. Penjelasan Kode

a. Aktivasi Softmax
   - Kode ini menerapkan fungsi aktivasi softmax ke tensor PyTorch x menggunakan torch.softmax(x, dim=0).
   - Kode ini juga menunjukkan cara alternatif untuk menerapkan softmax menggunakan modul nn.Softmax.
   - Fungsi softmax menormalkan tensor sepanjang dimensi yang ditentukan (dim=0) untuk mendapatkan distribusi probabilitas.

b. Aktivasi Sigmoid
   - Kode ini menerapkan fungsi aktivasi sigmoid pada tensor PyTorch x menggunakan torch.sigmoid(x).
   - Kode ini juga mendemonstrasikan penggunaan modul nn.Sigmoid.

- Fungsi sigmoid memetakan elemen-elemen tensor ke dalam rentang (0, 1), yang merepresentasikan probabilitas atau aktivasi biner.

c. Aktivasi Tanh
   - Kode ini menerapkan fungsi aktivasi tangen hiperbolik (tanh) ke tensor PyTorch x menggunakan torch.tanh(x).
   - Kode ini juga menunjukkan penggunaan modul nn.tanh.
   - Fungsi tanh memadatkan nilai tensor antara -1 dan 1, memberikan aktivasi negatif yang lebih kuat dibandingkan dengan sigmoid.

d. Aktivasi ReLU
   - Kode ini menerapkan fungsi aktivasi rectified linear unit (ReLU) ke tensor PyTorch x menggunakan torch.relu(x).
   - Kode ini juga mendemonstrasikan penggunaan modul nn.ReLU.
   - Fungsi ReLU menetapkan nilai negatif pada tensor menjadi 0, sambil mempertahankan nilai positif.

e. Aktivasi Leaky ReLU
   - Kode ini menerapkan fungsi aktivasi leaky ReLU ke tensor PyTorch x menggunakan F.leaky_relu(x).
   - Kode ini juga menunjukkan penggunaan modul nn.LeakyReLU.
   - Fungsi leaky ReLU berperilaku seperti ReLU tetapi memungkinkan kemiringan negatif yang kecil untuk input negatif, mencegah neuron mati.

f. Implementasi Neural Network
   - Kode ini menyediakan dua opsi untuk mengimplementasikan jaringan syaraf menggunakan fungsi aktivasi.
   - Opsi 1 menunjukkan penggunaan fungsi aktivasi sebagai instance nn.Module yang terpisah di dalam metode__init__dan forward pada neural network.
   - Opsi 2 menunjukkan bagaimana fungsi aktivasi dapat digunakan secara langsung di dalam forward pass tanpa instance nn.Module yang terpisah.
   - Kedua opsi tersebut membuat neural network dengan lapisan linier, fungsi aktivasi, dan aktivasi sigmoid pada lapisan output.

13. Feed Forward

13.1. Penjelasan Feed Forward

Neural network feed-forward, juga dikenal sebagai multilayer perceptron (MLP), adalah jenis neural network tiruan di mana informasi mengalir dalam satu arah, dari lapisan input ke lapisan output. Jaringan ini terdiri dari beberapa lapisan node (neuron) yang saling terhubung yang menerapkan bobot pada data input, melakukan komputasi menggunakan fungsi aktivasi, dan meneruskan hasilnya ke lapisan berikutnya.

Setiap lapisan dalam jaringan kecuali lapisan input disebut lapisan tersembunyi. Jaringan feed-forward dilatih menggunakan algoritma pembelajaran terawasi untuk memetakan data input ke output yang diinginkan.

13.2. Penjelasan Kode

- Kode dimulai dengan mengimpor library yang diperlukan, termasuk PyTorch, torchvision, dan matplotlib.
- Konfigurasi perangkat diatur ke GPU (cuda) jika tersedia; jika tidak, maka akan kembali ke CPU (cpu).
- Hiperparameter seperti ukuran input, ukuran tersembunyi, jumlah kelas, jumlah epoch, ukuran batch, dan laju pembelajaran didefinisikan.
- Dataset MNIST dimuat menggunakan torchvision dan ditransformasikan menjadi tensor.
- Pemuat data dibuat untuk memuat dataset pelatihan dan pengujian dalam beberapa kelompok untuk pelatihan yang efisien.
- Kode menampilkan enam contoh gambar dari dataset uji menggunakan Matplotlib.
- Model neural network didefinisikan sebagai subkelas dari nn.Module.
- Ini memiliki satu lapisan tersembunyi dengan fungsi aktivasi ReLU dan lapisan keluaran.
- Metode maju mendefinisikan lintasan maju dari jaringan.
- Sebuah contoh model neural network dibuat dan dipindahkan ke perangkat yang ditentukan (GPU atau CPU).
- Fungsi kerugian (cross-entropy) dan pengoptimal (Adam) didefinisikan.
- Model dilatih menggunakan loop bersarang selama beberapa epoch dan kumpulan data:
- Gambar dan label input dibentuk ulang dan dipindahkan ke perangkat.
- Penerusan dilakukan, dan kerugian dihitung.
- Backpropagation digunakan untuk menghitung gradien dan memperbarui parameter model.
- Kerugian dicetak pada langkah-langkah tertentu selama pelatihan.
- Setelah pelatihan, model diuji pada dataset uji:
- Gradien dinonaktifkan karena tidak ada backpropagation yang diperlukan.
- Label yang diprediksi dibandingkan dengan label yang sebenarnya untuk menghitung akurasi.
- Akurasi model pada dataset uji dicetak.

14. CNN

14.1. Penjelasan CNN

Convolutional Neural Network (CNN) adalah model pembelajaran mendalam yang dirancang khusus untuk menganalisis data visual, seperti gambar. Model ini sangat cocok untuk tugas-tugas seperti klasifikasi gambar, deteksi objek, dan segmentasi gambar. CNN terinspirasi oleh organisasi dan fungsi korteks

visual pada hewan. CNN menggunakan lapisan konvolusi untuk secara otomatis mempelajari representasi hirarkis fitur dari data piksel mentah.

CNN terdiri dari beberapa lapisan, termasuk lapisan konvolusional, lapisan penyatuan, dan lapisan yang terhubung sepenuhnya. Lapisan convolutional menerapkan filter convolutional (kernel) pada data input, mengekstraksi fitur lokal melalui konvolusi spasial. Lapisan pooling mengurangi dimensi spasial dari fitur-fitur, mempertahankan karakteristik esensialnya. Lapisan yang terhubung sepenuhnya menghubungkan semua neuron dari satu lapisan ke neuron dari lapisan berikutnya, memungkinkan pembelajaran dan klasifikasi fitur tingkat tinggi.

14.2. Penjelasan Kode

- Kode dimulai dengan mengimpor library yang diperlukan, termasuk PyTorch, torchvision, matplotlib, dan numpy.
- Konfigurasi perangkat diatur ke GPU (cuda) jika tersedia; jika tidak, maka akan kembali ke CPU (cpu).
- Hiperparameter seperti jumlah epoch, ukuran batch, dan laju pembelajaran ditentukan.
- Dataset CIFAR-10 dimuat, yang terdiri dari gambar berwarna yang termasuk dalam sepuluh kelas yang berbeda.
- Pemuat data dibuat untuk memuat dataset pelatihan dan pengujian dalam beberapa kelompok untuk pelatihan yang efisien.
- Fungsi imshow didefinisikan untuk menampilkan gambar dari dataset.
- Beberapa gambar pelatihan acak diambil, dan fungsi imshow digunakan untuk menampilkannya.
- Kelas ConvNet didefinisikan sebagai subkelas dari nn.Module.
- Kelas ini berisi lapisan konvolusi, lapisan penyatuan, dan lapisan yang terhubung sepenuhnya.
- Metode forward mendefinisikan lintasan maju jaringan.
- Contoh model CNN dibuat dan dipindahkan ke perangkat yang ditentukan.
- Fungsi kerugian (cross-entropy) dan pengoptimal (Stochastic Gradient Descent) didefinisikan.
- Model dilatih menggunakan loop bersarang di atas epoch dan kumpulan data:
- Gambar dan label input dipindahkan ke perangkat.
- Penerusan dilakukan, dan kerugian dihitung.
- Backpropagation digunakan untuk menghitung gradien dan memperbarui parameter model.
- Kerugian dicetak pada langkah-langkah tertentu selama pelatihan.
- Setelah pelatihan, model diuji pada dataset uji:
- Gradien dinonaktifkan karena tidak ada backpropagation yang diperlukan.

- Label yang diprediksi dibandingkan dengan label yang sebenarnya untuk menghitung akurasi.
- Akurasi keseluruhan model dan akurasi untuk setiap kelas dicetak.

15. Transfer Learning

15.1. Penjelasan Transfer Learning

Transfer learning adalah teknik pembelajaran mesin di mana model yang telah dilatih sebelumnya, yang telah dilatih pada kumpulan data yang besar, digunakan sebagai titik awal untuk menyelesaikan tugas baru yang terkait. Alih-alih melatih model dari awal pada tugas baru, pembelajaran transfer memanfaatkan pengetahuan yang dipelajari dari tugas asli untuk meningkatkan kinerja dan mengurangi waktu pelatihan pada tugas baru. Model yang telah dilatih sebelumnya menangkap fitur dan pola umum dari dataset asli, yang dapat bermanfaat untuk tugas-tugas serupa.

Dalam kode yang disediakan, pembelajaran transfer diterapkan menggunakan model ResNet-18, yang telah dilatih sebelumnya pada dataset ImageNet berskala besar. Lapisan akhir yang terhubung sepenuhnya dari model ResNet-18 dimodifikasi agar sesuai dengan jumlah kelas dalam tugas baru. Kode ini mendemonstrasikan dua skenario pembelajaran transfer: menyempurnakan seluruh model dan menggunakan model sebagai ekstraktor fitur tetap.

15.2. Penjelasan Kode

- Kode dimulai dengan mengimpor library yang diperlukan, termasuk torch, torchvision, dan modul terkait lainnya.
- Nilai rata-rata dan standar deviasi ditentukan untuk menormalkan gambar input selama transformasi data.
- Transformasi data ditentukan untuk set pelatihan dan validasi, termasuk pemangkasan acak, pembalikan, pengubahan ukuran, dan normalisasi.
- Direktori data ditetapkan, dan set data ImageFolder dibuat untuk set pelatihan dan validasi.
- Pemuat data dibuat untuk memuat dataset pelatihan dan validasi secara berkelompok untuk pelatihan yang efisien.
- Ukuran dataset dan nama kelas ditentukan dari set pelatihan.
- Perangkat diatur ke GPU (cuda: 0) jika tersedia; jika tidak, perangkat akan kembali ke CPU (cpu).
- Fungsi imshow didefinisikan untuk menampilkan gambar dari kumpulan data.
- Sekumpulan gambar pelatihan dan kelas yang sesuai diambil, dan fungsi imshow digunakan untuk menampilkannya.
- Fungsi train_model didefinisikan untuk melatih model menggunakan parameter yang disediakan.
- Di dalam fungsi train_model, bobot model, akurasi, dan salinan

bobot terbaik diinisialisasi.

- Perulangan pelatihan dimulai selama jumlah epoch yang ditentukan.
- Dalam setiap epoch, ada dua fase: "latih" dan "val" (validasi).
- Untuk setiap fase, model diatur ke mode yang sesuai (train atau eval).
- Kerugian yang berjalan dan jumlah prediksi yang benar diinisialisasi untuk fase saat ini.
- Data diulang secara bertahap, dan input serta label dipindahkan ke perangkat.
- Penerusan dilakukan, dan kerugian dihitung.
- Jika fase adalah "latih", backpropagation dilakukan, dan parameter pengoptimal diperbarui.
- Kehilangan yang sedang berjalan dan prediksi yang benar diperbarui untuk fase saat ini.
- Kehilangan epoch dan akurasi dihitung untuk fase saat ini.
- Kehilangan epoch dan akurasi dicetak untuk fase saat ini.
- Jika fase adalah "val" dan akurasinya lebih baik daripada akurasi terbaik sebelumnya, bobot terbaik model diperbarui.
- Setelah setiap epoch, waktu pelatihan dan akurasi validasi terbaik dicetak.
- Model dimuat dengan bobot terbaik yang diperoleh selama pelatihan.
- Model yang telah dilatih dikembalikan.
- Kode ini melanjutkan untuk mendemonstrasikan dua skenario pembelajaran transfer menggunakan model ResNet-18.
- Pada skenario pertama, seluruh model ResNet-18 dimuat dengan bobot yang telah dilatih sebelumnya, dan lapisan akhir yang terhubung sepenuhnya dimodifikasi untuk memiliki dua kelas output untuk tugas baru.
- Model dipindahkan ke perangkat yang ditentukan.
- Fungsi kerugian (cross-entropy) dan pengoptimal (SGD) ditentukan.
- Penjadwal laju pembelajaran (StepLR) dibuat untuk meluruhkan laju pembelajaran dari waktu ke waktu.
- Fungsi train_model dipanggil untuk melatih model pada tugas baru dengan menggunakan fine-tuning.
- Pada skenario kedua, model ResNet-18 baru dimuat, dan semua parameternya diatur untuk tidak memerlukan gradien (dibekukan) kecuali untuk lapisan akhir yang terhubung sepenuhnya.
- Model baru dipindahkan ke perangkat yang ditentukan.
- Fungsi kerugian, pengoptimal, dan penjadwal laju pembelajaran ditentukan.
- Fungsi train_model dipanggil untuk melatih model pada tugas baru

sambil menjaga lapisan konvolusi yang sudah dilatih tetap.

16. Tensorboard

16.1. Penjelasan Tensorboard

Tensorboard adalah alat visualisasi berbasis web yang disediakan oleh TensorFlow yang memungkinkan pengguna untuk menganalisis dan memvisualisasikan eksperimen pembelajaran mesin secara interaktif. Tensorboard menyediakan serangkaian visualisasi, termasuk metrik skalar, histogram, gambar, dan grafik, untuk melacak dan memahami proses pelatihan dan kinerja model. Tensorboard membantu para peneliti dan pengembang mendapatkan wawasan tentang model mereka, membandingkan eksperimen, dan men-debug masalah potensial.

16.2. Penjelasan Kode

- Kode dimulai dengan mengimpor yang diperlukan, termasuk torch, torch.nn, torchvision, transform, dan matplotlib.pyplot. Selain itu, kode ini juga mengimpor modul-modul yang diperlukan dari torch.utils.tensorboard untuk menggunakan Tensorboard.
- Modul ini memeriksa ketersediaan perangkat GPU dan menetapkannya ke variabel perangkat jika tersedia; jika tidak, modul ini akan kembali menggunakan CPU.
- Menetapkan hyperparameter seperti input_size, hidden_size, num_classes, num_epochs, batch_size, dan learning_rate.
- Memuat dataset MNIST menggunakan torchvision.datasets.MNIST, menentukan direktori data, transformasi, dan apakah itu set pelatihan atau pengujian.
- Membuat pemuat data untuk set data pelatihan dan pengujian menggunakan torch.utils.data.DataLoader, dengan menentukan set data, ukuran batch, dan opsi pengacakan.
- Mengambil sekumpulan gambar dan label dari dataset pengujian menggunakan iter(test_loader) dan next(examples).
- Menampilkan kisi-kisi 6 gambar contoh dari kumpulan menggunakan matplotlib.pyplot.
- Membuat objek SummaryWriter dari torch.utils.tensorboard untuk menulis log Tensorboard, dengan menetapkan direktori log sebagai 'runs/mnist1'.
- Menambahkan kisi-kisi gambar contoh ke Tensorboard menggunakan writer.add_image.
- Mendefinisikan kelas model jaringan syaraf NeuralNet yang diwarisi dari torch.nn.Module. Terdiri dari lapisan input (l1), fungsi aktivasi ReLU (relu), dan lapisan output (l2).
- Mengimplementasikan metode penerusan ke depan di kelas NeuralNet, yang mendefinisikan aliran data melalui model.
- Menginisialisasi instance kelas NeuralNet dengan ukuran

input, ukuran tersembunyi, dan jumlah kelas yang ditentukan, dan memindahkannya ke perangkat.

- Mendefinisikan fungsi loss sebagai cross-entropy loss (nn.CrossEntropyLoss) dan pengoptimal sebagai Adam (torch.optim.Adam) dengan parameter model dan laju pembelajaran.
- Menambahkan grafik model ke Tensorboard menggunakan writer.add_graph, memberikan contoh model dan contoh data yang dibentuk ulang agar sesuai dengan ukuran input.
- Menginisialisasi variabel untuk melacak kerugian yang sedang berjalan dan mengoreksi prediksi selama pelatihan.
- Memasuki loop pelatihan, mengulang jumlah epoch yang ditentukan.
- Di dalam setiap epoch, mengulang sekumpulan gambar dan label dari pemuat data pelatihan.
- Membentuk ulang gambar input dan memindahkannya ke perangkat.
- Melakukan penerusan maju, menghitung kerugian, dan melakukan backpropagasi untuk memperbarui parameter model.
- Melacak kehilangan yang sedang berjalan dan menghitung jumlah prediksi yang benar.
- Mencetak kerugian secara berkala (jika (i+1) % 100 == 0).
- Menambahkan running loss dan akurasi ke Tensorboard menggunakan writer.add_scalar.
- Setelah pelatihan, masuk ke tahap pengujian dengan menonaktifkan komputasi gradien (torch.no_grad()).
- Menginisialisasi variabel untuk menghitung prediksi yang benar dan total sampel.
- Mengulangi kumpulan gambar dan label dari pemuat data pengujian.
- Melakukan penerusan dan menghitung prediksi dan akurasi.
- Mengumpulkan probabilitas kelas dan label untuk analisis selanjutnya.
- Menghitung dan mencetak akurasi jaringan pada gambar uji.
- Menambahkan kurva precision-recall ke Tensorboard untuk setiap kelas menggunakan writer.add_pr_curve.

17. Save dan Load Models

17.1. Penjelasan Save dan Load Models

Menyimpan dan memuat model adalah tugas penting dalam pembelajaran mesin dan pembelajaran mendalam. Hal ini melibatkan penyimpanan parameter model terlatih, status pengoptimal, dan informasi lain yang diperlukan ke dalam disk, sehingga model dapat digunakan di kemudian hari untuk inferensi, penyempurnaan, atau melanjutkan pelatihan.

Dalam PyTorch, ada beberapa cara untuk menyimpan dan memuat model. Kode yang Anda berikan menunjukkan dua pendekatan umum: menyimpan seluruh model dan hanya menyimpan kamus state.

17.2. Penjelasan Kode

a. Menyimpan Keseluruhan Model
   - Dengan "cara malas", Anda dapat menyimpan seluruh model dengan menggunakan torch.save(model, PATH), di mana model adalah contoh kelas model Anda dan PATH adalah jalur file untuk menyimpan model.
   - Untuk memuat model yang telah disimpan, Anda bisa menggunakan model = torch.load(PATH), yang memuat seluruh objek model. Ingatlah untuk memanggil model.eval() untuk mengatur model dalam mode evaluasi.
   - Pendekatan ini menyimpan arsitektur model dan parameter yang dipelajari, sehingga Anda dapat melanjutkan pelatihan atau melakukan inferensi tanpa mendefinisikan ulang struktur model.

b. Hanya Menyimpan State Dictionary
   - Pendekatan yang direkomendasikan adalah hanya menyimpan kamus keadaan model menggunakan torch.save(model.state_dict(), PATH). Di sini, model.state_dict() mengembalikan kamus yang berisi semua parameter model yang dapat dipelajari.
   - Untuk memuat kamus state yang tersimpan, Anda perlu membuat instance model dengan arsitektur yang sama dan kemudian memuat kamus state menggunakan model.load_state_dict(torch.load(PATH)). Ingatlah untuk memanggil model.eval() setelah pemuatan jika Anda bermaksud melakukan inferensi.
   - Pendekatan ini memisahkan arsitektur model dari parameternya, sehingga lebih fleksibel. Anda dapat dengan mudah beralih di antara arsitektur model yang berbeda atau memuat parameter ke dalam model yang sudah ada.

c. Menyimpan Checkpoints
   - Checkpoints digunakan untuk menyimpan status peralihan dari proses pelatihan, termasuk kamus status model, status pengoptimal, dan informasi lain yang relevan.
   - Kode berikut ini menunjukkan penyimpanan kamus checkpoint yang berisi epoch, status model, dan status pengoptimal menggunakan torch.save(checkpoint, FILE).
   - Untuk memuat checkpoint, Anda dapat menggunakan checkpoint = torch.load(FILE) dan kemudian memuat model dan status pengoptimal menggunakan model.load_state_dict(checkpoint['model_state']) dan optimizer.load_state_dict(checkpoint['optim_state']).
   - Pendekatan ini berguna ketika Anda ingin melanjutkan pelatihan dari epoch tertentu atau mengembalikan status model dan pengoptimal untuk evaluasi atau penyempurnaan lebih lanjut.

d. Menghemat GPU/CPU

- Kode ini juga menunjukkan cara menyimpan dan memuat model saat menggunakan GPU. Ini mencakup skenario di mana model disimpan di GPU dan perlu dimuat di CPU, disimpan dan dimuat di GPU, atau disimpan di CPU dan dimuat di GPU.
- Untuk menyimpan model pada perangkat tertentu, Anda dapat menggunakan model.to(device) untuk memindahkan model ke perangkat tersebut sebelum memanggil torch.save(model.state_dict(), PATH).
- Ketika memuat model, Anda dapat memberikan argumen map_location untuk menentukan perangkat target atau menggunakan torch.device("cuda") untuk memuatnya ke perangkat default

III. Kesimpulan

Dalam laporan teknis ini, kami membahas berbagai topik dalam pembelajaran mesin dan pembelajaran mendalam menggunakan PyTorch. Kami memulai dengan dasar-dasar tensor dan instalasi, diikuti dengan mengeksplorasi autograd dan backpropagation untuk komputasi gradien yang efisien.

Selain itu, dibahas juga tentanggradient descent dan variannya untuk mengoptimalkan model, dan memberikan wawasan tentang pipeline pelatihan, termasuk pemuatan data dan prapemrosesan dengan Dataset dan DataLoader.

Regresi linier, regresi logistik, dan transformasi dataset juga dibahas, bersama dengan softmax dan cross-entropy untuk tugas-tugas klasifikasi. Fungsi aktivasi, neural network tiruan umpan maju, dan neural network tiruan konvolusi (CNN) juga dijelaskan, sehingga pembaca dapat mengembangkan dan melatih model.

Kemudian, kami mengeksplorasi pembelajaran transfer untuk meningkatkan model yang telah dilatih sebelumnya dan memperkenalkan TensorBoard untuk pemantauan dan visualisasi. Terakhir, kami membahas penyimpanan dan pemuatan model untuk penerapan.

Kami berharap laporan ini telah memberikan wawasan dan panduan yang berharga, sehingga pembaca dapat dengan percaya diri melanjutkan upaya pembelajaran mesin mereka menggunakan PyTorch.