Technical Report Machine Learning

"DEEP LEARNING WITH PYTORCH"

Oleh:

Nama : Juan Meta Sirgianto

NIM : 1103202092

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
2023

I. PENDAHULUAN

Deep learning merupakan salah satu cabang dalam bidang kecerdasan buatan (artificial intelligence) yang menekankan pada penggunaan jaringan saraf tiruan (artificial neural networks) yang dalam dan kompleks. Teknologi ini telah mengalami perkembangan pesat dalam beberapa tahun terakhir dan telah menjadi salah satu pendorong utama dalam banyak inovasi dan kemajuan di berbagai industri.

Deep learning memungkinkan komputer untuk belajar secara mandiri dari data yang diberikan, dengan menggunakan representasi yang mendalam dan hierarkis dari informasi tersebut. Hal ini memungkinkan komputer untuk mengenali pola yang kompleks dan melakukan tugas yang sebelumnya hanya bisa dilakukan oleh manusia, seperti pengenalan wajah, pemrosesan bahasa alami, pengenalan suara, pengenalan objek, dan banyak lagi.

Laporan ini bertujuan untuk memberikan pemahaman yang komprehensif tentang konsep dasar dalam deep learning menggunakan PyTorch. Kami akan mulai dengan menjelaskan tensor, yang merupakan struktur data fundamental dalam PyTorch. Selanjutnya, kami akan membahas topik seperti autograd, backpropagation, dan gradient descent yang kunci dalam pelatihan model deep learning. Kami juga akan mencakup topik-topik lain seperti fungsi aktivasi, arsitektur jaringan saraf tiruan, dan penggunaan dataset dalam PyTorch. Dengan menyajikan topik-topik ini, kami berharap laporan ini dapat memberikan pemahaman yang mendalam tentang konsep-konsep penting dalam deep learning dengan PyTorch, serta memberikan landasan yang kuat untuk eksplorasi lebih lanjut dalam bidang yang terus berkembang ini.

II. TENSOR

Tensor adalah struktur data fundamental dalam PyTorch dan deep learning secara umum. Secara sederhana, tensor dapat dianggap sebagai suatu array multidimensi yang dapat menyimpan data numerik. Tensor serupa dengan matriks dalam matematika, tetapi dapat memiliki lebih dari dua dimensi.

Setiap elemen dalam tensor memiliki posisi yang ditentukan oleh indeks multidimensi. Misalnya, tensor dua dimensi mirip dengan matriks dengan baris dan kolom, sedangkan tensor tiga dimensi mirip dengan kubus dengan panjang, lebar, dan tinggi.

Tensor dalam PyTorch dapat digunakan untuk menyimpan dan memanipulasi data numerik dalam konteks deep learning. Data ini dapat berupa input, parameter model, hasil prediksi, atau gradien yang digunakan dalam proses pembelajaran model.

Tensor dalam PyTorch juga mendukung komputasi GPU, yang memungkinkan percepatan yang signifikan dalam pelatihan model deep learning. Dengan menggunakan GPU, kita dapat melakukan operasi tensor secara paralel, meningkatkan kecepatan dan kinerja dalam proses deep learning.

III. AUTO GRAD

Autograd (automatic differentiation) adalah komponen kunci dalam framework PyTorch yang memungkinkan perhitungan diferensiasi otomatis. Autograd mengotomatisasi perhitungan gradien dalam deep learning, yang sangat penting dalam proses pelatihan model.

Dalam deep learning, perhitungan gradien adalah proses yang penting untuk mengoptimalkan parameter model. Gradien mengindikasikan arah dan tingkat perubahan fungsi objektif terhadap setiap parameter model, yang pada gilirannya digunakan untuk memperbarui parameter melalui metode optimisasi seperti gradient descent.

Autograd di PyTorch menyediakan mekanisme untuk secara otomatis menghitung gradien dari operasi yang diterapkan pada tensor. Ketika tensor ditandai sebagai memerlukan gradien (requires_grad=True), PyTorch akan melacak setiap operasi yang diterapkan pada tensor tersebut. Ini membentuk sebuah graf yang merepresentasikan hubungan antara tensor input, operasi, dan tensor output.

Autograd di PyTorch juga mendukung perhitungan gradien dengan metode chain rule dalam graf komputasi yang kompleks. Hal ini memungkinkan perhitungan gradien yang efisien bahkan pada model deep learning dengan banyak parameter.

Dengan adanya autograd, para pengembang dan peneliti dapat dengan mudah mengimplementasikan dan melatih model deep learning tanpa perlu secara manual menghitung dan mengimplementasikan gradien. Autograd mengurangi kesalahan manusia dan mempercepat pengembangan model, sehingga memungkinkan eksplorasi lebih lanjut dan inovasi dalam bidang deep learning.

IV. BACKPROPAGATION & GRADIENT DESCENT

Gradient descent adalah metode optimisasi yang digunakan untuk meminimalkan fungsi objektif (cost function) dalam proses pelatihan model dalam deep learning. Tujuan dari gradient descent adalah mencari nilai parameter yang menghasilkan nilai fungsi objektif yang optimal.

Pada dasarnya, gradient descent beroperasi dengan melakukan iterasi pada parameter model dan mengupdate nilainya berdasarkan gradien fungsi objektif terhadap parameter tersebut. Proses ini berulang hingga mencapai nilai parameter yang menghasilkan nilai fungsi objektif yang minimum atau konvergen.

Secara intuitif, gradient descent bergerak dalam arah kebalikan dari gradien (turunan) fungsi objektif. Jika gradien positif, parameter akan dikurangi agar fungsi objektif berkurang. Sebaliknya, jika gradien negatif, parameter akan ditambah agar fungsi objektif berkurang. Dengan demikian, gradient descent berusaha untuk menemukan titik minimum lokal atau global dalam ruang parameter model.

V. TRAINING PIPELINE

Training pipeline (pipa pelatihan) adalah serangkaian langkah yang dilakukan dalam proses pelatihan model dalam deep learning. Pipeline ini melibatkan tahap persiapan data, pembangunan model, pelatihan model, dan evaluasi kinerja model. Berikut adalah penjelasan singkat tentang setiap tahap dalam training pipeline:

1. Persiapan Data: Tahap ini melibatkan pengumpulan, pemrosesan, dan persiapan data untuk pelatihan model. Data dapat dibagi menjadi dataset pelatihan, validasi, dan pengujian. Dataset pelatihan digunakan untuk melatih model, dataset validasi digunakan untuk memonitor dan mengoptimalkan kinerja model selama pelatihan, dan dataset pengujian digunakan untuk menguji kinerja akhir model.
2. Pembangunan Model: Tahap ini melibatkan pemilihan arsitektur model dan konfigurasi parameter. Arsitektur model dapat mencakup berbagai jenis layer, fungsi aktivasi, dan parameter lainnya yang mempengaruhi cara model belajar dan melakukan prediksi. Model ini dibangun menggunakan framework deep learning seperti PyTorch.
3. Pelatihan Model: Tahap ini melibatkan pelatihan model menggunakan data pelatihan yang telah disiapkan. Proses ini melibatkan proses pengoptimalan, seperti gradient

descent, yang mengupdate parameter model berdasarkan gradien fungsi objektif. Pelatihan model dapat melibatkan pengaturan hiperparameter seperti learning rate, jumlah epoch, dan ukuran batch.

4. Evaluasi Kinerja: Tahap ini melibatkan evaluasi kinerja model yang telah dilatih menggunakan data validasi atau pengujian yang terpisah. Evaluasi kinerja melibatkan penggunaan metrik seperti akurasi, presisi, recall, dan F1-score untuk mengevaluasi sejauh mana model dapat menghasilkan prediksi yang akurat.
5. Penyetelan Model: Jika hasil evaluasi kinerja tidak memenuhi harapan, model dapat disetel ulang atau hiperparameter dapat disesuaikan untuk meningkatkan kinerja model. Proses ini melibatkan eksperimen dengan berbagai konfigurasi dan penyesuaian untuk mencapai hasil yang diinginkan.
6. Penggunaan Model: Setelah model telah dilatih dan kinerjanya dinilai memadai, model dapat digunakan untuk melakukan prediksi pada data baru. Model ini siap digunakan dalam produksi atau dalam pengaplikasian spesifik yang memerlukan kemampuan prediksi dari model.

Training pipeline merupakan alur kerja yang berulang dalam proses pengembangan dan iterasi model deep learning. Dengan melakukan siklus ini, model dapat terus diperbaiki dan disempurnakan seiring dengan kemajuan dalam pelatihan dan evaluasi.

VI. LINEAR REGRESSION

Linear regression (regresi linear) adalah salah satu metode yang paling sederhana dan umum digunakan dalam analisis regresi. Ini adalah teknik statistik yang digunakan untuk memodelkan hubungan linier antara variabel input (X) dan variabel target (y).

Tujuan regresi linear adalah untuk mempelajari persamaan garis lurus yang paling baik mendekati hubungan antara variabel input dan variabel target. Model regresi linear menggunakan pendekatan statistik untuk menentukan parameter yang optimal dalam persamaan garis, yaitu gradien (slope) dan intercept (intersep).

Secara matematis, persamaan regresi linear dapat ditulis sebagai:

$y = mx + c$

Di mana:

y adalah variabel target yang ingin diprediksi.

x adalah variabel input yang digunakan untuk memprediksi nilai y.

m adalah gradien atau slope yang menggambarkan kecuraman garis regresi.

c adalah intercept atau titik potong dengan sumbu y.

Proses pelatihan model regresi linear melibatkan estimasi atau perhitungan nilai optimal untuk gradien (m) dan intercept (c) berdasarkan data pelatihan yang ada. Ini dilakukan dengan menggunakan metode kuadrat terkecil (least squares method) atau metode lainnya untuk meminimalkan selisih antara nilai sebenarnya dan nilai yang diprediksi oleh model.

Setelah model regresi linear dilatih, itu dapat digunakan untuk melakukan prediksi nilai target (y) berdasarkan nilai input (x) yang baru atau belum dilihat sebelumnya. Dalam beberapa kasus, evaluasi kinerja model juga dilakukan menggunakan metrik seperti Mean Squared Error (MSE) atau R-squared untuk mengukur sejauh mana model dapat menjelaskan variasi dalam data.

VII. LOGISTIC REGRESSION

Logistic regression (regresi logistik) adalah metode statistik yang digunakan untuk memodelkan hubungan antara variabel input (X) dan variabel target biner (y). Tujuan regresi logistik adalah untuk memprediksi probabilitas kejadian suatu peristiwa berdasarkan variabel input.

Meskipun disebut "regresi," logistic regression sebenarnya adalah algoritma klasifikasi yang menggunakan fungsi logistik (sigmoid) untuk menghasilkan prediksi probabilitas. Regresi logistik dapat mengatasi masalah klasifikasi biner, di mana variabel target memiliki dua kemungkinan nilai, misalnya, 0 atau 1.

Fungsi logistik (sigmoid) dalam regresi logistik mengubah input linier menjadi output yang terbatas antara 0 dan 1. Persamaan regresi logistik dapat ditulis sebagai:

$p(y=1|X) = 1 / (1 + \exp(-z))$

Di mana:

$p(y=1|X)$ adalah probabilitas bahwa variabel target (y) memiliki nilai 1, diberikan nilai input (X).

z adalah kombinasi linier dari variabel input (X) dengan bobot (w) dan bias (b), yaitu $z = w^T X + b$.

Regresi logistik memiliki banyak aplikasi dalam berbagai bidang, termasuk prediksi risiko, analisis keuangan, pemodelan kesehatan, dan pengenalan pola. Ini juga sering digunakan sebagai bagian dari algoritma machine learning yang lebih kompleks, seperti dalam jaringan saraf tiruan atau dalam metode ensemble learning.

VIII. DATASET & DATALOADER

Dataset dan DataLoader adalah dua komponen penting dalam pemrosesan data dalam deep learning. Mereka digunakan untuk mengatur, memuat, dan memproses data yang akan digunakan dalam pelatihan model.

1. Dataset: Dataset merujuk pada kumpulan data yang digunakan untuk melatih atau menguji model deep learning. Dataset dapat terdiri dari pasangan data input dan target (supervised learning) atau hanya data input (unsupervised learning). Dataset juga dapat dibagi menjadi subset seperti dataset pelatihan, validasi, dan pengujian.
2. DataLoader: DataLoader adalah kelas yang digunakan untuk memuat data dari dataset dalam batch atau secara iteratif. DataLoader mengambil dataset sebagai masukan dan menyediakan antarmuka untuk mempermudah proses pengambilan batch data secara efisien. DataLoader memungkinkan pemrosesan paralel dan pengaturan beban kerja sehingga dapat mempercepat pelatihan model.

Dalam PyTorch, kita dapat menggunakan kelas DataLoader untuk mengatur proses pengambilan batch data dari dataset. DataLoader dapat dikonfigurasi dengan berbagai parameter seperti ukuran batch, pengacakan data, pemrosesan paralel menggunakan multiprocessing, dan banyak lagi. DataLoader menghasilkan batch data yang siap digunakan dalam pelatihan atau evaluasi model.

Dengan menggunakan Dataset dan DataLoader, pengguna dapat dengan mudah mengelola dan memproses data dalam deep learning. Mereka membantu dalam memisahkan data menjadi subset, memuat data secara efisien, mengelola ukuran batch, dan menyederhanakan tugas-tugas yang terkait dengan pemrosesan data. Ini memungkinkan pengguna untuk fokus pada pengembangan model dan pelatihan, sambil memastikan bahwa data dikelola dengan baik dan diolah secara efisien.

IX. DATASET TRANSFORM

Dataset transform dalam konteks deep learning merujuk pada serangkaian operasi yang diterapkan pada data dalam dataset untuk melakukan transformasi atau preprocessing sebelum data tersebut digunakan dalam pelatihan atau evaluasi model.

Transformasi dataset dapat melibatkan berbagai operasi seperti normalisasi, augmentasi data, cropping (pemotongan), pemutaran, pemindaian, dan transformasi lainnya. Tujuan transformasi ini adalah untuk meningkatkan kualitas dan keragaman data, mengurangi variabilitas, mengurangi overfitting, dan membantu model dalam mempelajari pola yang lebih baik.

Beberapa transformasi yang umum digunakan dalam PyTorch antara lain:

1. ToTensor: Transformasi ToTensor digunakan untuk mengubah data menjadi tensor. Hal ini penting karena model PyTorch bekerja dengan tensor sebagai struktur data dasar.

2. Normalize: Transformasi Normalize digunakan untuk normalisasi data dengan menyesuaikan rata-rata dan deviasi standar. Ini membantu mempercepat konvergensi model dan mempertahankan skala yang seragam antar fitur.

3. RandomHorizontalFlip dan RandomVerticalFlip: Transformasi ini secara acak membalikkan gambar secara horizontal atau vertikal. Ini dapat meningkatkan keragaman data dan membantu model dalam mempelajari invariansi terhadap pemutasian horizontal atau vertikal.

4. RandomCrop: Transformasi ini secara acak memotong bagian-bagian acak dari gambar. Ini dapat membantu meningkatkan keragaman data dan memastikan bahwa model dapat mengenali objek yang terletak di berbagai posisi dalam gambar.

5. Resize: Transformasi ini digunakan untuk mengubah ukuran gambar menjadi ukuran yang diinginkan. Ini penting ketika data dalam dataset memiliki ukuran yang berbeda-beda dan perlu diubah agar sesuai dengan ukuran yang diperlukan oleh model.

Dengan menerapkan transformasi dataset, pengguna dapat mempersiapkan data secara efisien dan efektif sebelum digunakan dalam pelatihan atau evaluasi model deep learning.

Transformasi dataset membantu meningkatkan kualitas data, mengurangi overfitting, dan memastikan bahwa model dapat belajar pola yang lebih baik dari data yang diberikan.

X. SOFTMAX & CROSSENTROPY

Softmax dan Cross Entropy adalah dua konsep yang sering digunakan dalam tahap akhir model deep learning untuk klasifikasi multi-kelas. Mereka berfungsi untuk menghitung probabilitas dan mengukur kesalahan prediksi model.

1. Softmax: Softmax adalah fungsi aktivasi yang mengubah keluaran model menjadi distribusi probabilitas yang valid. Ini digunakan untuk menghasilkan probabilitas kelas pada masalah klasifikasi multi-kelas. Softmax didefinisikan sebagai:

   softmax(x_i) = exp(x_i) / sum(exp(x_j)) untuk setiap x_i

   di mana x_i adalah keluaran model untuk kelas i, exp(x) adalah fungsi eksponensial, dan sum(exp(x_j)) adalah penjumlahan dari semua eksponensial keluaran model.

   Softmax memastikan bahwa probabilitas kelas yang dihasilkan oleh model adalah positif, berjumlah satu, dan mencerminkan kepercayaan relatif model terhadap setiap kelas.

2. Cross Entropy: Cross Entropy digunakan sebagai fungsi biaya (loss function) untuk membandingkan distribusi probabilitas yang dihasilkan oleh model (melalui softmax) dengan label sebenarnya dalam tugas klasifikasi multi-kelas.

   Cross Entropy Loss didefinisikan sebagai:

   cross_entropy(y, t) = - sum(t_i * log(y_i))

   di mana y adalah distribusi probabilitas yang dihasilkan oleh model untuk setiap kelas, t adalah vektor one-hot encoding dari label sebenarnya, sum adalah penjumlahan dari semua elemen, dan log adalah fungsi logaritma.

   Cross Entropy mengukur seberapa jauh probabilitas yang dihasilkan oleh model (y) dari probabilitas yang diharapkan (t). Semakin dekat distribusi probabilitas prediksi dengan distribusi probabilitas yang diharapkan, semakin rendah nilai Cross Entropy Loss.

   Tujuan dari pelatihan model adalah untuk mengoptimalkan parameter agar Cross Entropy Loss minimal, sehingga model dapat menghasilkan distribusi probabilitas yang mendekati distribusi probabilitas yang diharapkan.

Dalam konteks klasifikasi multi-kelas, softmax digunakan sebagai fungsi aktivasi pada lapisan output model untuk menghasilkan probabilitas kelas, sedangkan Cross Entropy Loss digunakan sebagai fungsi biaya untuk melatih model dengan membandingkan probabilitas yang dihasilkan dengan label sebenarnya. Dengan mengoptimalkan parameter model menggunakan teknik

seperti gradient descent, model dapat belajar untuk menghasilkan probabilitas yang akurat dan meminimalkan kesalahan prediksi.

XI. ACTIVATION FUNCTION

Fungsi aktivasi adalah fungsi matematika yang diterapkan pada output neuron dalam jaringan saraf tiruan. Fungsi ini memperkenalkan non-linearitas ke dalam jaringan dan memungkinkan model untuk mempelajari pola yang kompleks dalam data.

Berikut adalah beberapa fungsi aktivasi yang umum digunakan dalam deep learning:

1. ReLU (Rectified Linear Unit): Fungsi ReLU didefinisikan sebagai f(x) = max(0, x). Fungsi ini mengaktifkan neuron dengan mengeluarkan nilai positif, sedangkan nilai negatif dinonaktifkan (mengeluarkan nilai 0). ReLU adalah fungsi aktivasi yang paling umum digunakan karena sederhana, menghilangkan masalah gradien menghilang (vanishing gradient), dan efisien dalam komputasi.

2. Sigmoid: Fungsi sigmoid didefinisikan sebagai f(x) = 1 / (1 + exp(-x)). Fungsi ini menghasilkan output dalam rentang 0 hingga 1, yang dapat diinterpretasikan sebagai probabilitas. Sigmoid digunakan pada lapisan output dalam masalah klasifikasi biner atau masalah di mana output harus berada dalam rentang 0 hingga 1.

3. Tanh (Tangen Hiperbolik): Fungsi tanh didefinisikan sebagai f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x)). Fungsi ini menghasilkan output dalam rentang -1 hingga 1. Tanh mirip dengan sigmoid, tetapi rentangnya lebih luas dan simetris. Tanh sering digunakan pada lapisan tersembunyi dalam jaringan saraf.

4. Leaky ReLU: Leaky ReLU adalah variasi dari ReLU yang memperkenalkan gradien kecil untuk nilai negatif. Leaky ReLU didefinisikan sebagai f(x) = max(ax, x), di mana a adalah konstanta kecil. Hal ini membantu mengatasi masalah neuron yang "mati" (dead neurons) pada ReLU saat gradiennya negatif.

5. Softmax: Fungsi softmax digunakan pada lapisan output dalam masalah klasifikasi multi-kelas. Fungsi ini mengubah output neuron menjadi probabilitas kelas dengan memastikan bahwa probabilitas semua kelas jumlahnya sama dengan 1. Softmax digunakan bersama dengan fungsi biaya Cross Entropy untuk melatih model dan memperoleh distribusi probabilitas yang akurat untuk kelas-kelas yang mungkin.

Pemilihan fungsi aktivasi tergantung pada karakteristik masalah dan arsitektur jaringan. Pemilihan yang tepat dapat membantu jaringan untuk belajar dengan lebih baik dan menghasilkan prediksi yang akurat.

XII. FEEDFORWARD

Feedforward merupakan istilah yang merujuk pada arus data melalui jaringan saraf tiruan dalam satu arah, dari lapisan input ke lapisan output tanpa adanya siklus atau koneksi mundur. Ini adalah fase di mana model melakukan komputasi dan menghasilkan prediksi berdasarkan input yang diberikan.

Proses feedforward melibatkan beberapa tahap sebagai berikut:

1. Input Layer: Data input, seperti gambar atau teks, disajikan sebagai vektor fitur atau matriks ke lapisan input jaringan. Setiap node di lapisan input mewakili fitur atau atribut dari data yang akan diolah.

2. Hidden Layers: Setelah lapisan input, data mengalir ke serangkaian lapisan tersembunyi atau hidden layers. Setiap lapisan tersembunyi terdiri dari beberapa neuron atau unit yang mengimplementasikan fungsi aktivasi. Masing-masing neuron menerima masukan dari semua neuron di lapisan sebelumnya dan mengirimkan keluaran ke neuron di lapisan berikutnya.

3. Fungsi Aktivasi: Setiap neuron di lapisan tersembunyi dan lapisan output menggunakan fungsi aktivasi untuk mengubah masukan menjadi keluaran yang non-linear. Fungsi aktivasi membantu memperkenalkan non-linearitas ke dalam jaringan, memungkinkannya untuk mempelajari pola yang kompleks dalam data.

4. Bobot dan Bias: Setiap koneksi antara neuron di lapisan-lapisan memiliki bobot yang terkait. Bobot ini mengatur seberapa besar kontribusi setiap neuron pada lapisan sebelumnya terhadap neuron pada lapisan berikutnya. Selain itu, setiap neuron juga memiliki bias, yang merupakan konstanta yang ditambahkan ke input neuron sebelum aktivasi. Bobot dan bias ditentukan selama tahap pelatihan model untuk mencapai hasil yang diinginkan.

5. Output Layer: Lapisan output adalah lapisan terakhir dalam jaringan. Neuron-neuron di lapisan output menghasilkan keluaran yang mewakili prediksi atau klasifikasi model berdasarkan data input yang diberikan. Fungsi aktivasi yang digunakan di lapisan output tergantung pada jenis masalah yang sedang diselesaikan, seperti fungsi softmax untuk klasifikasi multi-kelas atau sigmoid untuk klasifikasi biner.

Pada tahap feedforward, input data mengalir melalui jaringan dan melalui serangkaian transformasi melalui neuron dan lapisan tersembunyi, hingga menghasilkan keluaran akhir. Proses ini memungkinkan model untuk menghasilkan prediksi berdasarkan pola yang telah dipelajari selama tahap pelatihan.

XIII. CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) adalah jenis jaringan saraf tiruan yang khusus dirancang untuk memproses data grid, seperti gambar atau data spasial lainnya. CNN sangat efektif dalam menangani tugas penglihatan komputer seperti klasifikasi gambar, deteksi objek, segmentasi, dan lainnya.

CNN terdiri dari beberapa lapisan yang khusus dirancang untuk memanfaatkan struktur spasial data input. Berikut adalah komponen utama dalam CNN:

1. Lapisan Konvolusi (Convolutional Layer): Lapisan konvolusi adalah inti dari CNN. Ini terdiri dari beberapa filter (kernel) yang melibatkan operasi konvolusi pada input gambar. Setiap filter berfungsi sebagai detektor fitur dan menghasilkan "peta fitur" yang menyoroti keberadaan fitur-fitur spesifik di input. Setiap filter bergerak secara bergantian di seluruh gambar dengan langkah yang ditentukan untuk menghasilkan peta fitur.

2. Lapisan Pooling (Pooling Layer): Lapisan pooling digunakan untuk mengurangi dimensi spasial peta fitur yang dihasilkan oleh lapisan konvolusi. Pooling dapat dilakukan dengan metode seperti max pooling atau average pooling, yang secara bertahap menggabungkan informasi dari wilayah tetangga pada peta fitur. Hal ini membantu mengurangi jumlah parameter yang diperlukan, menghasilkan invariansi terhadap translasi, dan mempertahankan informasi yang penting.

3. Lapisan Aktivasi (Activation Layer): Setelah operasi konvolusi atau pooling, fungsi aktivasi seperti ReLU (Rectified Linear Unit) sering diterapkan untuk memperkenalkan non-linearitas ke dalam jaringan. Ini membantu model untuk belajar pola yang lebih kompleks dan meningkatkan kapasitas representasi jaringan.

4. Lapisan Fully-Connected (Fully-Connected Layer): Setelah beberapa lapisan konvolusi dan pooling, lapisan fully-connected digunakan untuk menghubungkan neuron dari lapisan sebelumnya ke neuron di lapisan output. Lapisan fully-connected mirip dengan lapisan dalam jaringan saraf tiruan biasa, dan biasanya diikuti oleh fungsi aktivasi seperti softmax untuk menghasilkan prediksi kelas.

Keunggulan utama CNN adalah kemampuannya dalam menangani data grid, seperti gambar, dengan memanfaatkan pola lokal dan memperkenalkan invariansi terhadap translasi. Dengan demikian, CNN telah menjadi model yang sangat sukses dalam bidang penglihatan komputer dan banyak aplikasi lain yang melibatkan data spasial.

XIV. TRANSFER LEARNING

Transfer learning adalah pendekatan dalam deep learning di mana model yang telah dilatih sebelumnya pada tugas terkait atau sumber data yang berbeda digunakan sebagai titik awal atau basis untuk melatih model baru pada tugas yang baru atau domain yang berbeda. Pendekatan

ini memanfaatkan pengetahuan yang telah diperoleh oleh model yang sudah dilatih sebelumnya untuk membantu dalam pelatihan model baru.

Transfer learning memiliki beberapa keuntungan, antara lain:

- Membantu dalam melatih model yang efektif dengan dataset yang lebih kecil, karena pengetahuan yang sudah diperoleh dari model basis dapat membantu dalam generalisasi yang lebih baik.
- Mempercepat proses pelatihan, karena model basis yang sudah dilatih sebelumnya menyediakan inisialisasi yang baik untuk lapisan-lapisan awal yang dapat mempercepat konvergensi.
- Berguna ketika dataset yang baru memiliki karakteristik yang mirip dengan dataset yang sudah ada, sehingga pengetahuan yang sudah ada dapat dengan efisien digunakan pada tugas yang baru.

Namun, transfer learning juga memiliki batasan, terutama jika dataset yang baru memiliki karakteristik yang sangat berbeda dengan dataset yang sudah ada. Dalam hal ini, penyesuaian yang lebih lanjut pada lapisan-lapisan awal atau pemilihan model basis yang berbeda mungkin diperlukan.

XV. TENSORBOARD

TensorBoard adalah alat visualisasi yang dikembangkan oleh TensorFlow untuk membantu dalam memahami, memantau, dan menganalisis model dan proses pelatihan dalam deep learning. Ini menyediakan antarmuka web interaktif yang memungkinkan pengguna untuk memvisualisasikan data dan metrik yang relevan dengan model mereka.

Berikut adalah beberapa fitur utama yang ditawarkan oleh TensorBoard:

1. Visualisasi Grafik Model: TensorBoard memungkinkan pengguna untuk memvisualisasikan grafik model dari jaringan saraf mereka. Ini membantu dalam memahami struktur dan arus data dalam model secara intuitif. Pengguna dapat memeriksa lapisan, hubungan antara lapisan, dan parameter yang terkait.

2. Monitoring Metrik Pelatihan: Pengguna dapat melacak metrik penting seperti akurasi, loss, atau metrik evaluasi lainnya selama proses pelatihan model mereka. TensorBoard memberikan grafik dan plot yang interaktif untuk melihat perubahan metrik seiring berjalannya waktu, memungkinkan pemantauan yang lebih baik terhadap kemajuan pelatihan.

3. Visualisasi Histogram dan Distribusi: TensorBoard memungkinkan visualisasi histogram dan distribusi dari bobot dan bias dalam model. Ini membantu dalam memahami distribusi nilai-nilai dalam parameter jaringan dan memantau perubahan mereka selama pelatihan.

4. Visualisasi Data Input: Pengguna dapat memvisualisasikan data input yang digunakan untuk pelatihan model. Misalnya, dalam kasus penglihatan komputer, gambar input dapat ditampilkan dengan label yang sesuai, memungkinkan analisis dan pemahaman yang lebih baik tentang data yang digunakan dalam pelatihan.

5. Embedding dan t-SNE: TensorBoard mendukung visualisasi embedding, yang memungkinkan pengguna untuk memvisualisasikan ruang fitur dalam dimensi yang lebih rendah. Ini membantu dalam menganalisis hubungan antara data dan membantu dalam pemahaman representasi fitur dalam jaringan.

Selain fitur-fitur tersebut, TensorBoard juga menyediakan fungsi untuk melacak dan memvisualisasikan grafik komputasi, distribusi waktu eksekusi, serta penyajian hasil pelatihan dalam bentuk ringkasan dan catatan. Dengan menggunakan TensorBoard, pengguna dapat memahami dan menganalisis model mereka dengan lebih baik, serta melakukan tuning dan peningkatan yang lebih efisien dalam proses deep learning.

XVI. SAVE AND LOAD MODEL

Save and load model adalah proses untuk menyimpan dan memuat kembali model yang telah dilatih sehingga dapat digunakan kembali tanpa perlu melatih ulang dari awal. Ini sangat berguna ketika ingin menggunakan model yang sudah dilatih pada tugas yang sama atau untuk melakukan inferensi pada data baru.

Proses save dan load model melibatkan dua langkah utama:

1. Menyimpan Model:

- Menyimpan Arsitektur: Model dapat disimpan dengan menyimpan konfigurasi arsitektur yang telah didefinisikan. Ini mencakup informasi tentang lapisan-lapisan, hubungan antara lapisan, fungsi aktivasi, dan parameter lain yang terkait dengan arsitektur model.

- Menyimpan Bobot: Bobot (weights) model yang telah dioptimalkan selama pelatihan juga perlu disimpan. Bobot mencakup parameter-parameter yang telah diperbarui selama proses pelatihan dan yang bertanggung jawab untuk transformasi data yang diterapkan oleh model. Bobot dapat disimpan dalam berbagai format seperti Numpy array, TensorFlow checkpoint, atau PyTorch state dictionary.

- Opsional: Menyimpan Optimizer State: Jika ingin melanjutkan pelatihan dari titik terakhir di mana model disimpan, bisa menyimpan status optimizer saat ini. Ini mencakup informasi tentang keadaan optimizer seperti learning rate, momentum, dan parameter lain yang terkait dengan proses optimisasi.

2. Memuat Model:

- Membangun Arsitektur: Ketika memuat model, langkah pertama adalah membangun arsitektur model seperti yang telah didefinisikan sebelumnya. Ini melibatkan membuat lapisan-

lapisan dan mengatur hubungan antara lapisan-lapisan tersebut sesuai dengan konfigurasi yang telah disimpan.

- Memuat Bobot: Setelah arsitektur dibangun, bobot model yang disimpan dapat dimuat kembali ke model yang baru dibangun. Ini dilakukan dengan menginisialisasi bobot model dengan nilai-nilai yang telah disimpan sebelumnya.

- Opsional: Memuat Optimizer State: Jika diinginkan, bisa memuat kembali status optimizer yang telah disimpan. Ini membantu dalam melanjutkan pelatihan dari titik terakhir yang tercapai sebelum model disimpan.

Dengan menyimpan dan memuat model, dapat menghemat waktu dan sumber daya yang diperlukan untuk melatih ulang model dari awal. Model yang telah disimpan juga dapat dibagikan dengan orang lain atau digunakan di lingkungan produksi untuk melakukan inferensi pada data baru tanpa perlu melibatkan proses pelatihan yang rumit lagi.

XVII. KESIMPULAN

Pada pembahasan di atas, kita membahas berbagai topik terkait pemrosesan data dan model di PyTorch. Pertama mari kita lihat bagaimana transformasi data digunakan untuk persiapan kumpulan data, untuk mengubah ukuran gambar dan memuat dataset MNIST. Selanjutnya kita akan melihat bagaimana membangun jaringan saraf menggunakan n-module. Modul PyTorch. Kami mengetahui konsep lemparan ke depan, fungsi kerugian, pengoptimal dan proses pelatihan model menggunakan data pelatihan. Selain itu, penggunaan TensorBoard untuk memantau dan menganalisis pelatihan model juga dijelaskan. Selanjutnya, kita akan membahas menyimpan dan memuat model di PyTorch, menyimpan seluruh model atau hanya diet status. Terakhir, kami melihat cara mengonfigurasi model pemrosesan saat berjalan di GPU atau CPU. Pembicaraan ini akan memberikan pemahaman komprehensif tentang langkah-langkah kunci yang terlibat dalam pemrosesan data dan manajemen model dengan PyTorch.