Technical Report : Deep Learning with Pytorch

1. Pendahuluan

1.1 Latar Belakang

Dalam beberapa tahun terakhir, deep learning telah menjadi salah satu bidang yang paling menarik dalam kecerdasan buatan. Deep learning memungkinkan mesin untuk mempelajari pola yang kompleks dan mampu mengambil keputusan secara otomatis berdasarkan data yang diberikan. Ini telah menghasilkan kemajuan yang luar biasa dalam berbagai aplikasi seperti pengenalan gambar, pemrosesan bahasa alami, deteksi anomali, dan banyak lagi.

PyTorch adalah salah satu framework deep learning yang populer dan kuat yang menggunakan bahasa pemrograman Python. PyTorch menyediakan alat dan fungsi yang mudah digunakan untuk membangun, melatih, dan menerapkan model deep learning dengan cepat dan efisien. Dengan pendekatan yang intuitif dan fleksibel, PyTorch telah menjadi pilihan utama para peneliti dan praktisi dalam komunitas deep learning.

1.2 Tujuan Laporan

Tujuan laporan ini adalah untuk memberikan pemahaman mendalam tentang konsep dasar deep learning dan cara mengimplementasikannya menggunakan PyTorch. Laporan ini akan membahas berbagai aspek penting dalam deep learning, termasuk arsitektur jaringan saraf tiruan, proses pelatihan, evaluasi model, dan contoh implementasi praktis dengan menggunakan PyTorch.

Melalui laporan ini, pembaca akan memperoleh pemahaman yang kokoh tentang dasar-dasar deep learning dan keterampilan praktis dalam menggunakan PyTorch. Diharapkan laporan ini dapat menjadi sumber yang berharga bagi pembaca yang ingin memulai atau meningkatkan pengetahuan mereka dalam bidang deep learning dengan PyTorch.

2. Tensor

Dalam konteks deep learning dengan PyTorch, tensor adalah struktur data yang digunakan untuk menyimpan dan memanipulasi data numerik. Tensor pada PyTorch mirip dengan array multidimensi, tetapi dengan dukungan untuk operasi dan komputasi numerik yang efisien di GPU.

Tensor adalah struktur data sentral dalam deep learning dengan PyTorch. Dengan menggunakan tensor, Anda dapat mengelola dan memanipulasi data numerik yang diperlukan dalam proses pelatihan dan evaluasi model deep learning.

```
tensor([-2.9816e-11])
tensor([ 4.6687e-35,  0.0000e+00, -2.0702e-01])
tensor([[4.6697e-35, 0.0000e+00, 4.6697e-35],
        [0.0000e+00, 1.9005e-19, 5.0833e+31]])
tensor([[[-2.0702e-01,  4.5663e-41,  4.4016e-35],
         [ 0.0000e+00,  4.4842e-44,  0.0000e+00]],

        [[ 8.9683e-44,  0.0000e+00,  4.4070e-35],
         [ 0.0000e+00,  2.0319e-43,  0.0000e+00]]])
tensor([[0.7920, 0.2101, 0.3326],
        [0.1911, 0.0645, 0.0220],
        [0.6585, 0.8347, 0.0984],
        [0.6567, 0.7526, 0.8486],
        [0.5560, 0.7417, 0.8477]])
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]])
torch.Size([5, 3])
torch.float32
tensor([[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]], dtype=torch.float16)
torch.float16
torch.Size([2])
tensor([[0.4787, 0.7497, 0.1715],
        [0.9980, 0.6213, 0.0536],
        [0.3932, 0.3045, 0.3333],
        [0.0479, 0.7332, 0.4034],
        [0.0267, 0.0703, 0.3581]])
tensor([0.4787, 0.9980, 0.3932, 0.0479, 0.0267])
tensor([0.9980, 0.6213, 0.0536])
tensor(0.6213)
0.621333122253418
torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
<class 'numpy.ndarray'>
tensor([2., 2., 2., 2., 2.])
[2. 2. 2. 2. 2.]
[1. 1. 1. 1. 1.]
tensor([1., 1., 1., 1., 1.], dtype=torch.float64)
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
```

Pada Google Colab, digunakan beberapa sintaks dalam PyTorch seperti torch.empty(size) untuk membuat tensor kosong dengan ukuran yang ditentukan, torch.rand(size) untuk membuat tensor dengan angka acak, dan torch.zeros(size) untuk menginisialisasi tensor dengan nol. Operasi element-wise seperti penjumlahan, pengurangan, pembagian, dan perkalian dapat dilakukan menggunakan operator matematika standar seperti +, -, *, dan / antara dua tensor x dan y. Selain itu, PyTorch juga menyediakan fungsi-fungsi seperti torch.add(), torch.sub(), torch.mul(), dan torch.div() untuk operasi tensor yang serupa. Sintaks-sintaks ini memungkinkan pengguna untuk melakukan berbagai operasi pada tensor, yang sangat berguna dalam pemrosesan data dan pelatihan model dalam deep learning..

3. Autograd

Autograd adalah salah satu komponen inti dalam PyTorch yang memungkinkan perhitungan otomatis diferensiasi. Dalam deep learning, diferensiasi diperlukan untuk memperbarui parameter model melalui algoritma penurunan gradien, yang merupakan salah satu aspek kunci dalam proses pelatihan model.

Dalam PyTorch, autograd menyediakan mekanisme untuk menghitung gradien otomatis dari operasi-operasi yang dilakukan pada tensor. Gradien ini kemudian digunakan untuk memperbarui parameter model dengan algoritma optimisasi, seperti Stochastic Gradient Descent (SGD).

Autograd di PyTorch memberikan kemudahan dan fleksibilitas dalam menghitung gradien secara otomatis, yang sangat penting dalam proses pelatihan model deep learning. Dengan adanya autograd, kita dapat dengan mudah memperbarui parameter model dan melakukan optimisasi dengan algoritma penurunan gradien yang efisien.

```
tensor([-0.2572, -0.6876, -0.0402], requires_grad=True)
tensor([1.7428, 1.3124, 1.9598], grad_fn=<AddBackward0>)
<AddBackward0 object at 0x7f4aacc9d360>
tensor([ 9.1115,  5.1670, 11.5225], grad_fn=<MulBackward0>)
tensor(8.6003, grad_fn=<MeanBackward0>)
tensor([3.4855, 2.6248, 3.9196])
tensor([ -211.8870, -2834.6411, -2974.8743], grad_fn=<MulBackward0>)
torch.Size([3])
tensor([2.0480e+02, 2.0480e+03, 2.0480e-01])
False
None
True
<SumBackward0 object at 0x7f4aacc9d270>
True
False
True
False
tensor([3., 3., 3., 3.])
tensor([3., 3., 3., 3.])
tensor([3., 3., 3., 3.])
tensor([0.1000, 0.1000, 0.1000, 0.1000], requires_grad=True)
tensor(4.8000, grad_fn=<SumBackward0>)
```

Dalam Google Colab, kita dapat membuat Tensor x dengan argumen requires_grad=True untuk mengindikasikan bahwa kita ingin melacak operasi pada Tensor tersebut untuk perhitungan gradien. Selanjutnya, kita dapat melakukan operasi y = x + 2, yang menghasilkan Tensor y dengan atribut grad_fn yang menunjukkan fungsi yang menciptakan Tensor tersebut (dalam hal ini, penambahan).

Selain itu, penggunaan autograd pada Tensor non-skalar juga dapat dilakukan. Jika Tensor memiliki lebih dari satu elemen, perlu menyertakan argumen gradient saat memanggil metode .backward(). Argumen ini berupa Tensor yang memiliki bentuk yang sesuai dengan Tensor yang ingin dihitung gradiennya. Hal ini diperlukan untuk menghitung produk vektor-Jacobian.

Selama pelatihan model, penting untuk berhati-hati agar tidak melacak dan mengakumulasi gradien saat melakukan operasi yang tidak diperlukan, seperti pembaruan parameter. Untuk itu, terdapat beberapa cara yang dapat digunakan, seperti menggunakan metode .requires_grad_(False), .detach(), atau torch.no_grad() untuk menghentikan Tensor melacak histori atau menghapus gradiennya.

Pada akhirnya, sebelum melakukan setiap langkah optimisasi, penting untuk mereset gradien. Hal ini dapat dilakukan dengan menggunakan metode .zero_() untuk mengosongkan gradien sebelum melangkah ke langkah optimisasi berikutnya..

4. Backpropagation & Gradient Descent

Dalam deep learning, backpropagation dan gradient descent merupakan dua konsep yang sangat penting. Backpropagation adalah algoritma yang digunakan untuk menghitung gradien dari fungsi kerugian terhadap parameter-parameter model. Sementara itu, gradient descent adalah metode optimisasi yang menggunakan gradien tersebut untuk memperbarui parameter-parameter model agar model dapat belajar secara efektif. Dalam PyTorch, backpropagation dan gradient descent diimplementasikan secara otomatis melalui autograd dan optimizers..

PyTorch menyediakan berbagai optimizer yang mengimplementasikan varian-varian gradient descent, seperti stochastic gradient descent (SGD), Adam, dan RMSProp. Optimizers ini memudahkan proses optimisasi dengan mengelola pengelolaan gradien dan penyesuaian parameter secara otomatis.

Dalam PyTorch, dengan menggunakan autograd untuk backpropagation dan optimizers untuk gradient descent, kita dapat dengan mudah melatih model deep learning dan mengoptimalkan parameter-parameter model untuk meminimalkan fungsi kerugian.

```
tensor(1., grad_fn=<PowBackward0>)
tensor(-4.)
tensor(0.)
```

```
Prediction before training: f(5) = 0.000
epoch 1: w = 0.540, loss = 54.00000000
epoch 11: w = 1.937, loss = 0.09973580
epoch 21: w = 1.997, loss = 0.00018420
epoch 31: w = 2.000, loss = 0.00000034
epoch 41: w = 2.000, loss = 0.00000000
epoch 51: w = 2.000, loss = 0.00000000
epoch 61: w = 2.000, loss = 0.00000000
epoch 71: w = 2.000, loss = 0.00000000
epoch 81: w = 2.000, loss = 0.00000000
epoch 91: w = 2.000, loss = 0.00000000
Prediction after training: f(5) = 10.000
```

Pada Google Colab, setelah menghitung gradien terhadap parameter w, kita dapat menggunakan algoritma gradient descent untuk memperbarui nilai w. Dalam hal ini, kita menggunakan pernyataan w -= 0.01 * w.grad untuk mengurangi nilai w dengan faktor (learning rate) dikali dengan gradien w.grad. Dengan demikian, langkah yang diambil mengarah pada pengurangan nilai loss.

Penting untuk dicatat bahwa saat memperbarui parameter, operasi tersebut tidak boleh terlibat dalam perhitungan gradien. Oleh karena itu, kita menggunakan blok torch.no_grad() atau metode .detach() untuk memastikan bahwa operasi tersebut tidak dilacak dan tidak mempengaruhi perhitungan gradien.

Dengan menggunakan algoritma gradient descent, kita dapat secara iteratif memperbarui parameter-model berdasarkan gradien yang dihitung menggunakan backpropagation. Ini membantu dalam meminimalkan loss dan meningkatkan kinerja model dalam tugas yang diberikan..

5. Training Pipeline

1. Persiapan Data:
    - Memuat data pelatihan dan data validasi/test.
    - Melakukan preprocessing data, seperti normalisasi, pengkodean kategori, atau pembagian data menjadi batch.
2. Inisialisasi Model:
    - Mendefinisikan arsitektur model dengan memilih jenis layer, ukuran input/output, dan fungsi aktivasi.
    - Membuat objek model menggunakan kelas yang sesuai, misalnya menggunakan kelas nn.Module di PyTorch.
3. Penentuan Loss Function:
    - Memilih fungsi loss yang sesuai dengan jenis tugas yang dihadapi, misalnya mean squared error untuk regresi atau cross-entropy untuk klasifikasi.
4. Penentuan Optimizer:
    - Memilih algoritma optimisasi, seperti stochastic gradient descent (SGD), Adam, RMSprop, atau lainnya.
    - Mengatur learning rate dan parameter optimizer lainnya.
5. Pelatihan Model:
    - Iterasi melalui data pelatihan dalam batch-batch kecil.
    - Melakukan forward pass: memasukkan batch data ke dalam model untuk menghasilkan prediksi.
    - Menghitung loss antara prediksi dan label sebenarnya.
    - Melakukan backward pass: menghitung gradien loss terhadap parameter-model menggunakan backpropagation.
    - Memperbarui parameter-model menggunakan optimizer dengan menggunakan gradien yang dihitung.
    - Mengulangi langkah-langkah di atas untuk setiap batch hingga seluruh data pelatihan digunakan.
    - Evaluasi model pada data validasi/test untuk melihat performa aktual model.

6. Evaluasi Model:
    - Menggunakan data validasi/test yang tidak digunakan dalam pelatihan untuk mengevaluasi performa model.
    - Menghitung metrik evaluasi yang sesuai, seperti akurasi, presisi, recall, F1-score, atau MSE (mean squared error).
    - Melakukan analisis lebih lanjut jika diperlukan dan mengubah parameter-model atau arsitektur untuk meningkatkan performa.
7. Penggunaan Model:
    - Setelah model telah dilatih dan dievaluasi, model tersebut dapat digunakan untuk membuat prediksi pada data baru atau diimplementasikan dalam aplikasi yang sesuai.

Dengan mengikuti langkah-langkah ini, kita dapat membangun dan melatih model deep learning menggunakan PyTorch, serta menganalisis dan memperbaiki performa model untuk tugas yang diberikan.

Berikut adalah penjelasan langkah-langkah dalam contoh kode pada Google Colab:

- Data Preparation:
  Langkah 1 dilakukan secara implisit dengan memuat data X dan Y.

- Model Initialization:
  Langkah 2 dilakukan dengan mendefinisikan model menggunakan nn.Linear.

- Loss Function Selection:
  Langkah 3 dilakukan dengan menggunakan nn.MSELoss sebagai fungsi loss.

- Optimizer Selection:
  Langkah 4 dilakukan dengan menggunakan stochastic gradient descent (SGD) sebagai optimizer dengan learning rate 0.01.

- Model Training:
  Langkah 5 dilakukan dalam loop pelatihan, di mana dilakukan forward pass, backward pass, dan update parameter.

- Model Evaluation:
  Langkah 6 tidak terlihat pada contoh kode tersebut, tetapi evaluasi model dapat dilakukan dengan memprediksi nilai pada X_test dan membandingkannya dengan nilai sebenarnya.

- Model Usage:
  Langkah 7 tidak terlihat pada contoh kode tersebut, tetapi setelah model dilatih dan dievaluasi, model tersebut dapat digunakan untuk memprediksi nilai baru menggunakan model(X_test).

  Dengan menggunakan langkah-langkah ini, model dapat dilatih, dievaluasi, dan digunakan untuk membuat prediksi pada data baru.

6. Regresi Linear

Regresi linear adalah metode prediksi sederhana namun efektif dalam statistika dan machine learning. Tujuannya adalah memodelkan hubungan linier antara variabel input (X) dan variabel target (y).

Pada contoh kode di Colab, kita menggunakan regresi linear untuk memprediksi nilai target (y) berdasarkan satu fitur input (X). Data pelatihan dibuat menggunakan fungsi datasets.make_regression dari library scikit-learn. Data tersebut terdiri dari 100 sampel dengan satu fitur, serta termasuk noise sebesar 20.

Setelah melatih model selesai, kita dapat menggunakan model tersebut untuk melakukan prediksi pada data baru. Pada contoh kode, model digunakan untuk memprediksi nilai target (predicted) berdasarkan fitur input (X).

Terakhir, hasil prediksi dan data asli ditampilkan dalam plot menggunakan library matplotlib. Data asli direpresentasikan sebagai titik merah ('ro'), sementara hasil prediksi ditampilkan sebagai garis biru ('b').

Regresi linear berguna dalam berbagai aplikasi, terutama ketika terdapat hubungan linier yang kuat antara variabel input dan target. Namun, jika hubungan tersebut lebih kompleks, model regresi linear mungkin tidak mampu memodelkannya dengan baik..

7. Regresi Logistik

Regresi logistik adalah metode klasifikasi yang digunakan untuk memprediksi probabilitas keanggotaan dalam kategori atau kelas tertentu. Biasanya, regresi logistik digunakan dalam kasus di mana variabel target adalah biner, yaitu dalam klasifikasi biner.

Pada contoh kode di Colab, kami menggunakan regresi logistik untuk melakukan klasifikasi pada dataset Breast Cancer Wisconsin. Dataset ini terdiri dari fitur-fitur yang menggambarkan sel-sel dalam citra mikroskopik dari jaringan payudara, dan tujuannya adalah memprediksi apakah tumor tersebut bersifat jinak (0) atau ganas (1).

Setelah melatih model selesai, kami menggunakan model tersebut untuk melakukan prediksi pada data pengujian. Pada contoh kode, kami menggunakan model untuk memprediksi probabilitas klasifikasi (y_predicted) dan membulatkannya menjadi kelas yang dihasilkan (y_predicted_cls).

Terakhir, kami menghitung akurasi model dengan membandingkan prediksi model dengan label yang sebenarnya (y_test). Akurasi merupakan persentase jumlah prediksi yang benar dibandingkan dengan jumlah total sampel pada data pengujian.

Regresi logistik merupakan metode klasifikasi yang populer dan efektif untuk masalah klasifikasi biner. Namun, untuk masalah klasifikasi dengan lebih dari dua kelas, diperlukan pendekatan lain seperti regresi logistik multinomial atau metode klasifikasi lainnya seperti SVM atau Decision Trees..

8. Dataset dan Dataloader

Dataset:

- Dataset adalah kelas dasar yang digunakan untuk merepresentasikan dataset dalam PyTorch.
- Untuk membuat dataset kustom, kita perlu mengimplementasikan tiga metode: __init__, __getitem__, dan __len__.
- Dalam contoh WineDataset, data dibaca dari file CSV menggunakan NumPy dan diubah menjadi torch.Tensor.
- Metode__getitem__digunakan untuk mengakses sampel berdasarkan indeks.
- Metode__len__mengembalikan jumlah total sampel dalam dataset.

DataLoader:

- DataLoader digunakan untuk memuat dataset dengan batch dan menyediakan fungsionalitas pengacakan dan pemrosesan paralel.
- Kita menginisialisasi DataLoader dengan memberikan dataset sebagai argumen, serta menyebutkan batch_size, shuffle (untuk mengacak data), dan num_workers (untuk memuat data dengan proses sub).
- DataLoader menghasilkan iterator yang dapat digunakan untuk mengambil batch data secara iteratif.
- Pada contoh kode, kita mendapatkan satu batch data dari DataLoader menggunakan iter dan next.
- DataLoader secara otomatis membagi dataset menjadi batch-batch kecil dan mengeluarkan batch data saat diminta.

Penggunaan Dataset dan DataLoader:

- Setelah mengimplementasikan dataset dan DataLoader, kita dapat menggunakannya dalam pelatihan model.
- Dalam contoh kode, kita melakukan iterasi melalui DataLoader dalam loop pelatihan.
- DataLoader memungkinkan kita memuat data dalam batch kecil dan menjalankan proses pelatihan pada setiap batch.
- Dalam setiap iterasi, kita mendapatkan batch data (inputs dan labels) dari DataLoader dan menjalankan langkah pelatihan pada batch tersebut.
- Total iterasi untuk satu epoch dihitung berdasarkan jumlah total sampel dan ukuran batch.

9. Transformasi Dataset

Transformasi dataset dalam PyTorch digunakan untuk melakukan pra-pemrosesan data pada saat pembuatan dataset. Transformasi ini dapat diterapkan pada gambar PIL (Python Imaging Library), tensor, ndarrays (array multidimensi), atau data kustom.

Dalam contoh kode dicolab, kita mengimplementasikan transformasi kustom ToTensor yang mengonversi ndarrays menjadi tensor, dan transformasi kustom MulTransform yang mengalikan input dengan faktor tertentu. Kemudian, transformasi-transformasi ini digunakan dalam objek WineDataset dengan menggunakan argumen transform pada saat pembuatan dataset.

Dengan menggunakan transformasi dataset, kita dapat melakukan berbagai pra-pemrosesan data seperti normalisasi, pengubahan skala, pemotongan, dan transformasi kustom pada dataset sebelum digunakan dalam pelatihan model. Transformasi dataset memungkinkan kita untuk dengan mudah menerapkan operasi-operasi ini secara efisien dan konsisten pada setiap sampel data dalam dataset.

10. Softmax dan Crossentropy

Fungsi Softmax:

- Fungsi softmax didefinisikan dalam bentuk numpy dan PyTorch.
- Fungsi softmax digunakan untuk mengubah skor (logits) menjadi probabilitas yang ternormalisasi.
- Pada implementasi numpy, fungsi softmax menggunakan eksponensial dari setiap elemen input dan membaginya dengan jumlah eksponensial dari semua elemen tersebut.
- Pada implementasi PyTorch, fungsi softmax menggunakan tensor sebagai input dan melakukan normalisasi softmax pada dimensi yang diberikan.

Fungsi Cross Entropy:

- Fungsi cross-entropy didefinisikan dalam bentuk numpy.
- Fungsi cross-entropy mengukur kinerja model klasifikasi dengan output berupa nilai probabilitas antara 0 dan 1.
- Implementasi numpy dari fungsi cross-entropy menggunakan rumus yang menghitung divergensi antara probabilitas aktual (one-hot encoded) dan probabilitas yang diprediksi.
- Pada implementasi PyTorch, modul nn.CrossEntropyLoss digunakan, yang menggabungkan operasi softmax dan negative log likelihood loss (NLLLoss).
- Pada implementasi PyTorch, input yang diberikan kepada nn.CrossEntropyLoss adalah logits (belum melalui softmax), dan target berisi label kelas yang bukan one-hot encoded.

11. Fungsi Aktivasi

Dalam implementasi model menggunakan PyTorch, terdapat beberapa pilihan untuk

menerapkan fungsi aktivasi. Saya akan menjelaskan menggunakan beberapa contoh.

Fungsi Softmax:

- Fungsi softmax digunakan untuk menghasilkan probabilitas yang ternormalisasi dari output model.
- Pada implementasi PyTorch, saya dapat menggunakan torch.softmax atau modul nn.Softmax.
- Saya memasukkan output linier model sebagai input dan mendapatkan probabilitas yang telah dinormalisasi.
- Ini berguna ketika saya ingin mengklasifikasikan input ke dalam beberapa kategori yang saling eksklusif.

Fungsi Sigmoid:

- Fungsi sigmoid merupakan fungsi aktivasi yang menghasilkan keluaran antara 0 dan 1.
- Pada implementasi PyTorch, saya dapat menggunakan torch.sigmoid atau modul nn.Sigmoid.
- Saya menerapkan fungsi sigmoid pada output linier model untuk menghasilkan prediksi yang berada dalam rentang 0 hingga 1.
- Fungsi ini berguna ketika saya ingin melakukan tugas klasifikasi biner, di mana output model merupakan probabilitas untuk kelas positif.

Fungsi Tanh:

- Fungsi tanh merupakan fungsi aktivasi yang menghasilkan keluaran antara -1 dan 1.
- Pada implementasi PyTorch, saya dapat menggunakan torch.tanh atau modul nn.Tanh.
- Saya menerapkan fungsi tanh pada output linier model untuk mengenalkan non-linearitas pada rentang nilai yang lebih luas.
- Fungsi ini berguna ketika saya ingin mengatasi masalah klasifikasi yang memiliki variasi nilai target yang lebih besar.

12. Feed Forward Net

Pertama, saya mempersiapkan lingkungan kerja dan mengatur parameter seperti ukuran input, ukuran lapisan tersembunyi, jumlah kelas (digit 0-9), jumlah epoch, ukuran batch, dan learning rate.

Selanjutnya, saya memuat dataset MNIST dan membuat data loader untuk melatih dan menguji model. Saya juga menampilkan beberapa contoh gambar dari dataset menggunakan matplotlib.

Arsitektur jaringan saraf yang saya gunakan adalah jaringan saraf berlapis penuh (fully connected) dengan satu lapisan tersembunyi. Lapisan tersembunyi memiliki 500 unit, dan ukuran inputnya adalah 784 (28x28 piksel gambar MNIST yang diubah menjadi vektor).

Saya mendefinisikan kelas **NeuralNet** sebagai turunan dari **nn.Module** di PyTorch. Dalam konstruktor, saya mendefinisikan lapisan-lapisan yang akan digunakan dalam jaringan, yaitu lapisan linear pertama (**self.l1**), fungsi aktivasi ReLU (**self.relu**), dan lapisan linear kedua (**self.l2**).

Metode **forward** digunakan untuk menentukan aliran maju (forward pass) dalam jaringan. Saya menerapkan langkah-langkah sebagai berikut:

1. Input diteruskan melalui lapisan linear pertama.
2. Hasilnya diteruskan melalui fungsi aktivasi ReLU.
3. Output dari fungsi aktivasi diteruskan melalui lapisan linear kedua.

Setelah mendefinisikan arsitektur jaringan, saya menginisialisasi objek **model** dari kelas **NeuralNet** dan memindahkannya ke perangkat yang tepat (GPU jika tersedia).

Selanjutnya, saya mendefinisikan fungsi loss yang akan digunakan, yaitu **nn.CrossEntropyLoss**, dan optimizer **torch.optim.Adam** dengan learning rate yang telah ditentukan sebelumnya.

Selama pelatihan model, saya melakukan loop melalui data loader pelatihan. Setiap iterasi, saya melakukan langkah-langkah sebagai berikut:

1. Gambar-gambar dan label-label dari batch saat ini dipindahkan ke perangkat yang tepat.
2. Melakukan langkah maju (forward pass) dengan meneruskan gambar-gambar ke dalam model.
3. Menghitung loss antara output model dan label yang sebenarnya.
4. Melakukan langkah mundur (backward pass) untuk menghitung gradien loss terhadap parameter-model.
5. Mengoptimalkan parameter-model dengan mengatur gradien menggunakan optimizer.

Selama pengujian model, saya menonaktifkan perhitungan gradien dengan **torch.no_grad()**. Saya menggunakan model yang telah dilatih sebelumnya untuk membuat prediksi pada set data pengujian. Prediksi dilakukan dengan mengambil nilai maksimum dari output model. Akurasi model dihitung dengan membandingkan prediksi dengan label sebenarnya.

13. Convolutional Neural Networks (CNN)

Pertama, saya mengatur konfigurasi perangkat, seperti menggunakan GPU jika tersedia, atau CPU jika tidak. Kemudian, saya menentukan hyperparameter seperti jumlah epoch, ukuran batch, dan learning rate.

Dataset CIFAR-10 terdiri dari 60.000 gambar warna 32x32 piksel yang dibagi menjadi 10 kelas, dengan 6.000 gambar per kelas. Saya menggunakan transformasi transform untuk mengubah gambar-gambar tersebut menjadi tensor dan melakukan normalisasi.

Selanjutnya, saya membuat data loader untuk pelatihan dan pengujian menggunakan dataset CIFAR-10 yang telah dimuat sebelumnya.Saya juga mendefinisikan kelas-kelas yang ada dalam dataset CIFAR-10, seperti 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', dan 'truck'.

Untuk memvisualisasikan beberapa gambar acak dari dataset pelatihan, saya mengambil beberapa gambar dan labelnya menggunakan iter(train_loader) dan menampilkan gambar-gambar tersebut menggunakan fungsi imshow.

Arsitektur CNN yang saya gunakan terdiri dari beberapa lapisan konvolusi, pooling, dan lapisan linear. Dalam kelas ConvNet, saya mendefinisikan lapisan-lapisan tersebut menggunakan modul-modul yang disediakan oleh PyTorch seperti nn.Conv2d, nn.MaxPool2d, dan nn.Linear. Metode forward mendefinisikan aliran maju (forward pass) dalam jaringan, di mana input gambar melewati lapisan-lapisan konvolusi, aktivasi ReLU, pooling, dan lapisan linear. Output akhir adalah kelas prediksi untuk setiap gambar.

Setelah mendefinisikan arsitektur jaringan, saya menginisialisasi objek model dari kelas ConvNet dan memindahkannya ke perangkat yang tepat (GPU jika tersedia).

Saya menggunakan fungsi loss nn.CrossEntropyLoss dan optimizer torch.optim.SGD untuk melatih model.

Selama pengujian, saya menonaktifkan perhitungan gradien dengan torch.no_grad(). Saya menggunakan model yang telah dilatih sebelumnya untuk membuat prediksi pada dataset pengujian. Akurasi model dihitung dengan membandingkan prediksi dengan label sebenarnya. Selain itu, saya juga menghitung akurasi untuk setiap kelas dalam dataset CIFAR-10.

14. Transfer Learning

Dalam penerapan transfer learning untuk pengenalan objek pada dataset Hymenoptera, langkah-langkah yang diambil adalah sebagai berikut:

1. Memuat dataset Hymenoptera dan menerapkan transformasi data yang diperlukan menggunakan modul `transforms` dari PyTorch.
2. Mengatur data loader untuk pelatihan dan validasi menggunakan `torch.utils.data.DataLoader`.
3. Mempersiapkan variabel seperti `dataset_sizes` dan `class_names` untuk menyimpan informasi tentang dataset.
4. Memeriksa ketersediaan perangkat CUDA (GPU) dan mengatur perangkat yang akan digunakan.
5. Membuat fungsi `imshow` untuk menampilkan gambar-gambar dalam tensor.
6. Menampilkan contoh data yang akan dilatih menggunakan `imshow`.
7. Membuat fungsi `train_model` untuk melatih model dengan pelatihan dan validasi pada setiap epoch.
8. Melatih model dengan menggunakan model ResNet-18 yang telah dilatih sebelumnya, mengganti lapisan terakhir dengan lapisan linear baru, dan meng-update semua parameter dalam model.
9. Melatih model dengan menggunakan model ResNet-18 yang telah dilatih sebelumnya, membekukan semua lapisan kecuali lapisan terakhir, dan hanya meng-update parameter dalam lapisan terakhir.
10. Meload model terbaik setelah pelatihan selesai pada kedua skenario.

Dengan menggunakan transfer learning, berhasil dilatih model pengenalan objek pada dataset Hymenoptera dengan efisiensi yang tinggi. Model ini dapat digunakan untuk mengklasifikasikan gambar-gambar baru antara lebah dan semut.

15. Tensorboard

Kode di collab merupakan implementasi jaringan saraf berbasis fully connected neural network (FCNN) untuk pengenalan digit menggunakan dataset MNIST. Berikut adalah analisis hasil dari kode tersebut:

1. Dataset MNIST digunakan, yang terdiri dari gambar-gambar digit tulisan tangan.
2. Dilakukan pemrosesan dataset menggunakan `torchvision.transforms.ToTensor()` untuk mengubah gambar menjadi tensor.
3. Dilakukan pembagian dataset menjadi data pelatihan dan data pengujian menggunakan `torch.utils.data.DataLoader`.
4. Dilakukan visualisasi beberapa contoh gambar dari dataset menggunakan `matplotlib`.
5. Dibuat arsitektur model FCNN dengan satu lapisan tersembunyi (`hidden_size`) menggunakan `nn.Module`.
6. Fungsi `forward` digunakan untuk melakukan perhitungan maju dalam model.
7. Fungsi loss yang digunakan adalah `nn.CrossEntropyLoss`.
8. Optimizer yang digunakan adalah Adam dengan learning rate (`learning_rate`) 0.001.
9. Dilakukan pelatihan model dengan loop for yang berjalan selama jumlah epoch (`num_epochs`) yang ditentukan.
10. Di dalam loop pelatihan, dilakukan perhitungan maju dan mundur (backward pass) serta optimasi parameter model.
11. Loss dan akurasi pelatihan dicatat dan ditampilkan setiap 100 langkah.
12. Dilakukan pengujian model pada data pengujian untuk menghitung akurasi model pada gambar-gambar yang belum pernah dilihat sebelumnya.
13. Hasil akurasi pengujian ditampilkan.
14. Dilakukan visualisasi Precision-Recall curve untuk setiap kelas digit menggunakan TensorBoard.

Kode tersebut juga menggunakan TensorBoard untuk mencatat metrik dan visualisasi model yang berguna dalam analisis dan pemantauan pelatihan.

16. Menyimpan dan Memuat Model

Metode pertama yang dijelaskan adalah menggunakan fungsi torch.save dan torch.load untuk menyimpan dan memuat seluruh model. Dalam contoh tersebut, model lengkap disimpan ke dalam file dengan ekstensi .pth. Kemudian, model dapat dimuat kembali dengan menggunakan torch.load. Perlu diingat bahwa setelah memuat model, perlu memanggil metode model.eval() untuk mengatur model ke mode evaluasi.

Metode kedua yang dijelaskan adalah menyimpan dan memuat hanya state_dict dari model. State_dict adalah dictionary yang berisi parameter dan tensor yang merupakan bagian dari model. Dalam contoh tersebut, state_dict model disimpan ke dalam file .pth. Kemudian, model baru dibuat dan state_dict dimuat menggunakan torch.load_state_dict. Setelah memuat state_dict, perlu memanggil model.eval() untuk mengatur model ke mode evaluasi.

Selanjutnya, dijelaskan juga cara menyimpan dan memuat checkpoint yang mencakup state_dict model dan state_dict optimizer. Checkpoint ini berguna jika ingin melanjutkan pelatihan model dari titik yang disimpan. Checkpoint disimpan ke dalam file .pth menggunakan torch.save. Kemudian, model dan optimizer baru dibuat, dan state_dict model dan optimizer dimuat menggunakan torch.load_state_dict. Epoch juga dapat dipulihkan dari checkpoint untuk melanjutkan pelatihan dari titik terakhir.

Kemudian, dijelaskan tentang menyimpan dan memuat model ketika digunakan GPU atau CPU. Jika model dilatih dan disimpan menggunakan GPU, saat memuat model, perlu memastikan bahwa model dan tensornya diatur untuk digunakan di GPU yang sama. Hal ini dapat dilakukan dengan memanggil model.to(device) di model dan torch.load(..., map_location=device) saat memuat model. Jika model disimpan menggunakan CPU dan akan dimuat ke GPU, model dapat dimuat dengan torch.load dan kemudian dipindahkan ke GPU dengan model.to(device).

Terakhir, disarankan agar ketika menggunakan GPU, perlu memanggil model.eval() atau model.train() setelah memuat model untuk mengatur dropout dan batch normalization layers ke mode evaluasi atau pelatihan yang benar.

Dalam keseluruhan, kodingan ini memberikan pemahaman tentang berbagai metode untuk menyimpan dan memuat model dalam PyTorch, dan juga mengingatkan pentingnya mengatur model ke mode yang benar setelah memuatnya.

17. Kesimpulan

Dalam diskusi di atas, kita membahas beberapa topik terkait dengan pemrosesan data dan model dalam PyTorch. Pertama, kita melihat bagaimana menggunakan transformasi data untuk mempersiapkan dataset, seperti mengubah ukuran gambar dan mengunduh dataset MNIST. Selanjutnya, kita melihat bagaimana membangun jaringan saraf tiruan (neural network) dengan menggunakan modul nn.Module dari PyTorch. Kita mengenal konsep forward pass, loss function, optimizer, dan proses pelatihan model menggunakan data pelatihan. Selain itu, kita juga menjelaskan penggunaan TensorBoard untuk memantau dan menganalisis pelatihan model. Selanjutnya, kita membahas cara menyimpan dan memuat model dalam PyTorch, baik dengan menyimpan seluruh model maupun hanya state_dict. Terakhir, kita melihat bagaimana mengatur pemrosesan model saat digunakan di GPU atau CPU. Diskusi ini memberikan pemahaman yang komprehensif tentang langkah-langkah penting dalam pengolahan data dan pengelolaan model menggunakan PyTorch.