Bab 2

Arsitektur MLP dan Fungsi Aktivasi

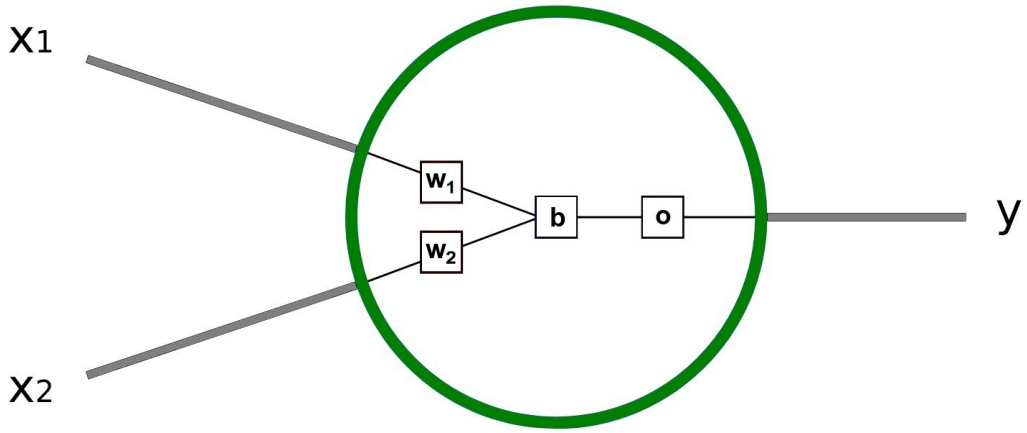
2.1 Multi-Layer Perceptron (MLP)

Jaringan syaraf tidak serumit itu! Istilah "neural network" sering digunakan sebagai kata kunci, tetapi pada kenyataannya, jaringan syaraf tiruan sering kali jauh lebih sederhana daripada yang dibayangkan kebanyakan orang. Catatan kecil ini ditujukan untuk pemula atau bahkan tidak memiliki pengetahuan dasar tentang pembelajaran mesin. Kita akan memahami cara kerja neural network. Neural Network sederhana yang tersusun atas beberapa neuron, input dan output layer, disebut sebagai perceptron. Dengan tambahan dari hidden layer, satu hingga tiga layer, maka disebut sebagai Multi-Layered Perceptron.

Contoh sebuah neuron adalah seperti pada Gambar 2.1:

Pertama, kita harus berbicara tentang neuron, unit dasar dari jaringan saraf. Sebuah neuron menerima input, melakukan beberapa perhitungan dengan input tersebut, dan menghasilkan satu output. Gambar 1 merupakan bentuk neuron sederhana dengan 2 inputan. Ada beberapa hal yang terjadi di sini:

1. Setiap inputan x_1 dan x_2 dikalikan dengan suatu bobot. Di sini, x_1



Gambar 2.1: Neuron Sederhana

dikalikan dengan bobot w_1, sedangkan x_2 dikalikan dengan bobot w_2. Bobot sini adalah faktor pengali yang menentukan seberapa besar pengaruh setiap inputan terhadap output. Apabila dituliskan dalam bentuk persamaan, hasilnya adalah:

$$z_1 = x_1 \cdot w_1 \tag{2.1}$$

$$z_2 = x_2 \cdot w_2 \tag{2.2}$$

Dari semua hasil perkalian sebelumnya (z_1, z_2), dijumlahkan dan dikalikan dengan bias b. Bias adalah nilai tambahan yang ditambahkan ke dalam hasil perkalian antara input dan bobot sebelum diterapkan fungsi aktivasi. Tanpa bias, model hanya dapat mempelajari fungsi yang selalu melewati titik nol. Bias memungkinkan model untuk mempelajari fungsi yang tidak melewati titik nol. Untuk memperjelas apa itu bias, perhatikan contoh berikut. Misalkan kita memiliki persamaan

$$z = x \cdot w + b \tag{2.3}$$

Yang perlu diperhatikan di sini adalah

1. Apabila nilai $b = 0$, maka nilai z sangat bergantung pada nilai $x \cdot w$.

2. Jika nilai $b \neq 0$, maka nilai b akan berfungsi seperti interceptor c pada per- samaan fungsi linier, yaitu menggeser nilai output z.

Nilai luaran hasil penjumlahan dan perkalian antara input, bobot, dan bias, bisa dituliskan dengan:

$$x_1 \cdot w_1 + x_2 \cdot w_2 + b \tag{2.4}$$

2.2 Fungsi Aktivasi

Secara definisi, fungsi aktivasi adalah fungsi matematis yang mengubah suatu nilai dalam jaringan saraf menjadi nilai tertentu. Fungsi ini membatasi suatu nilai agar bisa menangani kasus yang lebih kompleks. Tanpa fungsi aktivasi non-linear, jaringan saraf hanya akan melakukan transformasi linear, yang sangat terbatas dalam memecahkan masalah kompleks. Adapun keuntungan dari non-linieritas dalam neural network adalah memungkinkan jaringan untuk mempelajari pola-pola yang rumit dan hubungan data non-linier.

Terdapat berbagai jenis fungsi aktivasi, masing-masing dengan karakteristik dan kegunaannya sendiri. Beberapa contoh yang umum dipakai adalah Sigmoid, ReLU, Tanh, Softmax.

2.2.1 Sigmoid

Menghasilkan output dalam rentang 0 hingga 1, sering digunakan dalam masalah klasifikasi biner.

2.2.2 ReLU

ReLU (Rectified Linear Unit): Menghasilkan output 0 jika input negatif, dan output sama dengan input jika input positif. Sangat populer karena efisiensinya.

2.2.3 Tanh

Tanh (tangent hiperbolik): Menghasilkan output dalam rentang -1 hingga 1, mirip dengan sigmoid tetapi dengan rentang output yang berbeda.

2.2.4 Softmax

Softmax: Sering digunakan dalam lapisan output untuk masalah klasifikasi multi-kelas, menghasilkan distribusi probabilitas atas kelas-kelas yang berbeda

Fungsi aktivasi sangat penting untuk kemampuan jaringan saraf dalam mempelajari dan memodelkan data yang kompleks. Pemilihan fungsi aktivasi yang tepat dapat sangat mempengaruhi kinerja dan akurasi model.

2.3 Fungsi Loss

Fungsi loss adalah sebuah fungsi yang digunakan sebagai ukuran matematis dari kesalahan prediksi model. Fungsi ini mengukur perbedaan antara nilai yang diprediksi dan nilai aktual.

Tujuan utamanya adalah untuk mengevaluasi kinerja model, dengan nilai loss yang lebih rendah menunjukkan akurasi yang lebih tinggi, dan untuk memandu algoritma optimasi seperti gradient descent dalam menyesuaikan parameter model untuk meminimalkan kesalahan ini.

Dengan memengaruhi cara model belajar, fungsi kerugian yang berbeda, seperti Mean Squared Error (MSE) untuk regresi atau Cross-Entropy Loss untuk klasifikasi, memenuhi kebutuhan tugas tertentu. Pada dasarnya, fungsi lossbertindak sebagai mekanisme umpan balik yang penting, mengarahkan model ke arah kemampuan prediksi yang lebih baik selama proses pelatihan.

Adapun MSE diformulasikan sebagai:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2 \tag{2.5}$$

di mana:

1. n adalah jumlah dari data

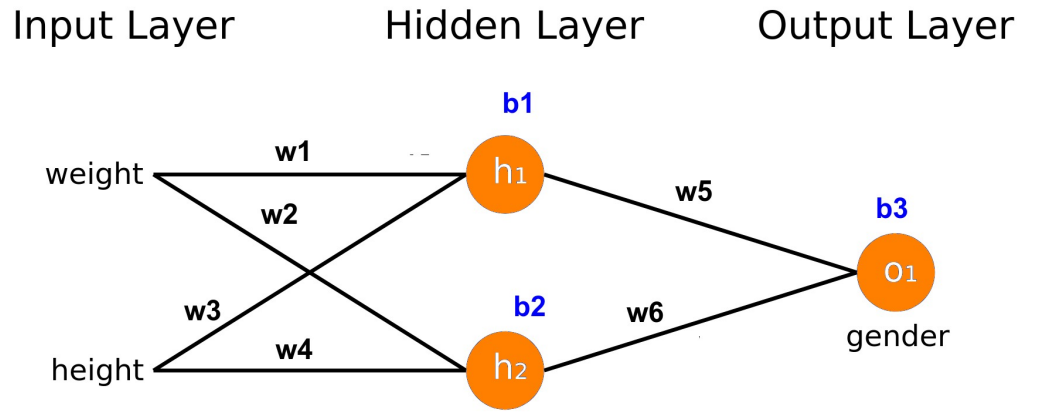
2. y adalah nilai aktual

3. dan $\hat{y}$ adalah nilai hasil prediksi

$(y - \hat{y})^2$ dikenal sebagai eror kuadrat. Fungsi loss kita sangat sederhana, yaitu cukup menghitung nilai rata-rata keseluruhan selisih/eror. Jika nilai fungsi loss semakin kecil, maka prediksi dari model MLP semakin baik.

2.4 Feedforward dan Backpropagation

Jaringan saraf tidak lebih dari sekumpulan neuron yang terhubung bersama. Gambar 2.2 adalah tampilan jaringan saraf sederhana: Jaringan ini memiliki 2 input, hidden layer dengan 2 neuron (h_1 dan h_2), dan lapisan output dengan 1 neuron (o_1). Perhatikan bahwa input untuk o_1 adalah output dari h_1 dan h_2. Hidden layer adalah lapisan apa pun di antara lapisan input (pertama) dan lapisan output (terakhir). Di lapisan hidden ini, bisa ada beberapa lapisan.



Gambar 2.2: Contoh MLP

Adapun jumlah atau nilai akhir dari Persamaan 2.4, dilewatkan suatu fungsi aktivasi f berikut:

$$y = f(x_1 \cdot w_1 + x_2 \cdot w_2 + b) \tag{2.6}$$

2.4.1 Feedforward

Mari kita gunakan jaringan pada Gambar 2.2 dan kita asumsikan semua neuron memiliki bobot yang sama $w = [0, 1]$, bias yang sama $b = 0$, dan fungsi aktivasi sigmoid. Didefinisikan h_1, h_2, o_1 sebagai output dari neuron. Untuk contoh, mari kita hitung nilai o_1 berdasarkan input $x_1 = 2$ dan $x_2 = 3$ sebagai berikut:

$$h_1 = f(w \cdot x) + b \tag{2.7}$$
$$= f(0 * 2 + 1 * 3 + 0) \tag{2.8}$$
$$= f(3) \qquad = 0.9526 \tag{2.9}$$

Ingat, untuk kondisi neural network pada contoh sebelumnya, node pada layer hidden $h_1 = h_2$. Maka, nilai dari o_1 adalah:

$$o_1 = f(w_5 \cdot h_1 + w_6 \cdot h_5 + b_3) \tag{2.10}$$
$$= f(b_3) \tag{2.11}$$

Sebuah jaringan saraf dapat memiliki sejumlah lapisan dengan sejumlah neuron dalam lapisan tersebut. Ide dasarnya tetap sama: meneruskan input ke depan melalui neuron-neuron dalam jaringan untuk mendapatkan output pada akhirnya.

2.4.2 Backpropagation

Sebelum kita melatih model MLP, pertama-tama kita perlu cara untuk mengukur seberapa "baik" kinerja jaringan sehingga jaringan dapat mencoba untuk melakukan yang "lebih baik". Itulah yang dimaksud dengan loss. Di sini, kita menggunakan MSE untuk menghitung loss model MLP.

Sekarang, tujuan kita adalah membuat prediksi jaringan saraf menjadi lebih akurat. Kita tahu bahwa kita bisa mengubah 'pengaturan' jaringan (yaitu bobot dan bias) untuk memengaruhi hasil prediksi. Tapi, bagaimana cara kita mengubah 'pengaturan' ini supaya prediksi menjadi lebih tepat dan 'kesalahan' atau loss berkurang?

Dalam bagian ini, kita akan menggunakan beberapa konsep dari kalkulus multivariabel, yaitu cabang matematika yang mempelajari fungsi-fungsi dengan banyak variabel. Mungkin Anda akan melihat beberapa rumus dan perhitungan yang sedikit rumit. Namun, jika Anda merasa kesulitan atau belum terlalu familiar dengan kalkulus, tidak masalah. Anda bisa langsung melewati bagian yang penuh dengan rumus dan fokus pada penjelasan konsepnya. Tapi, di sini saya berusaha menyajikan penjelasan yang mudah dipahami tanpa harus terlalu terpaku pada detail matematis. Tujuan kami adalah agar Anda tetap bisa memahami prinsip-prinsip dasarnya, bahkan jika Anda tidak sepenuhnya memahami perhitungannya.

Jika merujuk pada Gambar 2.2, kita bisa menuliskan fungsi loss pada tiap-tiap variabel dengan:

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3) \tag{2.12}$$

Bayangkan kita ingin mengubah w_1. Bagaimana perubahan w_1 akan mempengaruhi nilai kerugian L? Untuk menjawab pertanyaan ini, kita perlu menghitung **turunan parsial** $\frac{\partial L}{\partial w_1}$. Namun, bagaimana cara menghitungnya? Matematika mulai menjadi lebih kompleks. Jangan khawatir! Saya sarankan untuk menyiapkan kertas dan pena agar lebih mudah memahami konsep ini.

2.4.2.1 Menggunakan Aturan Rantai

Langkah pertama adalah menuliskan turunan parsial dalam bentuk $\frac{\partial y_{\text{pred}}}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{\text{pred}}} \cdot \frac{\partial y_{\text{pred}}}{\partial w_1} \tag{2.13}$$

*Ini berlaku karena aturan rantai (*Chain Rule*).*

Kita bisa menghitung $\frac{\partial L}{\partial y_{\text{pred}}}$ karena sebelumnya kita telah mendefinisikan fungsi kerugian sebagai $L = (1 - y_{\text{pred}})^2$:

$$\frac{\partial L}{\partial y_{\text{pred}}} = \frac{\partial (1 - y_{\text{pred}})^2}{\partial y_{\text{pred}}} = -2(1 - y_{\text{pred}}) \tag{2.14}$$

2.4.2.2 Menentukan $\frac{\partial y_{\mathbf{pred}}}{\partial w_1}$

Sekarang, kita perlu mencari nilai $\frac{\partial y_{\text{pred}}}{\partial w_1}$. Sama seperti sebelumnya, kita misalkan bahwa h_1, h_2, dan o_1 adalah output dari neuron-neuron yang terlibat. Maka,

$$y_{\text{pred}} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3) \tag{2.15}$$

f adalah fungsi aktivasi sigmoid.

Karena w_1 hanya mempengaruhi h_1 (bukan h_2), kita bisa menuliskan:

$$\frac{\partial y_{\text{pred}}}{\partial w_1} = \frac{\partial y_{\text{pred}}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} \tag{2.16}$$

$$\frac{\partial y_{\text{pred}}}{\partial h_1} = w_5 \cdot f'(w_5 h_1 + w_6 h_2 + b_3) \tag{2.17}$$

2.4.2.3 Menghitung $\frac{\partial h_1}{\partial w_1}$

Kita melakukan hal yang sama untuk $\frac{\partial h_1}{\partial w_1}$:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1) \tag{2.18}$$

$$\frac{\partial h_1}{\partial w_1} = x_1 \cdot f'(w_1 x_1 + w_2 x_2 + b_1) \tag{2.19}$$

x_1 di sini adalah bobot, dan x_2 adalah tinggi. Ini adalah kedua kalinya kita melihat $f'(x)$ (turunan dari fungsi sigmoid). Mari kita turunkan:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.20}$$

$$f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = f(x) \cdot (1 - f(x)) \tag{2.21}$$

Bentuk ini akan kita gunakan nanti.

2.4.2.4 Menganalisis $\frac{\partial L}{\partial w_1}$

Kita telah berhasil memecah turunan parsial $\frac{\partial L}{\partial w_1}$ menjadi beberapa bagian yang dapat dihitung:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{\text{pred}}} \cdot \frac{\partial y_{\text{pred}}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} \tag{2.22}$$

Sistem perhitungan turunan parsial ini yang dilakukan secara mundur dikenal sebagai **backpropagation**, atau disingkat **backprop**.

2.4.2.5 Kesimpulan

Wah, itu banyak sekali simbol matematika! Tidak masalah jika masih terasa membingungkan. Mari kita coba sebuah contoh untuk melihat bagaimana cara kerjanya dalam aksi nyata!

2.5 Implementasi MLP dengan PyTorch

PyTorch adalah kerangka kerja deep learning sumber terbuka yang populer yang dikembangkan oleh Facebook AI Research Lab. PyTorch menyediakan pendekatan yang fleksibel dan dinamis untuk membangun dan melatih jaringan saraf, menjadikannya pilihan yang sangat baik untuk mengimplementasikan Multi-Layer Perceptron (MLP).

MLP adalah jenis jaringan saraf feedforward yang terdiri dari beberapa lapisan neuron, termasuk lapisan input, lapisan tersembunyi, dan lapisan output. Setiap neuron dalam satu lapisan terhubung ke setiap neuron di lapisan berikutnya, menjadikannya jaringan yang sepenuhnya terhubung. PyTorch membuat penerapan MLP menjadi mudah dengan pendekatan modularnya.

Untuk detil dari implementasi, bisa dilihat di https://s.id/task2DL

2.6 Asesmen

Judul Tugas

Tugas Individu: Klasifikasi Bunga Iris dengan Multilayer Perceptron (MLP) menggunakan PyTorch

Deskripsi Tugas

Pada tugas ini, mahasiswa diminta untuk mengimplementasikan Multilayer Perceptron (MLP) menggunakan PyTorch untuk melakukan klasifikasi bunga Iris berdasarkan dataset Iris dari Scikit-learn. Mahasiswa harus:

1. Memuat dataset Iris dan melakukan eksplorasi data.
2. Membuat arsitektur MLP dengan setidaknya 1 hidden layer menggunakan PyTorch.
3. Melatih model menggunakan dataset Iris dengan optimasi dan fungsi loss yang sesuai.
4. Melakukan evaluasi model menggunakan metrik accuracy.
5. Membuat visualisasi loss dan akurasi selama training.
6. Menganalisis hasil: Menjelaskan bagaimana performa model dan kemungkinan perbaikannya.
7. Bandingkan hasil model MLP yang Anda buat dengan model sederhana seperti Logistic Regression. Jelaskan mengapa MLP lebih unggul atau justru lebih buruk

Instruksi Teknis

1. Gunakan Python dan PyTorch untuk membangun model.
2. Gunakan Scikit-learn untuk memuat dataset.

3. Latih model selama minimal 100 epoch dengan optimizer Adam atau SGD.

4. Gunakan fungsi aktivasi ReLU pada hidden layer dan Softmax pada output layer.

5. Simpan model yang telah dilatih dan buat script untuk melakukan prediksi pada data baru.

6. Tambahkan di laporan bagaimana model yang dibuat dapat diterapkan dalam dunia nyata dan apa saja tantangan yang mungkin muncul

7. Laporan dikumpulkan dalam format notebook (.ipynb) dan PDF.

Rubrik Penilaian