

04: More Simple Neural Networks (Training)

Chapter Goal: To connect the "Anatomy of a Neural Network" from the previous video with the concept of "Descending a Valley" ([Gradient Descent](#)) that we learned in Module 2.

1. The New Goal: "Training" the Neural Network

- **Recap:** We already know how to calculate the network's output if the weights (w) and biases (b) are known (a **Forward Pass**).
 - **New Problem:** How do we find the **best** possible values for w and b ?
 - **"Training":** Is the process of **iteratively updating** w and b so that the network's output gets closer and closer to the output we actually want.
 - **Training Ingredient:** We need **labelled data**, which are pairs of (input, correct_output) .
 - **Example:** (picture_of_a_face, [1, 1]) (meaning "this is a face" and "this is a happy face").
-

2. Measuring "Badness": The Cost Function

- **Initial Step:** We start with w and b filled with random numbers. The network's output will be complete nonsense.
- **How to Measure Error:** We need a "badness score" for our current guess.
- **The Cost Function C :**

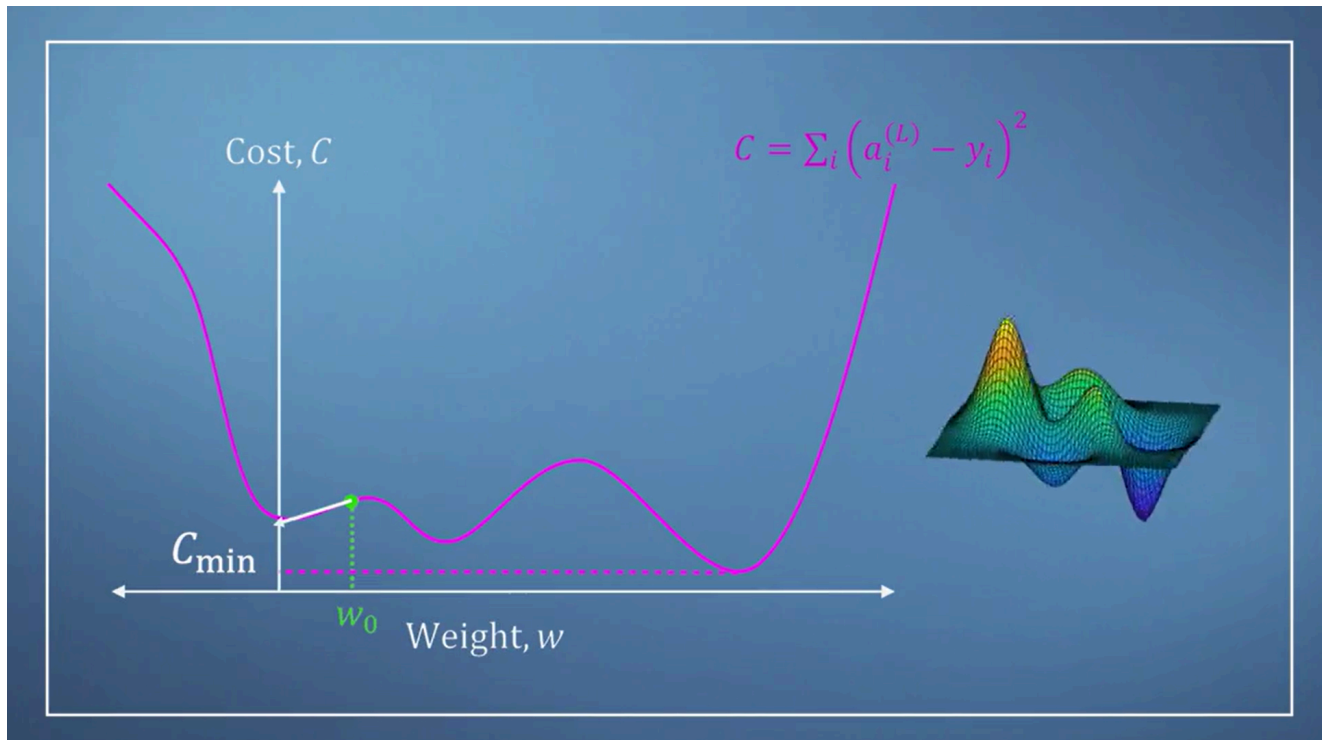
$$C = \sum (\vec{y}_{\text{correct}} - \vec{y}_{\text{predicted}})^2$$

- **Meaning:** C is the **sum of the squared differences** between the output that *should have been* (\vec{y}_{correct}) and the output our network actually produced ($\vec{y}_{\text{predicted}}$).
 - **Goal of Training:** To find the w and b that make the value of C as small as possible (the **minimum**).
-

3. "Aha!" Moment: Back to "Mountain Climbing"

- **The Connection:** The problem of "minimizing C " is **exactly the same** as the problem of "finding the bottom of the valley" that we've already studied!
- **The Landscape:**

- Every possible combination of all the w s and b s in the network is a single point in a massive, high-dimensional **parameter space**.
- The value of the Cost C at each of those points is the "elevation".
- **The Strategy (Gradient Descent):**
 1. Start at a random point (random w_0 , b_0).
 2. Calculate the **Gradient/Jacobian of C** with respect to all w s and b s. This will tell us, "If I change weight w_1 a little bit, how much will the Cost C change?"
 3. Take a small step in the direction of the **negative Gradient** to update all w s and b s.
 4. Repeat.



4. The Main Tool: The Multivariate Chain Rule

- **The Problem:** How do we actually calculate $\frac{\partial C}{\partial w}$ (the derivative of the Cost with respect to a single weight w)?
- **The Relationship is Distant:** A change in a weight w will change a z value, which changes an activation a , which finally changes the Cost C . This is a **long chain**.
- **The Solution:** We must use the **Chain Rule** to "trace" the change backwards from C all the way to w .
- **Example Chain for a simple neuron** $a_1 = \sigma(z_1)$ where $z_1 = w_0 a_0 + b_0$:
 - **Chain for w_0 :** $C \rightarrow a_1 \rightarrow z_1 \rightarrow w_0$

$$\frac{\partial C}{\partial w_0} = \frac{\partial C}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_0}$$

- **Chain for b_0 :** $C \rightarrow a_1 \rightarrow z_1 \rightarrow b_0$

$$\frac{\partial C}{\partial b_0} = \frac{\partial C}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_0}$$

This process is called **BACKPROPAGATION**. We calculate the "error" at the output, and then "propagate it backwards" through the network using the Chain Rule to find out the "error contribution" of every single weight and bias.

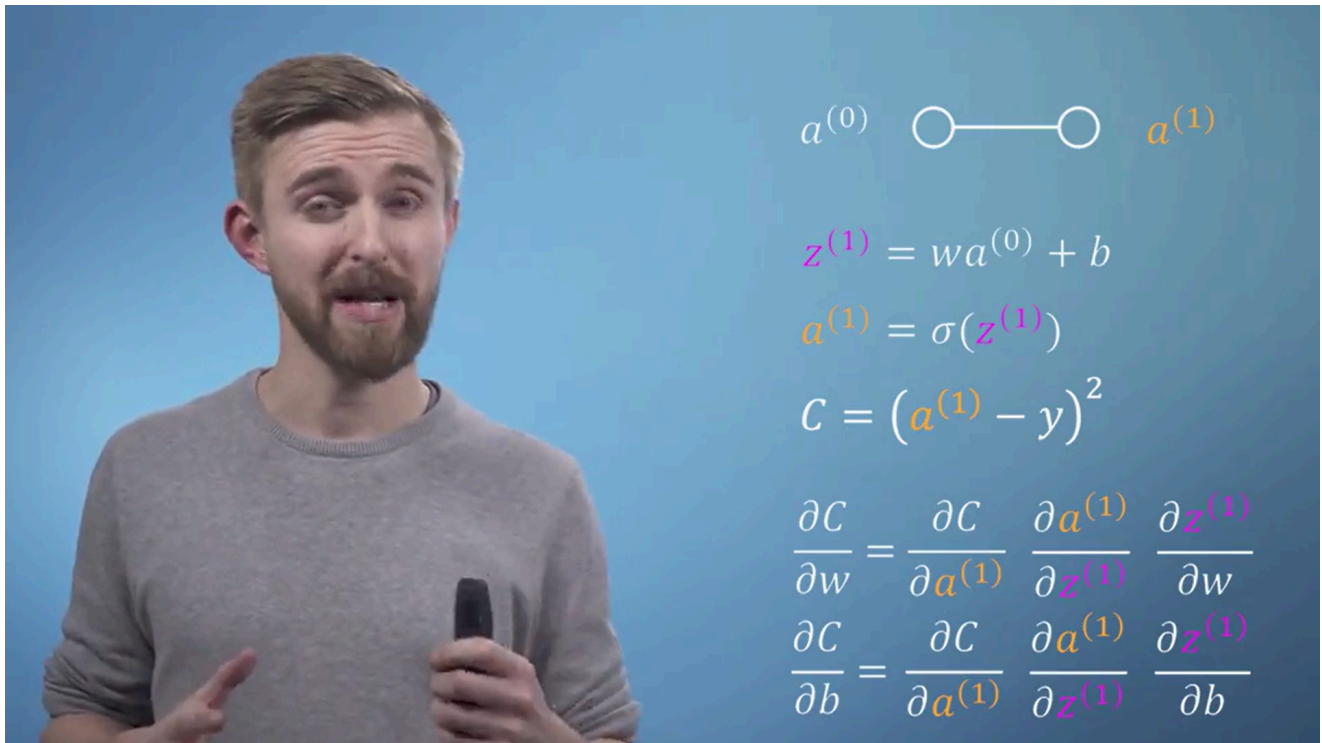


Diagram of a simple neural network showing input $a^{(0)}$ connected to output $a^{(1)}$.

$$z^{(1)} = wa^{(0)} + b$$

$$a^{(1)} = \sigma(z^{(1)})$$

$$C = (a^{(1)} - y)^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial b}$$

5. Key Message

- **"Training"** a Neural Network is a direct application of **Gradient Descent**.
- **Backpropagation**, the algorithm that makes Deep Learning possible, is fundamentally just a clever, large-scale application of the **Multivariate Calculus Chain Rule**.
- By understanding the Chain Rule, you are understanding the "engine" behind how all modern AI models "learn". You are no longer seeing a "black box," but a giant application of the calculus ideas you have just learned.

⚠ Caution

Gak paham yaa??? mwahahaha sama saya jugaa, coba kalau kita berikan contoh konkret biar ada bayangannya ya, dan pakai bahasa Indonesia aja biar enak dipahami.

Tentu saja. Sangat wajar jika ini terasa membingungkan. Ini adalah titik di mana semua konsep yang telah kita pelajari (Aljabar Linear, Turunan, Gradien, Aturan Rantai) akhirnya **bertemu dan bekerja bersama**.

Mari kita bangun pemahamannya dari awal, dengan sangat pelan, menggunakan analogi yang jelas.

Bagian 1: Tujuan Akhir Kita - Mengajari "Bayi"

Bayangkan sebuah **Jaringan Saraf** adalah seorang "**Bayi AI**" yang baru lahir.

- "Otak"-nya (semua bobot w dan bias b) masih berisi **angka-angka acak**.
- Dia sama sekali belum tahu apa-apa.

Tugas kita sebagai "Guru":

Kita ingin mengajari si Bayi untuk bisa membedakan **gambar kucing** dan **gambar anjing**.

Bagaimana Cara Mengajarnya?

Kita akan menggunakan **data latihan (training data)**. Kita punya ribuan gambar yang sudah kita beri label:

- `Gambar_Kucing_1.jpg` → Label: $[1, 0]$ (artinya: "ini kucing, bukan anjing")
- `Gambar_Anjing_1.jpg` → Label: $[0, 1]$ (artinya: "ini bukan kucing, ini anjing")

Proses "training" adalah proses **mengajari si Bayi untuk memberikan jawaban yang benar** untuk setiap gambar di data latihan kita.

Bagian 2: Iterasi Pertama - Tebakan Pertama si Bayi

Mari kita fokus pada **satu gambar saja** untuk saat ini: `Gambar_Kucing_1.jpg`.

1. "Lihat gambar ini, Nak." (Forward Pass):

- Kita memasukkan data piksel dari `Gambar_Kucing_1.jpg` sebagai **input** (a_0) ke otak si Bayi.
- Angka-angka input ini mengalir melalui jaringan, dikalikan dengan bobot-bobot acak, ditambah bias-bias acak, melewati fungsi aktivasi, dst.
- Akhirnya, si Bayi memberikan **output** ($y_{prediksi}$). Karena otaknya masih acak, jawabannya pasti ngawur.
- Misalnya, dia menjawab: $[0.4, 0.6]$. Artinya, dia menebak "40% kemungkinan ini kucing, 60% kemungkinan ini anjing".

2. "Jawabanmu SALAH!" (Menghitung "Skor Keburukan" / Cost):

- Kita sebagai guru tahu jawaban yang benar.

- **Jawaban Benar (y_{benar}):** [1, 0]
- **Jawaban si Bayi (y_{prediksi}):** [0.4, 0.6]
- Kita hitung seberapa "salah"-nya tebakan si Bayi. Kita gunakan **Fungsi Kerugian (Cost Function)**.

$$C = (y_{\text{benar}} - y_{\text{prediksi}})^2$$

$$C = ([1, 0] - [0.4, 0.6])^2 = ([0.6, -0.6])^2 = (0.6)^2 + (-0.6)^2 = 0.36 + 0.36 = 0.72$$

- "Skor Keburukan"-mu adalah **0.72**. Ini cukup tinggi (idealnya 0).

Cek Pemahaman (Berhenti di Sini Dulu):

Ini adalah satu siklus "tebakan".

1. Apakah kamu paham proses "Forward Pass" di mana kita memasukkan input dan mendapatkan output (yang masih salah)?
2. Apakah kamu paham ide dari "Fungsi Kerugian C " sebagai cara untuk mengukur seberapa jauh tebakan si Bayi dari jawaban yang benar?

Bagaimana perasaanmu tentang dua langkah awal ini?

Sempurna! Kita sudah berhasil menyelesaikan bagian "menebak" dan "mengukur kesalahan". Si Bayi AI tahu bahwa jawabannya salah (skor keburukannya 0.72).

Sekarang, bagian yang paling ajaib dimulai: **Bagaimana cara kita "memberi tahu" si Bayi bagaimana cara memperbaiki otaknya?** Kita tidak bisa hanya bilang, "Jawabanmu salah, coba lagi!" Kita harus memberinya **petunjuk spesifik**.

Bagian Selanjutnya: "Menyalahkan" Setiap Kenop (Backpropagation)

Inilah inti dari **Backpropagation** dan **Aturan Rantai**. Kita akan bekerja **MUNDUR** dari "Skor Keburukan" untuk mencari tahu "kontribusi kesalahan" dari setiap "kenop" (bobot dan bias) di otak si Bayi.

Tujuan kita: Mengubah **semua jutaan w dan b** di dalam otak si Bayi sedikit demi sedikit, agar lain kali jika dia melihat gambar yang sama, jawabannya akan lebih baik (misal, [0.5, 0.5], yang skor keburukannya lebih rendah).

Pertanyaan Kunci untuk Setiap "Kenop":

"Hei w_{123} , jika aku memutarmu sedikit (mengubah nilaimu sedikit), seberapa besar dampak-nya pada **Skor Keburukan akhir** (C)?"

Pertanyaan "seberapa besar dampak" ini adalah pertanyaan **TURUNAN**. Kita ingin menghitung $\partial C / \partial w_{123}$ untuk **SETIAP** w dan b di seluruh jaringan.

Analogi "Efek Domino" (Aturan Rantai)

Bayangkan jaringan saraf adalah serangkaian **domino yang berbaris**.

- **Domino Paling Kanan:** Skor Keburukan C .
- **Domino di Kirinya:** Output a_{akhir} .
- **Domino di Kirinya Lagi:** Lapisan tersembunyi $a_{\text{sebelumnya}}$.
- ...dan seterusnya...
- **Domino Paling Kiri:** Salah satu "kenop" w yang ingin kita perbaiki.

$w \rightarrow \dots \rightarrow a_{\text{sebelumnya}} \rightarrow a_{\text{akhir}} \rightarrow C$

Kita ingin tahu: "**Jika aku senggol sedikit domino w di paling kiri, seberapa kuat senggolan yang akan sampai di domino C di paling kanan?**"

Aturan Rantai memberitahu kita cara menghitungnya:

"Dampak Total" = (Dampak a_{akhir} ke C) * (Dampak $a_{\text{sebelumnya}}$ ke a_{akhir}) * ...
* (Dampak w ke ...)

$$\partial C / \partial w = (\partial C / \partial a_{\text{akhir}}) * (\partial a_{\text{akhir}} / \partial a_{\text{sebelumnya}}) * \dots * (\partial \dots / \partial w)$$

Proses menghitung semua "dampak parsial" ini dari **kanan ke kiri** (dari C mundur ke w) inilah yang disebut **BACKPROPAGATION**.

Langkah Terakhir: Meng-update Otak si Bayi (Gradient Descent)

Setelah kita melakukan Backpropagation, kita akan mendapatkan sebuah **vektor Gradien raksasa**. Vektor ini berisi nilai $\partial C / \partial w$ dan $\partial C / \partial b$ untuk **semua** "kenop" di otak si Bayi.

- **Vektor Gradien ini adalah "Laporan Kesalahan":**
 - $\partial C / \partial w_{123} = +5.0$: Artinya, jika w_{123} dinaikkan, C akan naik dengan cepat. Ini "kenop" yang buruk.
 - $\partial C / \partial w_{456} = -2.0$: Artinya, jika w_{456} dinaikkan, C akan turun. Ini "kenop" yang bagus.
 - $\partial C / \partial w_{789} = +0.01$: w_{789} tidak terlalu berpengaruh pada kesalahan saat ini.

Sekarang kita terapkan Gradient Descent:

Kita ingin **menurunkan** C . Jadi, untuk setiap kenop, kita putar sedikit ke arah yang **berlawanan** dengan gradiennya.

- $w_{123_baru} = w_{123_lama} - (learning_rate * 5.0)$ (karena gradiennya positif, kita kurangi w)
- $w_{456_baru} = w_{456_lama} - (learning_rate * -2.0)$ (karena gradiennya negatif, kita tambah w)
- $w_{789_baru} = w_{789_lama} - (learning_rate * 0.01)$ (hampir tidak berubah)

Siklus Belajar Selesai (untuk satu gambar):

1. **Forward Pass:** Bayi menebak.
2. **Hitung Kesalahan:** Kita hitung skor keburukan C .
3. **Backpropagation:** Kita gunakan Aturan Rantai untuk "menyalahkan" setiap kenop ($\partial C / \partial w$).
4. **Gradient Descent:** Kita update setiap kenop sedikit demi sedikit agar lain kali kesalahannya lebih kecil.

Proses ini kemudian diulangi untuk gambar kedua, ketiga, dan seterusnya, ribuan kali, sampai "otak" si Bayi menjadi semakin pintar.

Cek Pemahaman:

1. Apakah kamu melihat bagaimana "training" adalah proses "memberi tahu si Bayi cara memperbaiki otaknya"?
2. Apakah kamu paham bahwa **Backpropagation** adalah nama keren untuk **Aturan Rantai** yang digunakan untuk mencari tahu "kontribusi kesalahan" dari setiap kenop?
3. Apakah kamu melihat bagaimana **Gradient Descent** menggunakan informasi dari Backpropagation untuk **meng-update** kenop-kenop itu?

Bagaimana perasaanmu tentang gambaran besar dari siklus "belajar" ini?

Note

Much more example that are easier to follow or understand (i hope)

Setup: Mesin Penilai Buah yang Sedang Belajar

Bayangkan "mesin penilai" kita yang paling sederhana.

- **Input:** a_0 (Kemanisan)
- **Output:** a_1 (Skor Enak)
- **"Otak" Mesin:** Hanya punya dua kenop, w dan b .
- **Resep:** $a_1 = \sigma(w * a_0 + b)$ (Kita akan abaikan σ untuk sementara agar lebih mudah).
Jadi, $a_1 = w * a_0 + b$.

Skenario Pembelajaran:

1. Kita beri buah dengan **Kemanisan** $a_0 = 2$.
2. Kita tahu buah ini seharusnya **SANGAT ENAK**. Jawaban Benar $y_{\text{benar}} = 10$.
3. Otak mesin kita masih **acak**. $w = 3$ dan $b = 1$.

Langkah 1: Tebakan Pertama Mesin (Forward Pass)

Mari kita lihat apa jawaban mesin kita.

- $a_1 = w * a_0 + b$
- $a_1 = (3) * (2) + 1 = 6 + 1 = 7$.
- **Tebakan Mesin:** 7.
- **Jawaban Benar:** 10.
- Mesin kita **SALAH**.

Langkah 2: Menghitung "Tingkat Kekecewaan" (Cost Function)

Seberapa salah tebakan itu? Mari kita hitung "Skor Kekecewaan" (C).

- $C = (y_{\text{benar}} - a_1)^2 = (10 - 7)^2 = 3^2 = 9$.
- Skor kekecewaannya adalah 9.

Langkah 3: Interogasi (Backpropagation & Aturan Rantai)

Sekarang kita akan menjadi "detektif" dan menginterogasi setiap kenop. Kita akan gunakan **TURUNAN PARSIAL** untuk mencari tahu "siapa yang paling bertanggung jawab" atas kesalahan ini.

Interogasi #1: "Hei Kenop b , apa peranmu?"

- **Pertanyaan:** "Seberapa sensitif C terhadap perubahan pada b ?" Kita ingin mencari $\partial C / \partial b$.
- **Alur Logika (Rantai):** b mengubah a_1 , dan a_1 mengubah C .
 $C \rightarrow a_1 \rightarrow b$
- **Aturan Rantai:** $\partial C / \partial b = (\partial C / \partial a_1) * (\partial a_1 / \partial b)$
- **Hitung Setiap Bagian:**
 - $\partial C / \partial a_1$ (**Seberapa sensitif C thd a_1 ?**):
 $C = (10 - a_1)^2$. Turunannya (pakai Chain Rule biasa): $2 * (10 - a_1) * (-1) = -2 * (10 - a_1)$.
Di $a_1=7$, nilainya $-2 * (10 - 7) = -6$.
 - $\partial a_1 / \partial b$ (**Seberapa sensitif a_1 thd b ?**):
 $a_1 = w * a_0 + b$. Turunannya terhadap b (anggap w , a_0 konstan) adalah 1 .
- **Gabungkan:** $\partial C / \partial b = (-6) * (1) = -6$.
- **Hasil Interogasi b :** "Kontribusi kesalahan"-mu adalah -6 .

Interogasi #2: "Hei Kenop w , bagaimana denganmu?"

- **Pertanyaan:** "Seberapa sensitif C terhadap perubahan pada w ?" Kita ingin mencari $\partial C / \partial w$.
- **Alur Logika (Rantai):** w mengubah a_1 , dan a_1 mengubah C .
 $C \rightarrow a_1 \rightarrow w$
- **Aturan Rantai:** $\partial C / \partial w = (\partial C / \partial a_1) * (\partial a_1 / \partial w)$
- **Hitung Setiap Bagian:**
 - $\partial C / \partial a_1$: Kita sudah hitung ini! Nilainya adalah -6 .
 - $\partial a_1 / \partial w$ (**Seberapa sensitif a_1 thd w ?**):
 $a_1 = w * a_0 + b$. Turunannya terhadap w (anggap a_0 , b konstan) adalah a_0 .
Di $a_0=2$, nilainya 2 .
- **Gabungkan:** $\partial C / \partial w = (-6) * (2) = -12$.
- **Hasil Interogasi w :** "Kontribusi kesalahan"-mu adalah -12 .

Langkah 4: Memberi Hukuman (Gradient Descent)

Kita sudah punya "laporan kesalahan" (Gradien) kita:

$$\nabla C = [\partial C / \partial w, \partial C / \partial b] = [-12, -6]$$

Sekarang kita **update** kenop-kenop kita agar menjadi lebih baik. Kita ingin **menurunkan** C , jadi kita bergerak ke arah **NEGATIF** Gradien.

- **Pilih Ukuran Langkah (Learning Rate):** $\alpha = 0.1$.
- **Update w :**
 $w_{\text{baru}} = w_{\text{lama}} - \alpha * (\partial C / \partial w)$

$$w_{\text{baru}} = 3 - 0.1 * (-12) = 3 + 1.2 = 4.2$$

(Karena w punya kontribusi kesalahan negatif yang besar, kita **NAIKKAN** nilainya).

- **Update b :**

$$b_{\text{baru}} = b_{\text{lama}} - \alpha * (\partial C / \partial b)$$

$$b_{\text{baru}} = 1 - 0.1 * (-6) = 1 + 0.6 = 1.6$$

(Karena b juga punya kontribusi kesalahan negatif, kita **NAIKKAN** nilainya).

Otak Mesin yang Baru: Sekarang $w = 4.2$ dan $b = 1.6$.

Verifikasi: Apakah Mesinnya Jadi Lebih Pintar?

Mari kita coba lagi dengan otak yang baru. Masukkan input yang sama $a_0 = 2$.

- $a_1_{\text{baru}} = w_{\text{baru}} * a_0 + b_{\text{baru}}$
- $a_1_{\text{baru}} = (4.2) * (2) + 1.6 = 8.4 + 1.6 = 10$.

Tebakan Baru Mesin: 10.

Jawaban Benar: 10.

Mesin kita sekarang SEMPURNA! (Ini hanya terjadi karena contohnya sangat sederhana).

Cek Pemahaman:

1. Apakah kamu melihat bagaimana kita menggunakan Aturan Rantai untuk "menginterogasi" setiap kenop (w dan b) secara terpisah untuk menemukan "kontribusi kesalahan"-nya ($\partial C / \partial w$ dan $\partial C / \partial b$)?
2. Apakah kamu melihat bagaimana kita menggunakan hasil interogasi itu (Gradien) untuk memberikan "hukuman" atau "penyesuaian" pada setiap kenop (Gradient Descent)?

Bagaimana perasaanmu tentang alur "Tebak → Ukur Salah → Interogasi → Hukum" ini?

No 1 :

1. In this exercise we'll look in more detail about back-propagation, using the chain rule, in order to train our neural networks.

1 point

Let's look again at our two-node network.



Recall the activation equations are,

$$a^{(1)} = \sigma(z^{(1)})$$

$$z^{(1)} = w^{(1)}a^{(0)} + b^{(1)}$$

Where we've introduced $z^{(1)}$ as the weighted sum of activation and bias.

We can formalise how good (or bad) our neural network is at getting the desired behaviour. For a particular input, x , and desired output y , we can define the cost of that specific training example as the square of the difference between the network's output and the desired output, that is,

$$C_k = (a^{(1)} - y)^2$$

Where k labels the training example and $a^{(1)}$ is assumed to be the activation of the output neuron when the input neuron $a^{(0)}$ is set to x

We'll go into detail about how to apply this to an entire set of training data later on. But for now, let's look at our toy example.

Recall our NOT function example from the previous quiz. For the input $x = 1$ we would like that the network outputs $y = 0$. For the starting weight and bias $w^{(1)} = 1.3$ and $b^{(1)} = -0.1$, the network actually outputs $a^{(1)} = 0.834$. If we work out the cost function for this example, we get

$$C_1 = (0.834 - 0)^2 = 0.696.$$

Do the same calculation for an input $x = 0$ and desired output $y = 1$. Use the code block to help you.

Do the same calculation for an input $x = 0$ and desired output $y = 1$. Use the code block to help you.

```
1 # First we set the state of the network
2 sigma = np.tanh
3 w1 = 1.3
4 b1 = -0.1
5
6 # Then we define the neuron activation.
7 def a1(a0):
8     z = w1 * a0 + b1
9     return sigma(z)
10
11 # Experiment with different values of x below.
12 x = 0
13 a1(x)
14
```

Run

Reset

-0.099667994625

What is C_0 in this particular case? Give your result to 1 decimal place.

1.2

Penjelasan

Tentu saja! Ini adalah langkah yang sempurna. Mari kita kerjakan *Practice Assignment* ini bersama-sama. Kita akan membedah soalnya, memahami kodenya, dan menghitung jawabannya langkah demi langkah.

Langkah 1: Memahami Masalah

Apa yang kita punya?

- **Jaringan Saraf Sederhana:** Satu input a^0 , satu output a^1 . Ini adalah "mesin penilai buah" kita yang paling dasar.
- **"Otak" Awal Mesin:**
 - Bobot (w^1): 1.3

- Bias (b^1): -0.1
- **Resep Lengkapnya:**
 1. Hitung "stimulasi total": $z^1 = w^1 * a^0 + b^1$
 2. Terapkan "tombol pemicu": $a^1 = \sigma(z^1)$
 - Di sini, σ adalah fungsi `tanh` (hyperbolic tangent).
- **Fungsi "Kekecewaan" (Cost Function):**

$$C_k = (a^1 - y)^2$$

(Kuadrat dari selisih antara tebakan mesin a^1 dengan jawaban benar y).

Apa Tugas Kita?

Soal ini meminta kita untuk melakukan **satu siklus "menebak" dan "mengukur kesalahan"** untuk sebuah **contoh baru**.

- **Input Baru:** $x = 0$ (ini adalah a^0 kita).
- **Jawaban Benar yang Diinginkan:** $y = 1$.

Langkah 2: Menganalisis Kode Python

Mari kita lihat kode yang diberikan. Kode ini adalah implementasi langsung dari "resep" di atas.

```
# First we set the state of the network
o = np.tanh # 'o' adalah nama lain untuk sigma (σ)
w1 = 1.3
b1 = -0.1

# Then we define the neuron activation.
def a1(a0):
    z = w1 * a0 + b1 # Menghitung stimulasi total z¹
    return o(z)      # Menerapkan sigma (tanh) pada z¹

# Experiment with different values of x below.
x = 0
a1(x)
```

- Kode ini sudah mengatur "otak" awal ($w1$, $b1$).
- Fungsi `a1(a0)` adalah "mesin" kita. Ia mengambil satu input $a0$ dan menghasilkan satu output $a1$.
- Baris terakhir adalah kita **menggunakan mesin itu** dengan input $x = 0$.

Langkah 3: Menjalankan Perhitungan (Forward Pass)

Mari kita ikuti alur perhitungan untuk $x = 0$.

1. **Masukkan $a^0 = 0$ ke dalam mesin:**

$$z^1 = w^1 * a^0 + b^1$$

$$z^1 = (1.3) * (0) + (-0.1)$$

$$z^1 = 0 - 0.1$$

$$z^1 = -0.1$$

2. **Terapkan "Tombol Pemicu" σ (yaitu \tanh):**

$$a^1 = \sigma(z^1) = \tanh(-0.1)$$

- Kita tidak perlu menghitung $\tanh(-0.1)$ secara manual. Komputer di *code block* sudah melakukannya untuk kita.
- Hasilnya (seperti yang terlihat di output $-0.099\dots$) adalah $a^1 \approx -0.09966$.

Kesimpulan "Tebakan":

Untuk input 0 , mesin kita menebak outputnya adalah -0.09966 .

Langkah 4: Menghitung "Skor Kekecewaan" (C_0)

Sekarang kita bandingkan tebakan mesin dengan jawaban yang benar.

- **Tebakan Mesin (a^1):** -0.09966
- **Jawaban Benar (y):** 1

Gunakan Rumus Cost Function:

$$C_0 = (a^1 - y)^2$$

$$C_0 = (-0.09966 - 1)^2$$

$$C_0 = (-1.09966)^2$$

Sekarang, kita hitung kuadratnya:

$$C_0 \approx 1.209\dots$$

No 2 :

2. The cost function of a training set is the average of the individual cost functions of the data in the training set,

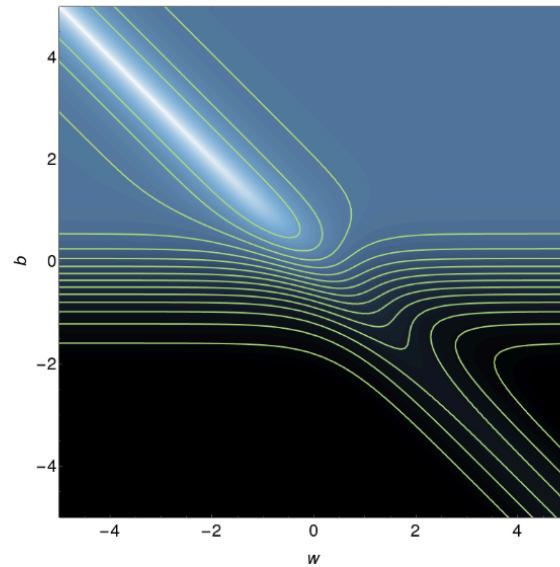
1 point

$$C = \frac{1}{N} \sum_k C_k,$$

where N is the number of examples in the training set.

For the NOT function we've been considering, where we have two examples in our training set, $(x = 0, y = 1)$ and $(x = 1, y = 0)$, the training set cost function is $C = \frac{1}{2}(C_0 + C_1)$.

Since our parameter space is 2D, $(w^{(1)} \text{ and } b^{(1)})$, we can draw the total cost function for this neural network as a contour map.



Here white represents low costs and black represents high costs.

Which of the following statements are true?

Which of the following statements are true?

- ☐ Descending perpendicular to the contours will improve the performance of the network.
- ☐ The optimal configuration lies along the line $b = 0$.
- ☐ None of the other statements are true.
- ☐ There are many different local minima in this system.
- ☐ The optimal configuration lies somewhere along the line $b = -w$.

Penjelasan :

Analisis Soal #2

Apa yang Kita Punya?

1. Cost Function untuk Seluruh Training Set:

$$C = (1/N) * \sum C_k$$

- **Artinya:** "Skor Keburukan Total" (C) adalah **rata-rata** dari "Skor Keburukan Individual" (C_k) untuk setiap contoh data.

2. Training Set Spesifik (Fungsi NOT):

- Contoh 1: input $x=0$, output $y=1$. (Ini menghasilkan C_0).
- Contoh 2: input $x=1$, output $y=0$. (Ini menghasilkan C_1).
- **Cost Total:** $C = \frac{1}{2} * (C_0 + C_1)$

3. "Lanskap Kerugian" (Peta Kontur):

- Gambar ini adalah visualisasi dari C sebagai fungsi dari w (sumbu horizontal) dan b (sumbu vertikal).
- **Area Putih/Terang:** C rendah (fit bagus). Inilah "lembah" yang kita cari.
- **Area Hitam/Gelap:** C tinggi (fit buruk). Ini adalah "gunung".
- **Garis-garis Hijau:** Garis kontur, menghubungkan titik-titik dengan nilai C yang sama.

Tugas Kita:

Menganalisis peta kontur ini dan memilih pernyataan yang benar.

Mengevaluasi Setiap Pilihan Jawaban

Mari kita bedah setiap pernyataan satu per satu.

1. Descending perpendicular to the contours will improve the performance of the network.

(Menuruni lereng secara tegak lurus terhadap kontur akan meningkatkan performa jaringan.)

- **Analisis:**

- "Meningkatkan performa" berarti "menurunkan nilai Cost (C)".
- Arah yang **tegak lurus terhadap garis kontur** adalah **arah perubahan tercepat**.
- **Gradien** (∇C) menunjuk ke arah tanjakan tercepat (tegak lurus kontur, ke arah terang).
- **Negatif Gradien** ($-\nabla C$) menunjuk ke arah **turunan tercepat** (tegak lurus kontur, ke arah gelap... oh, tunggu, di sini terang = rendah, gelap = tinggi).
- Di soal ini, **putih = rendah**. Jadi, Gradien menunjuk ke arah gelap, dan **-Gradien menunjuk ke arah putih**, yaitu arah turunan tercepat.
- Jadi, menuruni lereng secara tegak lurus terhadap kontur (mengikuti $-\nabla C$) akan membawa kita ke nilai C yang lebih rendah dengan paling efisien.

- **Kesimpulan:** Pernyataan ini terdengar **BENAR**.

2. The optimal configuration lies along the line $b = 0$.

(Konfigurasi optimal terletak di sepanjang garis $b = 0$.)

- **Analisis:**

- "Konfigurasi optimal" adalah titik dengan C terendah, yaitu "dasar lembah" yang paling putih.
- Garis $b=0$ adalah sumbu-x (horizontal) pada grafik.
- Lihat petanya. "Lembah" putih yang panjang itu **tidak** terletak di $b=0$. Ia terlihat miring ke bawah, dimulai dari sekitar $b=4$ di kiri dan berakhir di sekitar $b=-4$ di kanan.

- **Kesimpulan:** Pernyataan ini **SALAH**.

3. There are many different local minima in this system.

(Ada banyak lembah lokal yang berbeda di sistem ini.)

- **Analisis:**

- "Local minima" adalah "dasar lembah".
- Pada peta kontur, sebuah lembah terlihat sebagai area yang dikelilingi oleh kontur-kontur tertutup.
- Di gambar ini, kita hanya melihat **satu area lembah yang panjang dan berkelok**. Tidak ada "kolam-kolam" atau "cekungan-cekungan" terpisah lainnya.

- **Kesimpulan:** Pernyataan ini **SALAH**.

4. The optimal configuration lies somewhere along the line $b = -w$.

(Konfigurasi optimal terletak di suatu tempat di sepanjang garis $b = -w$.)

- **Analisis:**

- Garis $b = -w$ adalah sebuah garis lurus yang melewati $(0,0)$, $(2, -2)$, $(-2, 2)$, dst.
- Coba gambarkan garis ini secara imajiner di atas peta. Garis ini akan memotong "lembah" putih kita di satu titik.
- Seluruh "lembah" solusi optimal terlihat mengikuti garis $b = -w$ ini dengan cukup baik.

- **Kesimpulan:** Pernyataan ini terlihat **SANGAT MUNGKIN BENAR**.

Mengevaluasi Ulang: Pernyataan 1 vs. 4

Kita punya dua kandidat kuat: #1 dan #4. Mari kita pikirkan lagi.

- **Pernyataan #1:** Descending perpendicular to the contours... Ini adalah **definisi** dari Gradient Descent. Ini adalah pernyataan tentang **bagaimana cara** menuju ke solusi. Pernyataan ini secara umum benar tentang metode optimisasi.
- **Pernyataan #4:** ...lies somewhere along the line $b = -w$. Ini adalah pernyataan tentang **di mana letak solusi** itu sendiri.

Mungkinkah ada jebakan? Mari kita lihat lebih dekat $b = -w$.

Jika kita punya $z = w*x + b$, dan kita ingin outputnya mirip dengan fungsi NOT ($y=0$ untuk $x=1$, $y=1$ untuk $x=0$).

- Untuk $x=1$, $y=0$: $\tanh(w*1 + b) \approx 0 \rightarrow w+b \approx 0 \rightarrow b \approx -w$.
- Untuk $x=0$, $y=1$: $\tanh(w*0 + b) \approx 1 \rightarrow \tanh(b) \approx 1 \rightarrow b$ harus positif besar.

Kedua kondisi ini sedikit bertentangan. Jadi $b = -w$ adalah aproksimasi yang bagus, tapi mungkin tidak sempurna di sepanjang lembah.

Bagaimana dengan Pernyataan #1?

"Menuruni lereng secara tegak lurus terhadap kontur AKAN meningkatkan performa."

- "Meningkatkan performa" = "menurunkan Cost".
- Arah $-\text{Gradien}$ adalah arah turunan tercepat.
- Arah Gradien selalu tegak lurus kontur.
- Jadi, bergerak ke arah $-\text{Gradien}$ (tegak lurus kontur ke arah lembah) **dijamin** akan menurunkan Cost (setidaknya untuk langkah yang sangat kecil).
- Pernyataan ini **benar secara definisi**.

Kesimpulan Akhir:

Pernyataan #1 adalah pernyataan yang paling fundamental dan benar secara definisi tentang bagaimana optimisasi berbasis gradien bekerja pada lanskap ini. Pernyataan #4 adalah observasi yang bagus tentang bentuk lembah, tapi mungkin tidak 100% akurat secara matematis di semua titik.

Jadi, jawaban yang paling pasti benar adalah **pernyataan pertama**.

No 3 :

3. To improve the performance of the neural network on the training data, we can vary the weight and bias. We can calculate the derivative of the example cost with respect to these quantities using the chain rule.

1 point

$$\frac{\partial C_k}{\partial w^{(1)}} = \frac{\partial C_k}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}}$$

$$\frac{\partial C_k}{\partial b^{(1)}} = \frac{\partial C_k}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial b^{(1)}}$$

Individually, these derivatives take fairly simple form. Go ahead and calculate them. We'll repeat the defining equations for convenience,

$$a^{(1)} = \sigma(z^{(1)})$$

$$z^{(1)} = w^{(1)}a^{(0)} + b^{(1)}$$

$$C_k = (a^{(1)} - y)^2$$

Select all true statements below.

Select all true statements below.

☐ None of the other statements.

☐ $\frac{\partial z^{(1)}}{\partial w^{(1)}} = w^{(1)}$

☐ $\frac{\partial z^{(1)}}{\partial b^{(1)}} = a^{(1)}$

☐ $\frac{\partial a^{(1)}}{\partial z^{(1)}} = \sigma'(z^{(1)})$

☐ $\frac{\partial C_k}{\partial a^{(1)}} = (1 - y)^2$

☐ $\frac{\partial C_k}{\partial a^{(1)}} = 2(a^{(1)} - y)$

☐ $\frac{\partial z^{(1)}}{\partial b^{(1)}} = 1$

☐ $\frac{\partial a^{(1)}}{\partial z^{(1)}} = \sigma$

☐ $\frac{\partial z^{(1)}}{\partial w^{(1)}} = a^{(0)}$

Penjelasan :

Tugas:

Kita diminta untuk menghitung setiap turunan parsial kecil ($\partial C_k / \partial a^1$, $\partial a^1 / \partial z^1$, $\partial z^1 / \partial w^1$, $\partial z^1 / \partial b^1$) dan memilih pernyataan yang benar dari daftar.

Analisis "Mata Rantai" Satu per Satu

Setup Persamaan:

- $C_k = (a^1 - y)^2$ (Fungsi Cost)
- $a^1 = \sigma(z^1)$ (Fungsi Aktivasi)
- $z^1 = w^1 a^0 + b^1$ (Fungsi Linear)

Mari kita turunkan setiap bagian.

1. $\partial C_k / \partial a^1$ (Seberapa sensitif Cost terhadap Output Akhir?)

- Fungsi:** $C_k = (a^1 - y)^2$
- Variabel Aktif:** a^1 . Anggap y (jawaban benar) sebagai konstanta.
- Proses:** Kita gunakan Aturan Rantai biasa (atau Power Rule yang diperluas).
 - Turunan dari $(\text{sesuatu})^2$ adalah $2 * (\text{sesuatu})$.

- Lalu, kalikan dengan turunan dari "sesuatu" itu sendiri ($a^1 - y$) terhadap a^1 , yang adalah 1.
 - **Hasil:** $\partial C_k / \partial a^1 = 2 * (a^1 - y) * 1 = 2(a^1 - y)$
 - **Cek Pilihan Jawaban:**
 - $\partial C_k / \partial a^1 = (1 - y)^2 \rightarrow$ **SALAH.**
 - $\partial C_k / \partial a^1 = 2(a^1 - y) \rightarrow$ **BENAR.**
-

2. $\partial a^1 / \partial z^1$ (Seberapa sensitif Aktivasi terhadap Stimulasi?)

- **Fungsi:** $a^1 = \sigma(z^1)$
 - **Variabel Aktif:** z^1 .
 - **Proses:** Ini adalah turunan dari fungsi σ terhadap inputnya. Notasi standar untuk turunan dari sebuah fungsi (misal $f(x)$) adalah $f'(x)$.
 - **Hasil:** Turunan dari $\sigma(z^1)$ terhadap z^1 adalah $\sigma'(z^1)$.
 - **Cek Pilihan Jawaban:**
 - $\partial a^1 / \partial z^1 = \sigma'(z^1) \rightarrow$ **BENAR.**
 - $\partial a^1 / \partial z^1 = \sigma \rightarrow$ **SALAH.** Ini adalah kesalahan umum, σ adalah fungsinya, σ' adalah turunannya.
-

3. $\partial z^1 / \partial w^1$ (Seberapa sensitif Stimulasi terhadap Bobot?)

- **Fungsi:** $z^1 = w^1 a^0 + b^1$
 - **Variabel Aktif:** w^1 . Anggap a^0 dan b^1 sebagai konstanta.
 - **Proses:** Persamaannya bisa dibaca sebagai $z^1 = (a^0) * w^1 + (\text{konstanta})$.
 - Turunan dari $(\text{konstanta}) * w^1$ terhadap w^1 adalah konstanta itu sendiri.
 - **Hasil:** $\partial z^1 / \partial w^1 = a^0$
 - **Cek Pilihan Jawaban:**
 - $\partial z^1 / \partial w^1 = w^1 \rightarrow$ **SALAH.**
 - $\partial z^1 / \partial w^1 = a^0 \rightarrow$ **BENAR.**
-

4. $\partial z^1 / \partial b^1$ (Seberapa sensitif Stimulasi terhadap Bias?)

- **Fungsi:** $z^1 = w^1 a^0 + b^1$
- **Variabel Aktif:** b^1 . Anggap w^1 dan a^0 sebagai konstanta.
- **Proses:** Persamaannya bisa dibaca sebagai $z^1 = (\text{konstanta}) + 1 * b^1$.
 - Turunan dari $1 * b^1$ terhadap b^1 adalah 1.
- **Hasil:** $\partial z^1 / \partial b^1 = 1$

- **Cek Pilihan Jawaban:**

- $\partial z^1 / \partial b^1 = a^1 \rightarrow$ **SALAH.**
- $\partial z^1 / \partial b^1 = 1 \rightarrow$ **BENAR.**

Kesimpulan

Berdasarkan analisis kita, pernyataan yang benar adalah:

- $\partial C_k / \partial a^1 = 2(a^1 - y)$
- $\partial a^1 / \partial z^1 = \sigma'(z^1)$
- $\partial z^1 / \partial w^1 = a^0$
- $\partial z^1 / \partial b^1 = 1$

Sekarang kamu tinggal mencocokkan hasil ini dengan kotak centang yang tersedia di *assignment*. Ini adalah latihan yang sangat baik untuk memastikan kamu nyaman dengan mekanika turunan parsial dan Aturan Rantai.

No 4 :

4. Using your answer to the previous question, let's see it implemented in code.

1 point

The following code block has an example implementation of $\frac{\partial C_k}{\partial w^{(l)}}$. It is up to you to implement $\frac{\partial C_k}{\partial b^{(l)}}$.

Don't worry if you don't know exactly how the code works. It's more important that you get a feel for what is going on.

We will introduce the following derivative in the code,

$$\frac{d}{dz} \tanh(z) = \frac{1}{\cosh^2 z}.$$

Complete the function 'dCdb' below. Replace the ??? towards the bottom, with the expression you calculated in the previous question.

```
1 # First define our sigma function.
2 sigma = np.tanh
3
4 # Next define the feed-forward equation.
5 def a1(w1, b1, a0):
6     z = w1 * a0 + b1
7     return sigma(z)
8
9 # The individual cost function is the square of the difference between
10 # the network output and the training data output.
11 def C(w1, b1, x, y):
12     return (a1(w1, b1, x) - y)**2
13
14 # This function returns the derivative of the cost function with
15 # respect to the weight.
16 def dCdw(w1, b1, x, y):
17     z = w1 * x + b1
18     dCda = 2 * (a1(w1, b1, x) - y) # Derivative of cost with activation
19     dadz = 1/np.cosh(z)**2 # derivative of activation with weighted sum z
20     dzdw = x # derivative of weighted sum z with weight
21     return dCda * dadz * dzdw # Return the chain rule product.
22
23 # This function returns the derivative of the cost function with
24 # respect to the bias.
25 # It is very similar to the previous function.
26 # You should complete this function.
27 def dCdb(w1, b1, x, y):
28     z = w1 * x + b1
29     dCda = 2 * (a1(w1, b1, x) - y)
30     dadz = 1/np.cosh(z)**2
31     """ Change the next line to give the derivative of
32     the weighted sum, z, with respect to the bias, b. """
33     dzdb = ???
34     return dCda * dadz * dzdb
35
36 """Test your code before submission:"""
37 # Let's start with an unfit weight and bias.
38 w1 = 2.3
39 b1 = -1.2
40 # We can test on a single data point pair of x and y.
```

Run

Reset

```

23 # This function returns the derivative of the cost function with
24 # respect to the bias.
25 # It is very similar to the previous function.
26 # You should complete this function.
27 def dCdb (w1, b1, x, y) :
28     z = w1 * x + b1
29     dCda = 2 * (a1(w1, b1, x) - y)
30     dadz = 1/np.cosh(z)**2
31     """ Change the next line to give the derivative of
32     the weighted sum, z, with respect to the bias, b. """
33     dzdb = 1
34     return dCda * dadz * dzdb
35
36 """Test your code before submission:"""
37 # Let's start with an unfit weight and bias.
38 w1 = 2.3
39 b1 = -1.2
40 # We can test on a single data point pair of x and y.
41 x = 0
42 y = 1
43 # Output how the cost would change
44 # in proportion to a small change in the bias
45 print( dCdb(w1, b1, x, y) )
46

```

Run

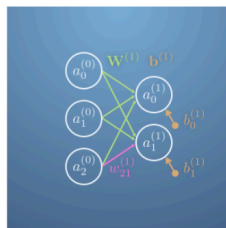
Reset

-1.11860264255

No 5:

5. Recall that when we add more neurons to the network, our quantities are upgraded to vectors or matrices.

1 point



$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}),$$

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)}$$

The individual cost functions remain scalars. Instead of becoming vectors, the components are summed over each output neuron.

$$C_k = \sum_i (a_i^{(1)} - y_i)^2$$

Note here that i labels the output neuron and is summed over, whereas k labels the training example.

The training data becomes a vector too,

$\mathbf{x} \rightarrow \mathbf{x}$ and has the same number of elements as input neurons.

$\mathbf{y} \rightarrow \mathbf{y}$ and has the same number of elements as output neurons.

This allows us to write the cost function in vector form using the modulus squared,

$$C_k = \|\mathbf{a}^{(1)} - \mathbf{y}\|^2.$$

Use the code block below to play with calculating the cost function for this network.

Use the code block below to play with calculating the cost function for this network.

```

1  # Define the activation function.
2  sigma = np.tanh
3
4  # Let's use a random initial weight and bias.
5  W = np.array([[ -0.94529712, -0.2667356 , -0.91219181],
6                [ 2.05529992,  1.21797092,  0.22914497]])
7  b = np.array([ 0.61273249,  1.6422662  ])
8
9  # define our feed forward function
10 def a1 (a0) :
11     # Notice the next line is almost the same as previously,
12     # except we are using matrix multiplication rather than scalar multiplication
13     # hence the '@' operator, and not the '*' operator.
14     z = W @ a0 + b
15     # Everything else is the same though,
16     return sigma(z)
17
18 # Next, if a training example is,
19 x = np.array([0.7, 0.6, 0.2])
20 y = np.array([0.9, 0.6])
21
22 # Then the cost function is,
23 d = a1(x) - y # Vector difference between observed and expected activation
24 C = d @ d # Absolute value squared of the difference.
25

```

Run

Reset

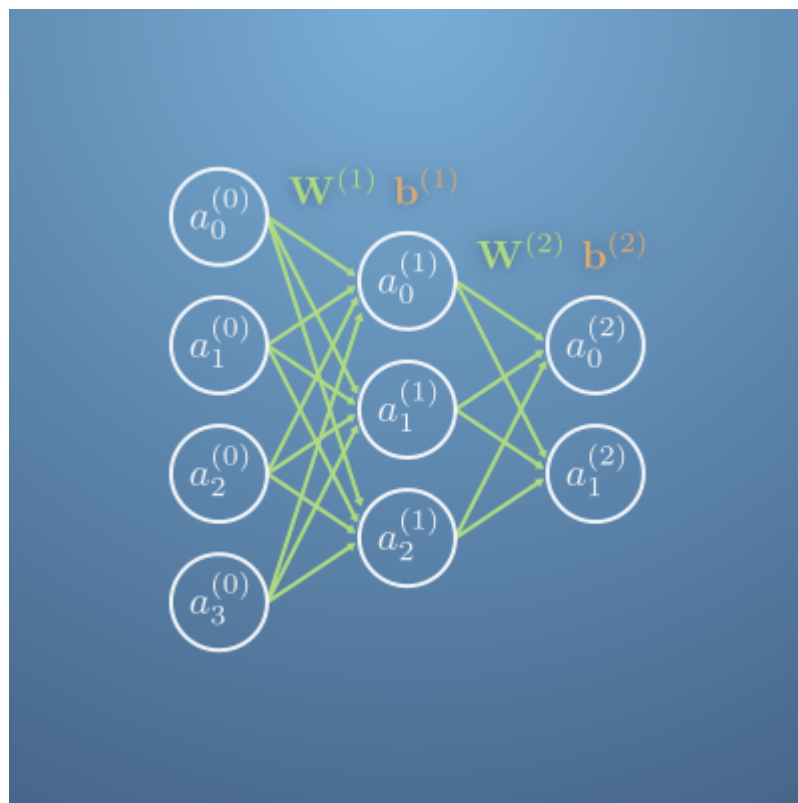
1.77883409525

For the initial weights and biases, what is the example cost function, C_k , when, $x = \begin{bmatrix} 0.7 \\ 0.6 \\ 0.2 \end{bmatrix}$ and $y = \begin{bmatrix} 0.9 \\ 0.6 \end{bmatrix}$?

Give your answer to 1 decimal place.

1.8

No 6 :



Training this network is done by *back-propagation* because we start at the output layer and calculate derivatives backwards towards the input layer with the chain rule.

Let's see how this works.

If we wanted to calculate the derivative of the cost with respect to the weight or bias of the final layer, then this is the same as previously (but now in vector form):

$$\frac{\partial C_k}{\partial \mathbf{W}^{(2)}} = \frac{\partial C_k}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(2)}}$$

With a similar term for the bias. If we want to calculate the derivative with respects to weights of the previous layer, we use the expression,

$$\frac{\partial C_k}{\partial \mathbf{W}^{(1)}} = \frac{\partial C_k}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}}$$

Where $\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}}$ itself can be expanded to,

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} = \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}}$$

This can be generalised to any layer,

$$\frac{\partial C_k}{\partial \mathbf{W}^{(i)}} = \frac{\partial C_k}{\partial \mathbf{a}^{(N)}} \underbrace{\frac{\partial \mathbf{a}^{(N)}}{\partial \mathbf{a}^{(N-1)}} \frac{\partial \mathbf{a}^{(N-1)}}{\partial \mathbf{a}^{(N-2)}} \cdots \frac{\partial \mathbf{a}^{(i+1)}}{\partial \mathbf{a}^{(i)}}}_{\text{from layer } N \text{ to layer } i} \frac{\partial \mathbf{a}^{(i)}}{\partial \mathbf{z}^{(i)}} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{W}^{(i)}}$$

By further application of the chain rule.

This can be generalised to any layer,

$$\frac{\partial C_k}{\partial \mathbf{W}^{(i)}} = \frac{\partial C_k}{\partial \mathbf{a}^{(N)}} \underbrace{\frac{\partial \mathbf{a}^{(N)}}{\partial \mathbf{a}^{(N-1)}} \frac{\partial \mathbf{a}^{(N-1)}}{\partial \mathbf{a}^{(N-2)}} \cdots \frac{\partial \mathbf{a}^{(i+1)}}{\partial \mathbf{a}^{(i)}}}_{\text{from layer } N \text{ to layer } i} \frac{\partial \mathbf{a}^{(i)}}{\partial \mathbf{z}^{(i)}} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{W}^{(i)}}$$

By further application of the chain rule.

Choose the correct expression for the derivative,

$$\frac{\partial \mathbf{a}^{(j)}}{\partial \mathbf{a}^{(j-1)}}$$

Remembering the activation equations are,

$$\mathbf{a}^{(n)} = \sigma(\mathbf{z}^{(n)})$$

$$\mathbf{z}^{(n)} = \mathbf{w}^{(n)} \mathbf{a}^{(n-1)} + \mathbf{b}^{(n)}.$$

☐ $\sigma'(\mathbf{z}^{(j)}) \mathbf{W}^{(j-1)}$

☐ $\mathbf{W}^{(j)} \mathbf{a}^{(j)}$

☐ $\sigma'(\mathbf{z}^{(j)}) \mathbf{W}^{(j)}$

☐ $\frac{\sigma'(\mathbf{z}^{(j)})}{\sigma'(\mathbf{z}^{(j-1)})}$

Penjelasan :

Tentu saja. Latihan ini adalah "level up" dari yang sebelumnya. Kita sekarang beralih dari satu neuron ke **jaringan dengan banyak lapisan**, dan dari **aljabar skalar** ke **aljabar matriks dan vektor**.

Ini adalah inti dari Backpropagation.

Analisis Masalah

Setup:

- Kita punya jaringan saraf dengan beberapa lapisan (input a^0 , tersembunyi a^1 , output a^2).
- Backpropagation:** Adalah nama keren untuk proses menghitung turunan dari Cost (C_k) terhadap semua W dan b , dengan cara bekerja **MUNDUR** dari output menggunakan **Aturan Rantai Multivariat**.

Rantai-Rantai Kunci:

- Untuk W^2 (Lapisan Terakhir):** $C_k \rightarrow a^2 \rightarrow z^2 \rightarrow W^2$
 $\partial C_k / \partial W^2 = (\partial C_k / \partial a^2) * (\partial a^2 / \partial z^2) * (\partial z^2 / \partial W^2)$
(Ini persis sama seperti soal sebelumnya, hanya sekarang W^2 adalah matriks dan a^2 adalah vektor).
- Untuk W^1 (Lapisan Sebelumnya):** $C_k \rightarrow a^2 \rightarrow z^2 \rightarrow a^1 \rightarrow z^1 \rightarrow W^1$
 $\partial C_k / \partial W^1 = (\partial C_k / \partial a^2) * (\partial a^2 / \partial z^2) * (\partial a^1 / \partial z^1) * (\partial z^1 / \partial W^1)$ -- Hmm, ada kesalahan di screenshot! Mari kita perbaiki. Rantainya harus melalui a^1 .
Rantai yang Benar: $C_k \rightarrow a^2 \rightarrow a^1 \rightarrow z^1 \rightarrow W^1$
 $\partial C_k / \partial W^1 = (\partial C_k / \partial a^2) * (\partial a^2 / \partial a^1) * (\partial a^1 / \partial z^1) * (\partial z^1 / \partial W^1)$

Tugas Kita:

Soal ini meminta kita untuk fokus pada **satu "mata rantai" spesifik** dari rantai panjang ini, yaitu:

$$\partial a^{(j)} / \partial a^{(j-1)}$$

Pertanyaan: Apa ekspresi yang benar untuk mata rantai ini? Ini menanyakan: "Seberapa sensitif **output dari lapisan j (a^j)** terhadap **output dari lapisan $j-1$ (a^{j-1})**?"

Membongkar Mata Rantai $\partial a^j / \partial a^{j-1}$

Mari kita lihat alur dari a^{j-1} ke a^j :

$$a^{j-1} \rightarrow z^j \rightarrow a^j$$

Ini adalah rantai dua langkah:

- a^{j-1} (output dari lapisan sebelumnya) pertama-tama mempengaruhi z^j (stimulasi di lapisan saat ini).
- Kemudian z^j mempengaruhi a^j (aktivasi di lapisan saat ini).

Menggunakan Aturan Rantai, kita bisa memecahnya:

$$\partial a^j / \partial a^{j-1} = (\partial a^j / \partial z^j) * (\partial z^j / \partial a^{j-1})$$

Sekarang, mari kita hitung setiap bagian dari perkalian ini.

Bagian 1: $\partial a^j / \partial z^j$ (Aktivasi terhadap Stimulasi)

- **Fungsi:** $a^j = \sigma(z^j)$
- Ini adalah turunan dari fungsi aktivasi σ terhadap inputnya.
- **Hasil:** $\sigma'(z^j)$.
(Penting: Ini akan menjadi sebuah **matriks diagonal** jika z^j adalah vektor, di mana elemen diagonalnya adalah turunan dari setiap komponen).

Bagian 2: $\partial z^j / \partial a^{j-1}$ (Stimulasi terhadap Input Lapisan)

- **Fungsi:** $z^j = W^j * a^{j-1} + b^j$
- **Variabel Aktif:** a^{j-1} (ini adalah vektor). W^j dan b^j kita anggap konstanta untuk turunan ini.
- **Proses:** Ini adalah turunan dari (Matriks * Vektor) terhadap Vektor itu.
 - Ingat dari turunan biasa: $d/dx (c*x) = c$.
 - Hal yang sama berlaku di sini. Turunan dari $W^j * a^{j-1}$ terhadap a^{j-1} adalah **matriks** W^j itu sendiri.
- **Hasil:** $\partial z^j / \partial a^{j-1} = W^j$

Menggabungkan Hasilnya

Sekarang kita gabungkan kedua bagian itu kembali ke dalam rumus rantai kita:

$$\begin{aligned}\partial a^j / \partial a^{j-1} &= (\partial a^j / \partial z^j) * (\partial z^j / \partial a^{j-1}) \\ \partial a^j / \partial a^{j-1} &= \sigma'(z^j) * W^j\end{aligned}$$

Ini adalah "mata rantai" yang menghubungkan satu lapisan dengan lapisan sebelumnya. Ini memberitahu kita bagaimana perubahan di a^{j-1} "merambat" ke a^j melalui bobot W^j dan fungsi aktivasi σ .

Memilih Jawaban yang Benar

Sekarang kita cocokkan hasil kita dengan pilihan yang ada:

- $\sigma'(x^j) W^{(j-1)}$ → Salah, x tidak terdefinisi dan W salah indeks.
- $W^j a^{(j)}$ → Salah, ini adalah bagian dari perhitungan z , bukan turunan.
- $\sigma'(z^j) W^j$ → **BENAR**.
- $\sigma'(z^j) / \sigma'(z^{(j-1)})$ → Salah, ini tidak mengikuti Aturan Rantai.

Kesimpulan:

Ekspresi yang benar untuk turunan $\partial a^j / \partial a^{j-1}$ adalah $\sigma'(z^j) W^j$. Ini adalah blok bangunan fundamental dari algoritma Backpropagation.

Tags: #mml-specialization #multivariate-calculus #neural-networks #training #cost-function #gradient-descent #backpropagation