

03: Simple Neural Networks

Chapter Goal: To understand that a **Neural Network** is not a mysterious black box, but simply a complex mathematical **function**, and to build it from its most basic "Lego brick": a single neuron.

1. Core Idea: A Neural Network is Just a Function

- **Don't be intimidated:** A Neural Network diagram with its circles ("neurons") and lines ("connections") is just a visual representation of a complex mathematical **FUNCTION**.
 - **Its Job:** Like any other function, it takes an **input** (which can be a vector) and produces an **output** (which can also be a vector).
-

2. The Most Basic "Lego Brick": A Single Neuron

- **Scenario:** One input a_0 (a number) → One output a_1 (a number).
- **The Mathematical "Recipe":**



$$a_1 = \sigma(w \cdot a_0 + b)$$

- **Anatomy of the Recipe:**
 - a (**Activations**): The input value (a_0) and the output value (a_1).
 - w (**Weight**): A number that controls the "strength" of the connection. It's like a "volume knob" for the input a_0 .
 - b (**Bias**): A number that gets added. It's like a "slider" that can shift the final result up or down, regardless of the input.
 - σ (**Sigma / Activation Function**): This is the most important part. σ is a **non-linear function** that is "applied" to the result of $w \cdot a_0 + b$.

3. Why Do We Need an Activation Function σ ? (The Key to a Neural Network's Power)

- **Brain Inspiration:** Neurons in the brain don't respond linearly. They "stay quiet" until the total stimulation reaches a certain threshold, and then they "fire" (activate).
- The σ function mimics this behavior. It adds **non-linearity** to the system.
- **Example:** A **Sigmoid** function (S-shaped), like \tanh (hyperbolic tangent).

- This function will "squash" very large or very small inputs into a limited range (e.g., between -1 and 1).
 - **Why it matters:** This allows the network to learn much more complex relationships than just straight lines. **Without σ , the entire neural network would just be one giant Transformasi Linear.**
-

4. Building a Network (Stacking the "Lego Bricks")

A. Adding More Inputs (n inputs → 1 output)

- **Scenario:** n inputs (a_{00}, a_{01}, \dots) → 1 output (a_1).
- **Logic:** Each input will have its own "path" with its own weight (w).
- **Recipe:** $a_1 = \sigma((w_0 a_{00} + w_1 a_{01} + \dots) + b)$
- **"Aha!" Moment (Linear Algebra):** The part ($w_0 a_{00} + w_1 a_{01} + \dots$) is a [DOT PRODUCT](#) between the weight vector \vec{w} and the input vector \vec{a}_0 .
- **Vector Form:**

$$a_1 = \sigma(\vec{w} \cdot \vec{a}_0 + b)$$

B. Adding More Outputs (n inputs → m outputs)

- **Scenario:** n inputs → m outputs.
- **Logic:** Each output neuron will have its **own set of weights** (w) and its **own bias** (b).
- **Result:** We will have m separate equations.
- **"Aha!" Moment (Linear Algebra):** The collection of m weight vectors w can be organized into a **WEIGHT MATRIX** W . The collection of m biases b becomes a **bias vector** b .
- **Matrix Form:**

$$\vec{a}_1 = \sigma(W\vec{a}_0 + \vec{b})$$

- \vec{a}_1 : The $m \times 1$ output vector.
- W : The $m \times n$ weight matrix.
- \vec{a}_0 : The $n \times 1$ input vector.
- \vec{b} : The $m \times 1$ bias vector.
- σ is applied element-wise to the result.

C. Adding More Layers (Hidden Layers)

- **Logic:** The output (a_1) from one layer becomes the input for the next layer.
- This creates a very deep chain of function compositions:

$$\vec{a}_2 = \sigma_2(W_2\vec{a}_1 + \vec{b}_2) = \sigma_2(W_2\sigma_1(W_1\vec{a}_0 + \vec{b}_1) + \vec{b}_2).$$

5. Final Conclusion

- A neural network, no matter how complex, is just a **series of matrix operations (linear transformations) interspersed with non-linear activation functions (σ)**.
- We now know how to perform a "**forward pass**": calculating the output from an input if we already know all the weights w and biases b .
- **The Next Challenge:** How do we "teach" this network, i.e., how do we find the correct values for w and b ? The answer lies in [Multivariate Calculus](#) and the [Chain Rule](#).

⚠ 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. Kamu benar, ini adalah lompatan besar dari matematika murni ke aplikasi yang sangat abstrak. Sangat wajar jika terasa pusing dan tidak ada yang "ngeh".

Mari kita **reset total**. Lupakan semua tentang matriks, dot product, dan lapisan-lapisan. Kita akan mulai dari **satu "bata lego" paling dasar**, dengan analogi yang sangat sederhana.

Bagian 1: "Neuron" sebagai Mesin Penilai Sederhana

Bayangkan kita ingin membuat sebuah "mesin" super sederhana yang bisa menebak apakah sebuah buah itu "**Enak**" atau "**Tidak Enak**".

Skenario Paling Sederhana: Hanya Satu Input

- Kita hanya melihat **satu faktor: Tingkat Kemanisan** dari buah itu.
- Input a_0 : Angka yang mewakili tingkat kemanisan (misal, dari 0 sampai 10).
- Output a_1 : Angka yang mewakili seberapa "enak" buah itu (misal, dari -1 sampai 1).

Bagaimana Cara Mesin Ini Bekerja?

Kita akan membangun "otak" dari mesin ini (satu "neuron") dengan dua "kenop" pengatur dan satu "tombol pemicu".

Langkah 1: Kenop Bobot (w - Weight)

$w * a_0$

- **Fungsi:** Mengatur "seberapa penting" tingkat kemanisan.
- Jika w besar (misal, $w=2$): Artinya, kemanisan adalah faktor yang **sangat penting** bagimu.
- Jika w kecil (misal, $w=0.1$): Kemanisan tidak terlalu penting.
- Jika w negatif (misal, $w=-1$): Kamu justru suka buah yang asam!
- **Bobot (w)** adalah "kenop volume" untuk setiap input.

Langkah 2: Kenop Bias (b - Bias)

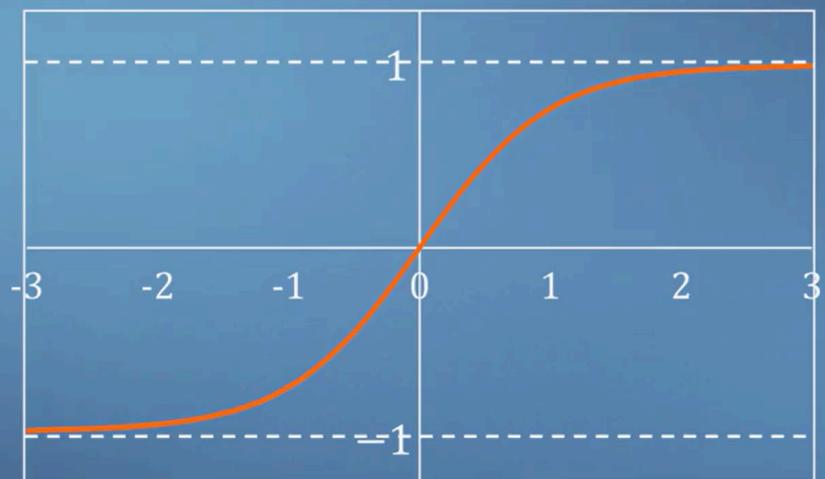
$$(w * a_0) + b$$

- **Fungsi:** Mengatur "standar dasar" atau "prasangka"-mu.
- Jika b besar (misal, $b=5$): Kamu pada dasarnya **optimis**. Bahkan sebelum merasakan manisnya, kamu sudah cenderung berpikir buah itu akan enak.
- Jika b kecil (misal, $b=-5$): Kamu **pesimis**. Kamu butuh buah yang sangat manis untuk bisa bilang itu enak.
- **Bias (b)** adalah "kenop geser" yang menggeser seluruh hasil ke atas atau ke bawah.

Langkah 3: Tombol Pemicu (Fungsi Aktivasi σ)

$$a_1 = \sigma(w * a_0 + b)$$

- **Masalah:** Hasil dari $w*a_0 + b$ bisa berupa angka berapa pun (misal, 100, -50). Tapi kita ingin output "Enak/Tidak Enak" kita berada dalam rentang yang jelas, misalnya antara -1 (sangat tidak enak) dan 1 (sangat enak).
- **Fungsi σ (Sigma):** Ini adalah "tombol pemicu" atau "penekan". Fungsinya adalah mengambil hasil perhitungan tadi dan "**memetakan**"-nya ke rentang yang kita inginkan.
- **Contoh (tanh):**
 - Jika $w*a_0 + b$ hasilnya 100 (sangat besar), $\tanh(100)$ akan mendekati 1.
 - Jika $w*a_0 + b$ hasilnya -50 (sangat kecil), $\tanh(-50)$ akan mendekati -1.
 - Jika $w*a_0 + b$ hasilnya 0, $\tanh(0)$ adalah 0.
- Fungsi σ inilah yang meniru "tombol pemicu" neuron di otak dan menambahkan **non-linearitas** yang krusial.

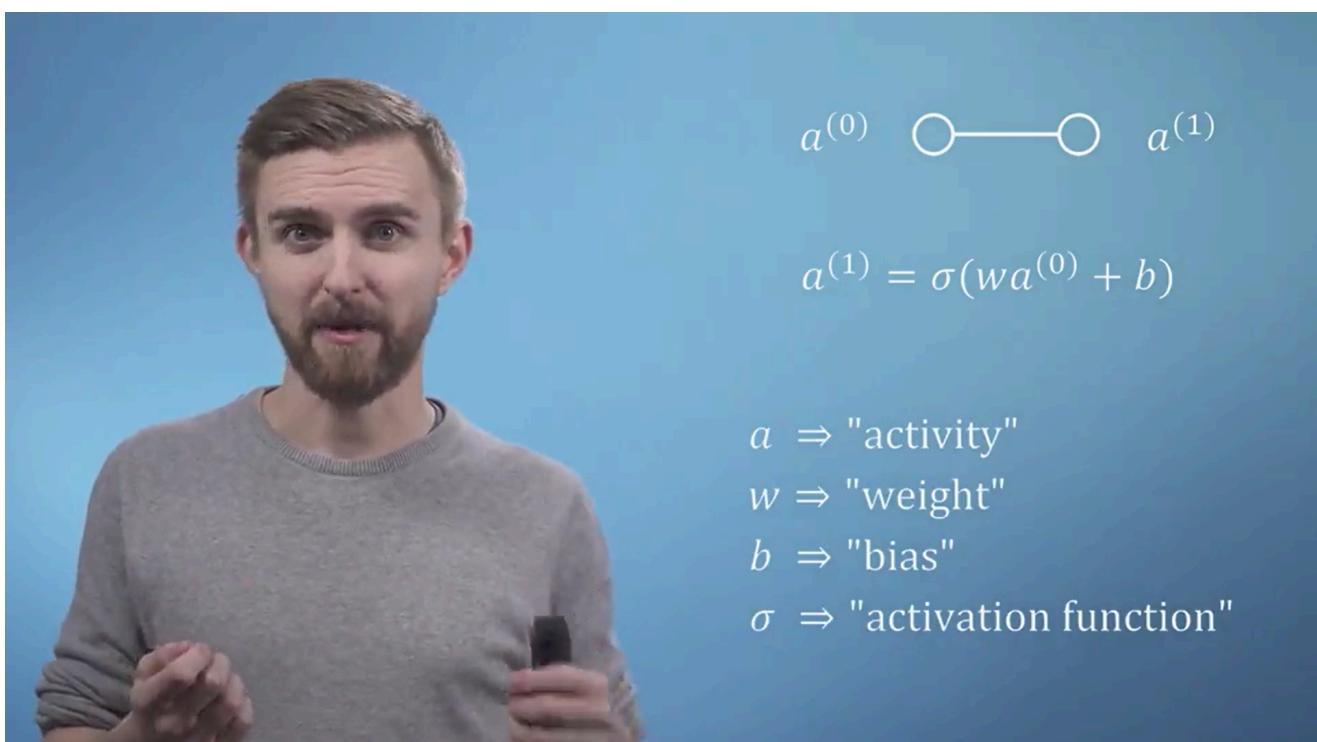


$$\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Cek Pemahaman (Berhenti di Sini Dulu):

Ini adalah "bata lego" paling dasar dari seluruh Neural Network.

1. Apakah kamu bisa melihat peran dari tiga komponen ini secara terpisah?
 - w (Bobot): Seberapa **penting** input ini?
 - b (Bias): Apa **standar dasar**-ku?
 - σ (Aktivasi): "**Tekan**" hasilnya ke rentang yang wajar.
2. Apakah kamu paham bahwa $a_1 = \sigma(w \cdot a_0 + b)$ adalah resep lengkap untuk satu neuron dengan satu input?



Bagaimana perasaanmu tentang "bata lego" dasar ini?

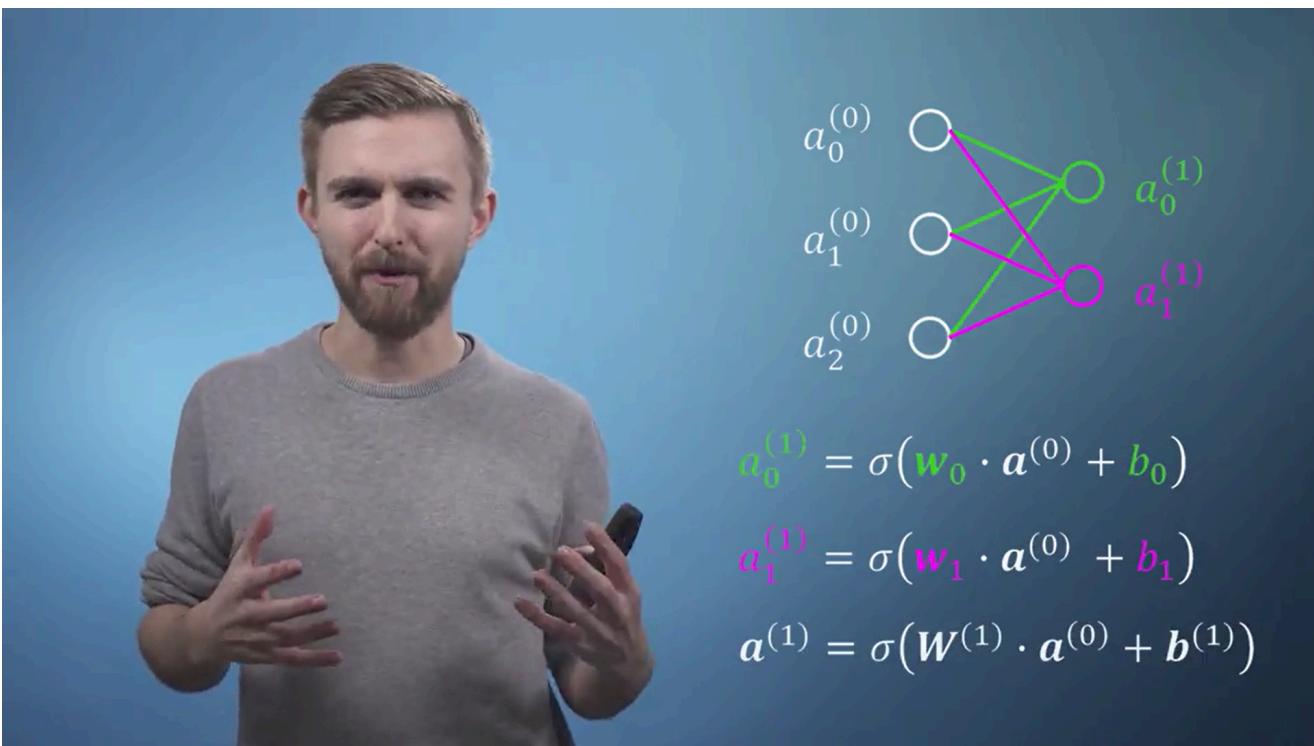
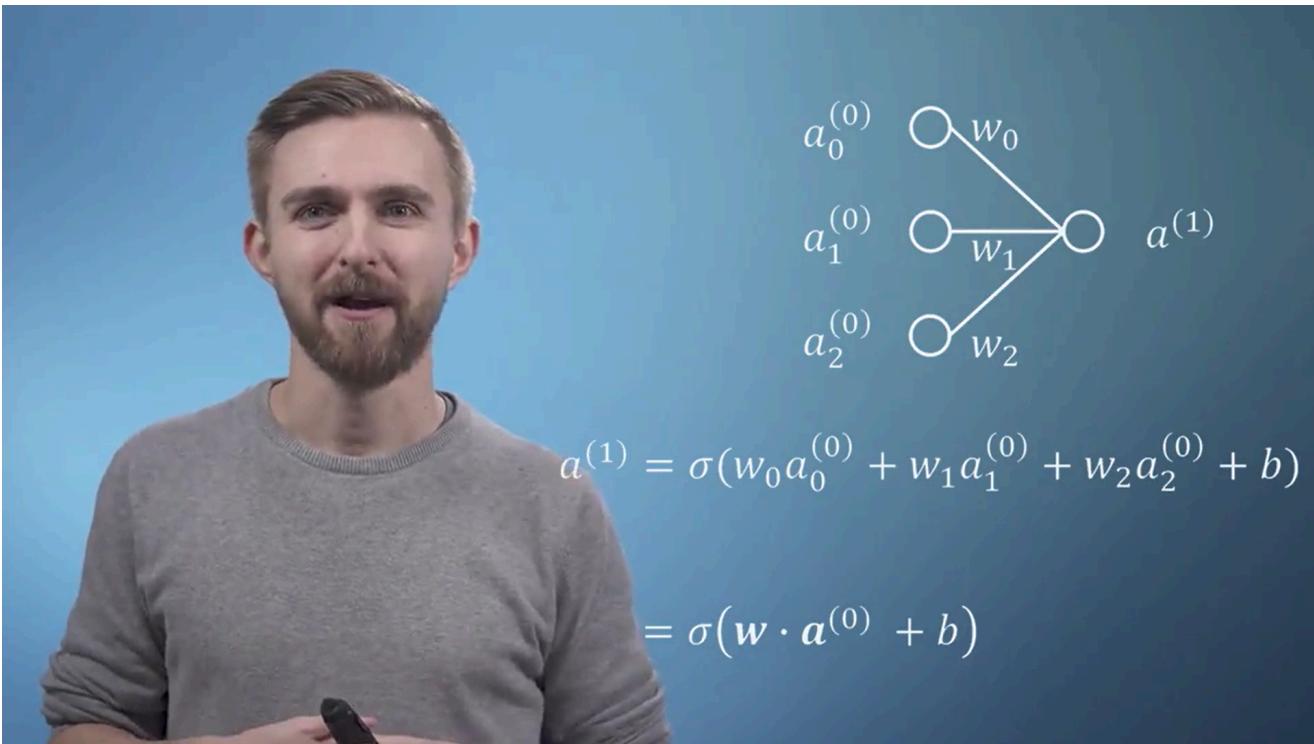
terus kalau misalnya cuman gitu doank, kan bisa saja masih gak cukup untuk melakukan pekerjaan yang lebih kompleks, nah coba kalau kita tambahann yaitu nambahin neurons nya

we simply say that a_1 equals sigma of the sum of these two inputs,

dst.....

$$a^{(1)} = \sigma(w_0 a_0^{(0)} + w_1 a_1^{(0)} + w_2 a_2^{(0)} + b)$$
$$= \sigma\left(\left(\sum_{j=0}^n w_j a_j^{(0)}\right) + b\right)$$

yang bisa dengan mudah disederhanakan menggunakan dot product



Penjelasan lebih lanjut gambar diatas, menggunakan contoh lebih konkret :

Rekap Cepat Terakhir Kali:

Kita sudah berhasil memahami "bata lego" paling dasar dari sebuah Neural Network, yaitu **satu neuron dengan satu input.**

- **Resepnya:** $a_1 = \sigma(w * a_0 + b)$
- **Anatomii:**
 - **w** (Bobot): Mengatur "**seberapa penting**" input.
 - **b** (Bias): Mengatur "**standar dasar**" atau prasangka.

- σ (Aktivasi): "Menekan" hasil ke rentang yang wajar dan menambahkan non-linearitas.

Sekarang, mari kita buat "mesin penilai buah" kita menjadi sedikit lebih pintar dengan menambahkan lebih banyak informasi.

Bagian Selanjutnya: Menambah Input (dari 1 Input ke Banyak Input)

Teks: "If we now add an additional neuron to our input... we can call them a_{00} and a_{01} ... we simply say that a_1 equals sigma of the sum of these two inputs, each multiplied by their own weighting plus the bias."

(*Jika sekarang kita menambahkan neuron tambahan ke input kita... kita bisa sebut mereka a_{00} dan a_{01} ... kita tinggal katakan bahwa a_1 sama dengan sigma dari jumlah kedua input ini, masing-masing dikalikan dengan bobotnya sendiri ditambah bias.*)

Skenario Baru:

Sekarang, untuk menilai apakah sebuah buah itu "Enak", kita tidak hanya melihat "Kemanisan", tapi juga "**Kerenyahan**".

- Input #0 (a_{00}): Tingkat Kemanisan.
- Input #1 (a_{01}): Tingkat Kerenyahan.
- Output (a_1): Skor "Enak/Tidak Enak" (masih satu angka).

Bagaimana Otak "Neuron" Kita Berpikir Sekarang?

Logikanya sangat sederhana: otak akan mempertimbangkan kedua faktor itu secara terpisah, lalu menggabungkannya.

1. Evaluasi Kemanisan:

- Seberapa penting Kemanisan? Ini diatur oleh "kenop" w_0 .
- Kontribusi dari kemanisan: $w_0 * a_{00}$.

2. Evaluasi Kerenyahan:

- Seberapa penting Kerenyahan? Ini diatur oleh "kenop" w_1 yang **berbeda**. Mungkin kamu sangat suka renyah, jadi w_1 nilainya besar.
- Kontribusi dari kerenyahan: $w_1 * a_{01}$.

3. Gabungkan dan Tambahkan "Prasangka":

- Otak akan menjumlahkan semua kontribusi: $(w_0 * a_{00}) + (w_1 * a_{01})$
- Lalu, ia akan menambahkan "prasangka" dasarnya (b).
- Total Stimulasi: $w_0 * a_{00} + w_1 * a_{01} + b$

4. Tekan dengan Tombol Pemicu:

- Hasil akhir a_1 adalah: $\sigma(w_0 * a_{00} + w_1 * a_{01} + b)$

we simply say that a_1 equals sigma of the sum of these two inputs,

"Aha!" Moment: Ini adalah Dot Product!

Teks: "...we can make a vector of weights and a vector of inputs, and then just take the dot product to achieve the same effect."

(...kita bisa membuat sebuah vektor bobot dan sebuah vektor input, lalu tinggal lakukan dot product untuk mencapai efek yang sama.)

Lihat kembali bagian $w_0 \cdot a_{00} + w_1 \cdot a_{01}$. Ini adalah **pola yang sangat kita kenal!**

- Mari kita "kemas" semua input kita ke dalam **vektor input** a_0 :

$$a_0 = [a_{00}, a_{01}]$$
- Mari kita "kemas" semua "kenop kepentingan" kita ke dalam **vektor bobot** w :

$$w = [w_0, w_1]$$
- Perhitungan $w_0 \cdot a_{00} + w_1 \cdot a_{01}$ adalah **DOT PRODUCT** antara w dan a_0 .

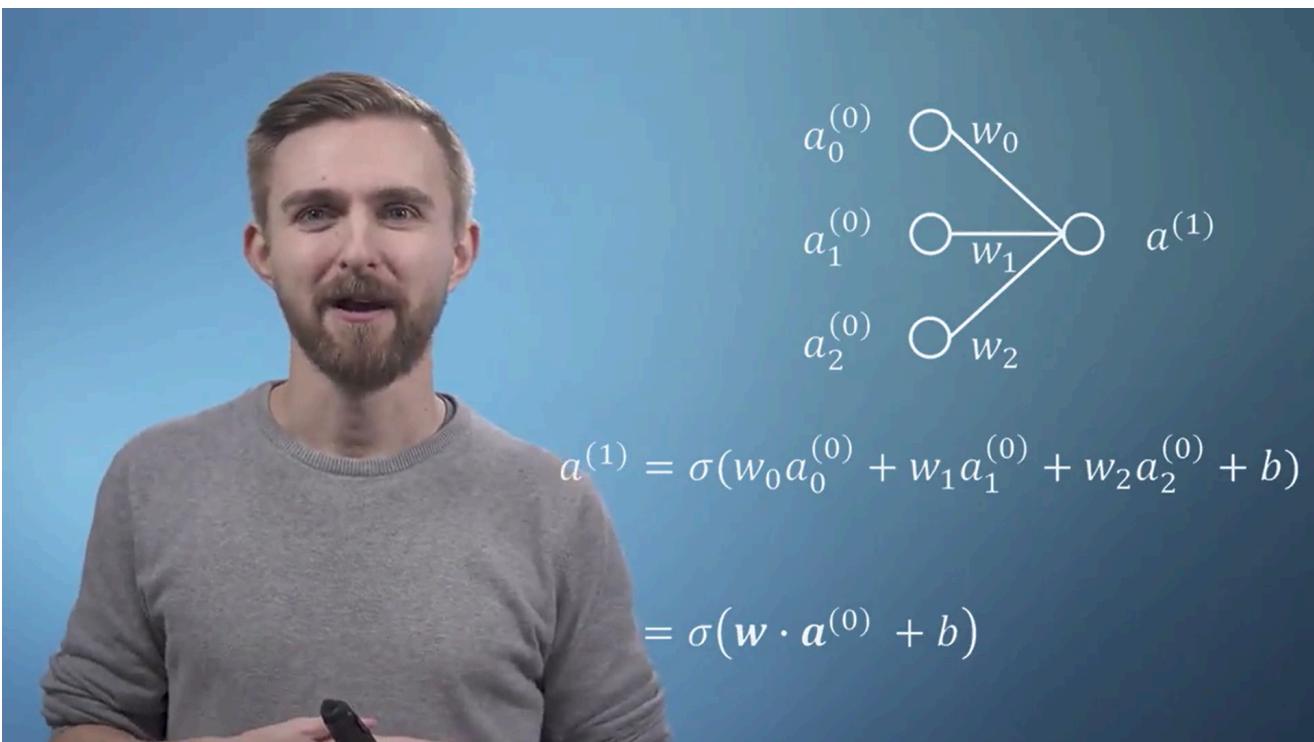
Sekarang kita bisa menulis ulang resep kita dengan "bahasa" Aljabar Linear yang jauh lebih ringkas dan elegan:

$$a_1 = \sigma(w \cdot a_0 + b)$$

Ini adalah resep untuk satu neuron dengan **banyak input**.



menjadi



Cek Pemahaman (Berhenti di Sini Dulu):

1. Apakah kamu melihat bagaimana, saat kita menambah input baru, kita hanya perlu memberinya "kenop bobot" (w) sendiri dan menambahkannya ke dalam total?
2. Apakah kamu melihat bagaimana proses penjumlahan ($w_0 * a_{00} + w_1 * a_{01} + \dots$) ini secara alami adalah sebuah **Dot Product**?
3. Apakah kamu paham bahwa rumus $a_1 = \sigma(\mathbf{w} \cdot \mathbf{a}_0 + b)$ adalah versi generalisasi dari rumus awal kita?

Bagaimana perasaanmu tentang langkah ini?

Note

Tambahan catatan mengenai betapa kuatnya dot product itu, dan betapa pentingnya dalam menyelesaikan masalah sehari hari.

Pertanyaan #1: Tentang Notasi Penjumlahan (Σ)

"... bisa menjadi pendek ituu, aku kalau gak salah ingett ada juga yang kek penulisan einstein apa gituu aku tnya lupa gitu, sheingga bisa ditulis kek gituuu dannn apa gituu."

Kamu benar sekali! Ingatanmu sangat tajam.

$w_0 a_0 + w_1 a_1 + w_2 a_2 + \dots$

Ini adalah penjumlahan dari perkalian komponen-komponen yang bersesuaian. Ada beberapa cara untuk menuliskannya secara lebih singkat:

1. Notasi Sigma (Σ) (yang ada di gambar):

Ini adalah cara paling umum di matematika untuk menulis penjumlahan yang panjang.

Σ (dari $j=0$ sampai n) $w_j a_j$

- **Artinya:** "Mulai dari $j=0$, hitung $w_j a_j$. Lalu, lakukan hal yang sama untuk $j=1$, $j=2$, dst., sampai $j=n$. Terakhir, **jumlahkan semua hasilnya.**"
- Ini adalah notasi yang sangat eksplisit dan tidak ambigu.

2. Notasi Dot Product (\cdot):

Ini adalah cara Aljabar Linear untuk menulisnya.

$w \cdot a$

- **Artinya:** "Ambil vektor w dan vektor a , lalu lakukan operasi Dot Product (kalikan komponen yang bersesuaian lalu jumlahkan)."
- Ini lebih ringkas daripada Notasi Sigma.

3. Konvensi Penjumlahan Einstein (yang pernah kita bahas):

Ini adalah cara paling "malas" dan paling canggih.

$w_j a_j$

- **Artinya:** "Karena indeks j muncul berulang, aku secara otomatis tahu bahwa aku harus menjumlahkan suku ini di sepanjang semua kemungkinan nilai j ."
- Ini menghilangkan kebutuhan untuk menulis Σ . Ini sangat umum di fisika teoretis dan komputasi tingkat lanjut.

Kesimpulan: Ketiga notasi ini (Σ , \cdot , dan Konvensi Einstein) adalah **tiga "bahasa" yang berbeda untuk mendeskripsikan PROSES yang SAMA PERSIS**: yaitu "kalikan komponen per komponen, lalu jumlahkan".

Pertanyaan #2: Kenapa Selalu "Ujung-ujungnya Dot Product"?

"Tapi aku penasaran juga yang kenapa kok dikit dikitt dot product gitu an sebagainyaaa, kek apa -apa dot product..."

Ini adalah observasi yang sangat cerdas. Kamu benar, Dot Product muncul di mana-mana. Kenapa?

Jawabannya adalah karena Dot Product secara fundamental adalah operasi "**Pengukur Kemiripan Tertimbang**" (**Weighted Similarity Measurement**).

Mari kita bongkar. Apa yang sebenarnya dihitung oleh $w \cdot a$?

$$w_0a_0 + w_1a_1 + w_2a_2 + \dots$$

Ini adalah sebuah **skor gabungan**. Mari kita kembali ke analogi **Menilai Buah**:

- a : Vektor yang merepresentasikan **fitur-fitur buah** ([kemanisan, kerenyahan, keasaman, ...]).
- w : Vektor yang merepresentasikan **preferensimu** ([pentingnya kemanisan, pentingnya kerenyahan, pentingnya keasaman, ...]).

$w \cdot a$ adalah **skor "kecocokan"** antara preferensimu dengan fitur buah.

- Jika buahnya **manis** (a_0 tinggi) dan kamu **sangat suka manis** (w_0 tinggi), maka suku w_0a_0 akan memberikan **kontribusi positif yang besar** pada skor total.
- Jika buahnya **asam** (a_2 tinggi) tapi kamu **tidak suka asam** (w_2 negatif), maka suku w_2a_2 akan memberikan **kontribusi negatif yang besar**, menurunkan skor total.
- Jika buahnya **renyah** (a_1 tinggi) tapi kamu **tidak peduli** dengan kerenyahan (w_1 mendekati nol), maka suku w_1a_1 hampir tidak memberikan kontribusi apa pun.

Dot Product adalah cara yang sangat efisien untuk menghitung "**skor total**" dari interaksi antara dua "daftar" informasi, di mana satu daftar adalah "data" (a) dan daftar lainnya adalah "filter" atau "pembobot" (w).

Kenapa ini muncul di mana-mana?

Karena banyak sekali masalah di dunia ini yang bisa direduksi menjadi masalah "pencocokan pola tertimbang":

- **Neural Network**: "Seberapa cocok input dari neuron sebelumnya dengan 'preferensi' dari neuron ini?" → **Dot Product**.
- **Aturan Rantai Multivariat**: "Seberapa besar perubahan total (df/dt)?" → "Seberapa 'cocok' arah gerakanmu (dx/dt) dengan arah 'peta kecuraman' (∇f)?" → **Dot Product**.
- **Proyeksi**: "Seberapa 'mirip' vektor v dengan arah u ?" → **Dot Product**.

Jadi, Dot Product bukan hanya "salah satu" operasi. Ia adalah salah satu **operasi paling fundamental** dalam Aljabar Linear karena ia adalah cara matematis untuk menjawab pertanyaan "Seberapa cocok/mirip/selaras kedua hal ini, dengan mempertimbangkan bobot masing-masing komponen?"

Tentu saja. Mari kita lanjutkan persis dari pemahaman terakhir kita yang sudah solid.

Rekap Cepat Terakhir Kali:

- Kita sudah berhasil memahami "bata lego" dari Neural Network: **satu neuron dengan banyak input**.
- Kita menemukan bahwa proses intinya adalah sebuah **Dot Product** antara **vektor input** (a_0) dengan **vektor bobot** (w), ditambah **bias** (b), lalu dimasukkan ke **fungsi aktivasi** (σ).
- Rumus ringkasnya: $a_1 = \sigma(w \cdot a_0 + b)$.

Sekarang, kita akan membuat jaringan kita menjadi lebih kuat dengan menambahkan lebih banyak **output**.

Bagian Selanjutnya: Menambah Output (dari 1 Output ke Banyak Output)

Teks: "Adding a second output neuron, we'd call these two values a_{10} and a_{11} , where we now have twice the number of connectors, each one with its own weighting and each neuron has its own bias."

(Menambahkan neuron output kedua, kita akan sebut kedua nilai ini a_{10} dan a_{11} , di mana kita sekarang punya dua kali lipat jumlah koneksi, masing-masing dengan bobotnya sendiri dan setiap neuron punya biasnya sendiri.)

Skenario Baru:

Bayangkan "mesin penilai buah" kita sekarang tidak hanya memberikan satu skor "Enak/Tidak Enak". Kita ingin output yang lebih detail.

- **Input:** Masih sama, vektor fitur buah $a_0 = [\text{kemanisan}, \text{kerenyahan}]$.
- **Output Baru:** Sekarang berupa vektor a_1 dengan dua komponen:
 - a_{10} : Skor "**Rasa**" (dari -1 sampai 1).
 - a_{11} : Skor "**Tekstur**" (dari -1 sampai 1).

Bagaimana Otak Jaringan Ini Bekerja?

Sekarang kita punya **dua neuron output** yang bekerja secara **paralel dan independen**. Masing-masing punya "otak"-nya sendiri.

Neuron #0 (Penilai "Rasa"):

- Neuron ini punya **set preferensinya sendiri**. Mungkin dia sangat peduli pada kemanisan, tapi tidak peduli pada kerenyahan.
- Preferensinya disimpan dalam **vektor bobotnya sendiri**, $w_0 = [w_{00}, w_{01}]$.
- Dia juga punya **prasangkanya sendiri**, b_0 .
- **Resepnya**: $a_{10} = \sigma(w_0 \cdot a_0 + b_0)$

Neuron #1 (Penilai "Tekstur"):

- Neuron ini punya preferensi yang berbeda. Mungkin dia tidak peduli kemanisan, tapi sangat peduli pada kerenyahan.
- Preferensinya disimpan dalam **vektor bobotnya sendiri**, $w_1 = [w_{10}, w_{11}]$.
- Dia juga punya **prasangkanya sendiri**, b_1 .
- **Resepnya**: $a_{11} = \sigma(w_1 \cdot a_0 + b_1)$

Kita sekarang punya **dua persamaan terpisah**, satu untuk setiap neuron output.

"Aha!" Moment: Ini adalah Perkalian MATRIKS!

Teks: "...we can again crunch these two equations down to a more compact vector form, where... we now hold our two weight vectors in a **weight matrix**..."

(...kita bisa lagi merapatkan kedua persamaan ini menjadi bentuk vektor yang lebih ringkas, di mana... kita sekarang menyimpan kedua vektor bobot kita di dalam sebuah **matriks bobot**...)

Lihat kembali dua persamaan kita:

$$a_{10} = \sigma(w_0 \cdot a_0 + b_0)$$

$$a_{11} = \sigma(w_1 \cdot a_0 + b_1)$$

Ini adalah pola yang sangat terstruktur. Mari kita gunakan "bahasa" Aljabar Linear untuk membuatnya lebih rapi.

1. **Tumpuk Vektor Bobot:** Mari kita "tumpuk" kedua vektor bobot w_0 dan w_1 (yang merupakan vektor baris jika kita menggunakan notasi $w \cdot a$) menjadi sebuah **MATRIKS** W .

- Baris 1 dari W adalah w_0 .
- Baris 2 dari W adalah w_1 .

$$W = [[w_{00}, w_{01}], [w_{10}, w_{11}]]$$

2. **Lakukan Perkalian Matriks-Vektor:** Apa yang terjadi jika kita mengalikan W dengan a_0 ?

$$W * a_0 = [[w_{00}, w_{01}], [w_{10}, w_{11}]] * [a_{00}, a_{01}]$$

- Hasil Baris 1: $w_{00}*a_{00} + w_{01}*a_{01}$ (Ini adalah $w_0 \cdot a_0$)

- Hasil Baris 2: $w_{10} \cdot a_{00} + w_{11} \cdot a_{01}$ (Ini adalah $w_1 \cdot a_0$)
- Hasilnya adalah sebuah vektor: $[w_0 \cdot a_0, w_1 \cdot a_0]$

3. Gabungkan Semuanya:

Kita bisa menulis ulang kedua persamaan kita menjadi **satu persamaan matriks tunggal** yang sangat elegan.

$$a_1 = \sigma(W * a_0 + b)$$

Di mana:

- a_1 adalah **vektor output** $[a_{10}, a_{11}]$.
- W adalah **matriks bobot**.
- a_0 adalah **vektor input**.
- b adalah **vektor bias** $[b_0, b_1]$.
- σ diterapkan pada setiap elemen hasilnya.

Cek Pemahaman (Berhenti di Sini Dulu):

1. Apakah kamu paham bahwa untuk menghasilkan banyak output, kita hanya perlu banyak "penilai" (neuron) yang bekerja secara paralel, masing-masing dengan bobot dan biasnya sendiri?
2. Apakah kamu melihat bagaimana **menumpuk** vektor-vektor bobot itu secara alami membentuk sebuah **matriks**?
3. Apakah kamu melihat bahwa rumus $a_1 = \sigma(W * a_0 + b)$ adalah cara yang sangat ringkas untuk mendeskripsikan seluruh lapisan neuron?

Bagaimana perasaanmu tentang transisi dari "banyak dot product" menjadi "satu perkalian matriks"?

Luar biasa! Kamu sudah berhasil membangun "blok" fundamental dari sebuah lapisan jaringan saraf. Kamu sekarang tahu bagaimana sebuah lapisan dengan banyak input dan banyak output bisa direpresentasikan dengan satu persamaan matriks yang elegan:

$$a_{\text{output}} = \sigma(W * a_{\text{input}} + b)$$

Bagian Terakhir: Menambah Lapisan (Hidden Layers)

Teks: "The last piece of the puzzle is that... neural networks often have one or several layers of neurons between the inputs and the outputs. We refer to these as **hidden layers**, and they behave in **exactly the same way** as we've seen so far, except that **outputs are now the inputs of the next layer**."

(Bagian terakhir dari puzzle ini adalah... jaringan saraf sering kali punya satu atau beberapa lapisan neuron di antara input dan output. Kita menyebutnya sebagai **lapisan**

*(tersembunyi, dan mereka berperilaku **persis sama** seperti yang telah kita lihat, kecuali bahwa outputnya sekarang menjadi input dari lapisan berikutnya.)*

Apa artinya ini?

Bayangkan proses berpikir manusia. Kita tidak langsung melihat "kemanisan" dan "kerenyahan" lalu langsung menyimpulkan "enak". Otak kita melakukan pemrosesan di tengah-tengah.

Analogi Jaringan Saraf dengan "Hidden Layer":

1. Lapisan Input (a_0):

- Ini adalah data mentah kita.
- Neuron-neuron di sini hanya "memegang" nilai input.
- $a_0 = [\text{kemanisan}, \text{kerenyahan}, \text{keasaman}]$

2. Lapisan Tersembunyi #1 (a_1 - Lapisan "Konsep Abstrak"):

- Lapisan ini mengambil input mentah a_0 dan mencoba **mempelajari konsep-konsep yang lebih abstrak**.
- **Proses:** $a_1 = \sigma_1(W_1 * a_0 + b_1)$
- **Hasilnya (a_1):** Mungkin neuron pertama di a_1 belajar untuk aktif jika buahnya "manis dan renyah" (seperti apel). Neuron kedua mungkin aktif jika buahnya "manis dan lembut" (seperti pisang). Neuron ketiga mungkin aktif jika buahnya "asam" (seperti lemon).
- a_1 sekarang bukan lagi tentang fitur mentah, tapi tentang **konsep-konsep seperti "mirip-apel", "mirip-pisang", "mirip-lemon"**.

3. Lapisan Output (a_2 - Lapisan "Keputusan"):

- Lapisan ini **tidak lagi melihat data mentah**. Ia mengambil **input** dari **konsep-konsep abstrak** yang dihasilkan oleh lapisan tersembunyi (a_1).
- **Proses:** $a_2 = \sigma_2(W_2 * a_1 + b_2)$
- **Hasilnya (a_2):** Dengan melihat input [**mirip-apel**, **mirip-pisang**, **mirip-lemon**], lapisan ini bisa membuat keputusan akhir. Misalnya, jika neuron "mirip-apel" dan "mirip-pisang" aktif, ia akan menyimpulkan "enak". Jika neuron "mirip-lemon" yang aktif, ia menyimpulkan "tidak enak".

"Aha!" Moment:

Sebuah jaringan saraf yang dalam (deep neural network) hanyalah sebuah **RANTAI** dari persamaan-persamaan matriks yang sudah kita kenal, di mana output dari satu lapisan menjadi input bagi lapisan berikutnya.

$$f(x) = \dots \sigma_3(W_3 * \sigma_2(W_2 * \sigma_1(W_1 * x + b_1) + b_2) + b_3) \dots$$

Ini adalah **fungsi komposisi raksasa**. Dan sekarang kamu tahu kenapa **Aturan Rantai (Chain Rule)** akan menjadi sangat krusial untuk melatihnya.

Kesimpulan Akhir Video:

- Kita sekarang tahu cara kerja "**Forward Pass**": bagaimana cara menghitung output dari sebuah input jika kita sudah tahu semua bobot w dan bias b .
- **Pertanyaan Selanjutnya:** Bagaimana cara **menemukan nilai w dan b yang benar?** Ini adalah proses "**Training**" atau "**Belajar**".
- Jawabannya akan kita temukan di video-video berikutnya, dengan menggunakan **Kalkulus Multivariat dan Aturan Rantai** untuk "mengajari" jaringan ini.

Selesai!

Kamu telah berhasil membedah seluruh anatomi dari sebuah jaringan saraf sederhana.

Bagaimana perasaanmu sekarang setelah melihat gambaran lengkapnya, dari satu neuron hingga jaringan berlapis-lapis?

Uwahhhhhh langsung paham yaa kalau ngeliatt kek ginii, jadi lebih terbayang.

Note

Contoh - contoh latihan :

← Back Simple Artificial Neural Networks Practice Assignment • 20 min

1. Recall from the video the structure of one of the simplest neural networks,

1 point



Here there are only two neurons (or nodes), and they are linked by a single edge.

The activation of neurons in the final layer, (1), is determined by the activation of neurons in the previous layer, (0),

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

where $w^{(1)}$ is the weight of the connection between Neuron (0) and Neuron (1), and $b^{(1)}$ is the bias of the Neuron (1). These are then subject to the activation function, σ , to give the activation of Neuron (1)

Our small neural network won't be able to do a lot - it's far too simple. It is however worth plugging a few numbers into it to get a feel for the parts.

Let's assume we want to train the network to give a *NOT* function, that is if you input 1 it returns 0, and if you input 0 it returns 1.

For simplicity, let's use, $\sigma(z) = \tanh(z)$, for our activation function, and randomly initialise our weight and bias to $w^{(1)} = 1.3$ and $b^{(1)} = -0.1$.

Use the code block below to see what output values the neural network initially returns for training data.

```
1 # First we set the state of the network
2 σ = np.tanh
3 w1 = 1.3
4 b1 = -0.1
5
6 # Then we define the neuron activation.
7 def a1(a0):
8     return σ(w1 * a0 + b1)
9
10 # Finally let's try the network out!
11 # Replace x with 0 or 1 below,
12 a1(x)
13
```

Run
Reset

Use the code block below to see what output values the neural network initially returns for training data.

```

1 # First we set the state of the network
2 σ = np.tanh
3 w1 = 1.3
4 b1 = -0.1
5
6 # Then we define the neuron activation.
7 def a1(a0) :
8     return σ(w1 * a0 + b1)
9
10 # Finally let's try the network out!
11 # Replace x with 0 or 1 below,
12 a1(0)
13

```

Run
Reset

-0.099667994625

It's not very good! But it's not trained yet; experiment by changing the weight and bias and see what happens.

```

1 # First we set the state of the network
2 σ = np.tanh
3 w1 = -5
4 b1 = 5
5
6 # Then we define the neuron activation.
7 def a1(a0) :
8     return σ(w1 * a0 + b1)
9
10 # Finally let's try the network out!
11 # Replace x with 0 or 1 below,
12 a1(1)
13

```

Run
Reset

0.0

It's not very good! But it's not trained yet; experiment by changing the weight and bias and see what happens.

Choose the weight and bias that gives the best result for a *NOT function* out of all the options presented.

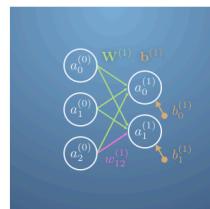
- $w^{(1)} = 3, b^{(1)} = 1$
- $w^{(1)} = -5, b^{(1)} = 5$
- $w^{(1)} = 0, b^{(1)} = 5$
- $w^{(1)} = 10, b^{(1)} = 0$
- $w^{(1)} = -3, b^{(1)} = 0$

[← Back](#) Simple Artificial Neural Networks
Practice Assignment • 20 min



2. Let's extend our simple network to include more neurons.

1 point



We now have a slightly changed notation. The neurons which are labelled by their layer with a superscript in brackets, are now also labelled with their number in that layer as a subscript, and form vectors $\mathbf{a}^{(0)}$ and $\mathbf{a}^{(1)}$.

The weights now form a matrix $\mathbf{W}^{(1)}$, where each element, $w_{ij}^{(1)}$, is the link between the neuron j in the previous layer and neuron i in the current layer. For example $w_{12}^{(1)}$ is highlighted linking $a_2^{(0)}$ to $a_1^{(1)}$.

The biases similarly form a vector $\mathbf{b}^{(1)}$.

We can update our activation function to give,

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

where all the quantities of interest have been *upgraded* to their vector and matrix form and σ acts upon each element of the resulting weighted sum vector separately.

For a network with weights, $\mathbf{W}^{(1)} = \begin{bmatrix} -2 & 4 & -1 \\ 6 & 0 & -3 \end{bmatrix}$, and bias $\mathbf{b} = \begin{bmatrix} 0.1 \\ -2.5 \end{bmatrix}$,

calculate the output, $\mathbf{a}^{(1)}$, given an input vector,

$$\mathbf{a}^{(0)} = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.1 \end{bmatrix}$$

For a network with weights, $\mathbf{W}^{(1)} = \begin{bmatrix} -2 & 4 & -1 \\ 6 & 0 & -3 \end{bmatrix}$, and bias $\mathbf{b} = \begin{bmatrix} 0.1 \\ -2.5 \end{bmatrix}$,

calculate the output, $\mathbf{a}^{(1)}$, given an input vector,

$$\mathbf{a}^{(0)} = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.1 \end{bmatrix}$$

You may do this calculation either by hand (to 2 decimal places), or by writing python code. Input your answer into the code block below.

(If you chose to code, remember that you can use the @ operator in Python to perform operate a matrix on a vector.)

```

1  # First set up the network.
2  sigma = np.tanh
3  W = np.array([[ -2, 4, -1],[6, 0, -3]])
4  b = np.array([0.1, -2.5])
5
6  # Define our input vector
7  x = np.array([0.3, 0.4, 0.1])
8
9  # Calculate the values by hand,
10 # and replace a1_0 and a1_1 here (to 2 decimal places)
11 # (Or if you feel adventurous, find the values with code!)
12 a1 = np.array([1, -1])
13

```

Run Reset

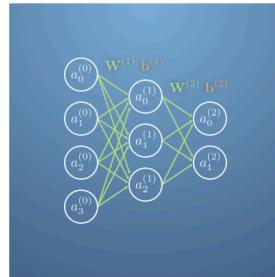
[1 -1]

[← Back](#) Simple Artificial Neural Networks
Practice Assignment • 20 min



3. Now let's look at a network with a *hidden layer*.

1 point



Here, data is input at layer (0), this activates neurons in layer (1), which become the inputs for neurons in layer (2).

(We've stopped explicitly drawing the biases here.)

Which of the following statements are true?

- This network can always be replaced with another one with the same amount of input and output neurons, but no hidden layers.
- This neural network has 9 biases.
- None of the other statements.
- The number of weights in a layer is the product of the input and output neurons to that layer.
- This neural network has 5 biases.
- The number of weights in a layer is the sum of the input and output neurons to that layer plus 1.

4. Which of the following statements about the neural network from the previous question are true?

1 point

- $\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(2)}\mathbf{W}^{(1)}\mathbf{a}^{(0)} + \mathbf{W}^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)})$
- $\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)})$,
- $\mathbf{a}^{(2)} = \sigma(\mathbf{W}^{(2)}\sigma(\mathbf{W}^{(1)}\mathbf{a}^{(0)} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$,
- None of the other statements.

5. So far, we have concentrated mainly on the *structure* of neural networks, let's look a bit closer at the *function*, and what the parts actually do.

1 point

We'll

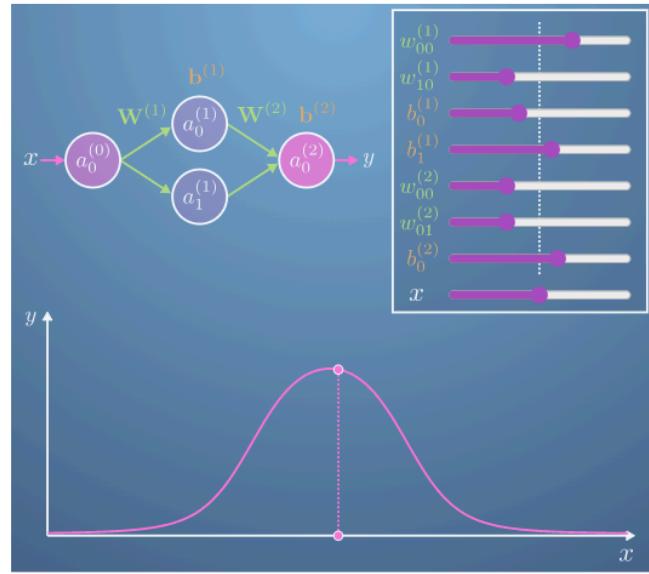
introduce another network, this time with a one dimensional input, a one dimensional output, and a hidden layer with two neurons.

Use

the tool below to change the values of the four weights and three biases, and observe what effect this has on the network's function.

With the weights and biases set here, observe how $a_0^{(1)}$ activates when $a_0^{(0)}$ is active, and $a_1^{(1)}$ activates when $a_0^{(0)}$ is inactive. Then the output neuron, $a_0^{(2)}$, activates when neither $a_0^{(1)}$ nor $a_1^{(1)}$ are too active.

(Interact with the plugin below to score the point for this question.)



Tags: #mml-specialization #multivariate-calculus #neural-networks #activation-function
#forward-pass