Your Attention Is All I Need

a high entropy presentation
about brains, computation, and more

Itinerary

- State of the relevant fields from most to least depressing
- Historical mathematical models in comp neuro
- Increasingly less bio inspired models
- TRANSFORMERS!
- Some side rants
- QUESTIONS!
- Idk how long this is gonna take so I have extra content on neuro

Some Background Info

- Brain emulation/computational neuroscience
  - We've been trying to emulate the same worm since the 1960s. And failing!
- Normal Neuroscience
  - Basically just biology and crappy fMRI studies
- Cellular automata
  - Held up by Wolfram but honestly quite promising
- Exotic computing (reversible, optic, quantum)
  - Good research, lotsa grifty companies, no real world usecase in sight
- Consciousness
  - The buddha figured it all out and now we're trying to math these insights
- AI
  - Lotsa money lotsa talent lotsa progress (!!!)

Computational Neuro!

- Trying to fit models to data we've recorded – spatial electrical recordings, proteins, behavior
- C elegans
- Small-n neuron sims
- Basically every AI model

Hebbian Learning → Hopfield network

- Neurons that fire together, wire together
- Also used in optimization problems

Updating one unit (node in the graph simulating the artificial neuron) in the Hopfield network is performed using the following rule:

$$s_i \leftarrow \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$

where:

- w_{ij} is the strength of the connection weight from unit j to unit i (the weight of the connection).
- s_i is the state of unit i.
- θ_i is the threshold of unit i.

Some code

```
# Initialize the weight matrix randomly
weights = np.random.choice([-1, 1], size=(num_neurons, num_neurons))
np.fill_diagonal(weights, 0)  # Set diagonal elements to 0

# Initialize the neuron states randomly
neuron_states = np.random.choice([-1, 1], size=num_neurons)

# Simulate the Hopfield network
for t in range(num_timesteps):
    # Select a random neuron
    neuron_idx = np.random.randint(num_neurons)

    # Calculate the input to the selected neuron
    input_sum = np.dot(weights[neuron_idx], neuron_states)

    # Update the state of the selected neuron
    if input_sum >= 0:
        neuron_states[neuron_idx] = 1
    else:
        neuron_states[neuron_idx] = -1
```

Hodgkin–Huxley model

- 4 dimensional differential equation, fits EM theory to observed neuron data

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_\mathrm{K} n^4 (V_m - V_K) + \bar{g}_\mathrm{Na} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l),$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

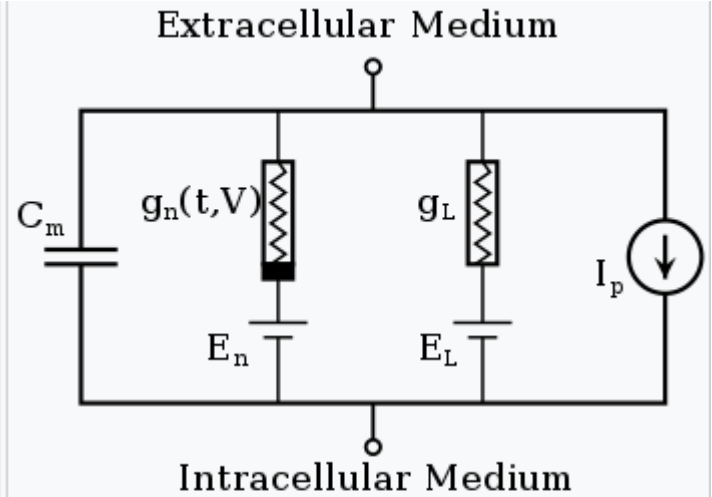
$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

where I is the current per unit area and α_i and β_i are rate constants for the i-th ion channel, which depend on voltage but not time. $\bar{g}_n$ is the maximal value of the conductance. n, m, and h are dimensionless probabilities between 0 and 1 that are associated with potassium channel subunit activation, sodium channel subunit activation, and sodium channel subunit inactivation, respectively. For instance, given that potassium channels in squid giant axon are made up of four subunits which all need to be in the open state for the channel to allow the passage of potassium ions, the n needs to be raised to the fourth power. For $p = (n, m, h)$, α_p and β_p take the form

$$\alpha_p(V_m) = p_\infty(V_m)/\tau_p$$

$$\beta_p(V_m) = (1 - p_\infty(V_m))/\tau_p.$$

p_∞ and $(1 - p_\infty)$ are the steady state values for activation and inactivation, respectively, and are usually represented by Boltzmann equations as functions of V_m. In the original paper by Hodgkin and Huxley,[1] the functions α and β are given by



Basic components of Hodgkin–Huxley-type models which represent the biophysical characteristic of cell membranes. The lipid bilayer is represented as a capacitance (C_m). Voltage-gated and leak ion channels are represented by nonlinear (g_n) and linear (g_L) conductances, respectively. The electrochemical gradients driving the flow of ions are represented by batteries (E), and ion pumps and exchangers are represented by current sources (I_p).

Leaky Integrate and Fire

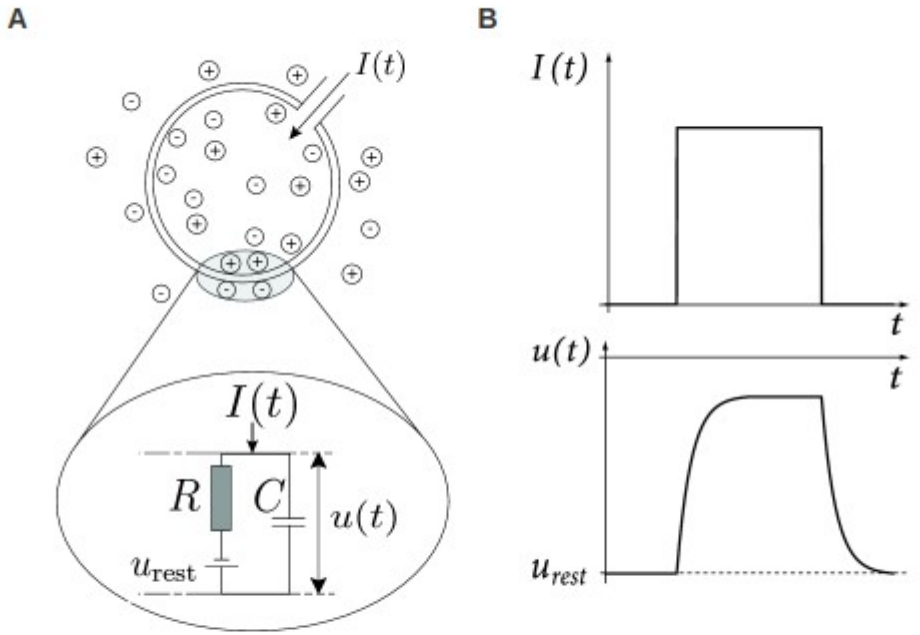


Fig. 1.6: Electrical properties of neurons: the passive membrane. **A**. A neuron, which is enclosed by the cell membrane (big circle), receives a (positive) input current $I(t)$ which increases the electrical charge inside the cell. The cell membrane acts like a capacitor in parallel with a resistor which is in line with a battery of potential u_{rest} (zoomed inset). **B**. The cell membrane reacts to a step current (top) with a smooth voltage trace (bottom).

be discussed in Part II of the book can be seen as variations of this basic model.

1.3.1 Integration of Inputs

The variable u_i describes the momentary value of the membrane potential of neuron i. In the absence of any input, the potential is at its resting value u_{rest}. If an experimentalist injects a current $I(t)$ into the neuron, or if the neuron receives synaptic input from other neurons, the potential u_i will be deflected from its resting value.

In order to arrive at an equation that links the momentary voltage $u_i(t) - u_{\text{rest}}$ to the input current $I(t)$, we use elementary laws from the theory of electricity. A neuron is surrounded by a cell membrane, which is a rather good insulator. If a short current pulse $I(t)$ is injected into the neuron, the additional electrical charge $q = \int I(t')\,dt'$ has to go somewhere: it will charge the cell membrane (Fig. 1.6A). The cell membrane therefore acts like a capacitor of capacity C. Because the insulator is not perfect, the charge will, over time, slowly leak through the cell membrane. The cell membrane can therefore be characterized by a finite leak resistance R.

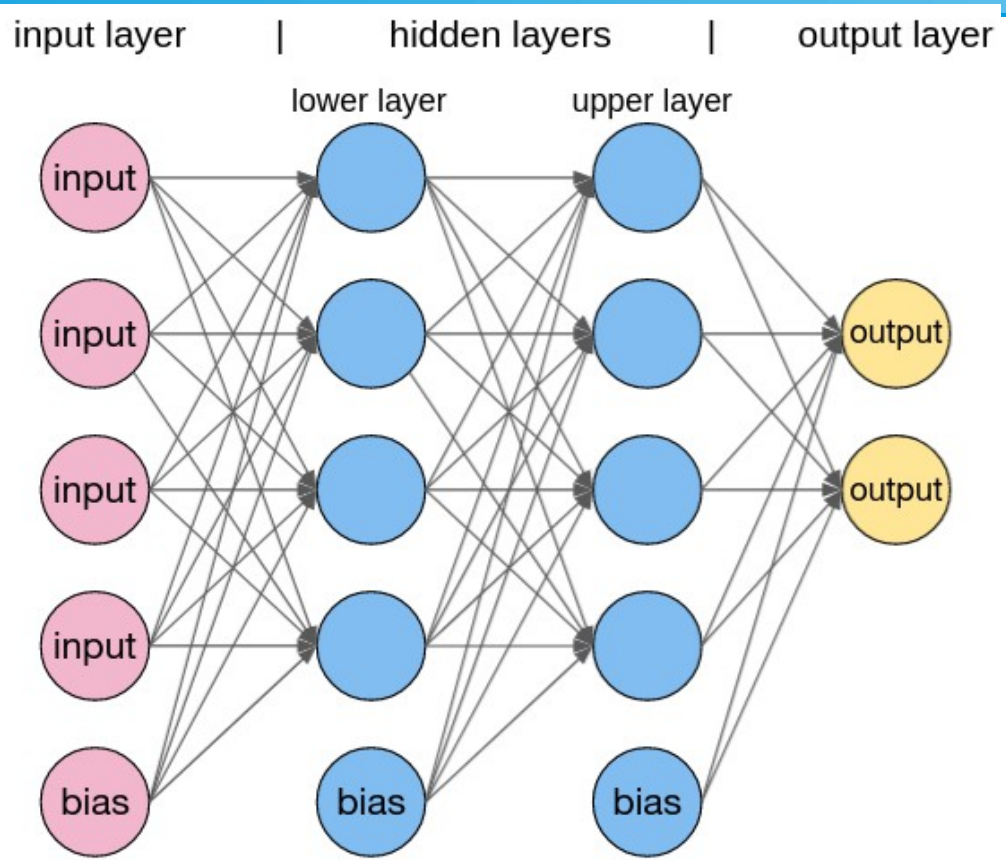
The basic electrical circuit representing a leaky integrate-and-fire model consists of a capacitor C in parallel with a resistor R driven by a current $I(t)$; see Fig. 1.6.

A B

Some fun ML facts

- Feedforward NNs with traditional activation functions are Universal Function Approximators!
- Geometric deep learning is a thing
- We've known how to do NNs for ages, we just didn't have compute (Moore's law!)
- Groq!
- OpenAI vs Anthropic vs [Gemini, conjecture, etc]

MLP / feedforward NNs



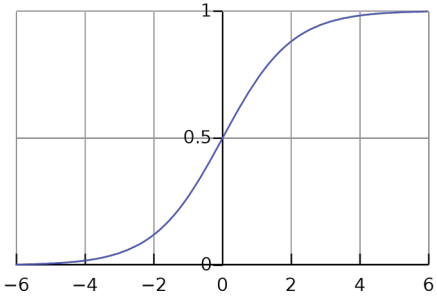
Forward prop

- Data gets fed to hidden layer
- Hidden layer multiplies it by weights (of shape INPUT_SIZE, LAYER_SIZE) and adds a bias sometimes (like y=mx+b)
- Example: input= [1, .1]
- Weights in hidden layer=[[.1, .2], [.5, .8]]
- final=[.1, .2]=.3, [.05,.08]=.11
- Each neuron's weights are summed, and then an "activation function" is applied to get out a new number
  - LET's TALK ABOUT ACTIVATION

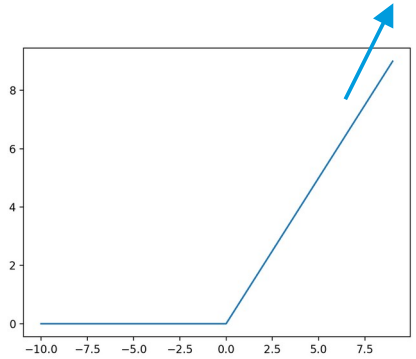
Activation functions

The weight initialization random dist you use depends on which activation function you choose!

- The sigmoid
  - Differentiable!
  - Non-linear!
  - Maps values → (0,1)
  - Vanishing gradients :(
- ReLU
  - Non-linear!
  - Sparsity (cuz it turns off neurons)
  - Kinda like a neuron
  - No vanishing gradients
  - Sometimes unstable gradients cuz unbounded



$$S(x) = \frac{1}{1 + e^{-x}}$$



max(0, value)

Backprop

- The last layer is our prediction!
- We calculate the error between the expected output and our prediction
- Mean squared error for each layer
- Error is multiplied by sigmoid_derivative(predicted_output)
- This derivative is multiplied with the weights in each layer, which updates them appropriately up or down. This is scaled by a “learning rate”

SGD, Adam

- GD moves all of the input data through the hidden layers, then calculates gradients for all of them and backprops error through (once per epoch)
- SGD moves batches of input data through the hidden layers, and calculates gradients after each batch (multiple calculations per epoch)
- Stochastic meaning random
- RMSProp updates learning rate (that thing you multiply gradients by) summing (with decay) root mean squared past gradients, which allows for faster convergence
- AdamW is the better version of this, used in modern models, maintains exponential moving average of the gradient and the squared gradient and uses that

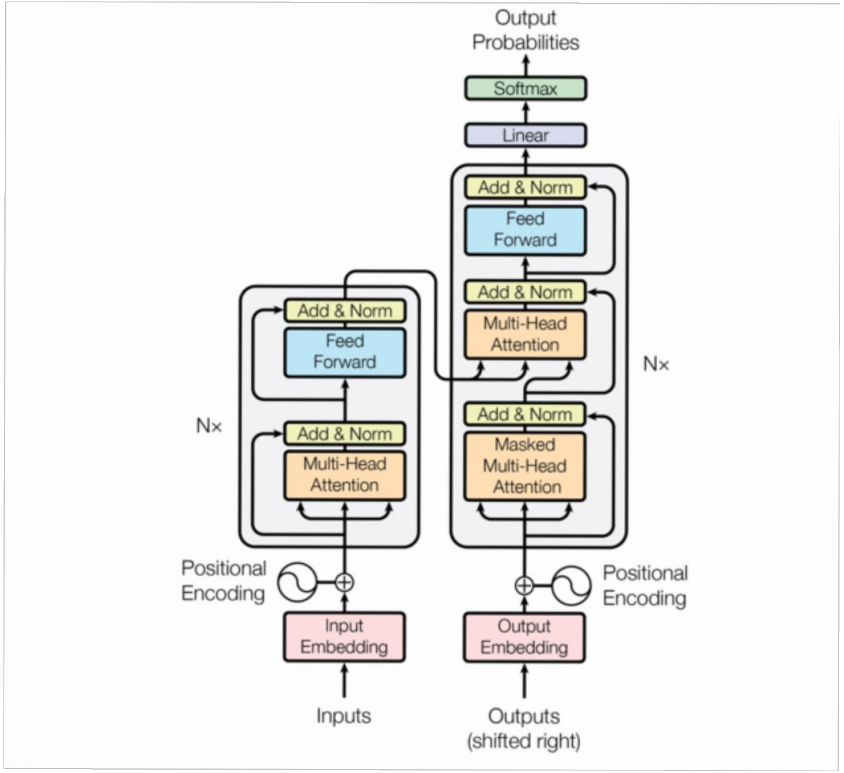
Softmax

- Turns a real vector into a probability dist from 0-1, exactly how you'd expect
- Very important normalization step
- Also dropout layers are a thing I don't wanna make a new slide

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K.$$

Traaaansssformeeeerssss

- Attention is all you need
- GPTs!
- Current best model for comp neuro
- Next token prediction



Tokens/Byte-pair encoding

- Turns a corpus of sequential words into a most-frequent-symbols vocabulary, which is good because it separates things like common prefixes and stems (-ing, pre-, etc) while compressing the corpus into its unique chunks
- Literally just a compresson alg from 1994
- Recursively replace recurring strings with “tokens” by frequency
- Differs from huffman coding because it deals in character pairs instead of one-passing byte strings
- Reversible and lossless by nature

Going back to our previous example, let's assume the words had the following frequencies:

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

meaning "hug" was present 10 times in the corpus, "pug" 5 times, "pun" 12 times, "bun" 4 times, and "hugs" 5 times. We start the training by splitting each word into characters (the ones that form our initial vocabulary) so we can see each word as a list of tokens:

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

Then we look at pairs. The pair `("h", "u")` is present in the words "hug" and "hugs", so 15 times total in the corpus. It's not the most frequent pair, though: that honor belongs to `("u", "g")`, which is present in "hug", "pug", and "hugs", for a grand total of 20 times in the vocabulary.

Thus, the first merge rule learned by the tokenizer is `("u", "g") -> "ug"`, which means that "ug" will be added to the vocabulary, and the pair should be merged in all the words of the corpus. At the end of this stage, the vocabulary and corpus look like this:

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

Embedding!

- GPTs decoder-only
- One-hot encoding, just indexing
- Bag of words – frequency of each word in vocab
- Sliding window/skipgram (word2vec) – basically MLP architecture where inputs are a word, outputs are surrounding word probabilities (both positive, as in the ones that *were* around, and negative, a sample of words that were not), and loss is SGD
- Queen-woman+man = king

Positional encoding

Positional Encoding Layer in Transformers

Let's dive straight into this. Suppose you have an input sequence of length L and require the position of the k^{th} object within this sequence. The positional encoding is given by sine and cosine functions of varying frequencies:

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Here:

k: Position of an object in the input sequence, $0 \leq k < L/2$

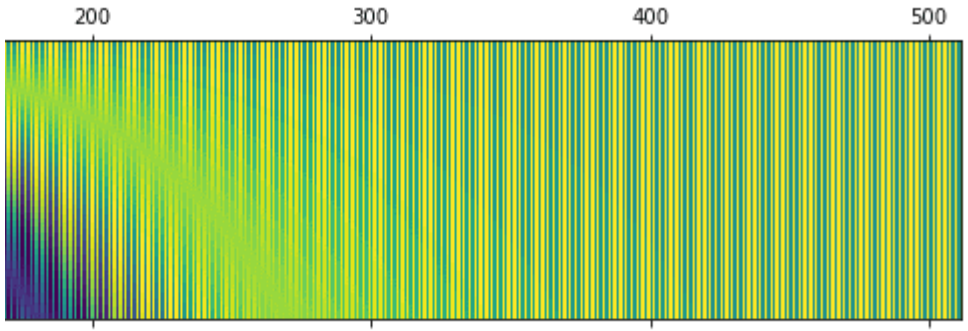
d: Dimension of the output embedding space

$P(k, j)$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

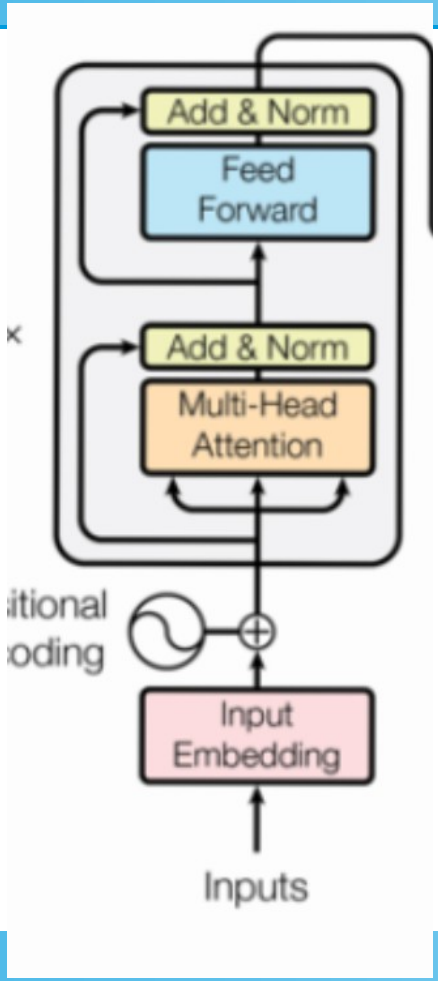
i: Used for mapping to column indices $0 \leq i < d/2$, with a single value of i maps to both sine and cosine functions

In the above expression, you can see that even positions correspond to a sine function and odd positions correspond to cosine functions.



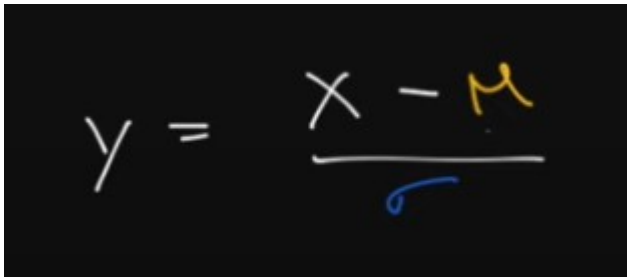
ENCODER LAYER

- encodes



LayerNorm

- Takes “residual”, which basically means those embeddings from earlier, not the attention vectors (this is done so original input keeps getting reintroduced and it doesn’t smooth out into nothingness (vanishing gradients))
- Take the means and standard deviations of each layer
- Calculate a scaling factor Y as on right
- Train learnable parameters gamma*y +b
- Learnable parameters get SGDed

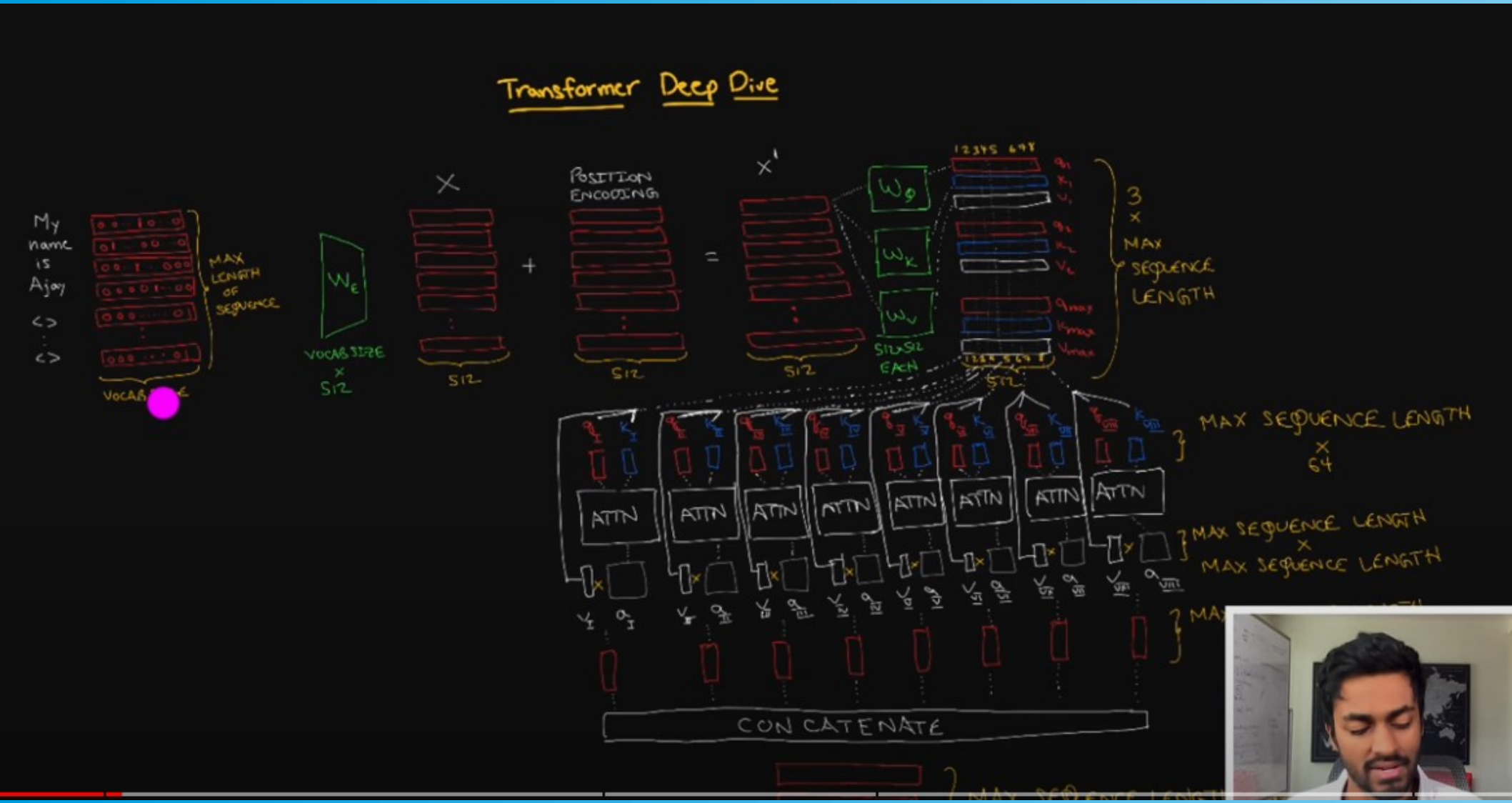


Self attention

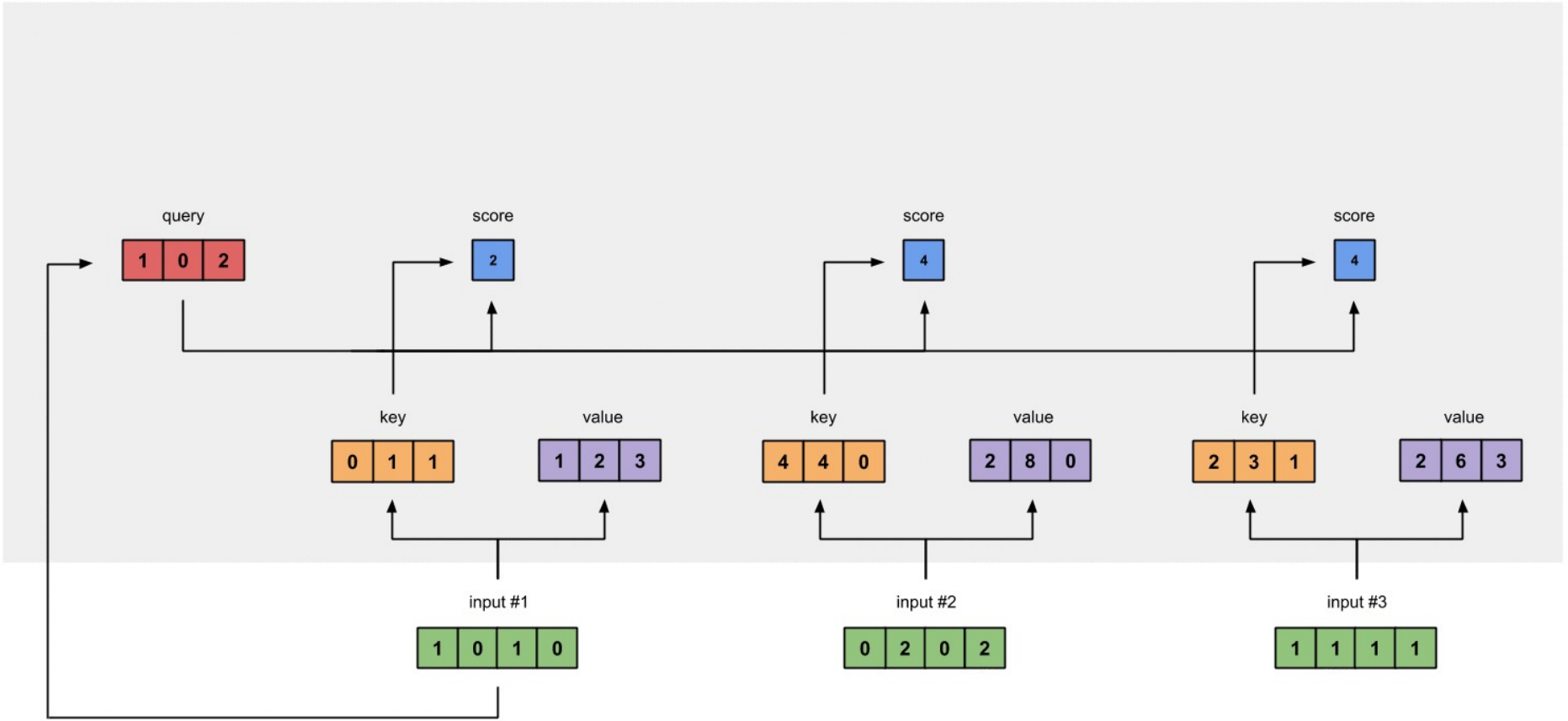
- We have Key, Query, and Value matrices, which are randomly initialized and then multiplied with the input vector
- Attention scores = Input 1’s query multiplied with all other inputs’ keys
- Scaling – divided by square root of key dimension
- Attention scores softmaxxed
- Weighted values = scores * values of each input (THIS IS WHERE GRADIENT DESCENT IS DONE)
- Sum(weighted values) = input one’s attention scores

MULTIPLE HEADS

- Do this like 7 more times
- Concatenate all of the attention vectors
- Ultimately this attention matrix is thrown into a residual in the next feedforward layer

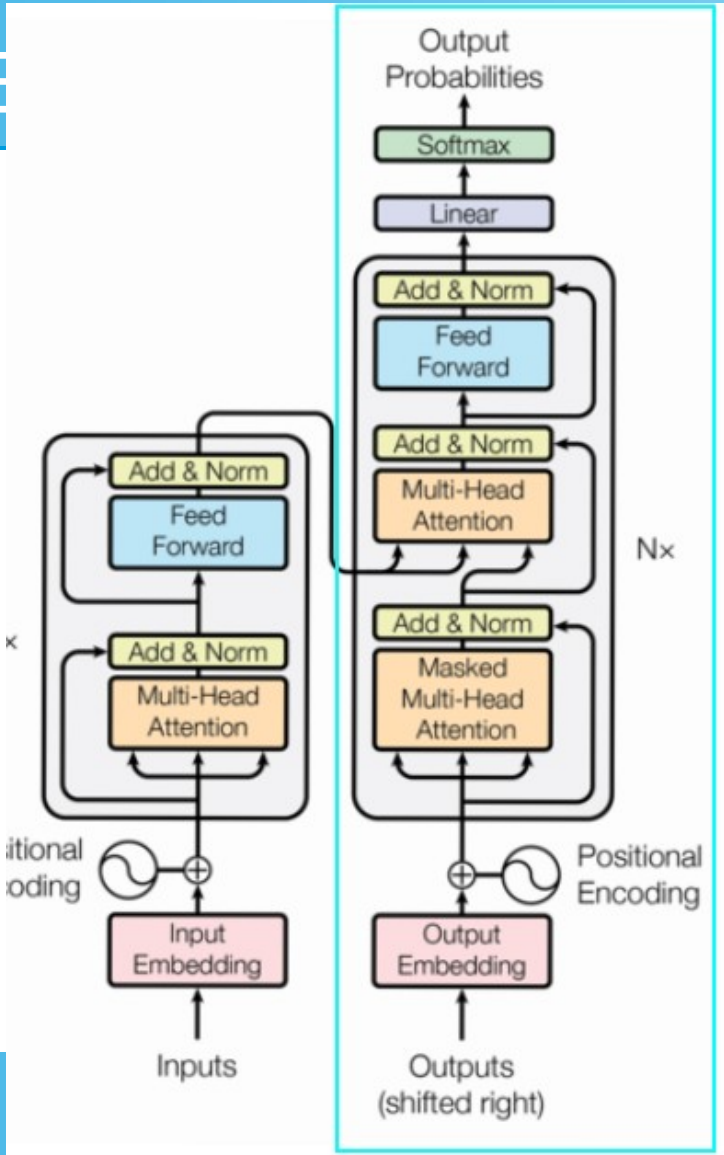

Transformer Deep Dive
My name is Ajay
<>
<>
MAX LENGTH OF SEQUENCE
VOCAB SIZE
VOCAB SIZE x 512
512
POSITION ENCODING
512
512
W_Q
W_K
W_V
512 x 512 EACH
512
3 x MAX SEQUENCE LENGTH
ATTN
ATTN
ATTN
ATTN
ATTN
ATTN
ATTN
ATTN
MAX SEQUENCE LENGTH x 64
MAX SEQUENCE LENGTH x MAX SEQUENCE LENGTH
CONCATENATE

Self-attention



DECODING TIME

- Final linear layer that acts as a classifier (literally just a weight matrix that gets multiplied, with some biases, no nonlinear activation necessary)
- Softmax to get logits – these are now interpretable as token probabilities



Attention masking

- PAD tokens
- Autoregressive tasks

$$M = \begin{array}{c} \\ a^Q \\ b^Q \\ c^Q \\ D^Q \end{array} \begin{array}{c} \begin{array}{cccc} a^K & b^K & c^K & D^K \end{array} \\ \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Matrix representation of the look-ahead mask

$$QK^T + M = \begin{array}{c} \\ a^Q \\ b^Q \\ c^Q \\ D^Q \end{array} \begin{array}{c} \begin{array}{cccc} a^K & b^K & c^K & D^K \end{array} \\ \begin{bmatrix} a^Q a^K & a^Q b^K & a^Q c^K & -\infty \\ b^Q a^K & b^Q b^K & b^Q c^K & -\infty \\ c^Q a^K & c^Q b^K & c^Q c^K & -\infty \\ D^Q a^K & D^Q b^K & D^Q c^K & -\infty \end{bmatrix} \end{array}$$

Query-key matrix added to the padding mask matrix

Anything added to -∞ becomes -∞, so the resulting column D^K is a column of -∞. Now, what happens when softmax is applied to the matrix?

$$Soft(QK^T + M) = \begin{array}{c} \\ a^Q \\ b^Q \\ c^Q \\ D^Q \end{array} \begin{array}{c} \begin{array}{cccc} a^K & b^K & c^K & D^K \end{array} \\ \begin{bmatrix} (a^Q a^K & a^Q b^K & a^Q c^K)_S & 0 \\ (b^Q a^K & b^Q b^K & b^Q c^K)_S & 0 \\ (c^Q a^K & c^Q b^K & c^Q c^K)_S & 0 \\ (D^Q a^K & D^Q b^K & D^Q c^K)_S & 0 \end{bmatrix} \end{array}$$

Softmax of the result from adding the query-key matrix to the mask matrix

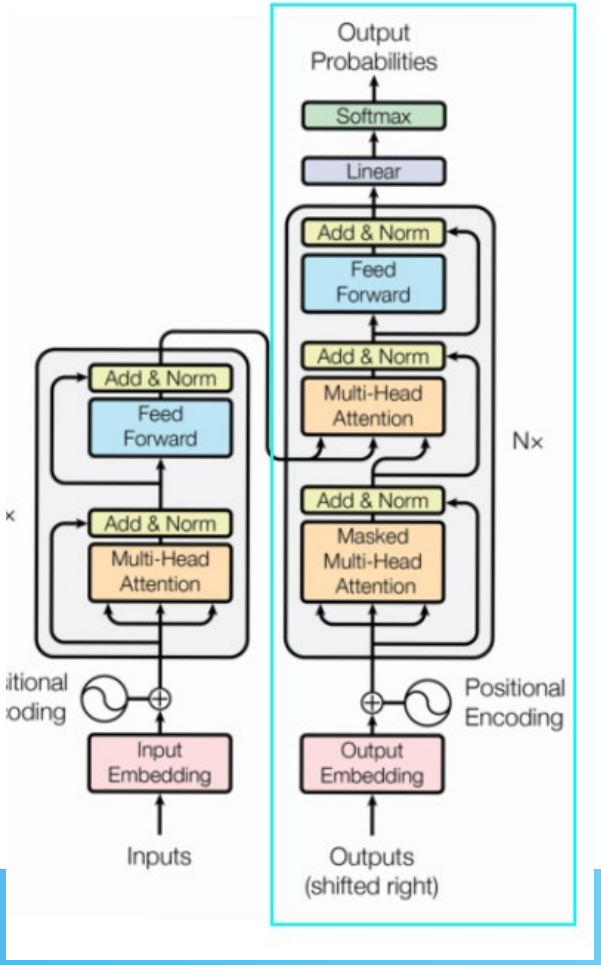
Error?

- Self supervised, so just checking if the logits for the next word are as predicted then backpropping with MSE (I think? Hard to find info here for sm reason)
- These are done in batches much like

```
when input is tensor([18]) the target: 47
when input is tensor([18, 47]) the target: 56
when input is tensor([18, 47, 56]) the target: 57
when input is tensor([18, 47, 56, 57]) the target: 58
when input is tensor([18, 47, 56, 57, 58]) the target: 1
when input is tensor([18, 47, 56, 57, 58,  1]) the target: 15
when input is tensor([18, 47, 56, 57, 58,  1, 15]) the target: 47
when input is tensor([18, 47, 56, 57, 58,  1, 15, 47]) the target: 58
```

Tada! That’s how transformers work

- GPT is decoder only. This means it doesn’t have any enc blocks and just does the masking I mentioned on the last slide



S’more random stuff

- Because why not
-
-
- Ring attention! ->

Step 3: Ring-Based Communication and Overlapping

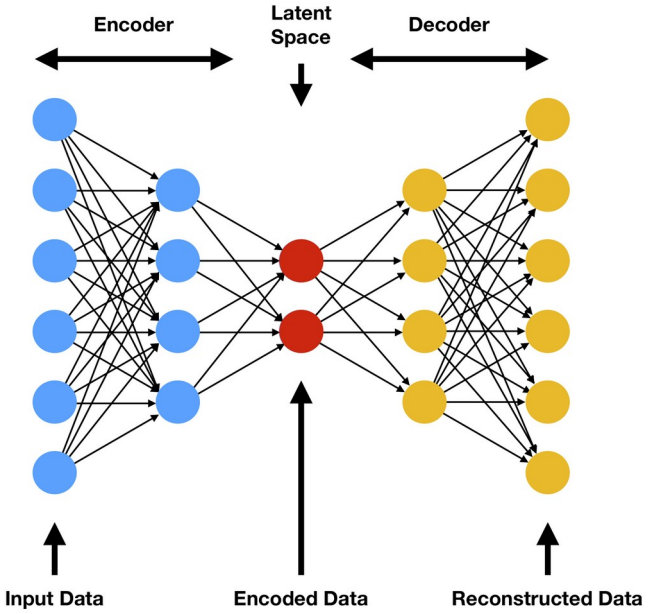
Once a host finishes processing its block, it begins to pass its key-value pairs to the next host in the ring. Simultaneously, it receives the key-value pairs from the previous host. This process is overlapped with computation, ensuring minimal idle time. For instance:

- Host 1 sends data to Host 2 and receives from Host 3.
- Host 2 sends data to Host 3 and receives from Host 1.
- Host 3 sends data to Host 1 and receives from Host 2.

This step ensures that each host gradually gets access to the key-value pairs from other blocks, which are necessary for calculating the attention scores that involve tokens from different blocks.

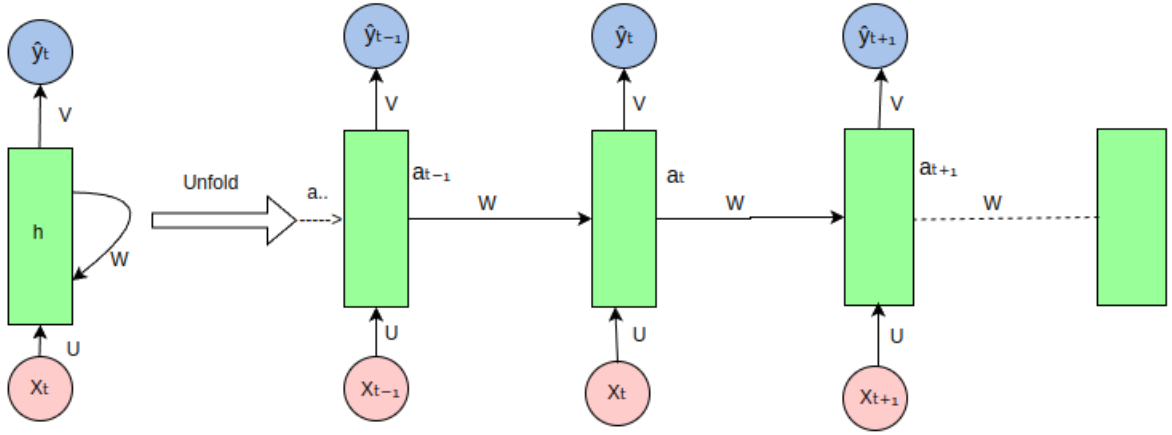
Neural coding- basically mechinterp

- VAEs
  - Autoencoder that enforces sparsity in its loss with KL divergence
- Linear probes
  - Literally just throw a bunch of data at your model and do classification on the inputs and outputs
- Find “features”
  - Try to interpret them



RNNs

- These are like a cross between transformers and MLPs- they can update their own weights during non-backprop parts of training, using the input *and* the state of the previous layer, and this makes them good at handling sequential data



Questions!!

Basically everything cool in neuroscience so far

- Place cells
- Sleep spindles
- Spatial vortexes
- The glymphatic system
- The role of glia
- hormonal signalling
- types of neurotransmitters
- neuromodulators
-

Grid cells

Mirror neurons

Synesthesia:

Neurogenesis

Optogenetics

Default mode network

Engrams

Neuroplasticity

Epigenetic modifications

Microglia

Neuronal oscillations

Neuronal avalanches

Synaptic pruning

Works cited aka my search history

https://stackoverflow.com/questions/6392739/what-does-the-at-symbol-do-in-python
https://medium.com/@ohadrubin/exploring-weight-decay-in-layer-normalization-challenges-and-a-reparameterization-solution-ad4d12c24950
https://stackoverflow.blog/2023/11/09/an-intuitive-introduction-to-text-embeddings/
https://jalammar.github.io/illustrated-transformer/
https://leimao.github.io/blog/Layer-Normalization/
https://en.wikipedia.org/wiki/Hebbian_theory
https://en.wikipedia.org/wiki/Feedforward_neural_network
https://arxiv.org/abs/1409.3215
https://www.youtube.com/watch?v=o0FppeD_xXQ&list=PL7m7hLIqA0hqsReJ0NYhyN3xiFQW9ko1h&index=2
https://www.pnas.org/action/oidcCallback?idpCode=connect&error=login_required&error_description=Login+required&state=anacHyh1K_51XzU1Qg1F9hyXpG2LLdSrCLeFWu-tPho
https://en.wikipedia.org/wiki/Feedforward_neural_network#Multilayer_perceptron
https://www.analyticsvidhya.com/blog/2021/03/forward-propagation-and-errors-in-a-neural-netwrok/
https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
https://en.wikipedia.org/wiki/Self-organized_criticality
https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3
https://www.youtube.com/watch?v=G45TuC6zRf4
https://neuronaldynamics.epfl.ch/online/Ch1.S3.html
https://en.wikipedia.org/wiki/Bisection_method
https://en.wikipedia.org/wiki/Spiking_neural_network
https://en.wikipedia.org/wiki/Multilayer_perceptron#:~:text=A%20multilayer%20perceptron%20(MLP)%20is,that%20is%20not%20linearly%20separable.
http://www.pmaweb.caltech.edu/Courses/ph136/yr2012/1203.1.K.pdf
https://medium.com/m/signin?operation=login&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fillustrated-self-attention-2d627e33b20a&source=post_page---two_column_layout_nav----------------------global_nav----------
https://paperswithcode.com/method/layer-normalization
https://pubs.acs.org/doi/abs/10.1021/acsami.0c14325#:~:text=The%20Bienenstock%E2%80%93Cooper%E2%80%93Munro%20(,spike%2Dtiming%2Ddependent%20plasticity.
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
https://en.wikipedia.org/wiki/Wolfe_conditions
https://en.wikipedia.org/wiki/Feedforward_neural_network#Perceptron
https://en.wikipedia.org/wiki/BCM_theory
https://pubmed.ncbi.nlm.nih.gov/23049852/
https://realpython.com/gradient-descent-algorithm-python/#basic-gradient-descent-algorithm
https://machinelearningmastery.com/transformer-models-with-attention/
https://github.com/ttngu207/najafi-2018-nwb/tree/master?tab=readme-ov-file
https://deepai.org/machine-learning-glossary-and-terms/rmsprop#:~:text=RMSProp%2C%20which%20stands%20for%20Root,in%20training%20deep%20neural%20networks.
https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31
https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643
https://www.reddit.com/r/singularity/comments/13vr70t/someone_managed_to_decode_a_tiny_transformer_the/
https://arxiv.org/pdf/2301.05217.pdf
https://hyperphysics.phy-astr.gsu.edu/hbase/quantum/disfcn.html
https://iopscience.iop.org/article/10.1088/0954-898X/8/4/002
https://www.neelnanda.io/modular-addition-walkthrough-2

https://mirror.explodie.org/universality_in_elementary_cellular_automata_by_matthew_cook.pdf
https://en.wikipedia.org/wiki/Reinforcement_learning
https://en.wikipedia.org/wiki/Hopfield_network
https://en.wikipedia.org/wiki/Distribution_function_(physics)
https://www.youtube.com/watch?v=XHMa-Ba-2Mo
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a
http://www.scholarpedia.org/article/Attractor_network
http://jalammar.github.io/illustrated-transformer/
https://datascience.stackexchange.com/questions/57996/is-gradient-descent-also-used-during-feed-forward-propagation-in-neural-network
https://www.science.org/doi/10.1126/science.aal4835
https://en.wikipedia.org/wiki/Seq2seq
https://medium.com/me/notifications?source=---two_column_layout_nav----------
https://stats.stackexchange.com/questions/474440/why-do-transformers-use-layer-norm-instead-of-batch-norm
https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452
https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a?source=-----2d627e33b20a----------post_regwall----------&skipOnboarding=1
https://stats.stackexchange.com/questions/31930/difference-between-som-and-hopfield
https://en.wikipedia.org/w/index.php?title=Attractor_network&action=edit&section=6
https://en.wikipedia.org/wiki/Phase_space
https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
https://medium.com/mlearning-ai/how-do-self-attention-masks-work-72ed9382510f
https://realpython.com/gradient-descent-algorithm-python/
https://stackoverflow.com/questions/72806582/do-layer-normalization-in-pytorch-without-learnable-parameters
https://ai.stackexchange.com/questions/25053/what-is-the-cost-function-of-a-transformer
https://en.wikipedia.org/wiki/Neural_coding#Correlation_coding
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
https://proceedings.neurips.cc/paper_files/paper/2019/file/2f4fe03d77724a7217006e5d16728874-Paper.pdf
https://nlp.seas.harvard.edu/2018/04/03/attention.html
https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse
https://en.wikipedia.org/wiki/Bifurcation_diagram
https://stackoverflow.com/questions/6392739/what-does-the-at-symbol-do-in-python/28997112#28997112
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5318375/
https://en.wikipedia.org/wiki/Gradient_boosting
https://en.wikipedia.org/wiki/Bifurcation_memory
https://neelnanda.io/grokking
https://linuxize.com/post/how-to-set-or-change-timezone-in-linux/
https://en.wikipedia.org/wiki/Neural_coding
https://en.wikipedia.org/wiki/Stochastic_gradient_descent#AdaGrad
https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34

http://papers.neurips.cc/paper/8689-understanding-and-improving-layer-normalization.pdf
https://en.wikipedia.org/wiki/Gradient_descent
http://arxiv.org/pdf/2301.05217
https://medium.com/get-started/plans
https://paperswithcode.com/method/sparse-autoencoder#:~:text=A%20Sparse%20Autoencoder%20is%20a,are%20penalized%20within%20a%20layer.
https://gmongaras.medium.com/how-do-self-attention-masks-work-72ed9382510f
https://en.wikipedia.org/wiki/Measure_(mathematics)
https://www.databricks.com/glossary/adagrad#:~:text=Adaptive%20Gradient%20Algorithm%20(Adagrad)%20is,incorporating%20knowledge%20of%20past%20observations.
https://www.pnas.org/action/oidcStart?redirectUri=%2Fdoi%2Ffull%2F10.1073%2Fpnas.132651299
https://discuss.huggingface.co/t/encoder-decoder-loss/4335
https://w11.death-note-manga.com/manga/death-note-chapter-63/
https://stackoverflow.com/questions/71581197/what-is-the-loss-function-used-in-trainer-from-the-transformers-library-of-huggi
https://qri.org/blog/symmetry-theory-of-valence-2020
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3895988/
https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
https://en.wikipedia.org/wiki/Feigenbaum_constants
https://en.wikipedia.org/wiki/Reinforcement_learning#Model-based_algorithms
https://www.youtube.com/watch?v=o0FppeD_xXQ&list=PL7m7hLIqA0hqsReJ0NYhyN3xiFQW9ko1h&index=3
https://www.neelnanda.io/mechanistic-interpretability/modular-addition-walkthrough
https://machinelearningmastery.com/the-transformer-model/
https://github.com/ttngu207/najafi-2018-nwb/blob/master/notebooks/Najafi-2018_example.ipynb
https://roman.computer/projects
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8016278/#:~:text=Long%2Dterm%20potentiation%20of%20synaptic,proteins%20%5B5%E2%80%937%5D.
https://medium.com/@seshu8hachi/stochastic-gradient-descent-vs-gradient-descent-exploring-the-differences-9c29698b3a9b#:~:text=Stochastic%20Gradient%20Descent%3A%20Faster%20convergence,entire%20dataset%20for%20each%20iteration.
https://medium.com/@tanuj22july/breaking-the-boundaries-understanding-context-window-limitations-and-the-idea-of-ring-attention-170e522d44b2
https://towardsdatascience.com/build-your-own-transformer-from-scratch-using-pytorch-84c850470dcb
https://guava.physics.uiuc.edu/~nigel/courses/563/Essays_2012/PDF/banerjee.pdf
https://arxiv.org/pdf/1706.03762
https://www.reddit.com/r/MachineLearning/comments/bnejs3/d_what_does_the_feedforward_neural_network_in/
https://ai.stackexchange.com/questions/40179/how-does-the-decoder-only-transformer-architecture-work
https://en.wikipedia.org/wiki/Residual_neural_network
https://builtin.com/machine-learning/common-loss-functions
https://datascience.stackexchange.com/questions/68220/how-are-q-k-and-v-vectors-trained-in-a-transformer-self-attention
https://builtin.com/artificial-intelligence/transformer-neural-network
https://en.wikipedia.org/wiki/Line_search
https://stackoverflow.com/questions/66633813/sequence-to-sequence-loss
https://ai.stackexchange.com/questions/38905/please-help-me-understand-the-role-of-loss-function-in-neural-networks
https://arxiv.org/abs/2310.01889
https://towardsdatascience.com/a-complete-guide-to-write-your-own-transformers-29e23f371ddd
https://en.wikipedia.org/wiki/Kuramoto_model
https://pytorch.org/docs/stable/generated/torch.tril.html
http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/disfcn.html
https://ethz.ch/content/dam/ethz/special-interest/mtec/chair-of-entrepreneurial-risks-dam/documents/Essay/0SX3463.pdf
https://arxiv.org/abs/1706.03762
https://www.quora.com/Whats-the-difference-between-a-single-output-RNN-and-an-MLP-whose-input-data-contains-all-of-the-features-of-the-given-time-steps
https://www.neelnanda.io/grokking
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
https://en.wikipedia.org/wiki/Attractor_network
https://en.wikipedia.org/wiki/Softmax_function
https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9313413/

https://en.wikipedia.org/wiki/Abelian_sandpile_model
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4315928/
https://medium.com/get-started/topics
https://www.pnas.org/doi/full/10.1073/pnas.132651299
https://scikit-learn.org/stable/modules/sgd.html
https://neelnanda.io/modular-addition-walkthrough-2
https://ai.stackexchange.com/questions/4320/why-are-the-initial-weights-of-neural-networks-randomly-initialised
https://math.stackexchange.com/questions/1973521/what-is-the-difference-between-line-search-and-gradient-descent
https://papers.neurips.cc/paper/8689-understanding-and-improving-layer-normalization.pdf
https://neuronaldynamics.epfl.ch/index.html
https://www.neelnanda.io/modular-addition-notebook
https://deepai.org/machine-learning-glossary-and-terms/bias-vector
https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0
https://medium.com/m/login-redirect?redirectUrl=https%3A%2F%2Ftowardsdatascience.com%2Fillustrated-self-attention-2d627e33b20a
https://en.wikipedia.org/wiki/Hodgkin%E2%80%93Huxley_model
https://medium.com/@kyeg/the-feedforward-demystified-a-core-operation-of-transformers-afcd3a136c4c
https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/
https://www.youtube.com/watch?v=HOxSKBxUVpg
https://en.wikipedia.org/wiki/Stochastic_gradient_descent
https://arxiv.org/pdf/2301.05217
https://en.wikipedia.org/wiki/Neural_coding#Population_coding
https://stackoverflow.com/a/28997112/229792
https://medium.com/analytics-vidhya/understanding-q-k-v-in-transformer-self-attention-9a5eddaa5960
https://en.wikipedia.org/wiki/Reinforcement_learning#Deep_reinforcement_learning
https://neelnanda.io/modular-addition-notebook
https://www.reddit.com/r/learnmachinelearning/comments/1bn3eyh/scale_and_shift_learnable_parameters_in_layer/
http://nlp.seas.harvard.edu/2018/04/03/attention.html
https://neuronaldynamics.epfl.ch/online/index.html
https://optimization.cbe.cornell.edu/index.php?title=AdaGrad
https://www.analyticsvidhya.com/blog/2021/07/lets-understand-the-problems-with-recurrent-neural-networks/
https://www.nature.com/articles/s41467-020-15158-3
https://en.wikipedia.org/wiki/Liouville%27s_theorem_(Hamiltonian)
https://en.wikipedia.org/wiki/Deep_reinforcement_learning
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a?gi=4a0fe648b1e4
https://pnas.scienceconnect.io/api/oauth/authorize?ui_locales=en&scope=affiliations+login_method+merged_users+openid+settings&response_type=code&redirect_uri=https%3A%2F%2Fwww.pnas.org%2Faction%2FoidcCallback%3FidpCode%3Dconnect&state=anacHyh1K_51XzU1Qg1F9hyXpG2LLdSrCLeFWu-tPho&prompt=none&nonce=TevCJK6%2B9hGXHqLdVck%2BcicrePJj6kw3yvHi%2F29SVrE%3D&client_id=pnas
https://towardsdatascience.com/gpt-3-rnns-and-all-that-deep-dive-into-language-modelling-7f67658ba0d5#:~:text=Instead%20of%20learning%20a%20set,function%20of%20this%20state%20vector.
https://www.galexander.org/osamaletter.html
https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/
https://www.pmaweb.caltech.edu/Courses/ph136/yr2012/1203.1.K.pdf
https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)