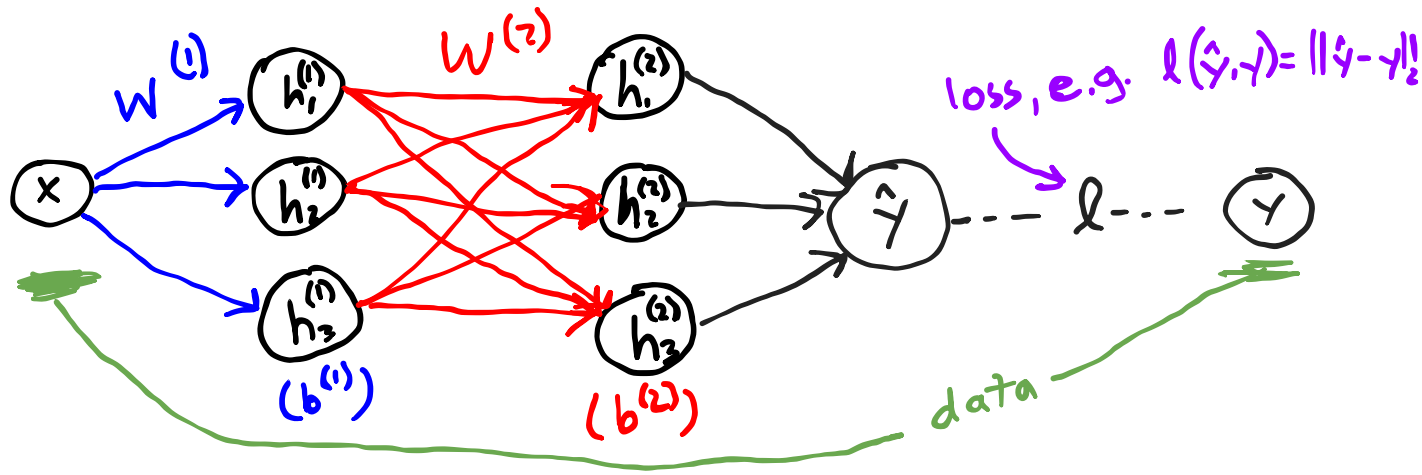


Neural Network Architectures for Sequential Inputs

Review from DMU: Neural Networks



Multi-Layered Perceptron (MLP)

$$h_i(x) = \sigma(W_i x + b_i)$$

$$\hat{y} = h_3(h_2(h_1(x)))$$

$$\theta^* = \arg \min_{\theta} \sum_{(x,y) \in \mathcal{D}} l(f_{\theta}(x), y)$$

("misnomer" because original perceptron used step function)

Stochastic Gradient Descent: $\theta \leftarrow \theta - \alpha \nabla_{\theta} l(f_{\theta}(x), y)$

Review: Why sequences are needed

In an MDP, for any $k \geq 0$:

$$P(r_{t+k} \mid \pi, s_t, a_{t-1}, s_{t-1}, \dots) = P(r_{t+k} \mid \pi, s_t),$$

so there is always an optimal policy based on the state.

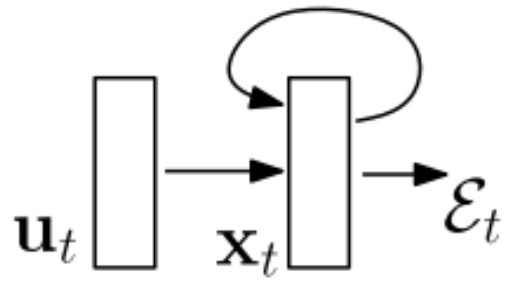
In a POMDP, there is no systematic way to simplify

$$P(r_{t+k} \mid \pi, o_t, a_{t-1}, o_{t-1}, \dots),$$

so an optimal policy may depend on the entire action-observation history.

A big MLP that takes the entire sequence in generally has too many parameters (and cannot handle variable length input)

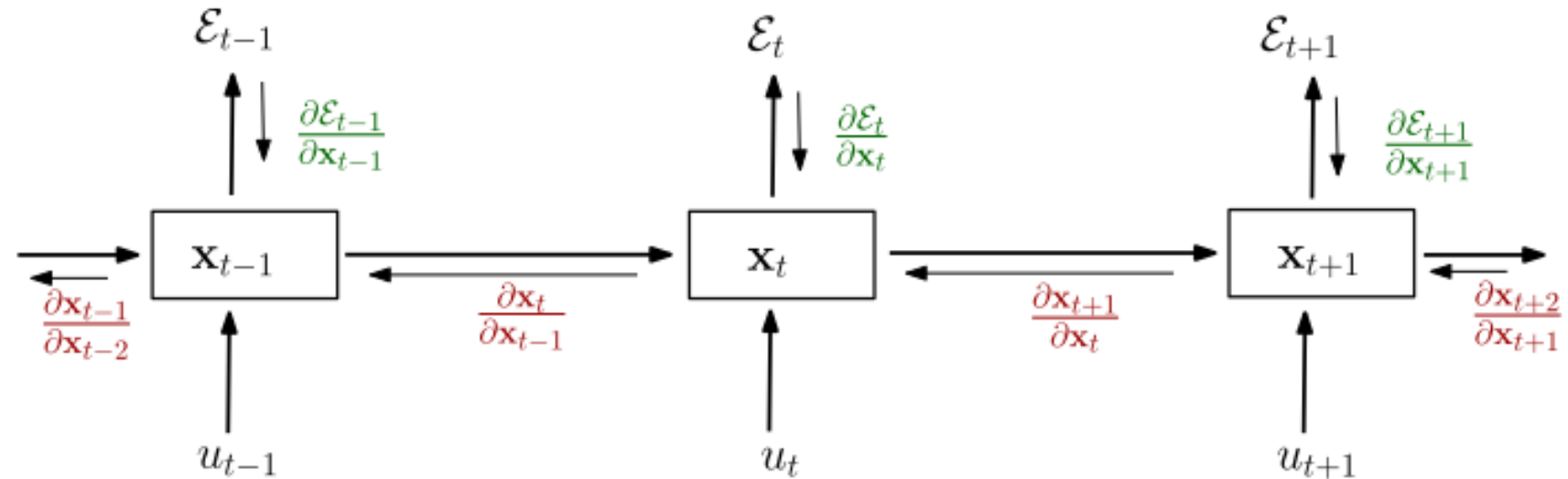
Recurrent neural networks



Input: u_t
State/Output: x_t
Cost: \mathcal{E}_t

$$\mathbf{x}_t = F(\mathbf{x}_{t-1}, \mathbf{u}_t, \theta)$$

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b}$$



Problem: Vanishing/exploding gradients

$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta}$$

$$\mathbf{x}_t = \mathbf{W}_{rec} \sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in} \mathbf{u}_t + \mathbf{b}$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$$

($\frac{\partial^+ \mathbf{x}_k}{\partial \theta}$ is partial with \mathbf{x}_{k-1} held constant)

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T \text{diag}(\sigma'(\mathbf{x}_{i-1}))$$

Generally, informally: If $\sigma' \in [0, 1]$ (e.g. sigmoid), product can become very small, even numerically zero.

Formally, for the linear case ($\sigma = \text{identity}$)

- Largest eigenvalue of $\mathbf{W}_{rec} < 1$: *sufficient* for vanishing
- Largest eigenvalue of $\mathbf{W}_{rec} > 1$: *necessary* for exploding

Important papers:

- Hochreiter, Sepp. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TU Munich, 1991.)
- Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." IEEE transactions on neural networks 5.2 (1994) [11k cit.]
- Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." International conference on machine learning, 2013. [6.5k cit.]

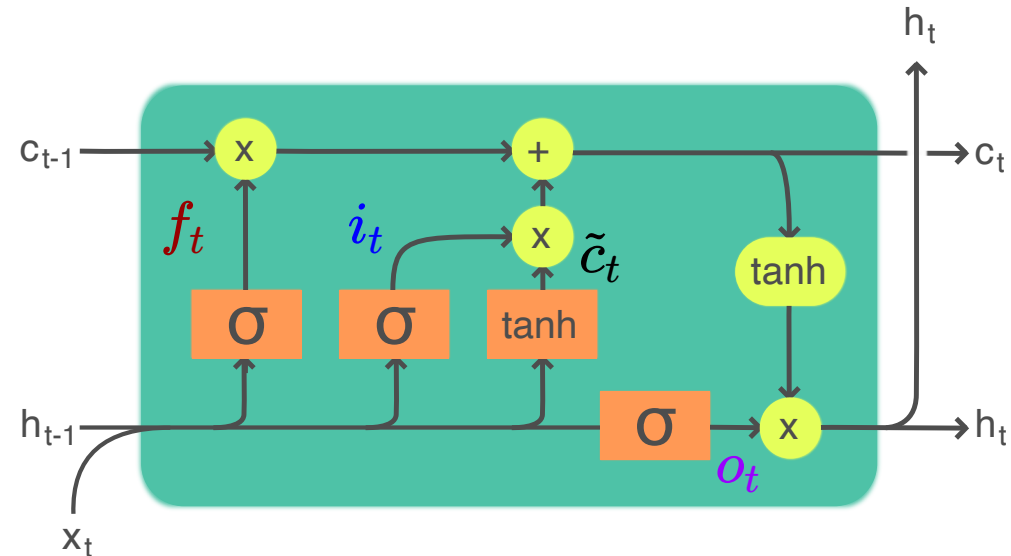
Long Short Term Memory (LSTM)

Input: x_t

Output: h_t

Cell state: c_t

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) && \text{Forget gate} \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) && \text{Input Gate} \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) && \text{Output Gate} \\ \tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \sigma_h(c_t) \end{aligned}$$



Legend:

Layer

Componentwise

Copy

Concatenate



Gated Recurrent Unit (GRU)

Input: x_t

Output: \hat{y}_t (h_t)

Recurrent State: h_t

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

$$\hat{h}_t = \phi(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

Fewer parameters than LSTM

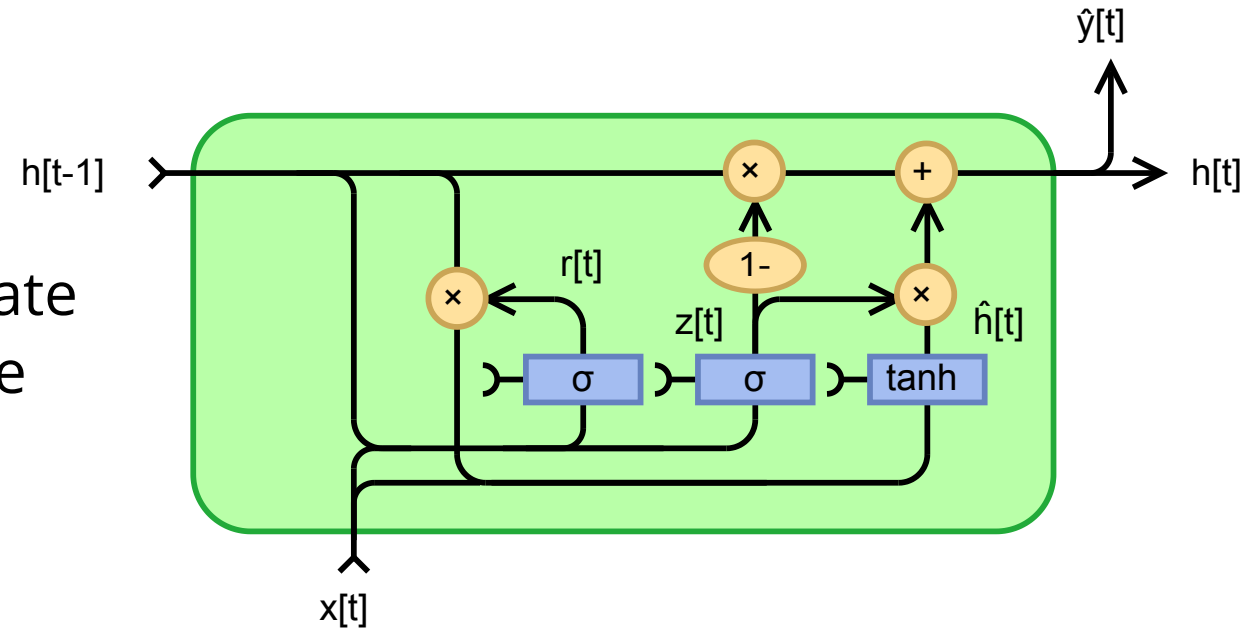
$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

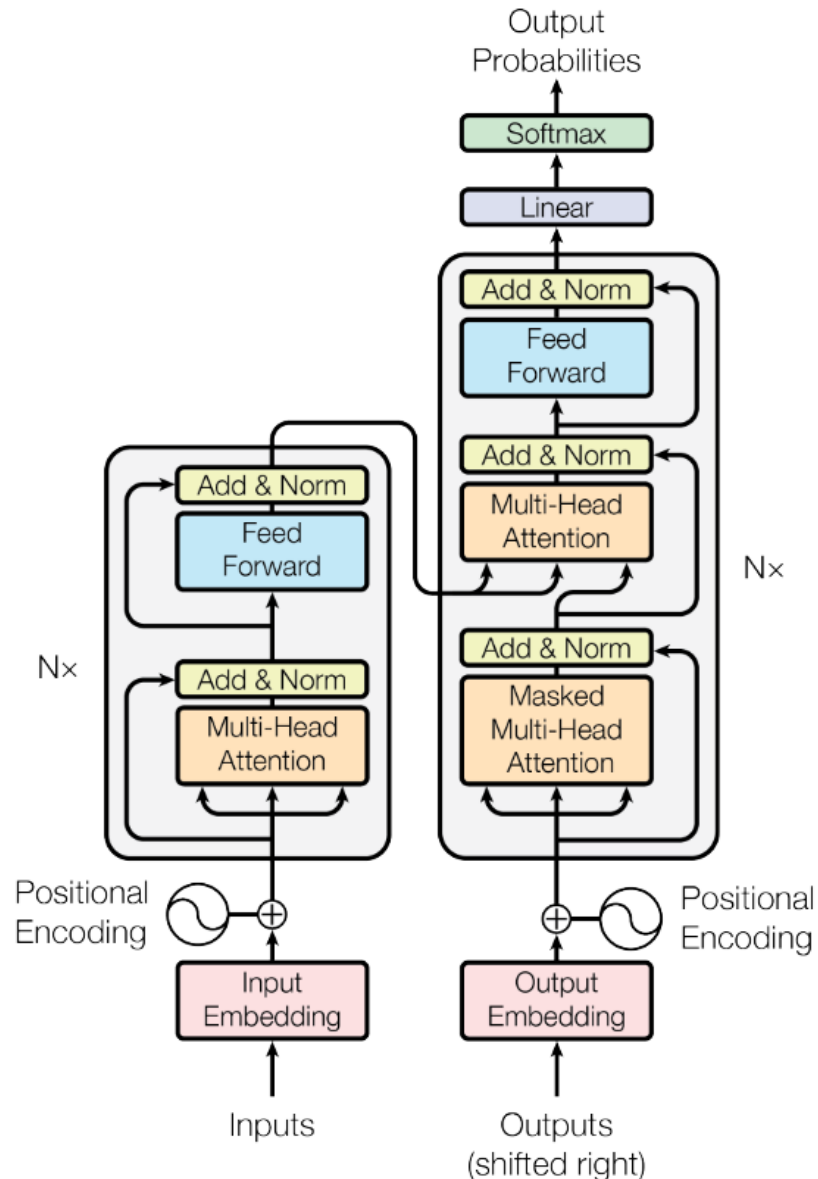
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

Update gate
Reset gate



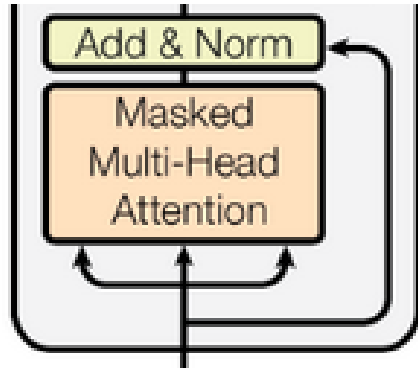
By Jebblad - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=66225938>

Transformer Overview



- **Multi-Head Attention**
- Skip Connections
- Embeddings
- Positional Encodings
- Layer Norm

Skip Connections (Resnets)



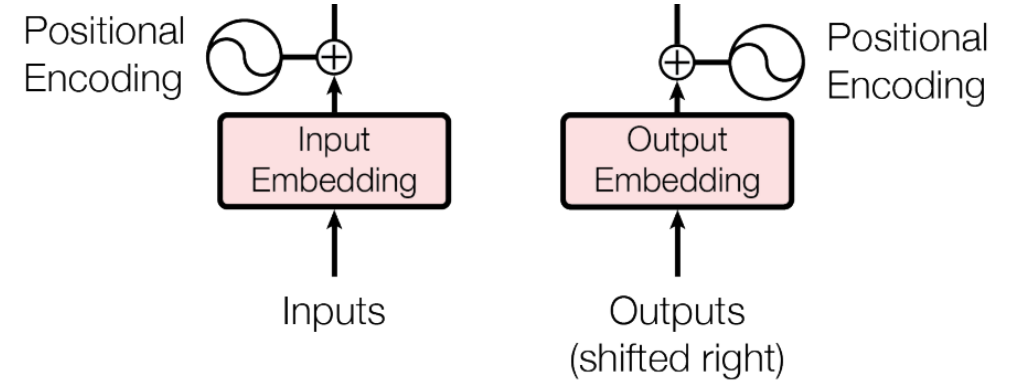
- Address the vanishing/exploding gradient problem
- Ensemble effect

Word (Token) Embeddings

Positional Encodings

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$



Layer Norm

$$\mu_l = \frac{1}{d} \sum_{i=1}^d x_i$$

$$\sigma_l^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu_l)^2$$

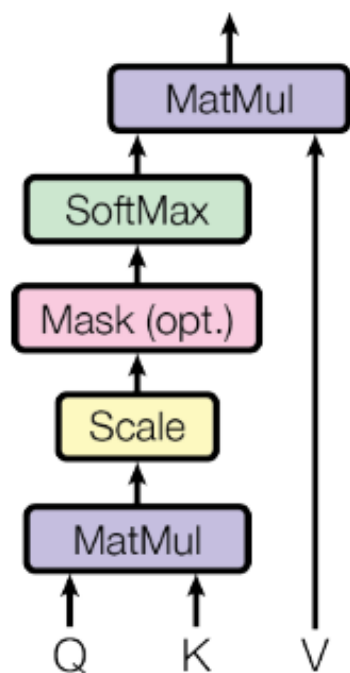
$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2}}$$

or

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}}$$

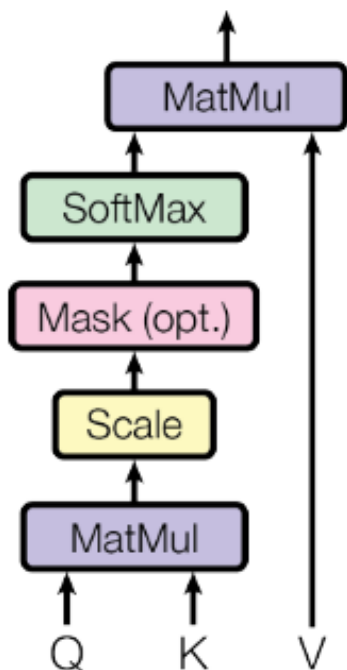
Attention: Fully differentiable key-value lookup

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



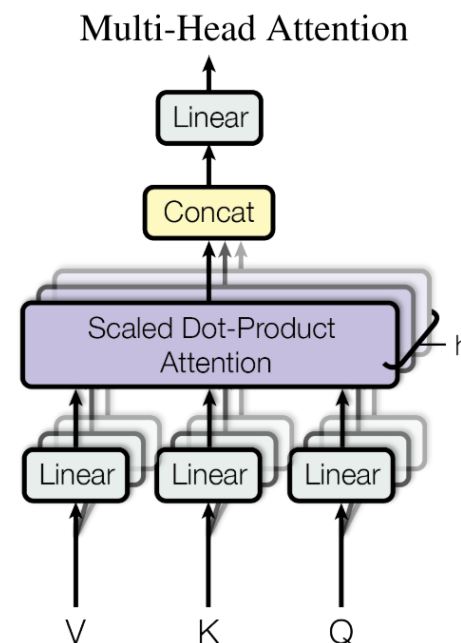
Attention: Fully differentiable key-value lookup

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$



Transformers: Putting it all together

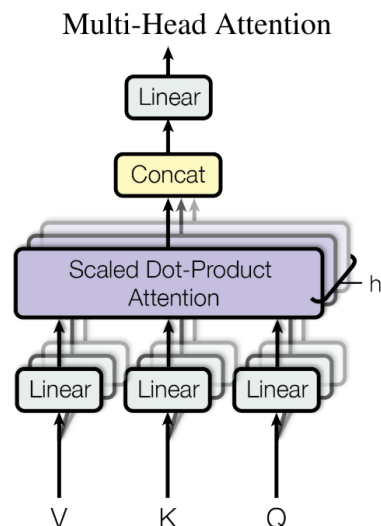
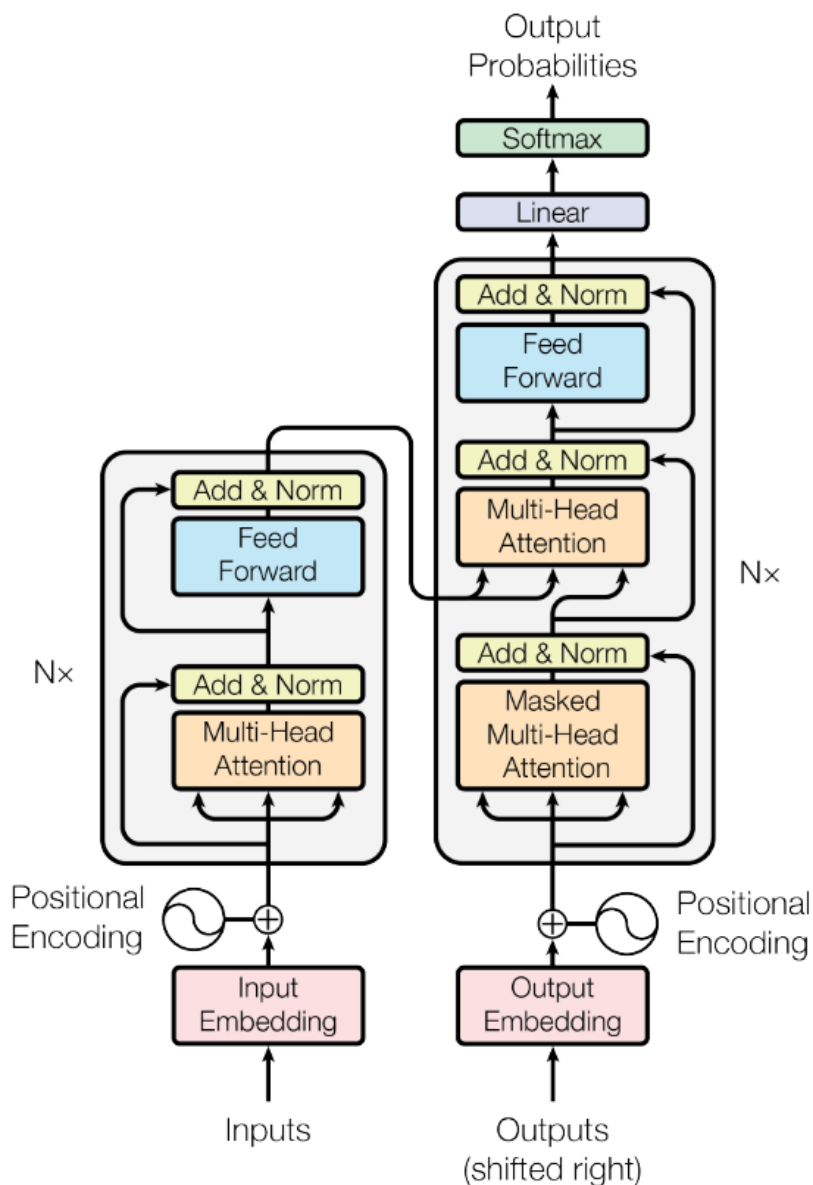


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

GPT-3

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020-December.

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

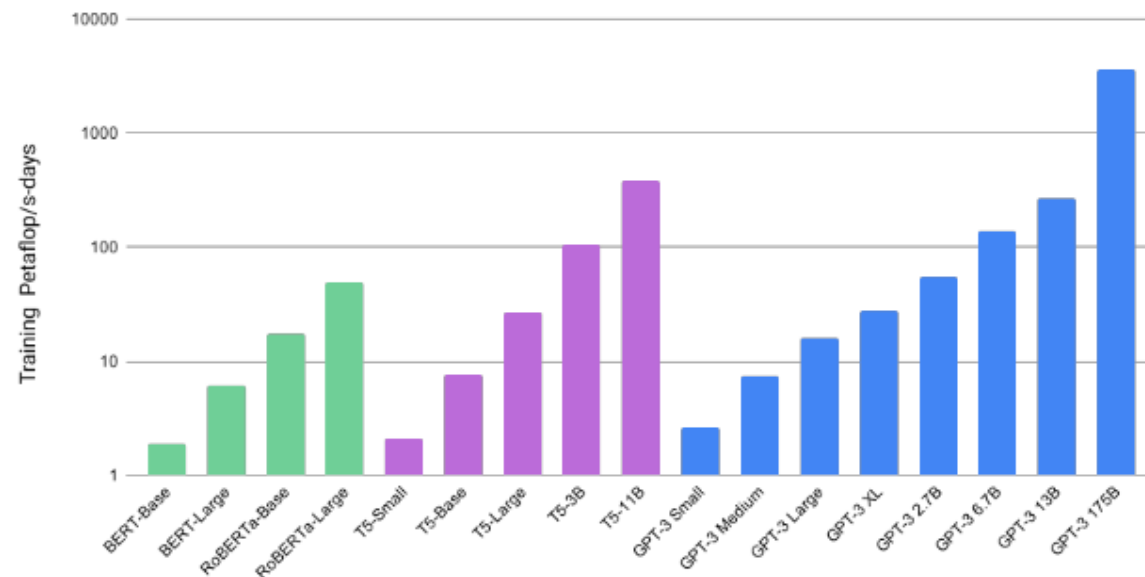


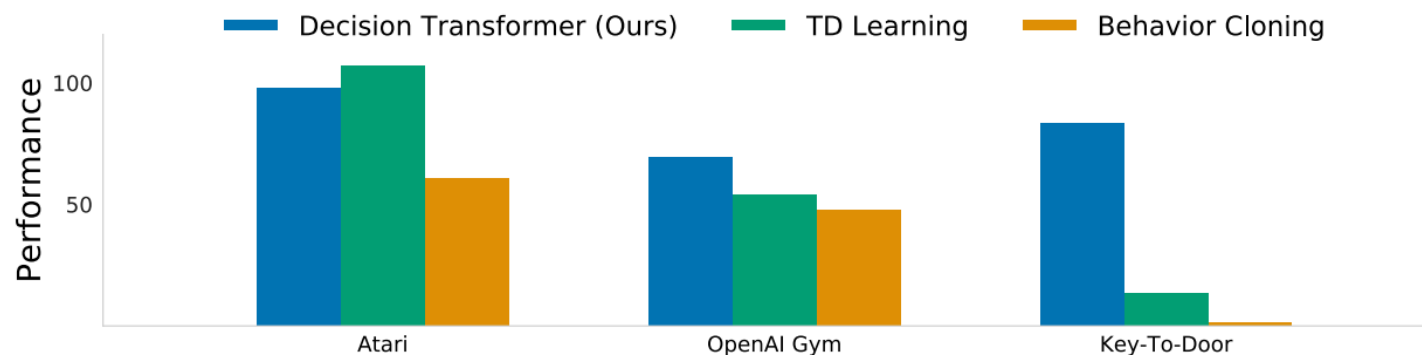
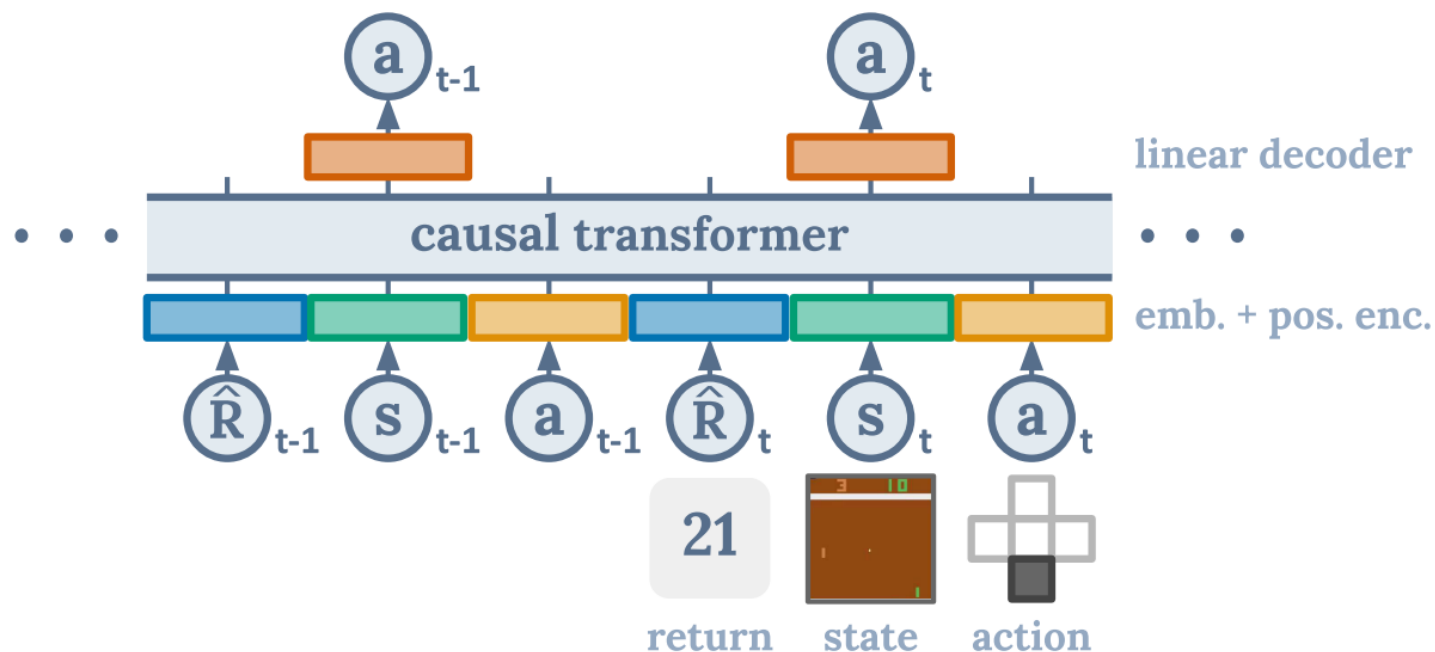
Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH⁺20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:

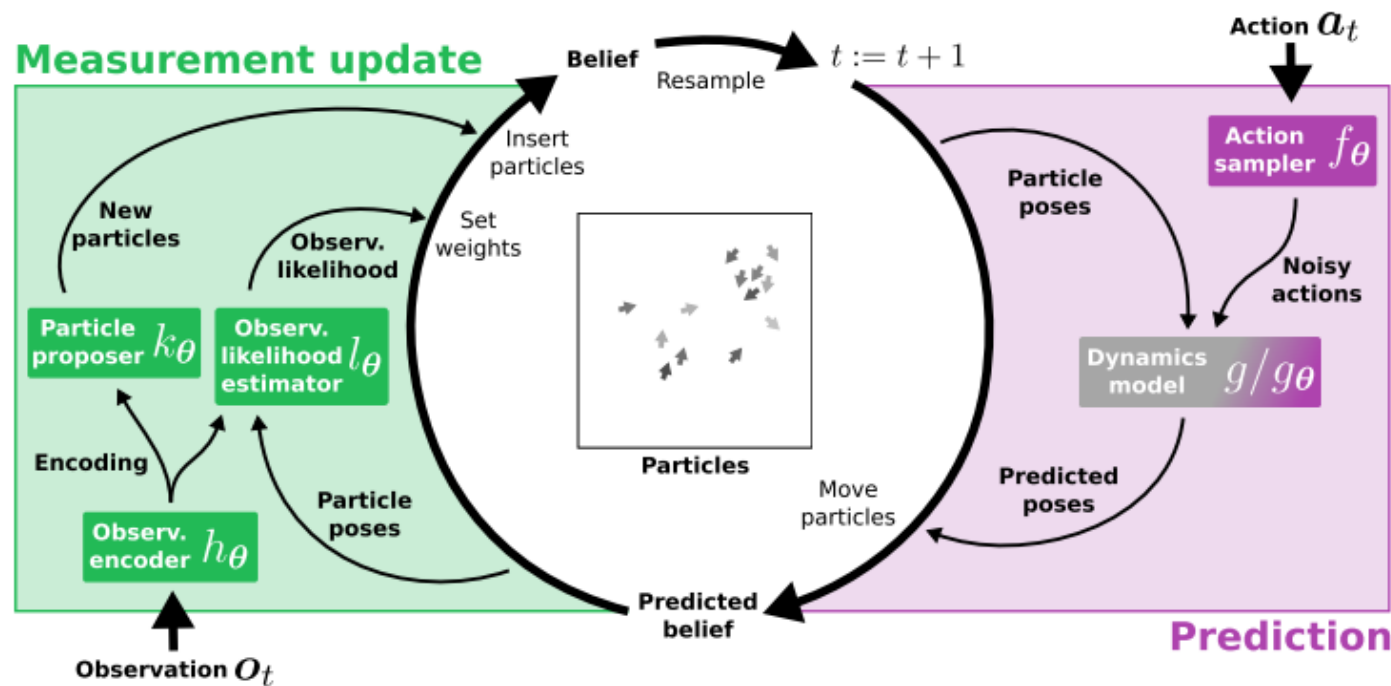
One day when I was playing tag with my little sister, she got really excited and she started doing these crazy farduddles.

Decision Transformer

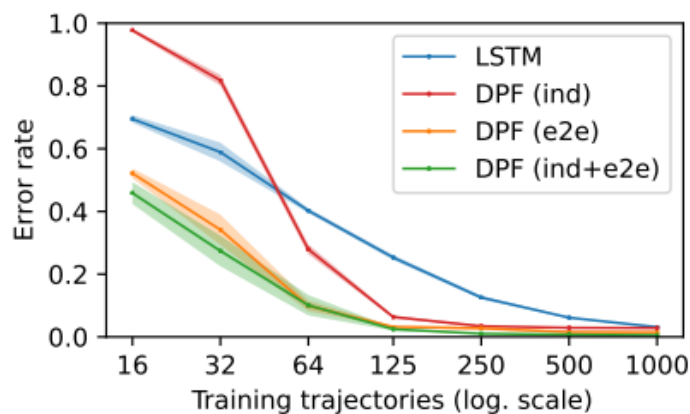
Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., & Mordatch, I. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling. *Advances in Neural Information Processing Systems*, 18, 15084–15097.



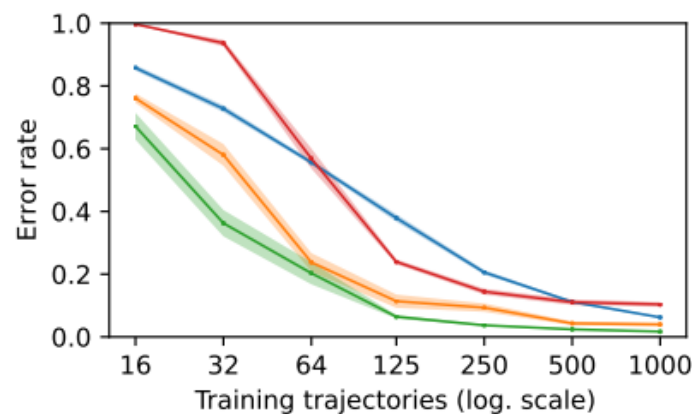
Differentiable Particle Filters



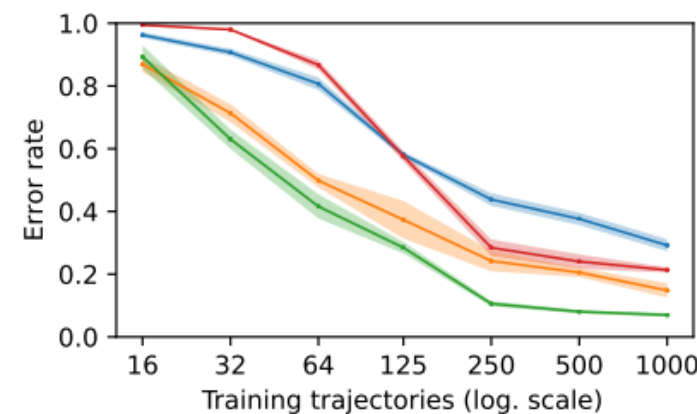
Jonschkowski, R., Rastogi, D., & Brock, O. (2018, June). Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors. *Proceedings of Robotics: Science and Systems*. <https://doi.org/10.15607/RSS.2018.XIV.001>



(a) Maze 1 (10x5)



(b) Maze 2 (15x9)



(c) Maze 3 (20x13)

My Thoughts

- Transformers work really well for interpreting structured data that is communicated in a time series (Language is a way of serializing ideas)
- State estimation is NOT usually the hardest part of a POMDP, so just use filtering if you have a decent model (?)
- Use an algorithmic prior when you are learning.
- In tasks where you have to interpret language, transformers are probably the way to go.