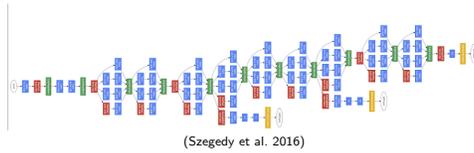## Motivation for Neural Networks

(https://www.cs.utexas.edu/ nn/-pages/research/neatdemo.html)

(Szegedy et al. 2016)

(Furber 2016)

(https://chatgpt.com)
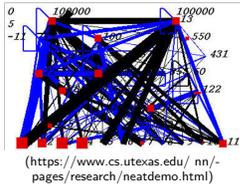
► Statistical machines with real-world applications
  ► Pattern recognition, control, behavior
  ► Biological and cognitive modeling
  ► Usually simulated, but also hardware

► Motivation from biological neural networks
  ► Abstracted into computational structures
  ► Massively parallel, simple operations
  ► Performance from scale
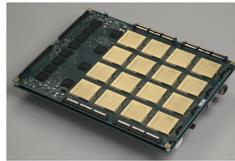
The Biological Neuron
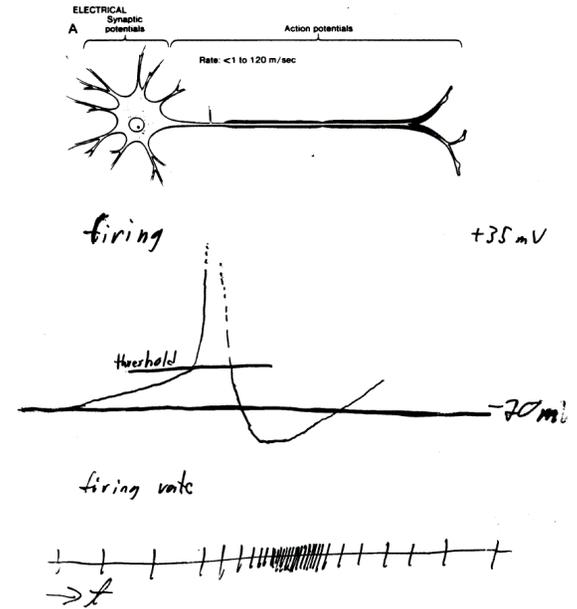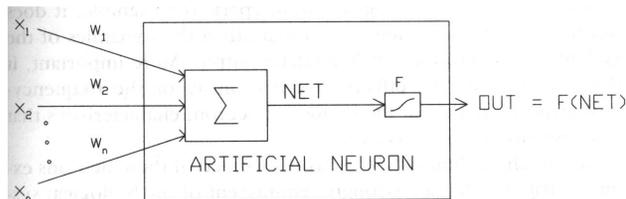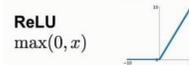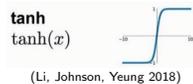

Artificial Neuron

Figure 1-3. Artificial Neuron with Activation Function
(Wasserman 1989)

► OUT = F(XW) = F($\sum w_i x_i$)
► Activity = Firing rate
► Nonlinear activation functions, e.g.:

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

tanh
$\tanh(x)$

ReLU
$\max(0, x)$

(Li, Johnson, Yeung 2018)
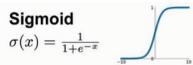
Feedforward Neural Networks (FNNs)

► Feedforward Neural Networks are the simplest type of artificial neural network.
► Consist of:
  ► Input layer
  ► One or more hidden layers
  ► Output layer
► Information flows in one direction, from input to output, without loops or cycles.

(Wasserman 1989)
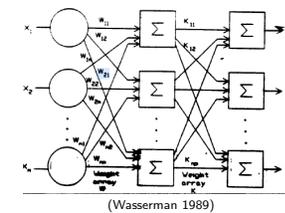
Details of Feedforward Neural Networks (FNNs)

- ▶ Input layer:
  - ▶ Receives raw data.
  - ▶ Each node corresponds to a feature or variable.
  - ▶ Passes input values to the next layer.
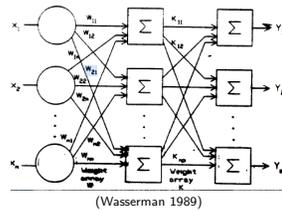- ▶ Hidden layers:
  - ▶ Perform computations.
  - ▶ Each neuron calculates a weighted sum of inputs, passes through an activation function (e.g., ReLU, Sigmoid, Tanh).
- ▶ Output layer:
  - ▶ Produces the network's prediction.
  - ▶ Number of neurons matches the number of possible outputs.

(Wasserman 1989)

Learning Through Gradient Descent

- ▶ Weights typically trained with gradient descent (a.k.a. backpropagation)
  - ▶ Form a training corpus of input-target pairs
  - ▶ Initialize the weights to be small and random
  - ▶ Then train until convergence or out of time
  - ▶ Evaluate on a test corpus of unseen data

(Wasserman 1989)
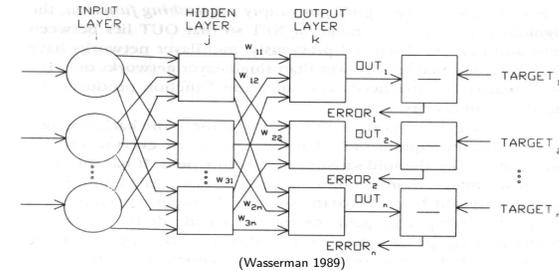
Learning Through Gradient Descent

- ▶ Gradient descent (online, i.e. stochastic version):
  - ▶ Select a training pair and present to the network
  - ▶ Compute output: $OUT = F(F(XU)W)$
  - ▶ Calculate error (i.e. loss): $\delta_k = F'(NET_k) * (T_k - OUT_k)$
  - ▶ Backpropagate error through the network: $\delta_j = F'(NET_j)\Sigma_k \delta_k W_{jk}$
  - ▶ Compute weight changes (i.e. the gradient):
    $\Delta W_{jk} = \eta \delta_k OUT_j; \qquad \Delta U_{ij} = \eta \delta_j OUT_i$
- ▶ Demo link: `https://playground.tensorflow.org`

- ▶ Alternatively, the weights can be discovered through evolution
  - ▶ Structure can be discovered through evolution as well

(Wasserman 1989)
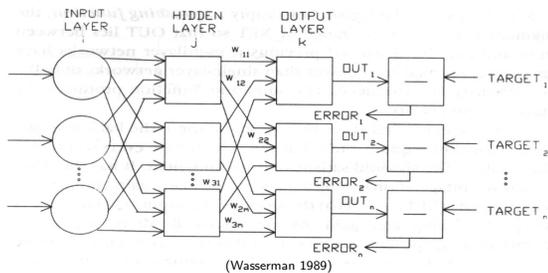
Example Evolved FFN: Pandemic Interventions

- ▶ The prescriptor receives an input vector of 21 days of past cases and NPIs
- ▶ It outputs stringency in 12 possible interventions
- ▶ Optimal strategy is not known; cannot use backprop
- ▶ Evolved with predictor as the fitness evaluator

(https://evolution.ml/demos/npidashboard/)

(Miikkulainen et al. 2021)
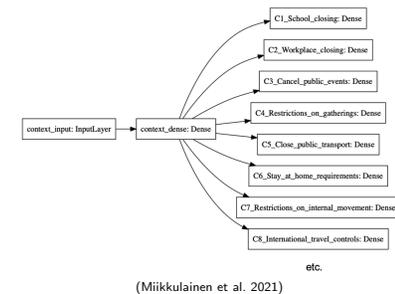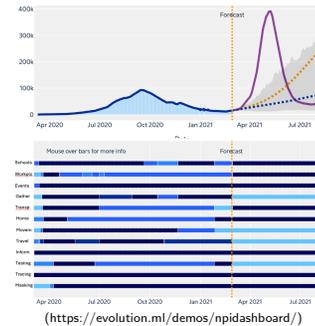
Recurrent Neural Networks (RNNs)

- ▶ Recurrent Neural Networks (RNNs) are designed to recognize patterns in sequences of data.
- ▶ RNNs have connections that loop back, allowing information to persist.
- ▶ Well-suited for tasks where context and order matter, such as time series, text, or audio.


(Wasserman 1989)
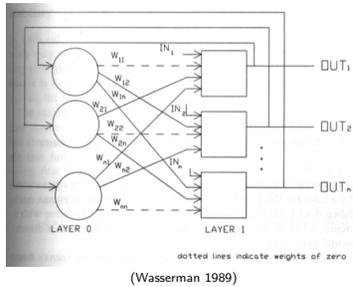
Details of Recurrent Neural Networks (RNNs)

- ▶ Neurons receive input from the previous layer and their previous states, allowing for memory.
- ▶ Process sequences one element at a time, maintaining a hidden state OUT_t that captures sequence information:

$$OUT_t = F(U \cdot IN_t + W \cdot OUT_{t-1})$$

- ▶ OUT_t: hidden state at time step t.
- ▶ IN_t: input at time step t.
- ▶ U, W: weight matrices.
- ▶ F: activation function (e.g., tanh, ReLU).
- ▶ Weights learned through backprop (e.g. BP through time), or evolved
- ▶ The recurrent structure can be customized through evolution

(Wasserman 1989)
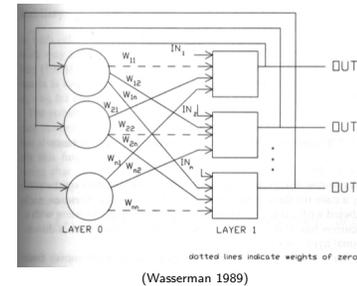
Example Evolved RNN: Controlling a Game Agent

- ▶ Sensors convey information about the current state
- ▶ Recurrency provides information from the past
  - ▶ E.g. speed, an opponent disappearing behind a wall...
- ▶ Combined to make a decision, implemented by effectors
- ▶ Evolved with success in game play as a fitness

(http://nerogame.org)
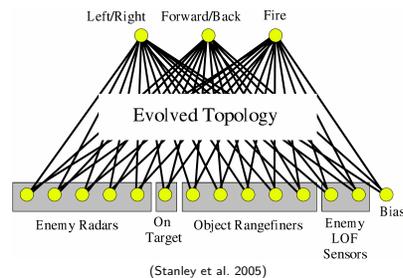
(Stanley et al. 2005)

Long Short-Term Memory Networks (LSTMs)

- ▶ LSTMs are a special type of Recurrent Neural Network (RNN) designed to learn and retain long-term dependencies.
- ▶ Overcome the limitations of traditional RNNs, particularly in handling long sequences.
- ▶ Highly effective for tasks involving sequential data, such as language modeling, speech recognition, and time-series forecasting.
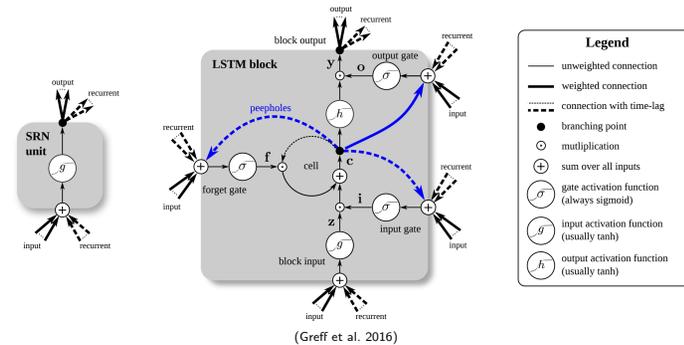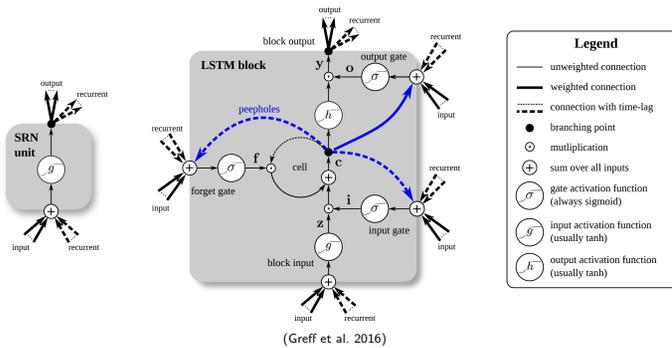

(Greff et al. 2016)

Left: Recurrent NN. Right: LSTM.

LSTM Cell Structure

- LSTM networks consist of LSTM cells, each with three main gates:
  - **Forget Gate:** Decides which parts of the cell's state to forget.
  - **Input Gate:** Determines which new information will be added to the cell state.
  - **Output Gate:** Controls the output based on the cell state.

(Greff et al. 2016)

LSTM Gate Operations

- **Forget Gate:** $\quad f_t = \sigma(W_f x_{f,t} + b_f)$
- **Input Gate:** $\quad i_t = \sigma(W_i x_{i,t} + b_i)$
  $$z_t = \tanh(W_z x_t + b_c)$$
- **Cell State:** $\quad c_t = f_t * c_{t-1} + i_t * z_t$
- **Output Gate:** $\quad o_t = \sigma(W_o x_{o,t} + b_o)$
  $$y_t = o_t * \tanh(c_t)$$
- Weights can be learned through gradient descent, or evolved
- The internal structure can be optimized through evolution


(Greff et al. 2016)
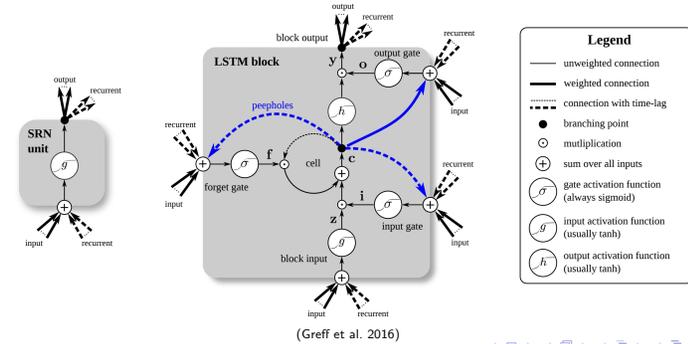
Examples of Evolved LSTM Designs

- Using the same components, but allowing more complexity
- More paths, nonlinearities, memory cells
- Still trained with gradient descent
- Improved 25-year old designs by 15% (in language modeling)

Original LSTM

Language

Music

(Rawal & Miikkulainen 2020)
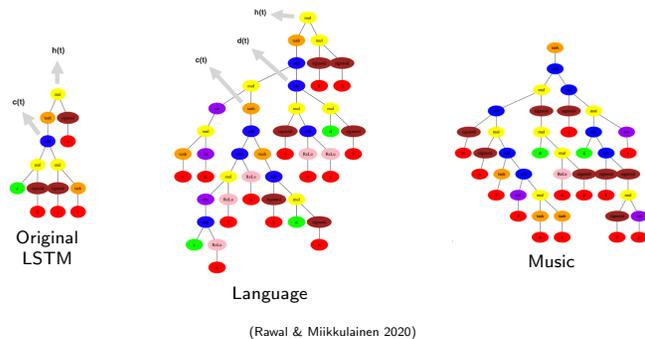
Convolutional Neural Networks (CNNs)

- CNNs are deep learning models designed to process grid-like data structures, such as images.
- Effective for tasks involving spatial hierarchies, such as image recognition, object detection, and video analysis.
- Inspired by the visual cortex, where neurons respond to overlapping regions in the visual field.

(Wikipedia 2025)
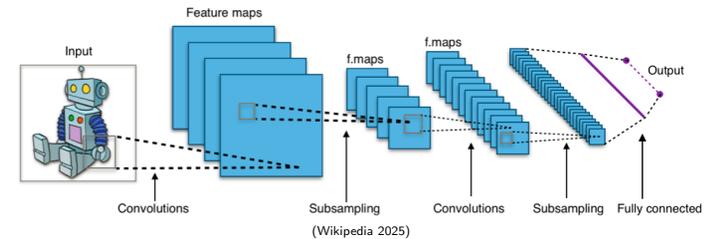
CNN Layers: Convolutional and Activation Layers
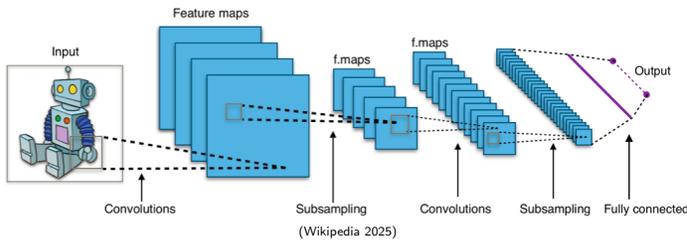
- **Convolutional Layer:**
  - Core component of a CNN.
  - Performs convolution operation using filters to extract spatial features.
  - With input I, kernel K, expressed as

$$(I * K)(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} I(x + i, y + j) \cdot K(i, j)$$

- **Activation Function:**
  - Typically uses the Rectified Linear Unit (ReLU).
  - Introduces non-linearity:

$$f(x) = \max(0, x)$$

(Wikipedia 2025)

CNN Layers: Pooling and Fully Connected Layers

- **Pooling Layer:**
  - Reduces spatial dimensions of feature maps.
  - Commonly uses Max Pooling to retain prominent features:

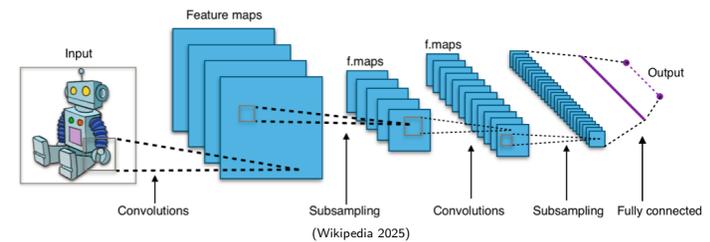$$P(x, y) = \max\{f(i, j) : i, j \in \text{window}(x, y)\}$$

- **Fully Connected Layer:**
  - High-level reasoning is performed.
  - Each neuron connects to every neuron in the previous layer.
  - Outputs class scores or other task-specific outputs:

$$y = W \cdot x + b$$

- Weights learned through gradient descent (limited evolution)
- Structure can be optimized through evolution

(Wikipedia 2025)

Example of an Evolved CNN Architecture

- $> 1M$ weights difficult to evolve directly
- But can evolve network structure and train it with gradient descent
  - Evolutionary Neural Architecture Search (NAS)!
- E.g. AmoebaNet: Given a search space of building blocks, evolve organization
- Scale up to more channels and widths
- State of the art in ImageNet in 2018

(Instawoot, Facebook 2022)   (Real et al. 2018)
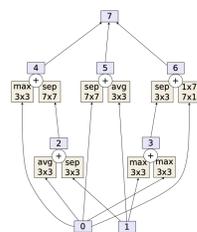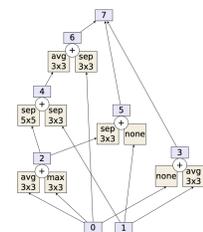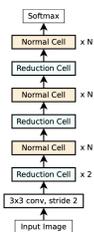
Transformer Networks

- Transformers are deep learning models that rely on self-attention mechanisms rather than traditional recurrent or convolutional layers.
- Highly effective for handling sequential data and long-range dependencies.
- Widely used in natural language processing (NLP) tasks such as machine translation, text generation, and summarization.


(Vaswani et al. 2017)
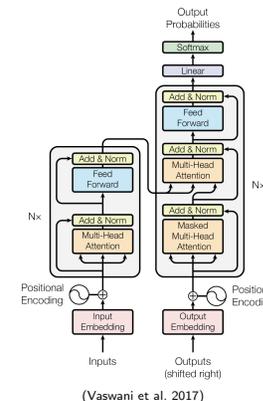
Transformer architecture: encoder (left) and decoder (right).

Transformer Architecture

- ▶ Trained to predict the next token (e.g. word, symbol) by gradient descent
  - ▶ Initially designed to beat LSTMs on perplexity
- ▶ The core idea is transformations of token embeddings based on their relationships with other words
  - ▶ Many different relationships in parallel; Many stacked transformations
  - ▶ Transformations are learned and opaque
  - ▶ When scaled up, a powerful sequence processing architecture

(Vaswani et al. 2017)

(Alammar 2018)
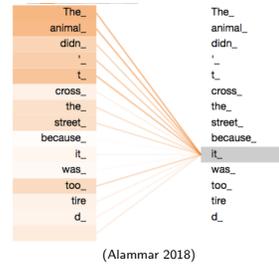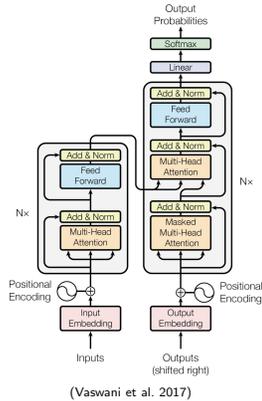
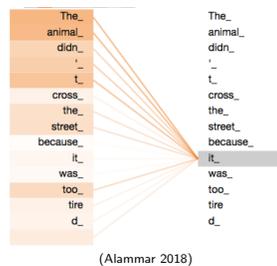## Self-Attention Mechanism in Transformers

- ▶ **A single attention head:**
  - ▶ Computes a transformation of each token representation vector (i.e. embedding) based on its relation (i.e. attention) to other tokens.
  - ▶ Based on projecting embeddings in three ways:
    - ▶ Query $Q_i = X_i W_i^Q$ formed from current word embedding X_i
    - ▶ Keys $K_j = X_j W_j^K$ formed from nearby word embeddings X_j
    - ▶ Values $V_j = X_j W_j^V$ formed from nearby word embeddings X_j
  - ▶ Self-attention matches queries with keys and forms a representation by combining the values proportionally.
    - ▶ Could do it simply with embeddings themselves, but only once
    - ▶ Q, K V mappings make it possible in many different ways.
    - ▶ They are learned; the process becomes opaque.

(Alammar 2018)
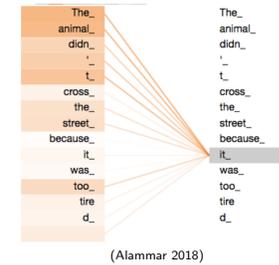
Self-Attention Mechanism in Transformers

- ▶ **Calculating attention scores:**
  - ▶ Take the dot product of Q and K, measuring relevance of each nearby word to the current word.
  - ▶ Scale by the square root of the dimensionality of K.
  - ▶ Take a softmax to scale between 0..1.
  - ▶ Interpret the result as weights determining how much each token j influences the transformed representation for token i

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

(Alammar 2018)

Self-Attention Mechanism in Transformers

- ▶ **Calculating the transformation:**
  - ▶ Form an AttentionScore-weighted sum of value vectors V_j.
  - ▶ The result is an attention-based representation for token i.
  - ▶ The collection of these representations for all tokens j is the output of the attention head.
  - ▶ It is a transformation of the token embeddings based on their relationships with other tokens, i.e. attention.

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

(Alammar 2018)
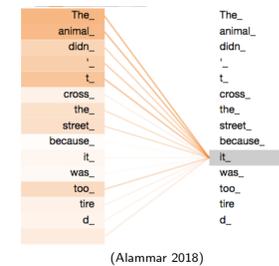
Self-Attention Mechanism in Transformers

- ▶ **Multihead attention:**
  - ▶ Multiple attention heads calculate different transformations in parallel, attending to different aspects (e.g. verb/subject, semantics, style, etc.)
  - ▶ Their outputs are combined in a feed-forward layer.
  - ▶ The output is passed on as input to the next layer
  - ▶ Many such transformations are stacked, forming more complex representations.


(Vaswani et al. 2017)
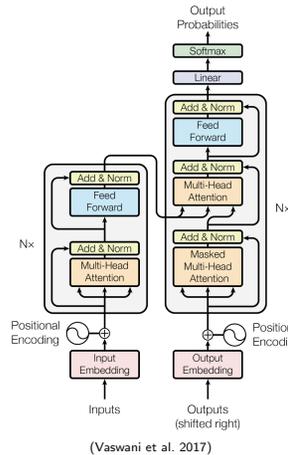
Transformer Encoder-Decoder Structure

- ▶ **Encoder-Decoder Structure:**
  - ▶ The encoder processes the input sequence into an internal representation.
  - ▶ The decoder generates an output sequence, based on input so far and the encoder's representation.
  - ▶ Depending on the task, only the encoder, only the decoder, or both may be needed.


(Vaswani et al. 2017)
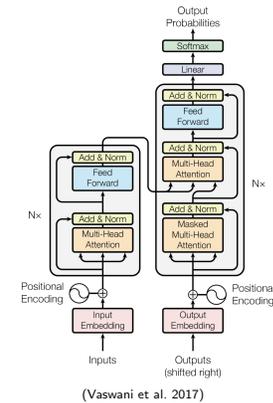
Making it Work

- ▶ **Positional Encoding**
  - ▶ Sequential structure is encoded into initial embeddings.
- ▶ **Gradient descent:**
  - ▶ Matrices W^Q, W^K, W^V, feedforward parameters.
  - ▶ The resulting representations are opaque and hard to understand.
- ▶ **Layer Normalization and Residual Connections:**
  - ▶ Layer normalization stabilizes and speeds up training.
  - ▶ Residual connections help prevent the vanishing gradient problem.

(Vaswani et al. 2017)
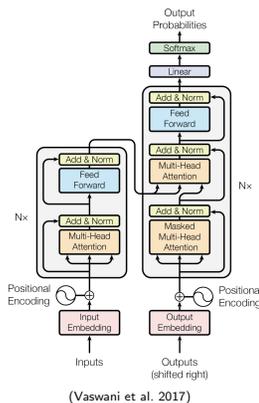
Example of an Evolved LLM Model Merging

- ▶ LLMs are large trainsformer models trained with language
- ▶ With 70B weights, difficult to train or evolve
- ▶ But can evolve an optimal way of combining existing models
  - ▶ Recombine layers, or recombine weights
- ▶ E.g. a model fine-tuned for Japanese and another for math
- ▶ The merged model can do both

(Akiba et al. 2025)
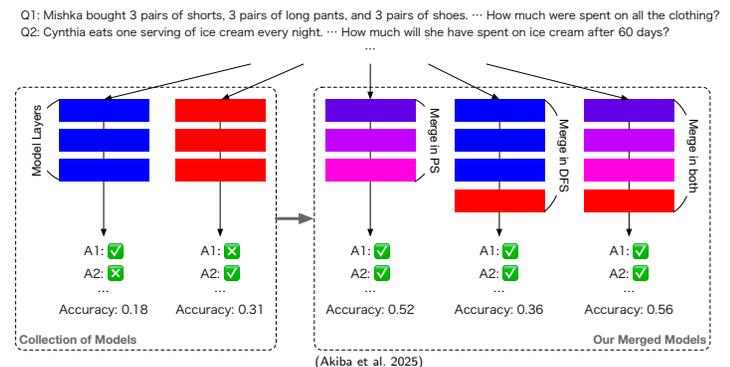
Conclusion

- ▶ Neural networks include several designs in size and complexity
- ▶ They make many difficult applications possible
- ▶ The common training method of gradient descent only works in known domains
  - ▶ Where the optimal outputs are known
- ▶ Neuroevolution allows training neural networks with only fitness information
- ▶ Can also be combined with gradient descent to find optimal designs
- ▶ Makes impactful applications possible that otherwise would not be.

(Wasserman 1909; Wasserman 1909; Rawal & Miikkulainen 2020; Real et al. 2018; Vaswani et al. 2017)