Generative Pretrained Transformer

Evolution and its Future

Jaideep Ganguly, Sc.D. (MIT)

June 5, 2023

Linear Regression

1. Regression is a statistical approach to find the relationship between variables between X_i and Y_i.
2. A common function used to model training data is a linear regression model. The model or the hypothesis is given by:

$$y_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} \cdots + \theta_n x_{in} \tag{1}$$

where x_{ij} is the observed value of the feature x_j, y_i is the predicted value of the outcome and θ_i are constants the values of which need to be determined. For now, we limit the the values of the features x_{ij} to numbers, positive or negative. Later on, we will study techniques on how to deal with the situation where the feature value is a string.

$$\text{Squared Error Loss (L)} = \frac{1}{2} \times \sum_{i=1}^{r} (\hat{y}_i - y_i)^2 \tag{2}$$

This loss function is always positive, there is a minimum, is monotonically increasing from that minimum value in both positive and negative directions. Such a function is called a convex function.
3. We need to minimize some loss function over the training data.

Gradient Descent

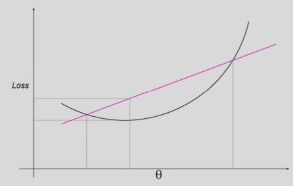


Figure: Convex Function

1. Minimize by setting the partial derivative of the loss wrt θ_j to zero.

$$\frac{\partial L}{\partial \theta_0} = \sum_{i=1}^{r} (\hat{y}_i - y_i) = 0 \quad \frac{\partial L}{\partial \theta_j} = \sum_{i=1}^{r} (\hat{y}_i - y_i)\, x_{ij} = 0 \tag{3}$$

2. Randomly assigning values to θ_j. For a convex function it does not matter what the initial weights are as it will always converge. $\hat{y}$ and x_{ij} are observed and y_i is computed. This means the slope of the loss function can be readily computed.

Gradient Descent

Now for a given θ_j, and keeping all other θ_i where $i \neq j$ constant, we change the value of θ_j slightly and compute a new value of θ_j

$$\triangle\theta_j = \alpha\theta_j$$
$$\theta_j^{'} = \theta_j + \triangle\theta_j$$

where α is small constant and is called the learning rate. Since the slope of $\frac{\partial L}{\partial \theta_j}$ is already calculated from the previous equations, we can compute the new value of the loss as follows:

$$L^{'} = L - \frac{\partial L}{\partial \theta_j}(\alpha\theta_j)$$

If $L^{'} < L$, we continue incrementing θ_j as long as the loss decreases. When the direction changes and the loss increases, we have found the θ_j for which the loss is minimum. If $L^{'} > L$, we travel in the reverse direction and follow the same process. Like this, we compute the value of each and every θ_j.

SoftMax

1. In the training data the values corresponding to the features x_j can have numerical values of different magnitudes. For example, one feature may have values ranging between $0 \ldots 10$, while another feature may have values ranging between $10,000 \cdots 100,000$. Some of these values can also be negative and the components will obviously not add up to 1.
2. The softmax function is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities in the interval 0...1 with the components adding up to 1.

$$\sigma(z)_i = \frac{e_i^z}{\sum_{j=1}^{K} e^{z_j}} \tag{4}$$

The feature values are normalized with the softmax function prior to inputing in a regression model.

One Hot Encoding

1. In the linear regression and the logistic regression models that we studied so far can only deal with numerical data. It cannot handle features that have text as values. In one-hot encoding, the string encoded variable is replaced with new variables of boolean type. For example, a particular feature, say color, can have "red", "green" and "blue" as possible values. This feature color, will be replaced by 3 features, red, blue and green which hold the values 1 or 0. We can then encode color, in terms of red, blue and green as follows:

red	green	blue
1	0	0
0	1	0
0	0	1

Note that if the boolean value of the feature is 1, it must be 0 for all the other features

Information & Uncertainty

1. In Clause Shannon's information theory, one bit of information reduces the uncertainty by 2. Similarly, if 3 bits of information are sent, then the reduction in uncertainly by 2^3, i.e., 8. This is intuitive. With 3 bits, there could be 8 possible values and so if a particular set of bits are transmitted, 8 possibilities are eliminated with 1 certainty.
2. Information Content When the information is probabilistic, the self-information I_x, or Information Content of *measuring a random variable* X *as outcome* x *is defined as:*

$$I_x = log\left(\frac{1}{p(x)}\right) = -log(p(x)) \tag{5}$$

where $p(x)$ is probability mass function.

3. Shannon Entropy of the random variable X is defined as:

$$H(X) = \sum_{x} -p(x)log(p(x)) = E(I_x) \tag{6}$$

It is the *expected information content* of the measurement of X.

Cross Entropy - Kullback–Leibler (KL) divegence

1. Cross-Entropy is defined as:

$$H(p, q) = -\sum_i p(i) log_2 q(i) \tag{7}$$

where p is the true distribution and q is the predicted distribution. If the predictions are perfect, then the cross-entropy is same as the entropy. If the prediction differs, then there is a divergence which is known as *Kullback–Leibler (KL) divergence*. Hence,

$$Cross\ Entropy = Entropy + KL\ Divergence$$

or,

$$KL\ Divergence = H(p, q) - H(p)$$

Hence, if the predicted distribution is closer to true distribution when KL divergence is low.

Neural Nets

Neural Net, RNN, LSTM

Deep Neural Net

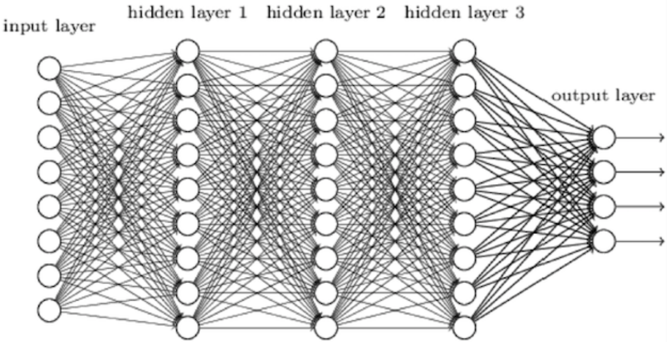


Figure: Deep Learning

Deep Neural Net

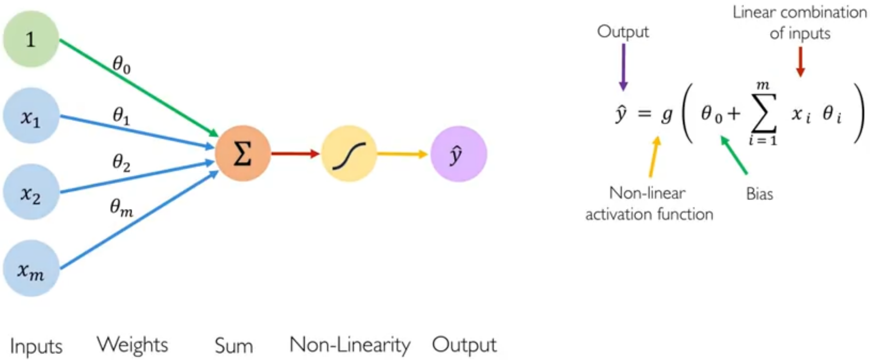
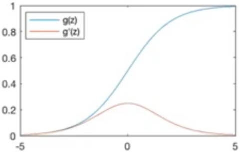


Figure: Forward Propagation

Source: MIT 6.S191 (2018) - Introduction to Deep Learning

Deep Neural Net

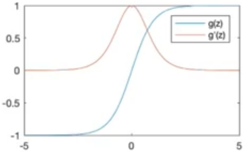
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

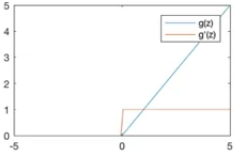
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

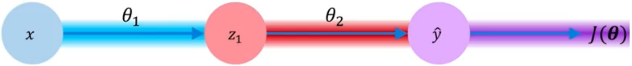


$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Figure: Activation Functions

Back Propagation



$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial \theta_1}$$

Figure: Loss Minimization

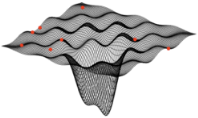


Figure: Global versus Local Minima

Repeat this for every weight in the network using gradients from later layers.

RNN

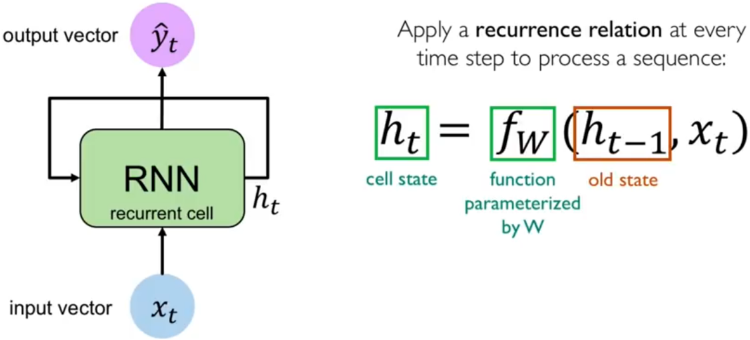
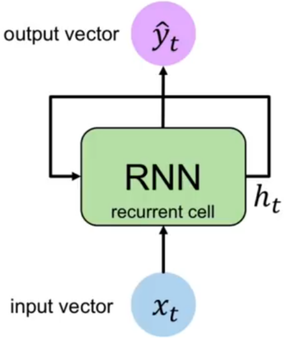


Figure: Same function and set of parameters are used in every step

Source - MIT 6.S191- Recurrent Neural Networks

RNN



Output Vector

$$\hat{y}_t = \boldsymbol{W_{hy}} h_t$$

Update Hidden State

$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

Input Vector

Figure: State Update and Output

RNN

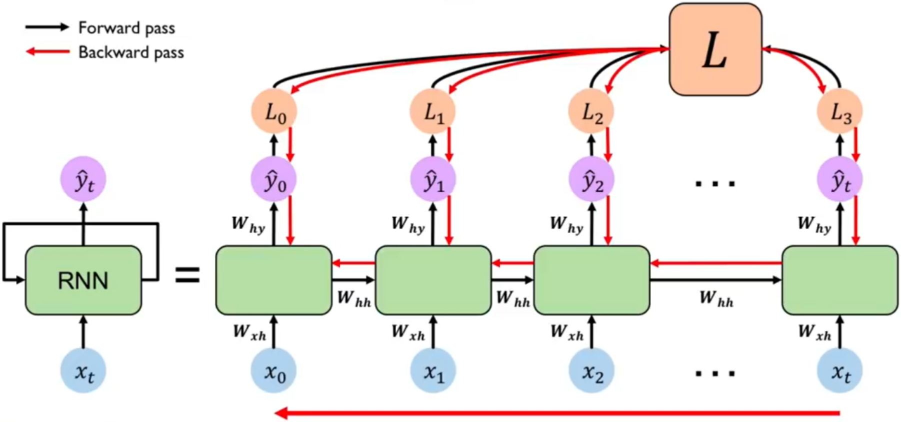
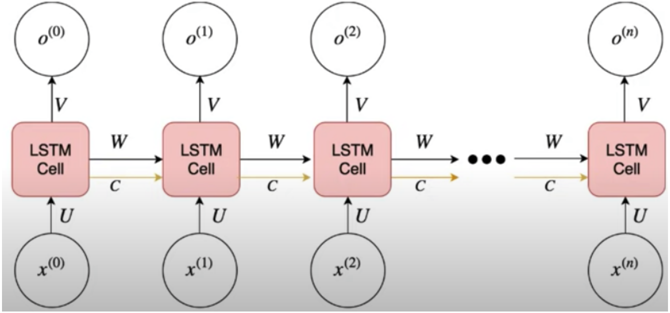


Figure: Backpropagation over time

Vanishing / Exploding Gradients

1. Learning with recurrent networks is challenging due to the difficulty of learning long-range dependencies, [Bengio et al. 1994], [Hochreiter et al, 2001].
2. Problems of vanishing and exploding gradients occur when backpropagating errors across many time steps.
3. Which of the two phenomena occurs depends on whether the weight of the recurrent edge $|wjj| > 1$ or $|wjj| < 1$ and on the activation function in the hidden node. For a sigmoid activation function, the vanishing gradient problem is more pressing, but with a rectified linear unit $max(0, x)$, it is easier to imagine the exploding gradient.

Long Short Term Memory (LSTM)



Maintain separate cell state to avoid vanishing gradients by allowing the network to selectively remember or forget information over long periods of time by incorporating a set of gates that control the flow of information through the network.

$$i_t = sigmoid(W_i[x_t, h_{t-1}] + b_i)$$

$$f_t = sigmoid(W_f[x_t, h_{t-1}] + b_f)$$

$$C_t = f_t * C_{t-1} + i_t * tanh(W_C[x_t, h_{t-1}] + b_C)$$

$$o_t = sigmoid(W_o[x_t, h_{t-1}] + b_o)$$

Attention & Masked Attention

Generalised Pretrained Transformer (GPT) - Dec 2017

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

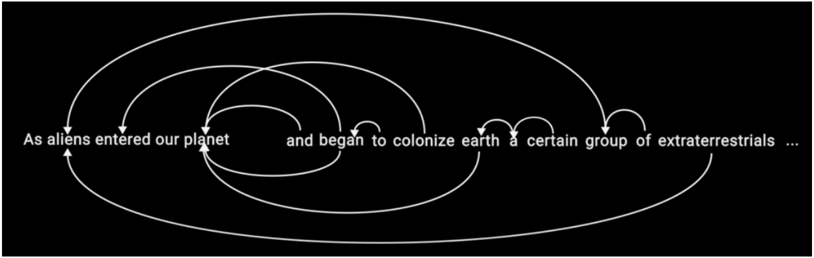
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

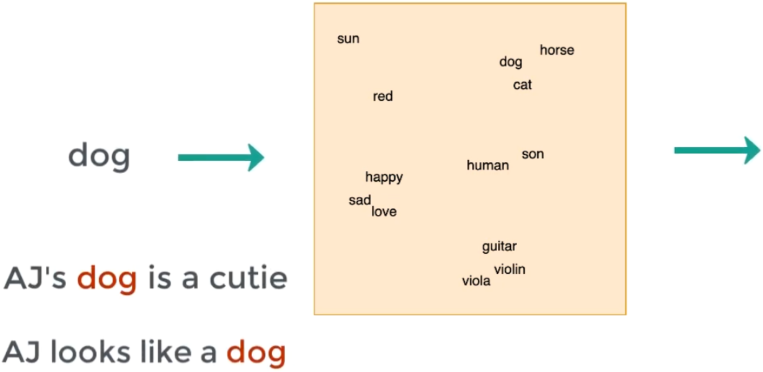
175 Billion parameters and counting!

Attention



1. Attention mechanism has an infinite reference window
2. In contrast, RNN has a short reference window, LSTM has a longer window.

Input Embedding



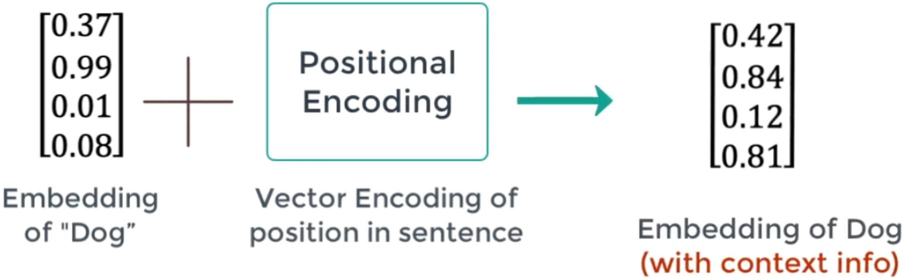
$$\begin{bmatrix} 0.37 \\ 0.99 \\ 0.01 \\ 0.08 \end{bmatrix}$$

Positional Encoding

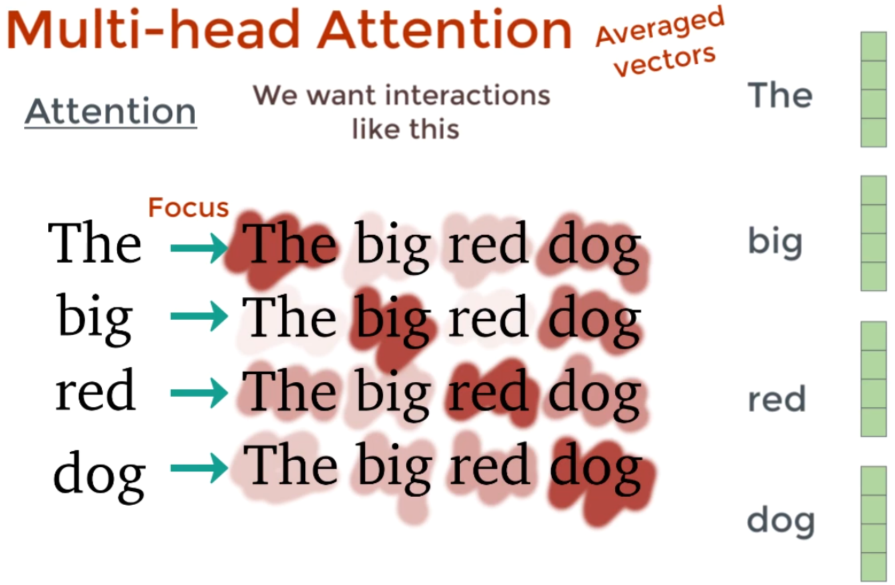
AJ's dog is a cutie ⟶ Position 2

AJ looks like a dog ⟶ Position 5

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



Multi Head Attention

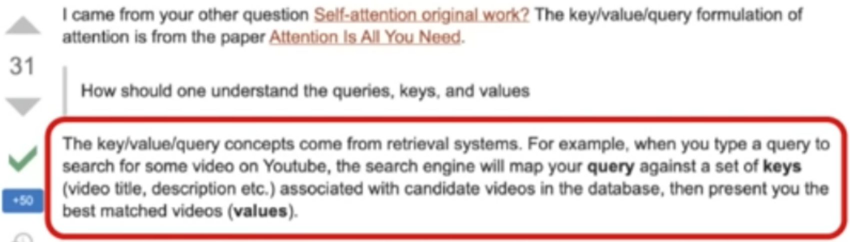


Attention: Query, Key, Value

Concept of:

1. Query (Q)
2. Key (K)
3. Value (V)

All constructed from the embedding.



I came from your other question Self-attention original work? The key/value/query formulation of attention is from the paper Attention Is All You Need.

> How should one understand the queries, keys, and values

The key/value/query concepts come from retrieval systems. For example, when you type a query to search for some video on Youtube, the search engine will map your **query** against a set of **keys** (video title, description etc.) associated with candidate videos in the database, then present you the best matched videos (**values**).

Mimic the retrieval of a value v_i for a query q based on a key k_i in DB.

$$attention(q, k, v) = \sum_i similarity(q, k_i) \times v_i \text{ (weighted)}$$

Attention: Q, K, V and weights Q_w, K_w, V_w

Weights are initialised randomly using a random distribution. Example:

input #1	input #2	input #3
1 0 1 0	0 2 0 2	1 1 1 1

Because every input has a dimension of 4, each set of the weights must have a shape of 4×3.
Weights are initialised randomly, it is done once before training.

Weights for **key**

```
[[0, 0, 1],
 [1, 1, 0],
 [0, 1, 0],
 [1, 1, 0]]
```

Weights for **query**

```
[[1, 0, 1],
 [1, 0, 0],
 [0, 0, 1],
 [0, 1, 1]]
```

Weights for **value**

```
[[0, 2, 0],
 [0, 3, 0],
 [1, 0, 3],
 [1, 1, 0]]
```

Key representation for input 1:

```
              [0, 0, 1]
[1, 0, 1, 0] x [1, 1, 0] = [0, 1, 1]
              [0, 1, 0]
              [1, 1, 0]
```

Key representation for input 2:

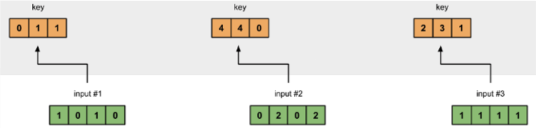
```
              [0, 0, 1]
[0, 2, 0, 2] x [1, 1, 0] = [4, 4, 0]
              [0, 1, 0]
              [1, 1, 0]
```

Key representation for input 3:

```
              [0, 0, 1]
[1, 1, 1, 1] x [1, 1, 0] = [2, 3, 1]
              [0, 1, 0]
              [1, 1, 0]
```

Key representation:
(Vectorise)

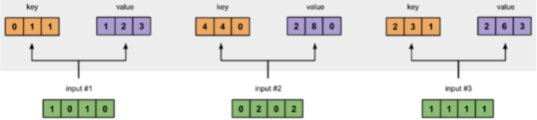
```
               [0, 0, 1]
[1, 0, 1, 0]   [1, 1, 0]   [0, 1, 1]
[0, 2, 0, 2] x [0, 1, 0] = [4, 4, 0]
[1, 1, 1, 1]   [1, 1, 0]   [2, 3, 1]
```



Attention: Q, K, V and weights Q_w, K_w, V_w

Value representation:

```
                [0, 2, 0]
[1, 0, 1, 0]    [0, 3, 0]    [1, 2, 3]
[0, 2, 0, 2] x  [1, 0, 3] =  [2, 8, 0]
[1, 1, 1, 1]    [1, 1, 0]    [2, 6, 3]
```



Query representation:

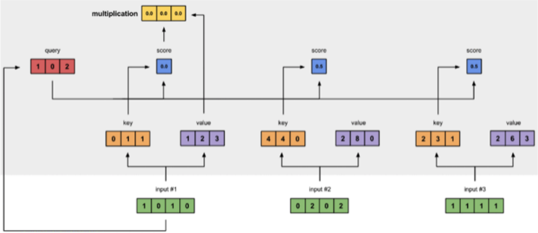
```
                [1, 0, 1]
[1, 0, 1, 0]    [1, 0, 0]    [1, 0, 2]
[0, 2, 0, 2] x  [0, 0, 1] =  [2, 2, 2]
[1, 1, 1, 1]    [0, 1, 1]    [2, 1, 3]
```

Attention: Q, K, V and weights Q_w, K_w, V_w

Attention scores - dot product between Input 1's query (red) with all keys (orange), including itself

```
             [0, 4, 2]
[1, 0, 2] x [1, 4, 3] = [2, 4, 4]
             [1, 0, 1]

softmax([2, 4, 4]) = [0.0, 0.5, 0.5]
```



Multiply Softmax attention scores for each input (blue) by its corresponding value (purple).

```
1: 0.0 * [1, 2, 3] = [0.0, 0.0, 0.0]
2: 0.5 * [2, 8, 0] = [1.0, 4.0, 0.0]
3: 0.5 * [2, 6, 3] = [1.0, 3.0, 1.5]
```

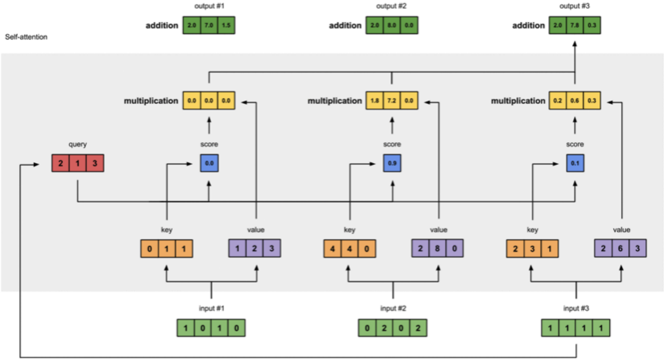
Attention: Q, K, V and weights Q_w, K_w, V_w

Weights are initialised randomly using a random distribution. Example:

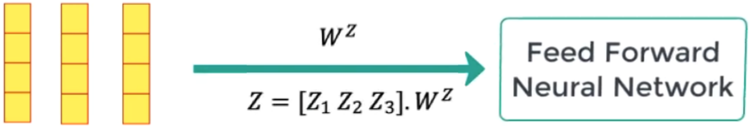
Sum weighted values to get Output 1

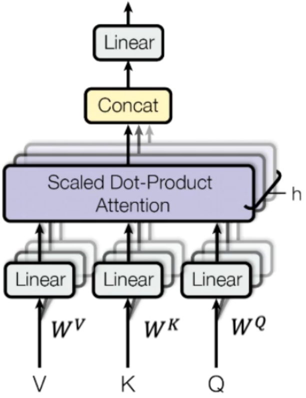
```
  [0.0, 0.0, 0.0]
+ [1.0, 4.0, 0.0]
+ [1.0, 3.0, 1.5]
----------------
= [2.0, 7.0, 1.5]
```

Repeat for Input 2 & Input 3



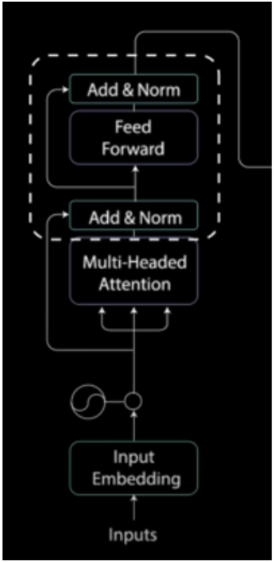
Multi Head Attention

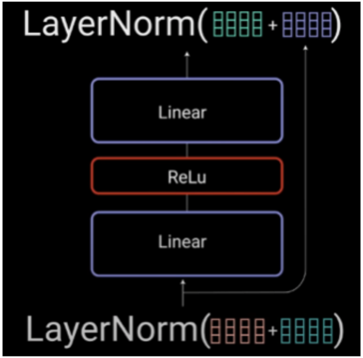




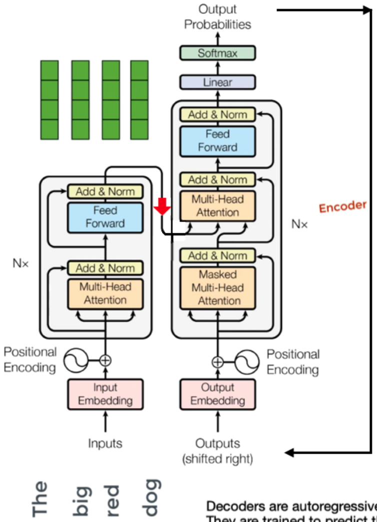
$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimension\ of\ vector\ Q, K\ or\ V}}\right).V$$

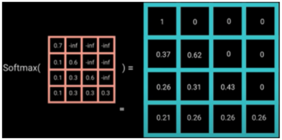
Add & Normalization





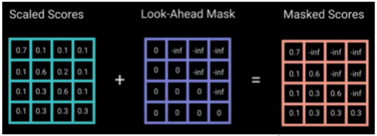
Decoder, Masked Attention





0.7	-inf	-inf	-inf
0.1	0.6	-inf	-inf
0.1	0.3	0.6	-inf
0.1	0.3	0.3	0.3

1	0	0	0
0.37	0.62	0	0
0.26	0.31	0.43	0
0.21	0.26	0.26	0.26



Scaled Scores

0.7	0.1	0.1	0.1
0.1	0.6	0.2	0.1
0.1	0.3	0.6	0.1
0.1	0.3	0.3	0.3

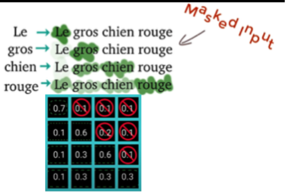
Look-Ahead Mask

0	-inf	-inf	-inf
0	0	-inf	-inf
0	0	0	-inf
0	0	0	0

Masked Scores

0.7	-inf	-inf	-inf
0.1	0.6	-inf	-inf
0.1	0.3	0.6	-inf
0.1	0.3	0.3	0.3

Decoder



0.7	0.1	0.1	0.1
0.1	0.6	0.2	0.1
0.1	0.3	0.6	0.1
0.1	0.3	0.3	0.3

Decoders are autoregressive models;
They are trained to predict the next token
after reading the preceding ones

Questions?

Thank You!