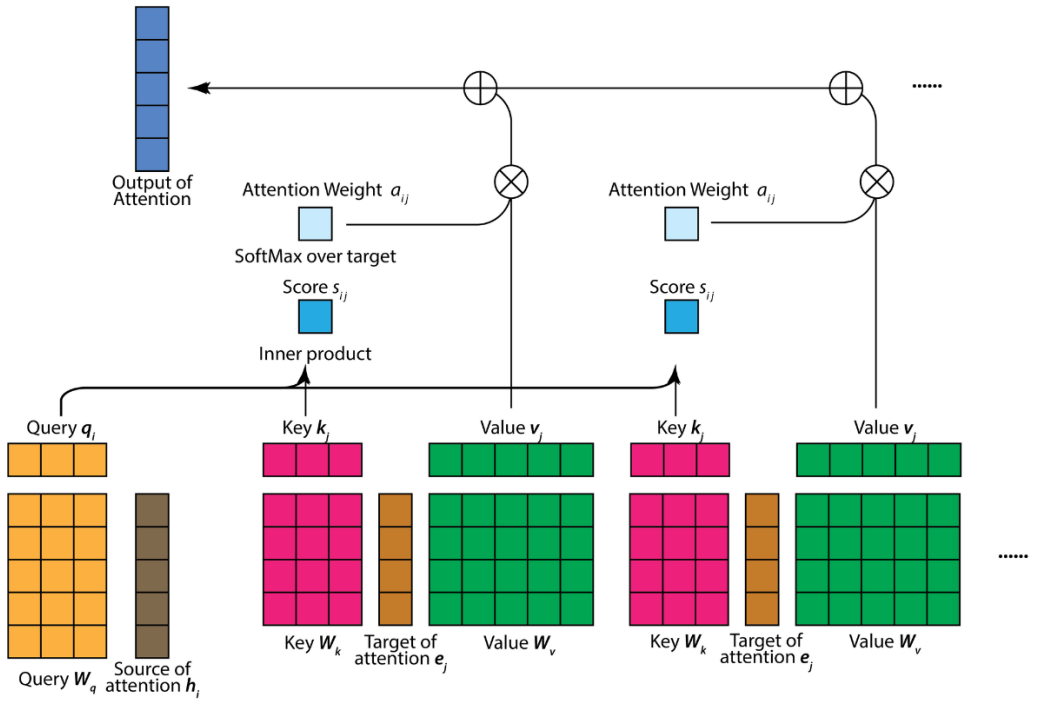
Transformer ဆိုတာဘာလဲ?

AI Field ထဲက neural network architecture နဲ့တည်ဆောက်ထားတဲ့ NLP မှာအသုံးများတဲ့ကောင်ကို ပြောပါဆိုရင် Transformer လို့ပြောလို့ရပါတယ်။ ၂၀၁၇ လောက်မှာ “Attention is All you Need” ဆိုတဲ့ seminal paper တစ်ခုနဲ့ အတူ Transformer ပေါ်ထွက်လာခဲ့ပါတယ်။ သူ့ကို Open AI ရဲ့ GPT, Meta ရဲ့ Llama နဲ့ Google ရဲ့ Gemini တို့မှာ deep learning models တစ်ခုအနေနဲ့သုံးနေခဲ့ကြတာပါ။

Transformer ကို အသုံးပြုပြီးတော့ audio generation တွေ image recognition တွေ၊ Protein structure prediction တွေနဲ့ game playing ပိုင်းတွေ ဖန်တီးလို့ရပါတယ်။ အခြေခံအားဖြင့် text-generative model (Transformer) ရဲ့ လုပ်ဆောင်ချက်က ရိုးရှင်းပါတယ်။ သူက user တွေရဲ့ပြောချင်တဲ့ words တွေ sentence တွေကို input အနေနဲ့ လက်ခံပြီး output အနေနဲ့ နောက်ထပ် ထပ်ထည့်လို့ရမယ့် meaningful words တွေ next-word prediction လုပ်ပေးတာပဲဖြစ်ပါတယ်။ သူ့ရဲ့ ဆန်းသစ်တဲ့ အလုပ်လုပ်ပုံတစ်ခုကတော့ sequence အလိုက်ဝင်လာတဲ့ sentences တွေ verb တွေ phrase တွေ process ကောင်းကောင်းလုပ်နိုင်ပြီး ရှည်လျားတဲ့ sentences တွေကိုလည်း ကောင်းကောင်း capture လုပ်ပေးနိုင်တဲ့ self-attention mechanism နဲ့ အလုပ်လုပ်နေတာဖြစ်ပါတယ်။

GPT-2 model က text-generative Transformers ကိုကြည့်လို့ရတဲ့ example ပါပဲ။ အဲ့ model မှာ parameters အရေအတွက် 124 million ရှိပါတယ်။



Transformer Architecture

Transformer အသုံးပြုပြီးဖန်တီးထားတဲ့ text-generative model တွေမှာ အပိုင်းသုံးပိုင်းကိုတွေ့ရပါတယ်။

1. Embedding : ဒီအပိုင်းကတော့ model ထဲကို input အနေနဲ့ sequence အလိုက်ဝင်လာကြတဲ့ sentences တွေကို သေးငယ်တဲ့ units တွေပုံစံဖြစ်အောင် token အနေနဲ့ပြန်ခွဲလိုက်တာပါ။ အဲ့လို token တွေက words တွေလည်းဖြစ်နိုင်သလို subwords တွေလည်းဖြစ်နိုင်ပါတယ်။ ခွဲထုတ်လိုပြီးသွားတဲ့ tokens တွေကို numerical vectors တွေအဖြစ် ပြောင်းလဲလိုက်တာပါ။ ပြီးရင် စာလုံးတွေရဲ့ ဝေါဟာရ ၊ ဆိုလိုရင် အဓိပ္ပါယ်တို့ကို capture လုပ်လိုက်ပါတယ်။
2. Transformer Block : model အတွက်လိုအပ်တဲ့ input data တွေကို စီမံပေးတဲ့ block တွေလိုပြောလို့ရပါတယ်။
    a. Attention mechanism: Transformer Block အတွက် အရေးကြီးတဲ့ component တစ်ခုပဲဖြစ်ပါတယ်။ ဒီ mechanism က tokens တွေ အချင်းချင်း communicate လုပ်နိုင်အောင်ရယ်၊ contextual အချက်အလက်တွေကို capture လုပ်ဖို့နဲ့ words တွေမှာရှိတဲ့ relationship ကို ဖော်ဆောင်ဖို့ပဲဖြစ်ပါတယ်။
    b. MLP (Multilayer Perceptron) Layer: ဒီ layer ကို neural network (deep learning layer) လို့သတ်မှတ်ပါတယ်။ ဒီ network layer ထဲကို ကျွန်တော်တို့ ထည့်သွင်းလိုက်တဲ့ input တွေကို independently operate လုပ်ပေးပါတယ်။ အဲ့တော့ attention layer က ရလာတဲ့ tokens တွေကြားထဲက route ကို MLP က tokens တွေကို representation လုပ်နိုင်ဖို့အတွက် ပြန်လည် refine လုပ်ပေးပါတယ်။
3. Output Probabilities: နောက်ဆုံး layer ပါ။ linear and SoftMax layers လို့သတ်မှတ်ပါတယ်။ အပေါ်မှာ embeddings လုပ်ပြီးသား ၊ representation လုပ်ပြီးသား sequence ထဲက Tokens တွေကို probabilities တွက်ချက်ပြီး next token (words or phrases) တွေကဘာဖြစ်နိုင်မလဲဆိုတာ predictions လုပ်ပေးတာပဲဖြစ်ပါတယ်။

Embedding

ကဲ ကျွန်တော်တို့ Transformer Model တစ်ခုကို အသုံးပြုပြီး text generate လုပ်ချင်တယ်ဆိုပါစို့။ "Data visualization empowers users to" ဆိုတဲ့ input လေးထည့်လိုက်မယ်။ ပထမဆုံးအနေနဲ့ ဒီ sentences ကို model က နားလည်အောင်လုပ်ဖို့လိုတယ်၊ အဲ့အချိန်မှာ embedding က ဝင်လာရော၊ သူက text တွေကို numerical representations အဖြစ်ပြောင်းလိုက်တယ်။

အဆင့် ၁ - Tokenize the input

အဆင့် ၂ - obtain token embeddings

အဆင့် ၃ - add positional information

အဆင့် ၄ - add up token and position encodings to get the final embedding

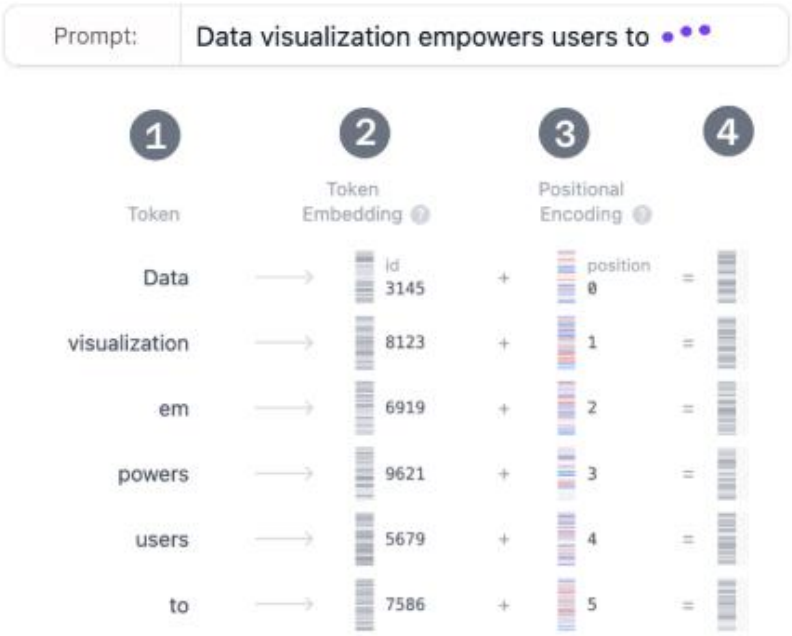


Figure 1. Expanding the Embedding layer view, showing how the input prompt is converted to a vector representation. The process involves (1) Tokenization, (2) Token Embedding, (3) Positional Encoding, and (4) Final Embedding.

Step 1: Tokenization

အပေါ်မှာပြောခဲ့သလိုပဲ Tokenization လုပ်တယ်ဆိုတာ input text ကို သေးငယ်တဲ့အစိတ်အပိုင်း tokens တွေဖြစ်အောင်ခွဲချလိုက်တာပါ။ ဒီလို vocabulary တွေကို ကျွန်တော်တို့ model မ train ခင်ကတည်းက Unique ID အတွေနဲ့ သတ်မှတ်ပြီးသားပါ။ ဥပမာ Data = ID 3145 ဒါမျိုးပေါ့ ၊ GPT2 မှာဆိုရင် unique tokens အရေအတွက် 50,257 တောင် ရှိပါတယ်။ အာ့လိုခွဲထားတဲ့ အတွက် tokens ID တွေကနေတစ်ဆင့် Tokens တစ်ခုချင်းစီရင် vector တွေရရှိလာပါပြီ။

Step 2: Token Embedding

GPT-2 (Small) Model မှာ Token တွေကို 768-dimensional vector တွေအဖြစ်သတ်မှတ်ထားပါတယ်။ အာ့ vector တွေရဲ့ dimensions တွေဟာ model ပေါ်မှာမူတည်ပါတယ်။ အာ့ embed လုပ်ထားတဲ့ vectors တွေကို (50, 257, 768) matrix shape ထဲမှာ store လုပ်ထားတာပါ။ ခန့်မှန်းခြေ parameters ပေါင်း 39 million ရှိသတဲ့။ ဒီ matrix တွေက model အတွက် လိုအပ်တဲ့ ဝေါဟာရ အဓိပ္ပါယ်တွေကို assign လုပ်ပေးနိုင်ပါတယ်။

Step 3: Positional Encoding

ဒီ Embedding Layer က input prompt ကနေ တစ်ဆင့် ဝင်ရောက်လာတဲ့ tokens ရဲ့ position တွေကိုလည်း encode လုပ်ပေးပါသေးတယ်။ ပေါ်ကပုံမှာကြည့်ပါ။ example အနေနဲ့ GPT-2 ဆိုရင် သူ့ရဲ့ ကိုယ်ပိုင် positional encoding matrix နဲ့ အစအဆုံးဖန်တီးထားပြီး training process ပြုလုပ်ထားတာဖြစ်ပါတယ်။ positional encode လုပ်တဲ့စတိုင်ကတော့ model တစ်ခုနဲ့ တစ်ခု မတူကြပါဘူး။

Step 4: Final Embedding

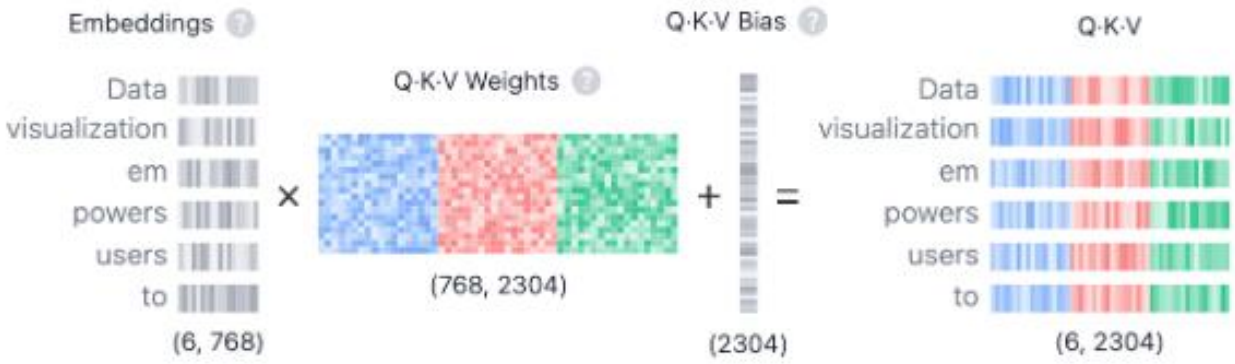
နောက်ဆုံးမှာတော့ token တွေရဲ့ vector တန်ဖိုးတွေနဲ့ positional encoding လုပ်ထားတဲ့ တန်ဖိုးတွေကို ပေါင်းပြီး final embedding representation ရအောင်တွက်ပါတယ်။ အာ့နှစ်ခုပေါင်းလိုက်တဲ့ အတွက် input sequences တစ်ခုရဲ့ token တွေရဲ့ meaning တွေနဲ့ position တွေကို capture လုပ်ပြီးသားဖြစ်သွားပါပြီ။

Transformer Block

Model အများစုမှာ blocks တွေကို sequence အလိုက်တစ်ခုနဲတစ်ခု ထပ်ပြီး တည်ဆောက်ကြပါတယ်။ Token တွေရဲ့ အဓိပ္ပါယ်တွေကို ဖော်ထုတ်ဖို့အတွက် အပေါ်က final embedding representation တွေကို ဒီ layer တွေနဲ့ ဖော်ထုတ်ရပါတယ်။ Self-attention နဲ့ Multi-layer perceptron layer နှစ်ခုပေါင်းစပ်ထားတဲ့ Transformer block ရဲ့ အလုပ်က tokens တွေရဲ့ ဆိုလိုရင်း အဓိပ္ပါယ်တွေကို model က အနုစိတ် နားလည်အောင် တည်ဆောက်ပေးရတာပဲဖြစ်ပါတယ်။ ဒီ layer blocks တွေက input တွေအတွက် higher-order representations ပုံစံနဲ့ အလုပ်လုပ်နေတာပါ။ GPT-2 (Small) မှာဆိုရင် Block ပေါင်း ၁၂ ခုရှိပါတယ်။

Multi-Head Self-Attention

Input sequence တွေမှာပါဝင်တဲ့ ရှုပ်ထွေးနေတဲ့ relationship တွေ dependencies တွေထဲက Token တွေမှာ ဘယ် Tokens ကို focus ထားရမလဲဆိုတာ ဆုံးဖြတ်ပေးတာပါ။ အဲ့ self-attention ကိုယ်လို တွက်သလဲဆိုရင် အောက်က ပုံမှာရှိုးကြည့်ပါ။



$$QKV_{ij} = (\sum_{d=1}^{768} \text{Embedding}_{i,d} \cdot \text{Weights}_{d,j}) + \text{Bias}_j$$

Figure 2. Computing Query, Key, and Value matrices from the original embedding.

Step 1: Query, Key, and Value Matrices

ဘယ်လို Tokens ကို focus ထားပြီး attention လုပ်ရမလဲဆိုတဲ့အခြေအနေကို တွက်ဖို့အတွက် ပထမဆုံး embedding vector ကို vector ၃ ခုခွဲထုတ်လိုက်ရပါတယ်။ Query (Q), Key (K) , Value(V) တို့ပဲဖြစ်ပါတယ်။ ဒီ Vector ၃ ခု ရဖို့အတွက် input ဝင်လာတဲ့ embedding matrix ရယ် ၊ model က train ပြီးသား weight matrices ရယ် multiplying လုပ်လိုက်ရပါတယ်။ ဒါက Linear Equation (y = mx+b) ပါပဲ။

- Query (Q): query လုပ်တယ်ဆိုတာက input ဝင်လာတဲ့ tokens တွေထဲမှာ user က ရှာဖွေချင်တဲ့ အပိုင်းကို ဖော်ထုတ်တာပဲဖြစ်ပါတယ်။ token we want to “find more information about”.
- Key (K): သူကတော့ Query ဆွဲလိုက်ပြီးရလာတဲ့ results တွေထဲမှာ ဖြစ်နိုင်တဲ့ Tokens ကို ကိုယ်စားပြုပါတယ်။
- Value (V): actual content ကိုရှာတွေ့ပါပြီ။ query ဆွဲလိုက်တဲ့ term နဲ့ ရရှိလာတဲ့ results တွေနဲ့ ပေါင်းစပ်ပြီး သင့်တော်တဲ့ အဖြေကို Value လို့ခေါ်ပါတယ်။

QKV values တွေကို အသုံးပြုပြီး model က attention scores တွေကို တွက်ချက်ပေးပါတယ်။ အဲ့ scores က predictions လုပ်တဲ့ အချိန်မှာ tokens တစ်ခုချင်းစီက ဘယ်လောက်ထိ focus ရလဲဆိုတာ တိုင်းတာပေးတာဖြစ်ပါတယ်။

Step 2: Multi-Head Splitting

Query, Key နဲ့ Values တွေဟာ Vectors တွေပါပဲ။ multiple heads လို့တင်စားလို့ရပါတယ်။ GPT-2 (small) model ရဲ့ Transformer Block တစ်ခုမှာ 12 Heads ရှိပါတယ်။ Head တစ်ခုချင်းစီဟာ embeddings တွေကို segment အပိုင်းတွေကို ခွဲထုတ်ပြီး မတူညီတဲ့ ဝေါဟာရ အဓိပ္ပါယ်တွေနဲ့ relationship တွေကို ဖမ်းယူပါတယ်။ ဒီလို ခေါင်းချည်းပဲ ၁၂ လုံးသုံးပြီး parallel learning လုပ်ထားတဲ့ စတိုင်ကြောင့် model ရဲ့ representation power က အတော်ကို မြင့်တက်သွားပါတယ်။ အဲ့တော့ example ခေါင်းသုံးလုံး အလုပ်လုပ်တဲ့ပုံလေးကိုကြည့်ရအောင်။

Step 3: Masked Self-Attention

အာ့တော့ ခေါင်းတစ်လုံးစီမှာ masked self-attention အတွက် တွက်ချက်မှုတွေလုပ်ကြရပါတယ်။ ဒီ နည်းလမ်းက model ကို future token တစ်ခုမထုတ်သေးခင်မှာ သင့်တော်တဲ့ input token ကို focus လုပ်ခိုင်းပြီး sequences တွေကိုအရင်ဆုံး generate လုပ်စေပါတယ်။

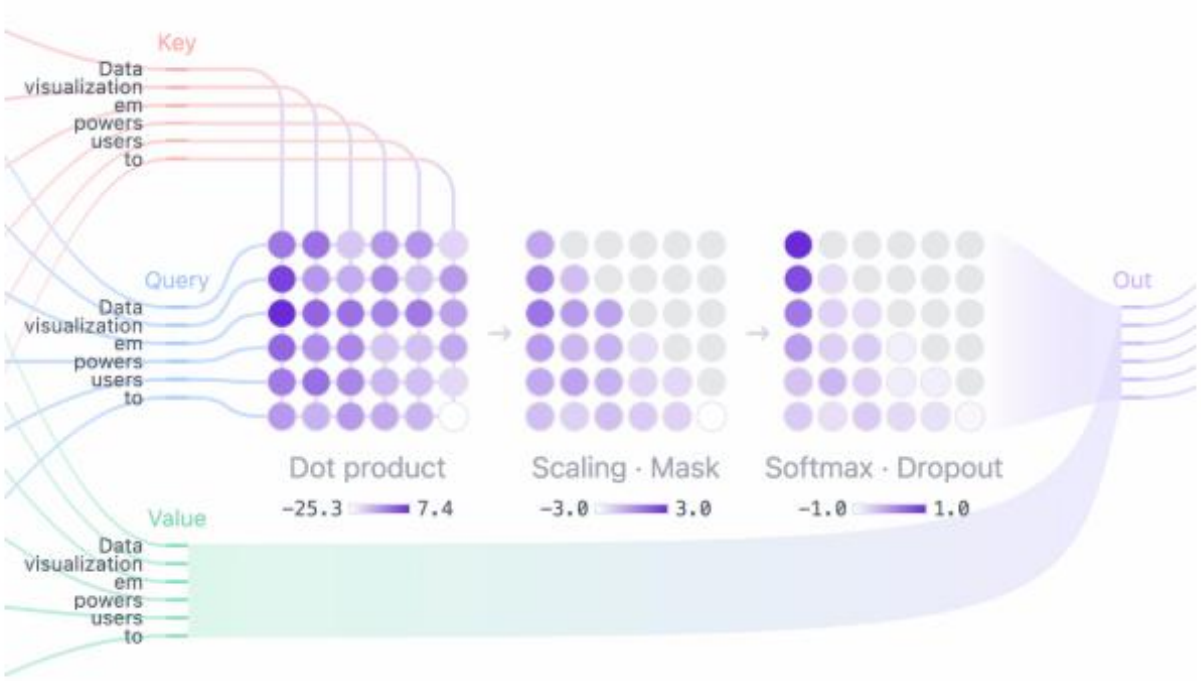


Figure 3. Using Query, Key, and Value matrices to calculate masked self-attention.

Attention Score:

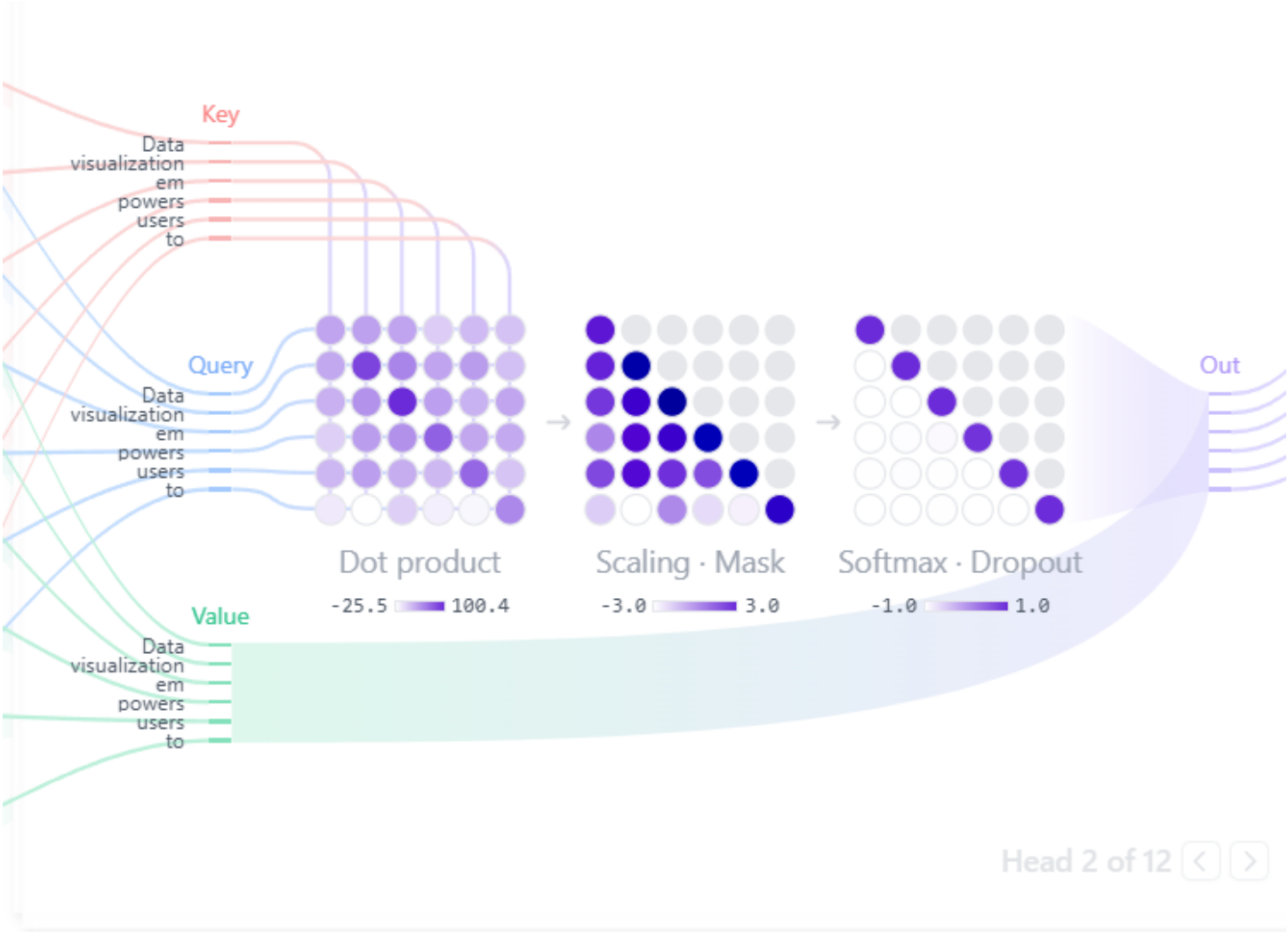
Query matrices နဲ့ Key matrices တို့ dot product လုပ်ပြီး square matrix တစ်ခုထုတ်ပေး လိုက်တာနဲ့ တစ်ပြိုင်တည်း အာ့ square matrix သည် input tokens အားလုံးရဲ့ relationship ကို represent လုပ်ပြီးသားဖြစ်သွားပါတယ်။

Masking:

Mask လုပ်တယ်ဆိုတာ Model ကို ကျွန်တော်တို့လိုချင်တဲ့ predicted (or) future token ကို accessing မလုပ်စေချင်လို့ပါ။ ဘာလို့လဲဆိုတော့ model ကို future token ဘယ်လို predict လုပ်ရမလဲဆိုတာ Learning လုပ်စေချင်လို့ပဲဖြစ်ပါတယ်။ ပုံထဲမှာဆိုရင် upper triangle ပုံစံနဲ့ attention matrix ကို mask ဖုံးအုပ်လိုက်တာကို တွေ့ရမှာပါ။

Softmax:

Mask လုပ်ပြီးရင်တော့ attention score ကို probability တွက်ချင်လို့ သူ့ရဲ့ exponent ကိုယူပြီး softmax operation လုပ်လိုက်ပါတယ်။ matrix ထဲမှာရှိတဲ့ row တစ်ခုချင်းစီပေါင်းပြီး အာ့ကောင်တွေရဲ့ဘယ်ဘက်မှာရှိတဲ့ token တွေရဲ့ relationship ကို ဖော်ထုတ်လိုက်တာပါ။



Step 4: Output and Concatenation

နောက်ဆုံး output ထွက်လာအောင် Softmax ကထွက်လာတဲ့ Masked self-attention scores နဲ့ Value Matrix နှစ်ခုကို မြှောက်လိုက်ပါတယ်။ ဒါကို self-attention mechanism လို့ခေါ်ပါတယ်။ GPT2 မှာ Self-attention heads ပေါင်း ၁၂ ခုနဲ့အလုပ်လုပ်ပါတယ်။ အဲ့ ၁၂ ခုလုံးဟာ တစ်ခုနဲ့တစ်ခု မတူညီတဲ့ လုပ်ဆောင်မှုပုံစံနဲ့ Tokens တွေရဲ့ မတူညီတဲ့ ဆက်သွယ်မှုကို ရှာဖွေဖော်ထုတ်ကြတာပဲဖြစ်ပါတယ်။ သူတို့စီက ရလာတဲ့ အဖြေကို Concatenated လုပ်ပြီး Linear Projection ထဲကို ဖြတ်သန်းစေပါတယ်။

MLP: Multi-layer perceptron

Input tokens တွေရဲ့ကြားထဲက ဆက်သွယ်မှုတွေကို Self-attention process နဲ့ multiple heads တွေက ရှာဖွေဖော်ထုတ်ပြီးနောက်မှာ၊ output တွေကို concatenated လုပ်ပြီးပါပြီ။ အဲ့ output တွေကို Multilayer perceptron (MLP) layer ထဲကို ဖြတ်သန်းပေးစေရမှာပါ။ ဒါမှသာလျှင် model ရဲ့ representational capacity ကို ကောင်းမွန်စေမှာပါ။

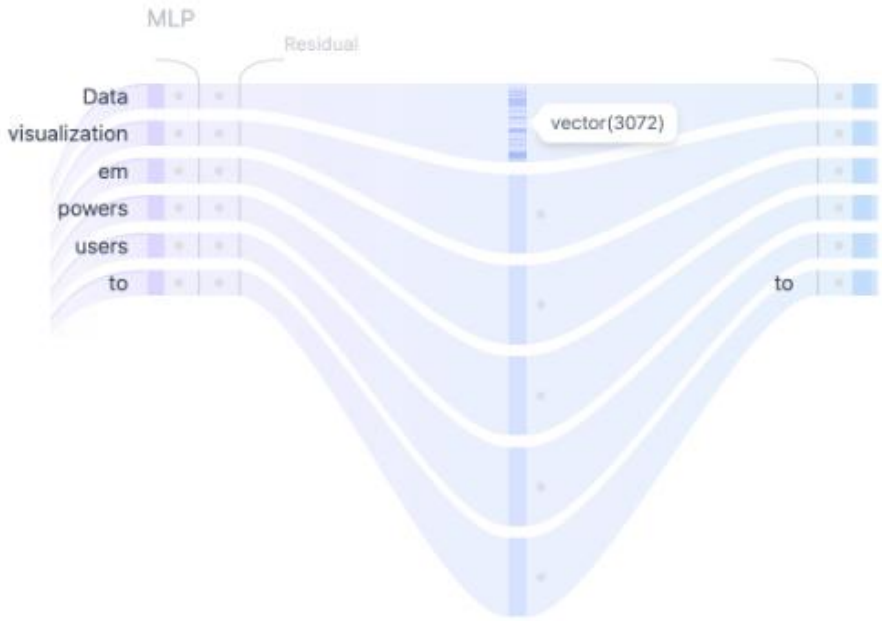
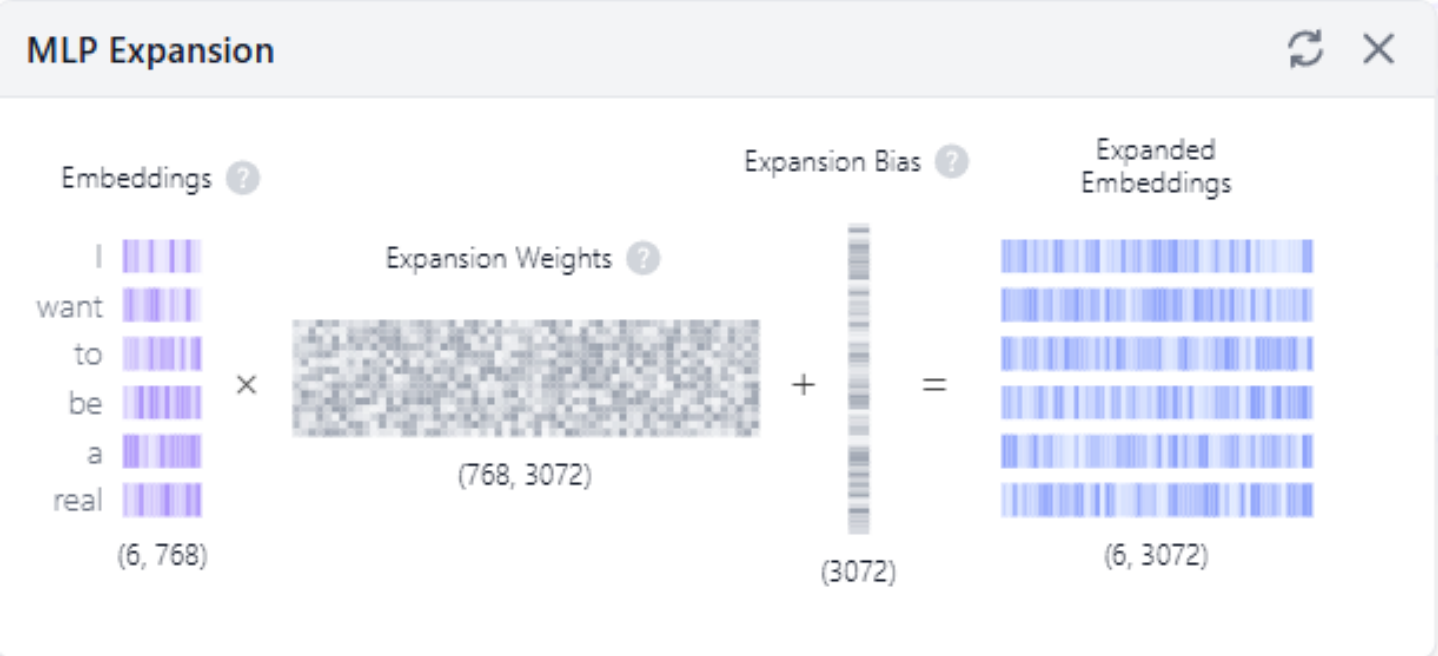


Figure 4. Using MLP layer to project the self-attention representations into higher dimensions to enhance the model's representational capacity.

ပုံမှာမြင်တွေ့ရတဲ့အတိုင်း MLP Block မှာ Linear transformations လုပ်ပေးနိုင်မယ့် GELU activation functions နှစ်ခုကို တွေ့ရမှာပါ။

ပထမတစ်ခုက input ရဲ့ dimension ကို တိုးပေးတာပါ။ အပေါ်မှာ ပြောခဲ့တဲ့ Embedding token မှာ Token တွေကို 768-dimensional vector အဖြစ်နဲ့ သိမ်းတယ်လို့ပြောခဲ့ပါတယ်။ အာ့တော့ ပထမ Linear Transformation က 768 က 3072 dimensional vector အဖြစ် increases လုပ်ပေးလိုက်တာပါ။



ပြီးရင်တော့ second linear transformation ဝင်လာပြီး original 768 dimensional vector အဖြစ် ပြန်ပြီး ချုပ်ပေးလိုက်ပါတယ်။ Transformation ပြုလုပ်ခြင်းအားဖြင့် ကျွန်တော်တို့ရဲ့ input vectors တွေဟာ layers တစ်ခုနဲ့တစ်ခု အကူးအပြောင်းမှာ တစ်သမှတ်တည်း တိကျမှုရှိမရှိ စစ်ဆေးပြီးသားဖြစ်သွားပါတယ်။ အဓိက vector dimensions တွေကို consistency ကောင်းမကောင်း စမ်းသပ်တဲ့သဘောပဲဖြစ်ပါတယ်။

ကျွန်တော်တို့ attention layer မှာတုန်းကလိုမျိုး Tokens တွေရဲ့ကြားမှာ ရှိတဲ့ meaning တွေ relationship တွေကို Tokens တွေကို positioning လုပ်ပြီးရှာဖွေတာမျိုးတွေမလုပ်တော့ပဲ၊ MLP က ဒီအတိုင်း input feature vector တွေကို transformation ပြုလုပ်ပြီး နောက်ထပ် Sequence layer (or) Transformer block ထဲကို Feed Forward လုပ်ပေးလိုက်တာဖြစ်ပါတယ်။

Output Probabilities

ကဲ့ input tokens တွေအားလုံး Transformer blocks တွေစီကနေ passed ဖြစ်လာပြီးပြီ ဆိုတာနဲ့ final layer မှာ token prediction လုပ်ကြရတော့မှာပါ။ ဒီ layer သည် final representation ကို 50,257 dimensional space နဲ့ဖော်ပြပါတယ်။ predict vocabulary list (Token List) ထဲမှာ ထွက်လာတဲ့ words တွေက သူတို့ သက်ဆိုင်ရာ Logits value နဲ့ match လုပ်ပြီးထွက်လာပါမယ်။

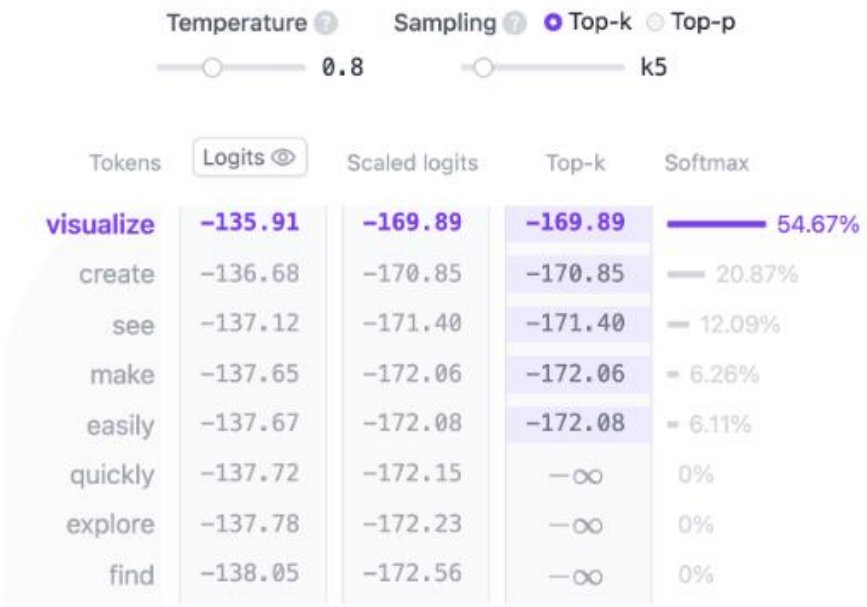


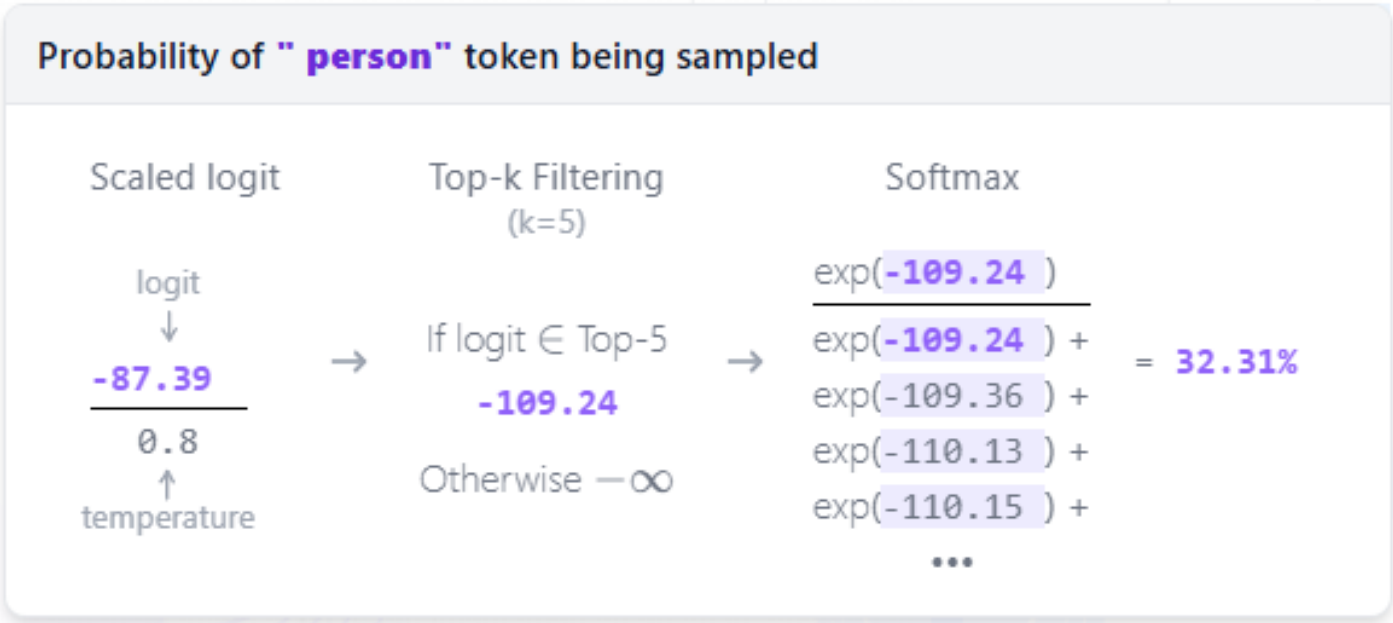
Figure 5. Each token in the vocabulary is assigned a probability based on the model's output logits. These probabilities determine the likelihood of each token being the next word in the sequence.

ထွက်လာတဲ့ Token list ထဲမှာရှိတဲ့ words အားလုံးက future token လို့သတ်မှတ်တဲ့ new word ဖြစ်မှာပါ။ အဲ့ထဲမှာ ဖြစ်တန်စွမ်း rank တွေကို likelihood နဲ့ တကွ ဖော်ပြပေးပါမယ်။ Logits ထဲက likehood တန်ဖိုးတွေတွကို probability distribution တွက်ပေးနိုင်တဲ့ softmax ထဲကို ထည့်လိုက်တာနဲ့ next token ရဲ့ probability distribution ကနေ sampling ရယူကြမှာဖြစ်ပါတယ်။

ကဲ့ ဒီနောက်ဆုံး step မှာ အရေးကြီးတဲ့ hyperparameter ကတော့ temperature ပါ။ logits ကထွက်လာတဲ့ output ကို temperature နဲ့ စားလိုက်တာပါပဲ ၊ သူကဘာလို့အရေးကြီးတာလဲ ဆက်လေ့လာရအောင်။

Temperature

- Temperature = 1: Logit output ကို 1 နဲ့ စားတာပဲဖြစ်တဲ့အတွက် softmax output က ဘာမှ ပြောင်းလဲမှာမဟုတ်ပါဘူး။
- Temperature < 1: ၁ ထက်ငယ်တာနဲ့ စားရင်တော့ probability distribution curve က sharpening ဖြစ်မှာ ဖြစ်တဲ့အတွက် ကျွန်တော်တို့အတွက် predict လုပ်ရတာ ပိုလွယ်သွားမှာပါ။
- Temperature > 1: ၁ ထက်ကြီးရင် softer probability distribution ပေါ့၊ ကျွန်တော်တို့အတွက် ရွေးချယ်စရာ next token တွေအများကြီးရှိလာလိမ့်မယ်။ ကိုယ်က RAP သီချင်း ဖန်တီးချင်တာမျိုး တို့ဆိုရင် ဒီ model ကိုသုံးသင့်တယ်၊ creativity လေ ဘေဘီ။



နောက်ထပ် သိရမယ့် hyperparameters လေးတွေကတော့ top-k နဲ့ top-p ပါ။ eg. top-k = 4 ဆိုပါစို့၊ ဒါဆိုရင် အမြင့် ဆုံး probabilities ရှိတဲ့လေးကောင်ကိုပဲ ယူမှာဖြစ်ပြီး အဲ့ထက်နဲရင် - infinity လုပ်ပစ်လိုက်ပြီး probabilities = 0 ဖြစ်သွားပါမယ်။ likelihood နဲတဲ့ ကောင်တွေကို filter လုပ်ချင်ရင် သုံးပါ။

top-p ကတော့ probability နဲ့တိုင်းတာတာပါ။ ကိုယ့်ရဲ့ model လိုအပ်ချက်ပေါ်မှာ ပုံစံမျိုးစုံနဲ့ ကြိုးခုံပြီး sample ယူချင်တဲ့သူတွေအတွက် tuning လုပ်ဖို့လိုအပ်ပါတယ်။

Step 4: Top-p Sampling (e.g. p = 0.95)

Start from highest and include tokens until cumulative probability ≥ 0.95:

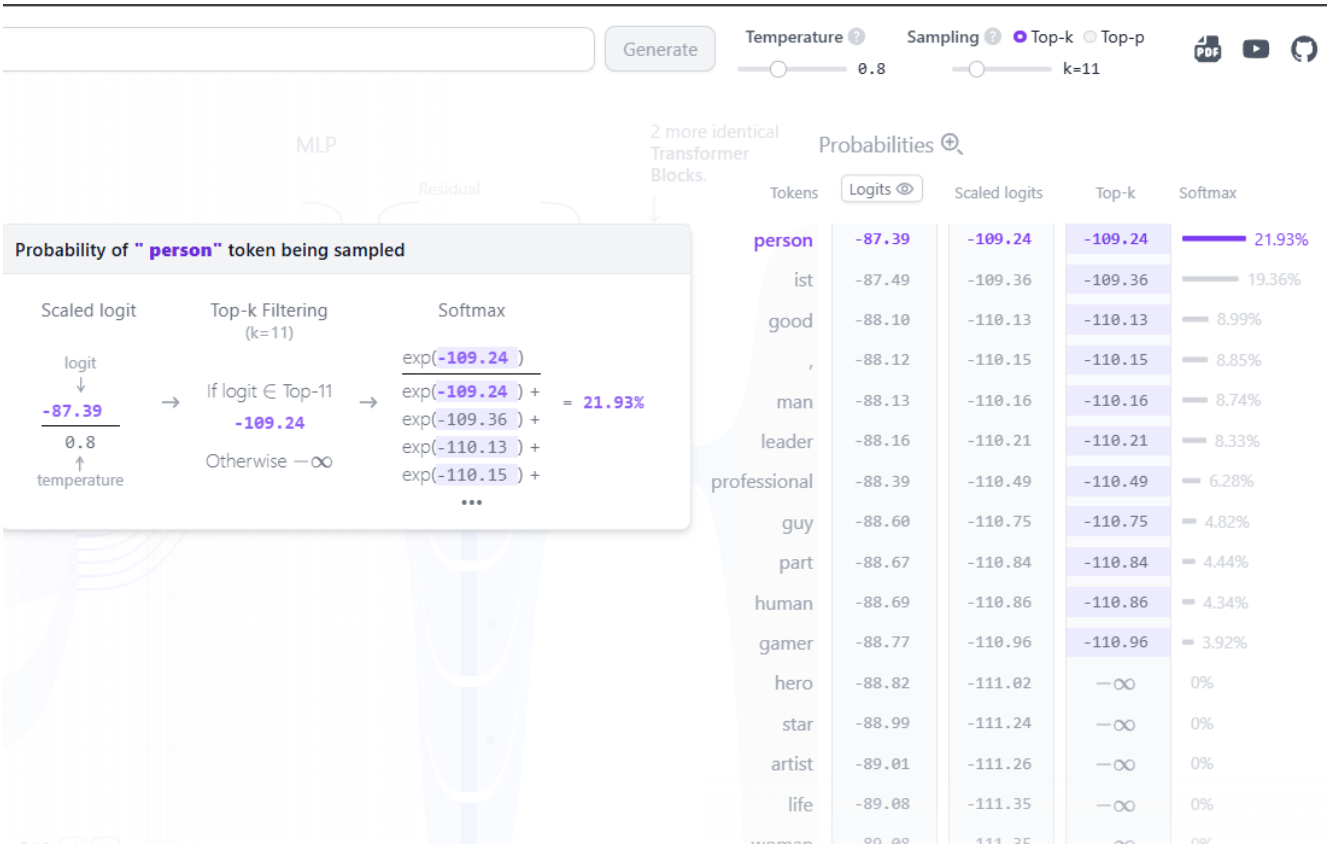
1. A → 85.9% (cumulative: 85.9%)
2. B → +11.6% (cumulative: **97.5%**) ✅ stop here

Only tokens **A and B** are kept.

- Re-normalize A and B:

Total = 403.43 + 54.60 = 458.03

Token	New Probability (%)
A	403.43 / 458.03 ≈ **88.1%**
B	54.60 / 458.03 ≈ 11.9%
C–E	0% (filtered out)



📐 Step 2: Softmax (No Top-k or Top-p Yet)

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Exponentiate each scaled logit:

Token	Scaled Logit	e^x
A	6.0	403.43
B	4.0	54.60
C	2.0	7.39
D	1.0	2.72
E	0.2	1.22

Total = 403.43 + 54.60 + 7.39 + 2.72 + 1.22 = 469.36

Now compute probabilities:

Token	Probability (%)
A	403.43 / 469.36 ≈ 85.9%
B	54.60 / 469.36 ≈ 11.6%
C	7.39 / 469.36 ≈ 1.6%
D	2.72 / 469.36 ≈ 0.6%
E	1.22 / 469.36 ≈ 0.3%

Advanced Architectural Features

Transformer Models ရဲ့ performance ကိုတိုင်းတာဖို့အတွက် ကျွန်တော်တို့တွေ အပေါ်မှာ ဆွေးနွေးခဲ့ ကြတဲ့ architectural concepts ချည်းပဲ အရေးကြီးတာမဟုတ်ပါဘူး။ Layer normalization, Dropout အသုံးပြုတာတွေ၊ Residual Connections တွေက ပိုအရေးကြီးပါတယ်။ Normalization လုပ်ပေးနိုင်တာ ဟာလည်း model training process မှာ converge ဖြစ်ဖို့ပိုမြန်ပါတယ်။

Layer Normalization

Layer Normalization helps to stabilize the training process and improves convergence. It works by normalizing the inputs across the features, ensuring that the mean and variance of the activations are consistent. This normalization helps mitigate issues related to internal covariate shift, allowing the model to learn more effectively and reducing the sensitivity to the initial weights. Layer Normalization is applied twice in each Transformer block, once before the self-attention mechanism and once before the MLP layer.

Dropout

Dropout is a regularization technique used to prevent overfitting in neural networks by randomly setting a fraction of model weights to zero during training. This encourages the model to learn more robust features and reduces dependency on specific neurons, helping the network generalize better to new, unseen data. During model inference, dropout is deactivated. This essentially means that we are using an ensemble of the trained subnetworks, which leads to a better model performance.

Residual Connections

Residual connections were first introduced in the ResNet model in 2015. This architectural innovation revolutionized deep learning by enabling the training of very deep neural networks. Essentially, residual connections are shortcuts that bypass one or more layers, adding the input of a layer to its output. This helps mitigate the vanishing gradient problem, making it easier to train deep networks with multiple Transformer blocks stacked on top of each other. In GPT-2, residual connections are used twice within each Transformer block: once before the MLP and once after, ensuring that gradients flow more easily, and earlier layers receive sufficient updates during backpropagation.

Ref: https://poloclub.github.io/transformer-explainer/

https://transformer-circuits.pub/2021/framework/index.html