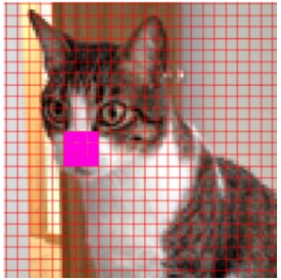
DeepLearning.AI

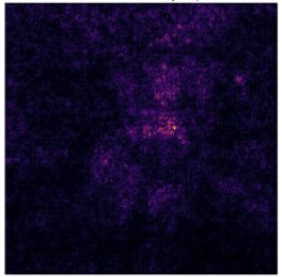
Transformers

Specialized Approaches to
Natural Language Processing in Pytorch

Module 2

Specialized Approaches to **Vision**





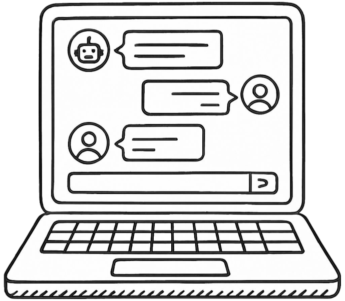
DeepLearning.AI

Laurence Moroney

Module 3

Specialized Approaches to **Language**

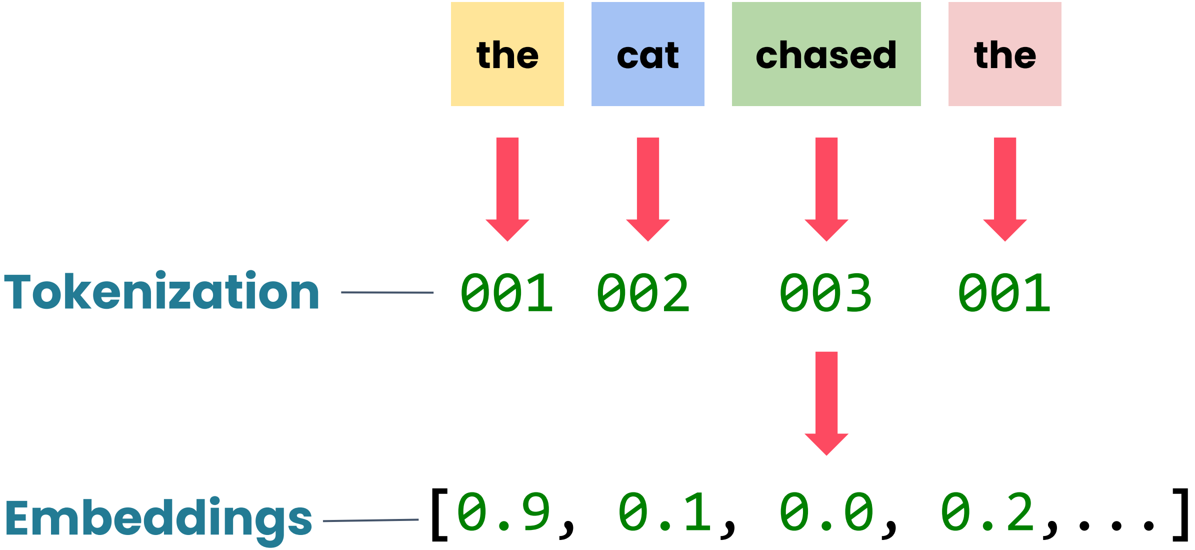
Transformers = Backbone of modern NLP



DeepLearning.AI

Laurence Moroney

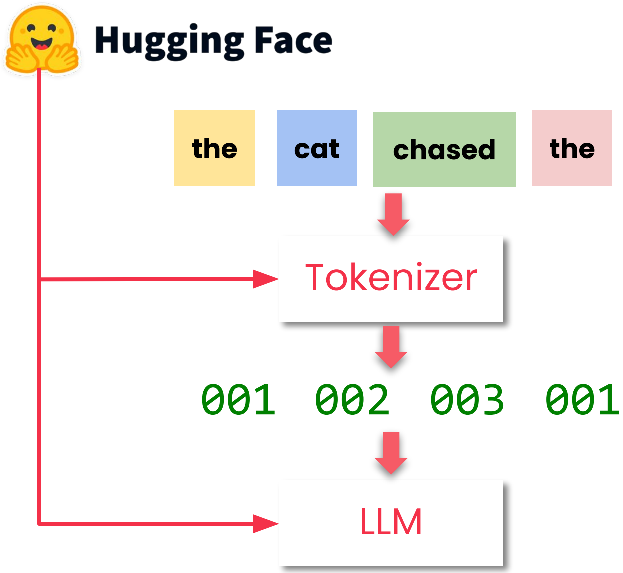
Tokenization and Embeddings



DeepLearning.AI

Laurence Moroney

Pre-Trained Models



Hugging Face

the cat chased the

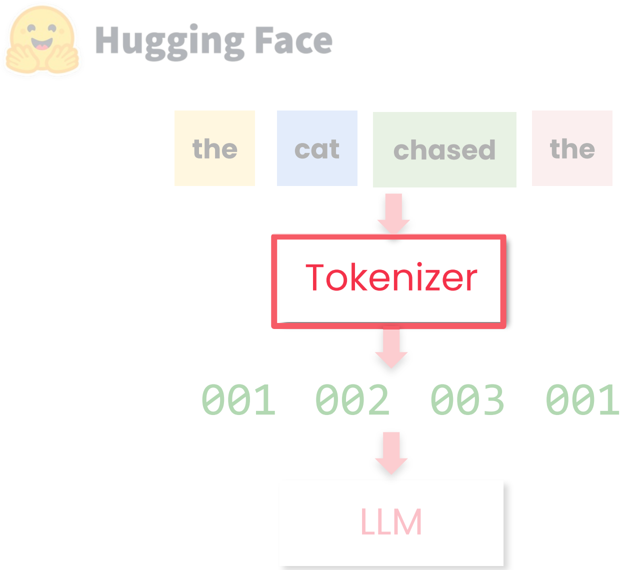
Tokenizer

001 002 003 001

LLM

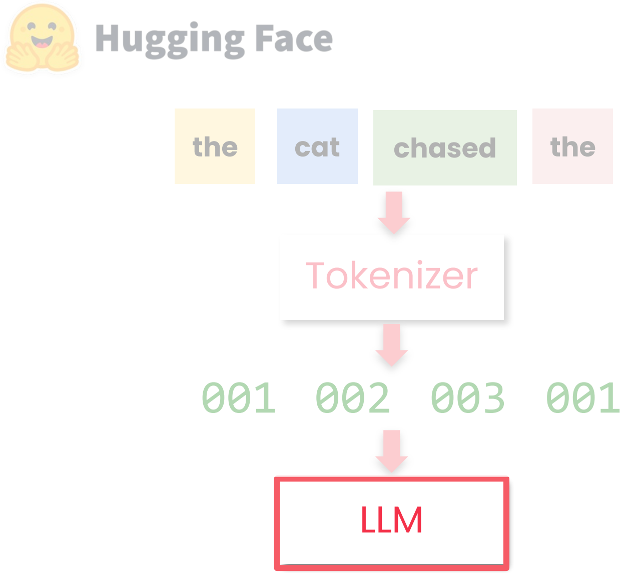
```python
from transformers import AutoTokenizer,
AutoModelForCausalLM

# Load a pretrained GPT-2 model and its tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Encode a short prompt
inputs = tokenizer("The cat chased the",
                   return_tensors="pt")

# Generate up to 10 tokens
outputs = model.generate(**inputs, max_length=10)
```

Pre-Trained Models



```
from transformers import AutoTokenizer,
AutoModelForCausalLM

# Load a pretrained GPT-2 model and its tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Encode a short prompt
inputs = tokenizer("The cat chased the",
                   return_tensors="pt")

# Generate up to 10 tokens
outputs = model.generate(**inputs, max_length=10)
```

Pre-Trained Models



Hugging Face

the cat chased the

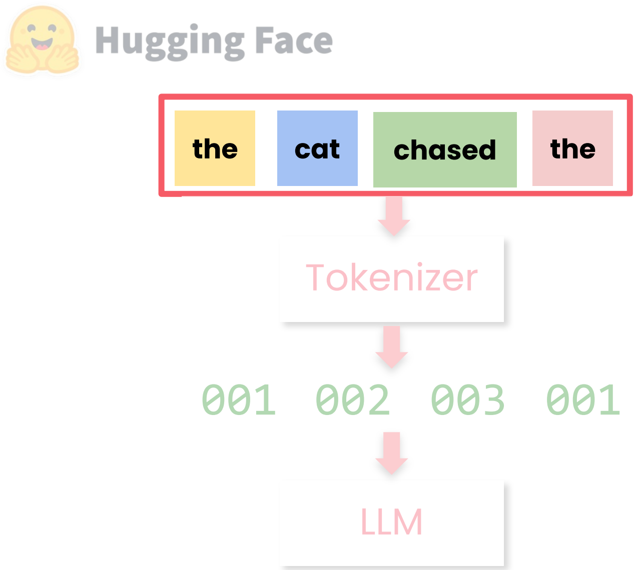
Tokenizer

001 002 003 001

LLM

```
from transformers import AutoTokenizer,
AutoModelForCausalLM

# Load a pretrained GPT-2 model and its tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Encode a short prompt
inputs = tokenizer("The cat chased the",
                   return_tensors="pt")

# Generate up to 10 tokens
outputs = model.generate(**inputs, max_length=10)
```

Pre-Trained Models



Hugging Face

the cat chased the

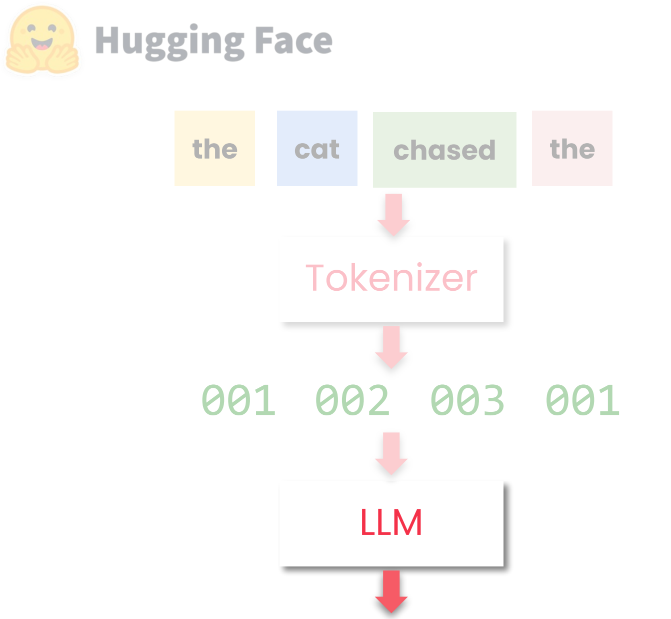
Tokenizer

001 002 003 001

LLM

```
from transformers import AutoTokenizer,
AutoModelForCausalLM

# Load a pretrained GPT-2 model and its tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Encode a short prompt
inputs = tokenizer("The cat chased the",
                   return_tensors="pt")

# Generate up to 10 tokens
outputs = model.generate(**inputs, max_length=10)
```

Pre-Trained Models



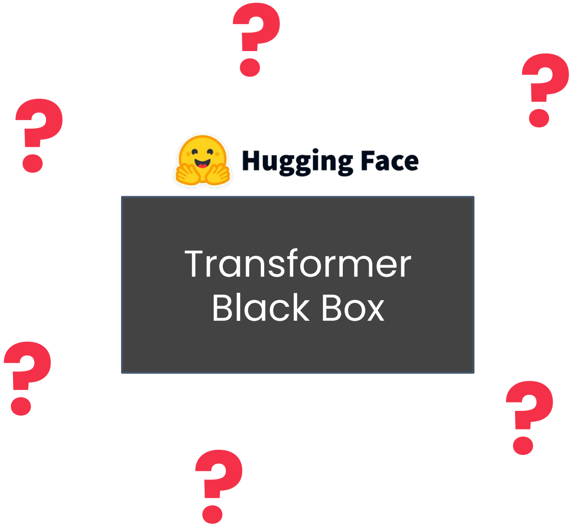
“The cat chased the mouse . . .”

```
from transformers import AutoTokenizer,
AutoModelForCausalLM

# Load a pretrained GPT-2 model and its tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Encode a short prompt
inputs = tokenizer("The cat chased the",
                   return_tensors="pt")

# Generate up to 10 tokens
outputs = model.generate(**inputs, max_length=10)
```

Transformers

You can't...

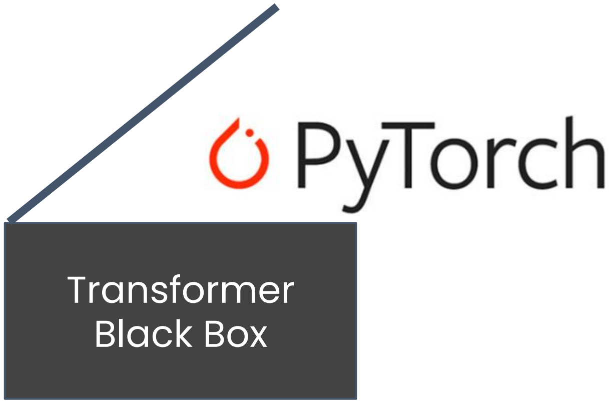
- Experiment
- Adapt
- Debug
- Explain



Hugging Face

Transformer
Black Box

Transformers

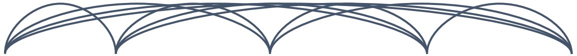


Early Language Models

RNNs & LSTMs

The cat chased the mouse quickly across the yard

Transformers



The cat chased the mouse

DeepLearning.AI

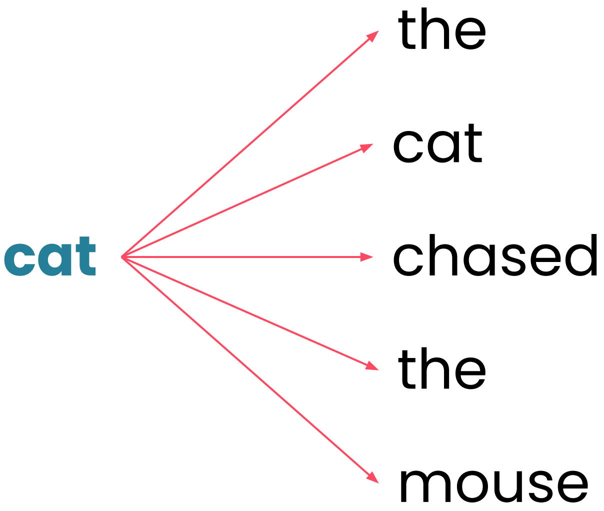
Laurence Moroney

Attention

“the cat chased the mouse”

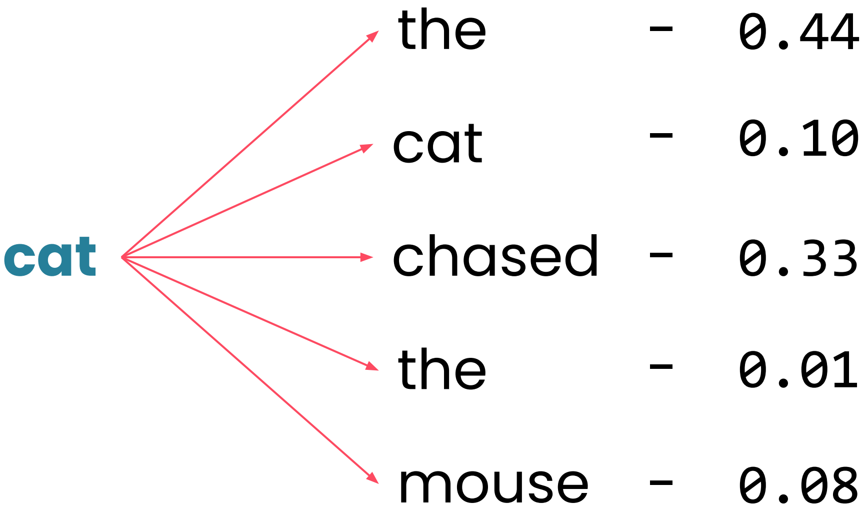
Attention

“the **cat** chased the mouse”



Attention

“the **cat** chased the mouse”

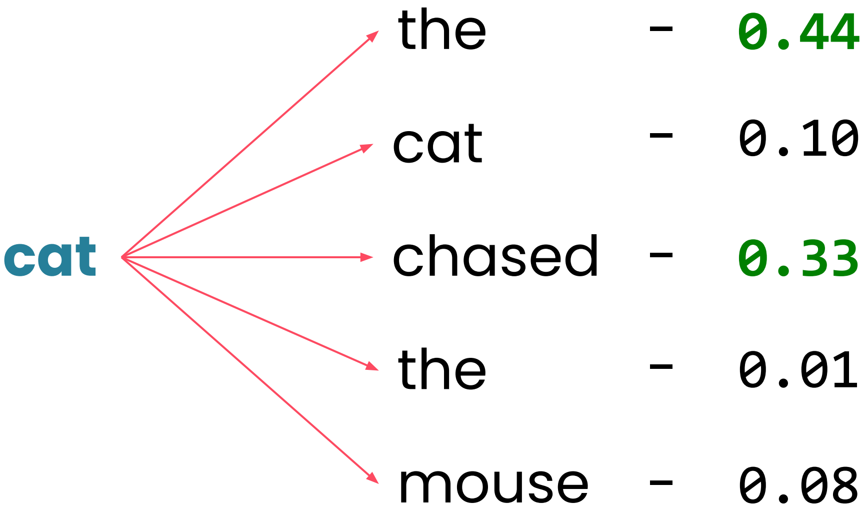


cat

the	-	0.44
cat	-	0.10
chased	-	0.33
the	-	0.01
mouse	-	0.08

Attention

“the **cat** chased the mouse”



cat

the - 0.44

cat - 0.10

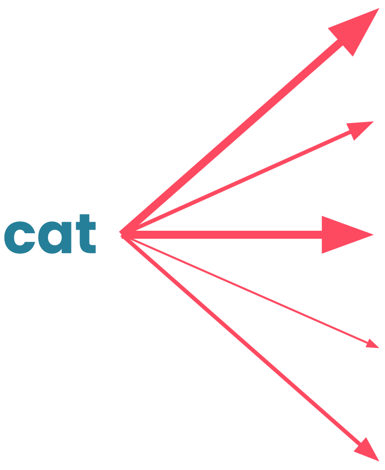
chased - 0.33

the - 0.01

mouse - 0.08

Attention

“the **cat** chased the mouse”



cat

the	-	0.44
cat	-	0.10
chased	-	0.33
the	-	0.01
mouse	-	0.08

Attention

The animal didn’t cross the street because it was too wide

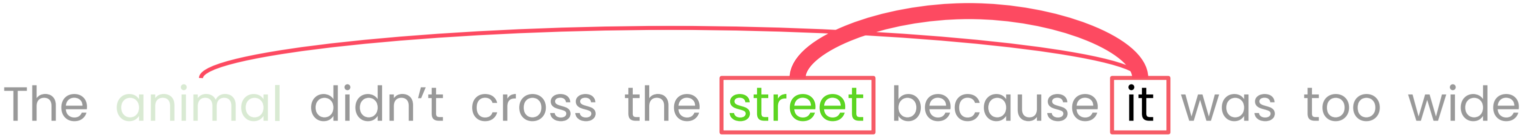
Attention

The animal didn’t cross the street because it was too wide

Attention

The animal didn’t cross the street because it was too wide

Attention

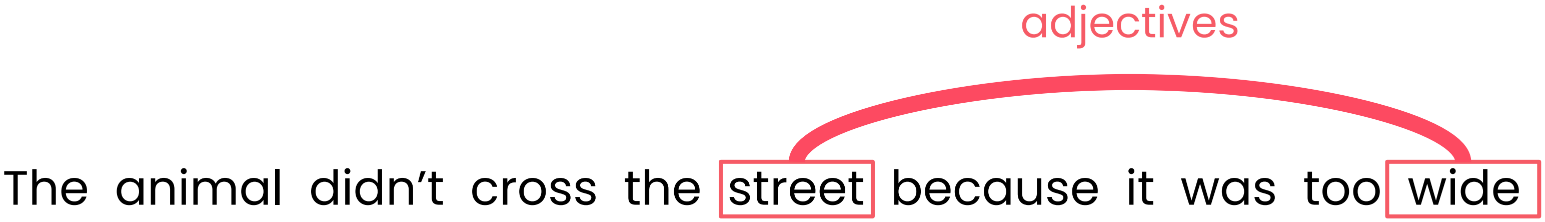


The animal didn’t cross the street because it was too wide

Attention

The animal didn’t cross the street because it was too wide

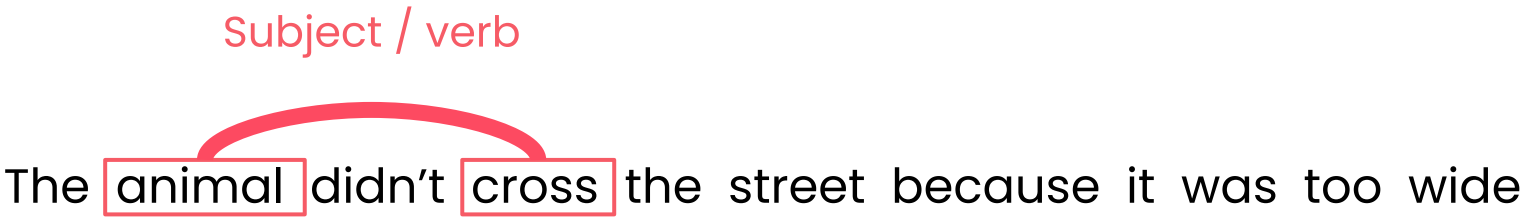
Attention



adjectives

The animal didn’t cross the street because it was too wide

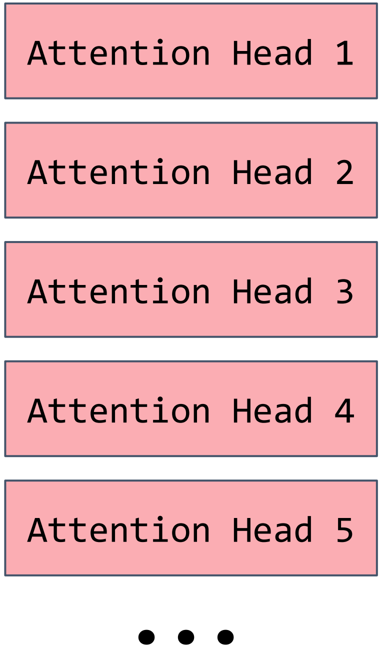
Attention



Subject / verb

The animal didn’t cross the street because it was too wide

Attention Heads



Attention Head 1

Attention Head 2

Attention Head 3

Attention Head 4

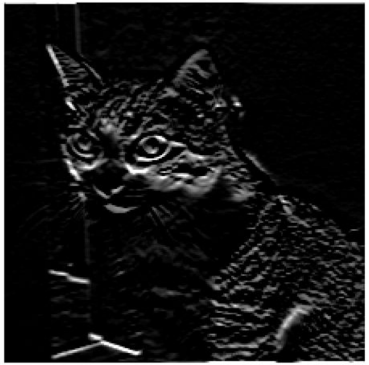
Attention Head 5

• • •

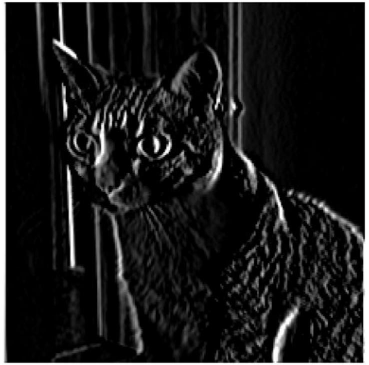
Each head learns different patterns

CNN Filters

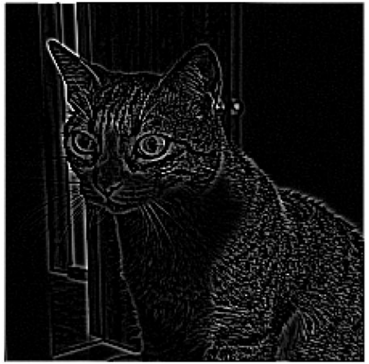
Horizontal Edge



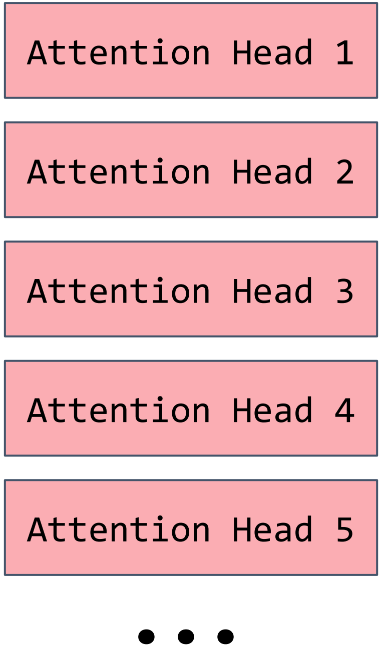
Vertical Edge



Ridge



Attention Heads



Attention Head 1

Attention Head 2

Attention Head 3

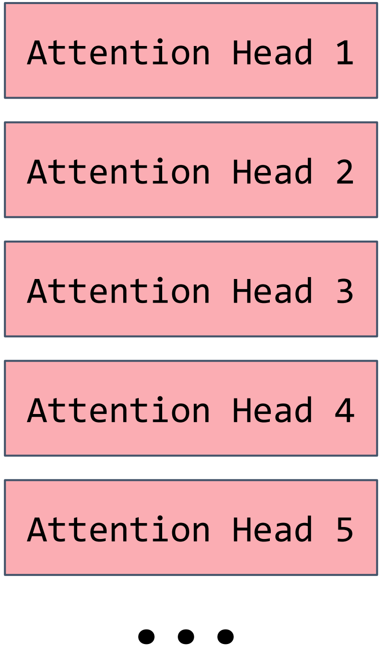
Attention Head 4

Attention Head 5

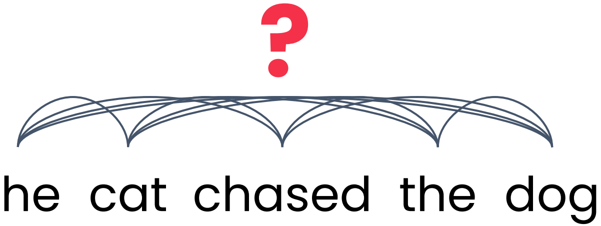
• • •

Attention Heads

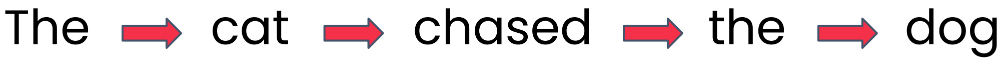
```python
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)
```



Position

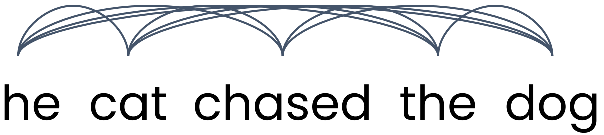


Position: RNNs

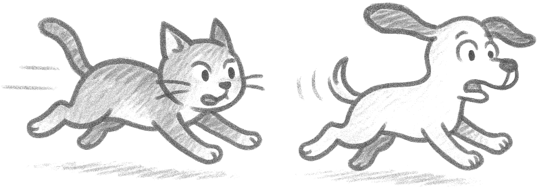


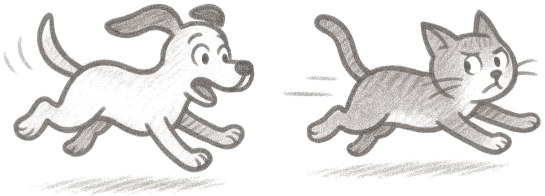
The → cat → chased → the → dog

Position: Transformers

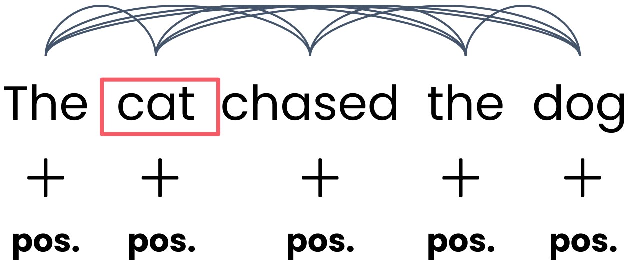


The cat chased the dog





Position: Transformers

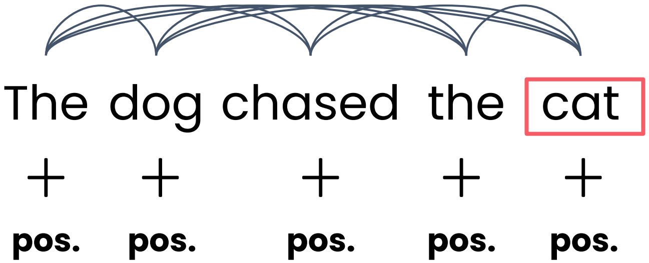


The cat chased the dog

+ \+ \+ \+ \+

pos. **pos.** **pos.** **pos.** **pos.**

Position: Transformers

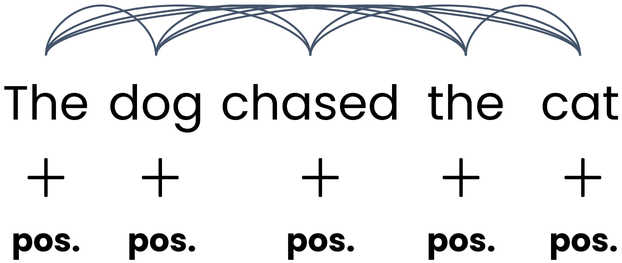


The dog chased the cat

+ \+ \+ \+ \+

pos. pos. pos. pos. pos.

Position: Transformers



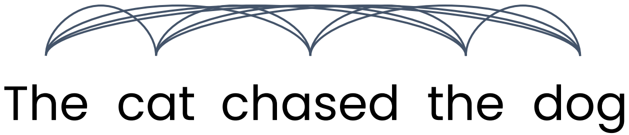
Positional info gives a sense of order

Transformer Architectures

- **Encoders**
- **Decoders**
- **Encoder-Decoders**

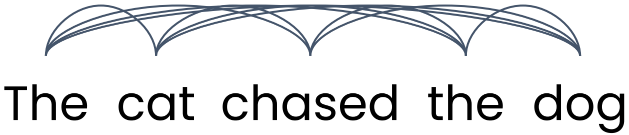
Encoders

- **Understanding**
- **Classification**
- **Named Entity**
- **BERT**



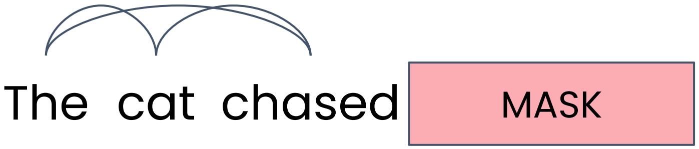
The cat chased the dog

Decoders



Decoders

- **Generation**
- **Chatbots**
- **GPT**

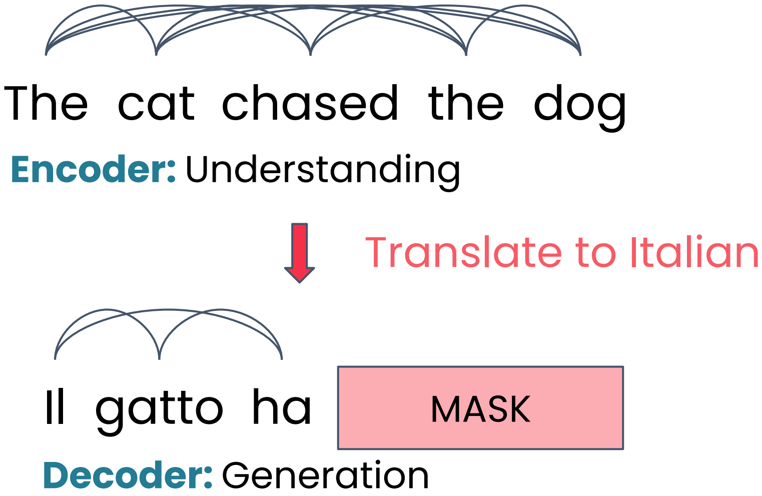


DeepLearning.AI

Laurence Moroney

Encoder - Decoders

- **Translation**
- **Summarization**



The cat chased the dog

Encoder: Understanding

Translate to Italian

Il gatto ha MASK

Decoder: Generation

Transformer Architectures

- **Encoders:** Understanding
- **Decoders:** Generation
- **Encoder-Decoders:** Transformation

Transformer Architectures

- **Encoders:** Understanding
- **Decoders:** Generation
- **Encoder-Decoders:** Transformation

Transformer Architectures

- **Encoders:** Understanding
- **Decoders:** Generation
- **Encoder-Decoders:** Transformation

Transformer Architectures

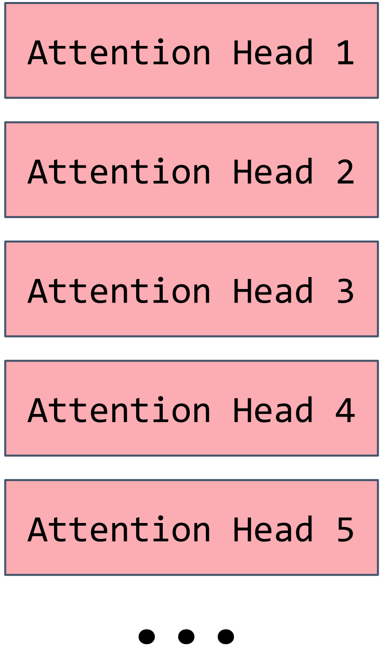
- **Encoders:** Understanding
- **Decoders:** Generation
- **Encoder-Decoders:** Transformation

DeepLearning.AI

Attention

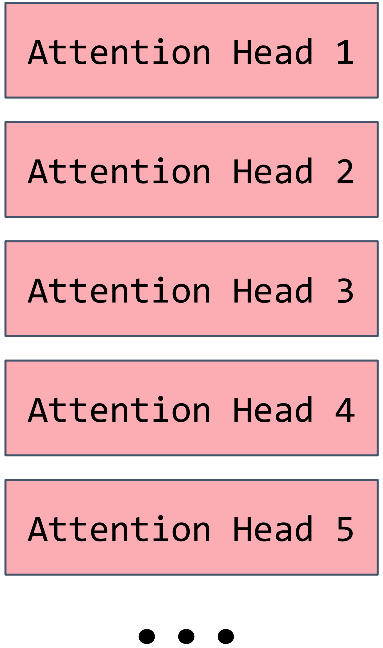
Specialized Approaches to
Natural Language Processing in Pytorch

Attention


Attention Head 1
Attention Head 2
Attention Head 3
Attention Head 4
Attention Head 5
• • •

Attention

```python
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)
```



Attention Head 1

Attention Head 2

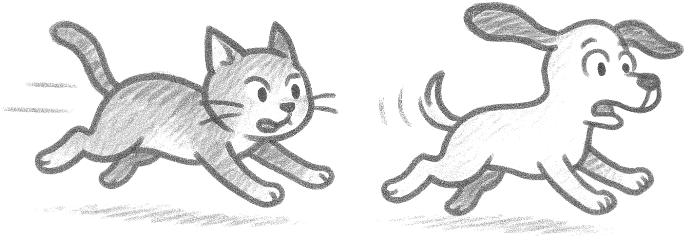
Attention Head 3

Attention Head 4

Attention Head 5

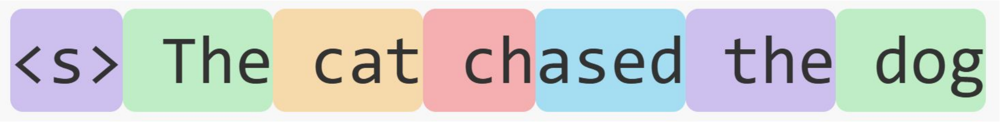
• • •

Attention



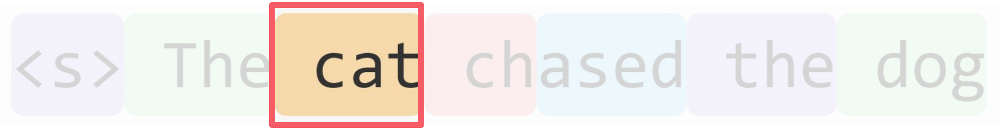
“The cat chased the dog”

Tokens



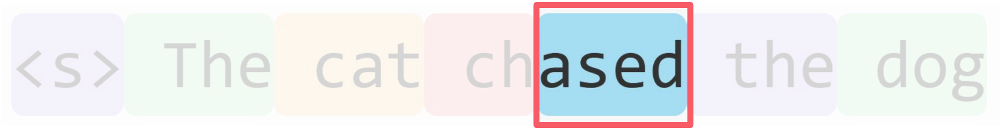
<s> The cat chased the dog

Tokens



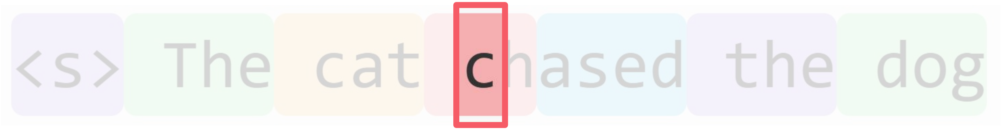
<s> The cat chased the dog

Tokens



<s> The cat ch ased the dog

Tokens



<s> The cat chased the dog

Tokens

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']]
```

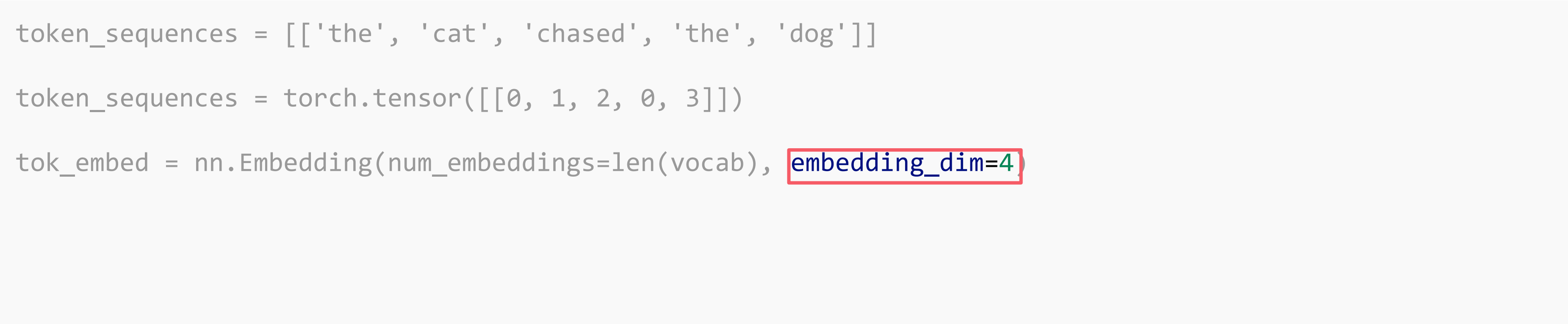
Tokens

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']]

token_sequences = torch.tensor([[0, 1, 2, 0, 3]])
```

Tokens

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']]

token_sequences = torch.tensor([[0, 1, 2, 0, 3]])

tok_embed = nn.Embedding(num_embeddings=len(vocab), embedding_dim=4)
```

Tokens



```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']]

token_sequences = torch.tensor([[0, 1, 2, 0, 3]])

tok_embed = nn.Embedding(num_embeddings=len(vocab), embedding_dim=4)
```

Tokens

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']]

token_sequences = torch.tensor([[0, 1, 2, 0, 3]])

tok_embed = nn.Embedding(num_embeddings=len(vocab), embedding_dim=4)
```

```
'cat'

[0.2,  0.8, -0.5, 0.1]
```

```
'dog'

[0.3,  0.7, -0.6, 0.2]
```

```
'carburetor'

[-0.9,  0.1, 0.8, -0.4]
```

Token Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

Token Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

Token Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

Token Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

Which word came first?

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

5

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the     0
               [-0.44,  0.91, -0.12, -0.77],   # cat     1
               [ 0.48,  0.02,  0.05,  0.39],   # chased  2
               [ 0.12, -0.55,  0.33,  0.10],   # the     3
               [-0.30,  0.14, -0.70,  0.81]]]) # dog     4
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the     0
               [-0.44,  0.91, -0.12, -0.77],   # cat     1
               [ 0.48,  0.02,  0.05,  0.39],   # chased  2
               [ 0.12, -0.55,  0.33,  0.10],   # the     3
               [-0.30,  0.14, -0.70,  0.81]]]) # dog     4


positions = torch.arange(seq_len, device=word_vecs.device)
```

Positional Embeddings

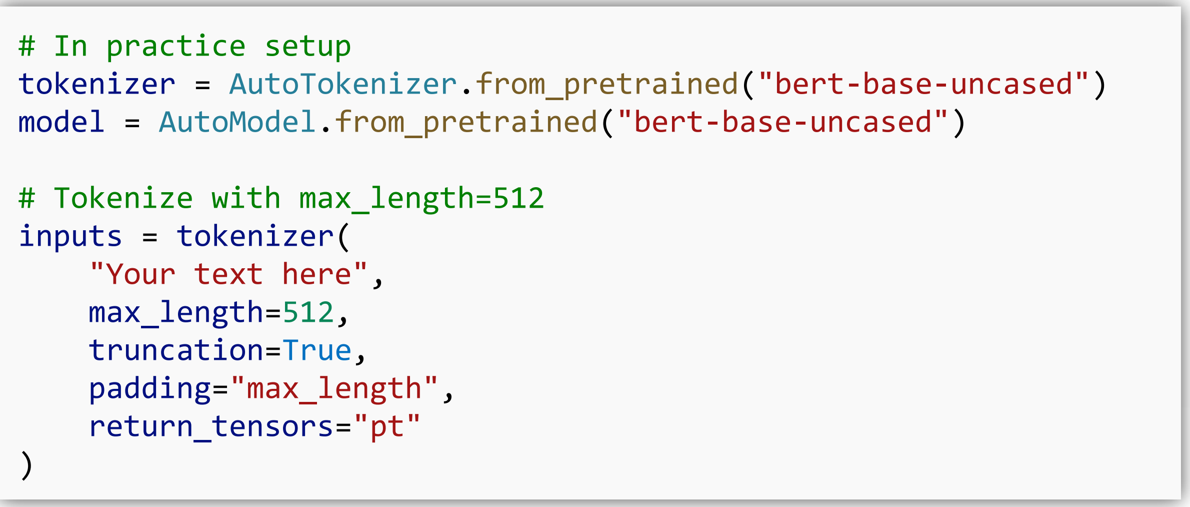
```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
                [-0.44,  0.91, -0.12, -0.77],   # cat
                [
                [
                [-


positions = torch
```



```
# In practice setup
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

# Tokenize with max_length=512
inputs = tokenizer(
    "Your text here",
    max_length=512,
    truncation=True,
    padding="max_length",
    return_tensors="pt"
)
```

DeepLearning.AI

Laurence Moroney

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [
               [
               [-


positions = torch
```

```
# In practice setup
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

# Tokenize with max_length=512
inputs = tokenizer(
    "Your text here",
    max_length=512,
    truncation=True,
    padding="max_length",
    return_tensors="pt"
)
```

DeepLearning.AI

Laurence Moroney

Positional Embeddings

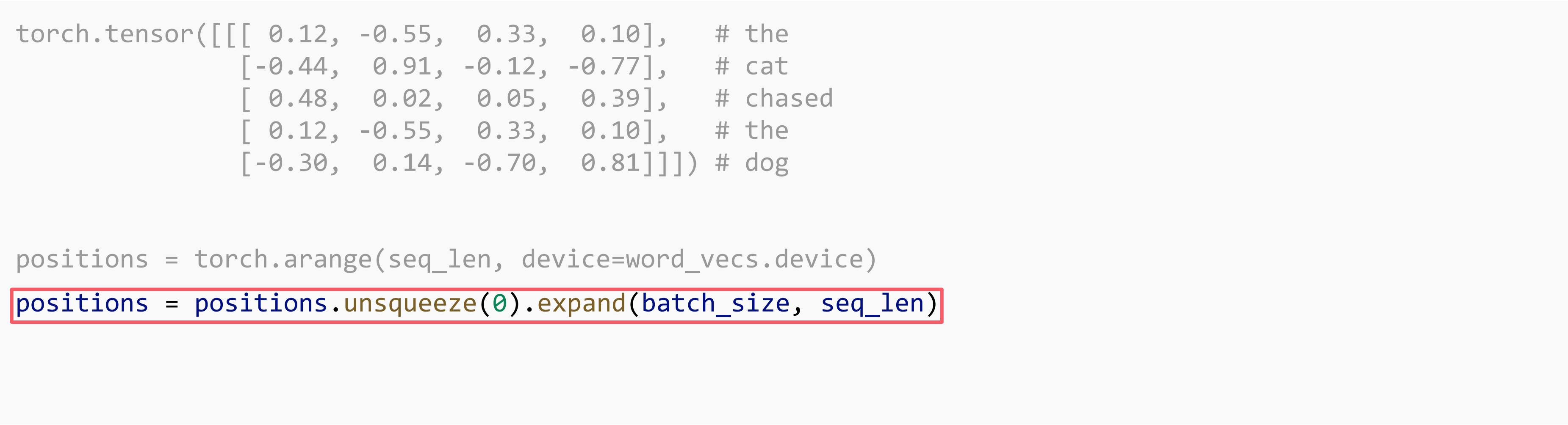
```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)
```

```
#positions
tensor([0, 1, 2, 3, 4])
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)
positions = positions.unsqueeze(0).expand(batch_size, seq_len)
```

Positional Embeddings



```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)

positions = positions.unsqueeze(0).expand(batch_size, seq_len)
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)
positions = positions.unsqueeze(0).expand(batch_size, seq_len)
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)
positions = positions.unsqueeze(0).expand(batch_size, seq_len)
```

Positional Embeddings

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog


positions = torch.arange(seq_len, device=word_vecs.device)

positions = positions.unsqueeze(0).expand(batch_size, seq_len)

pos_embed = nn.Embedding(num_embeddings=seq_len, embedding_dim=emb_dim)
```

Token and Positional Embeddings

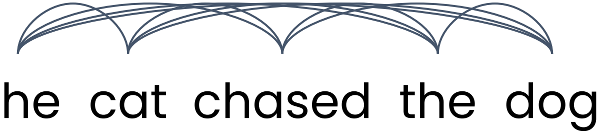
```
pos_vecs = pos_embed(positions)
word_vecs = tok_embed(token_ids)
```

Token and Positional Embeddings

```
pos_vecs = pos_embed(positions)
word_vecs = tok_embed(token_ids)


# add them together
input_vecs = word_vecs + pos_vecs
```

meaning position

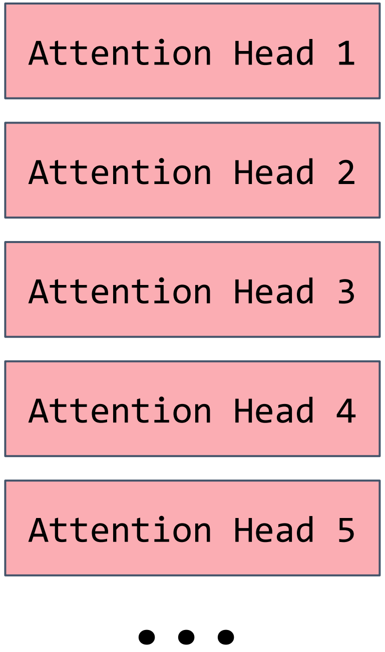
Attention



Pay attention to what?

… and how much attention?

Attention Heads



Attention Head 1

Attention Head 2

Attention Head 3

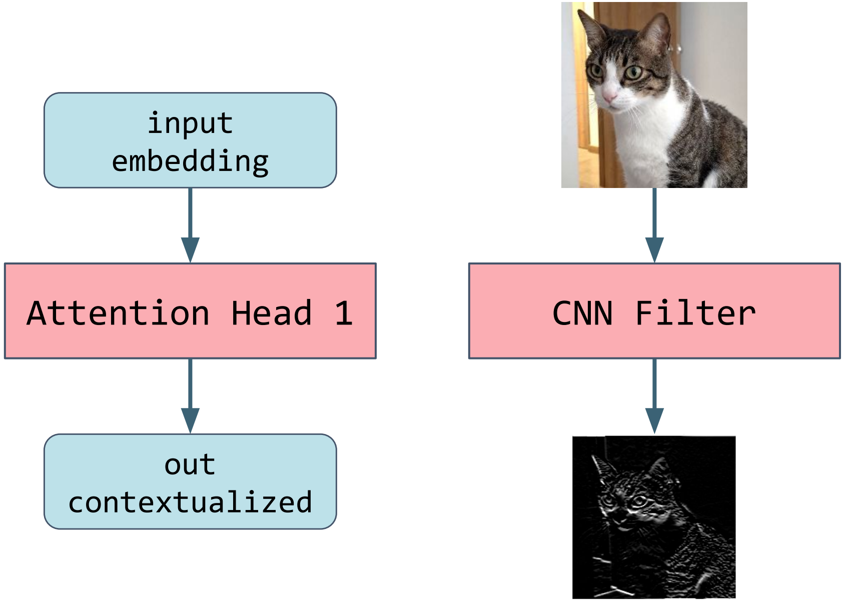
Attention Head 4

Attention Head 5

• • •

Each head learns different patterns

Attention Heads



input embedding

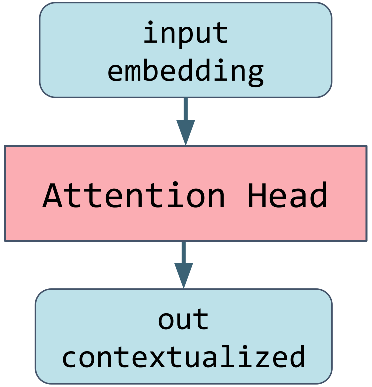
Attention Head 1

out contextualized

CNN Filter

Horizontal Edge

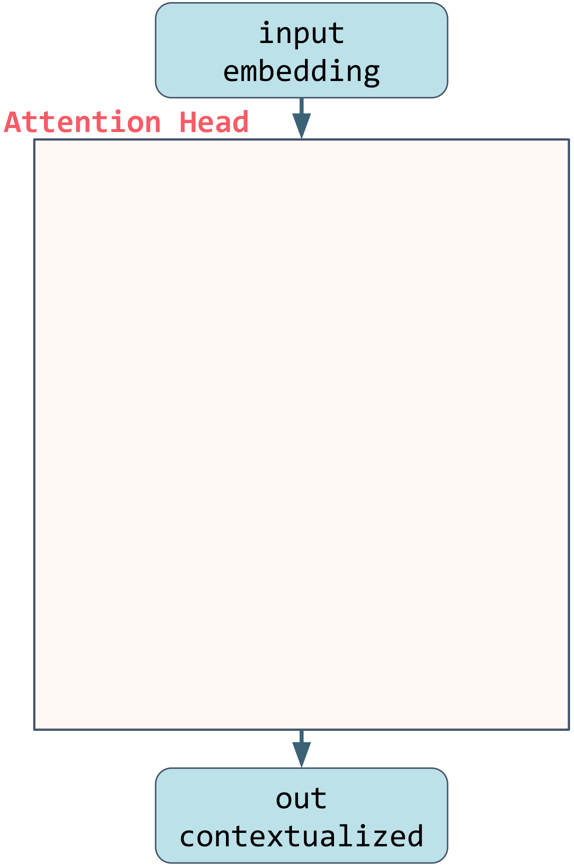
Attention Heads



DeepLearning.AI

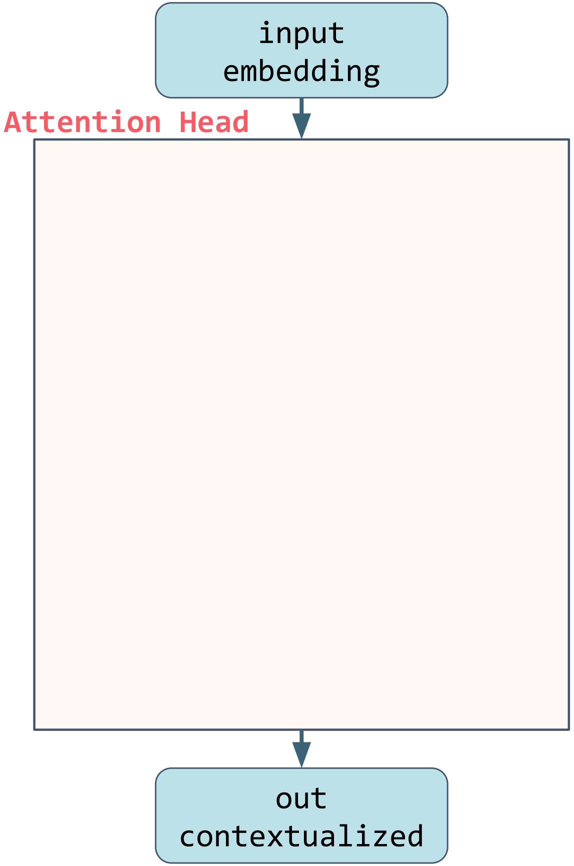
Laurence Moroney

Attention Heads



Attention Heads

```
to_k = nn.Linear(4, 4)
```

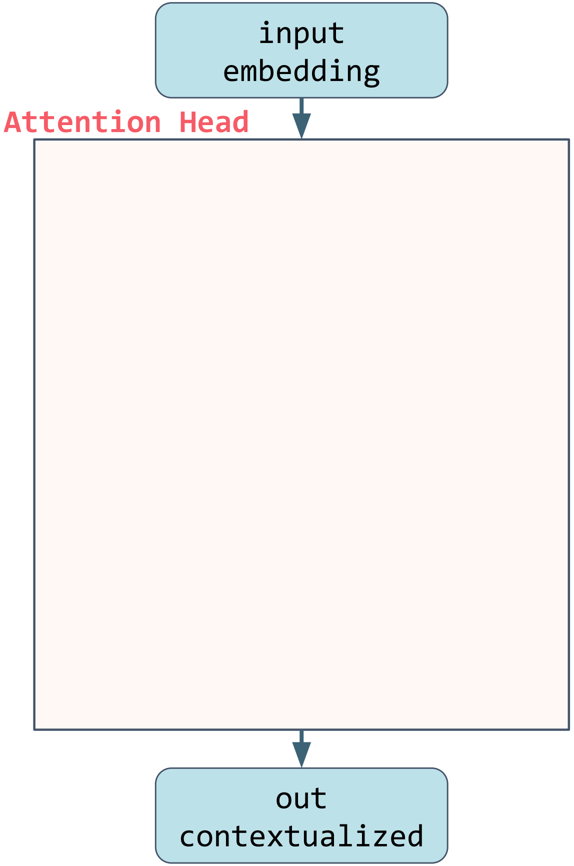


DeepLearning.AI

Laurence Moroney

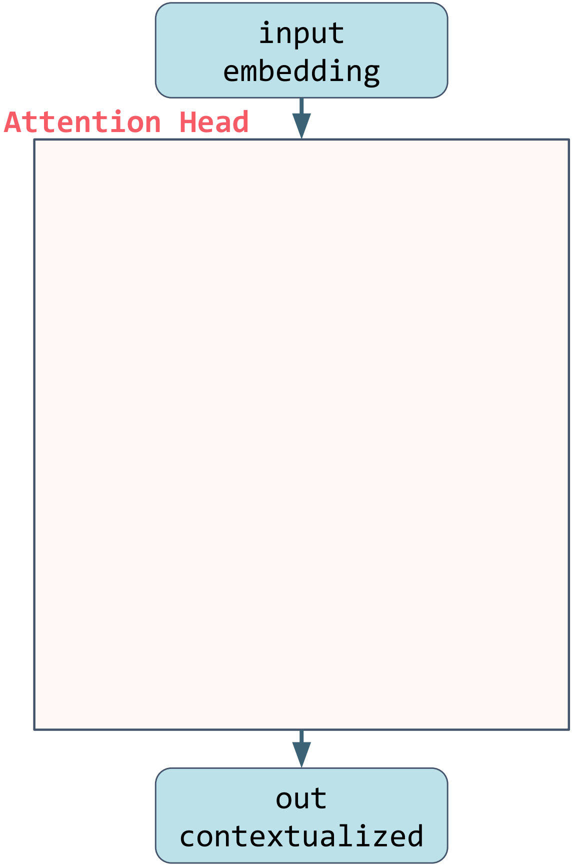
Attention Heads

```
to_k = nn.Linear(4, 4)
```



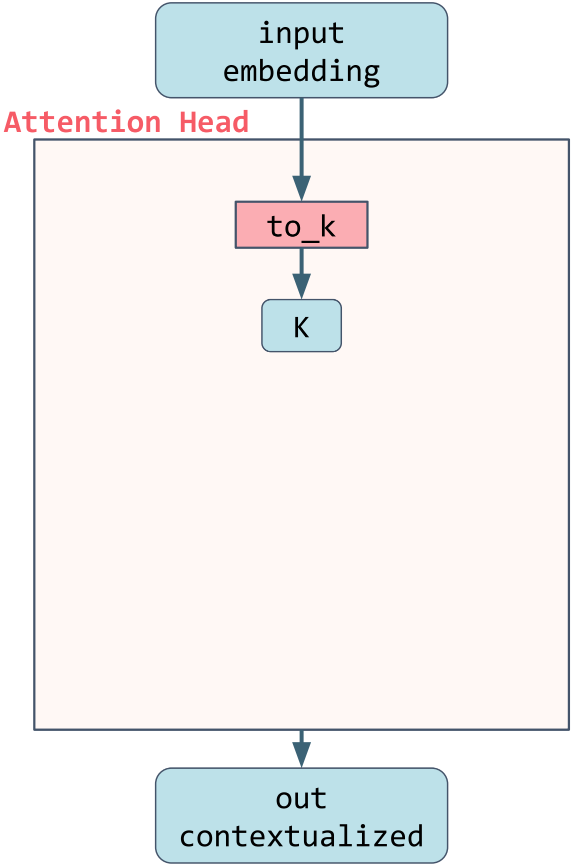
Attention Heads

```
to_k = nn.Linear(4, 4)

K = to_k(input_embedding)
```



Attention Heads

```
to_k = nn.Linear(4, 4)

K = to_k(input_embedding)
```



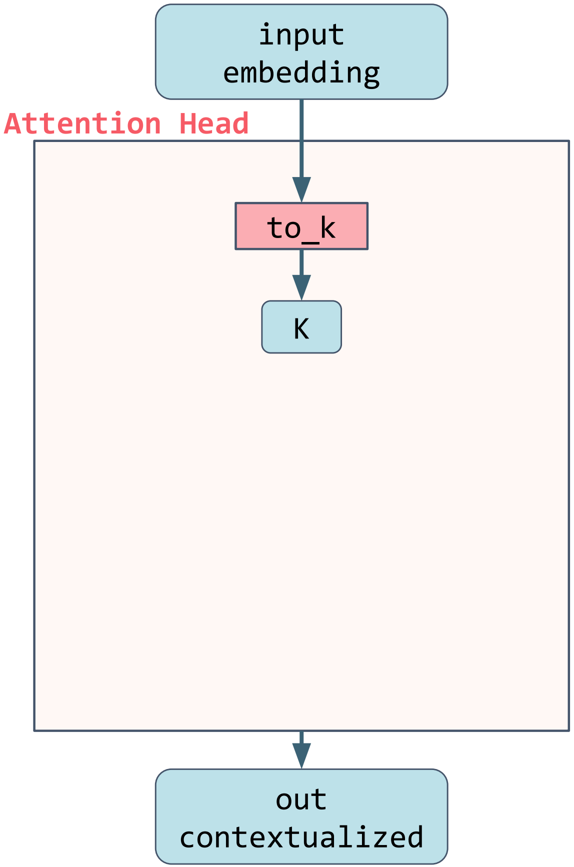
Key
What I look like in this space

But who should I pay attention to?

What am I looking for?

Attention Heads

```
to_k = nn.Linear(4, 4)

K = to_k(input_embedding)
```

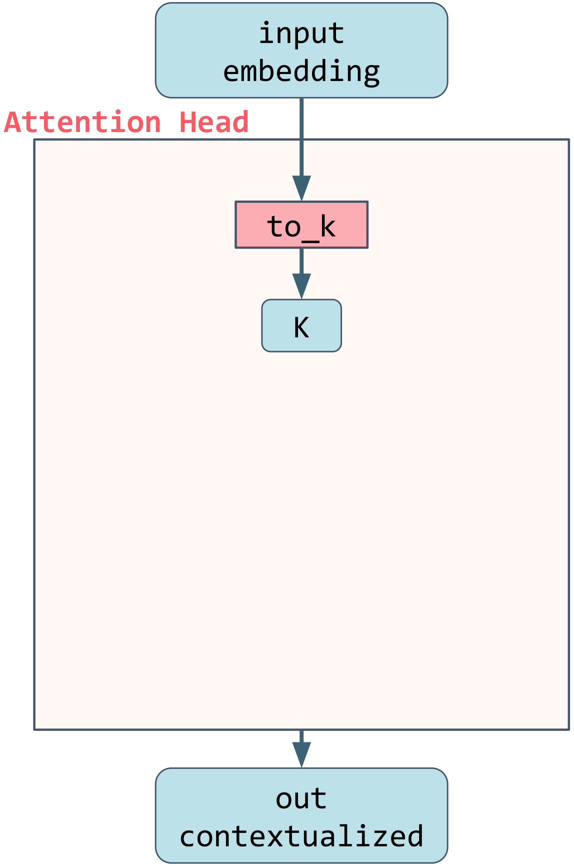


Query
What I'm looking for

Key
What I look like in this space

Attention Heads

```
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)
```

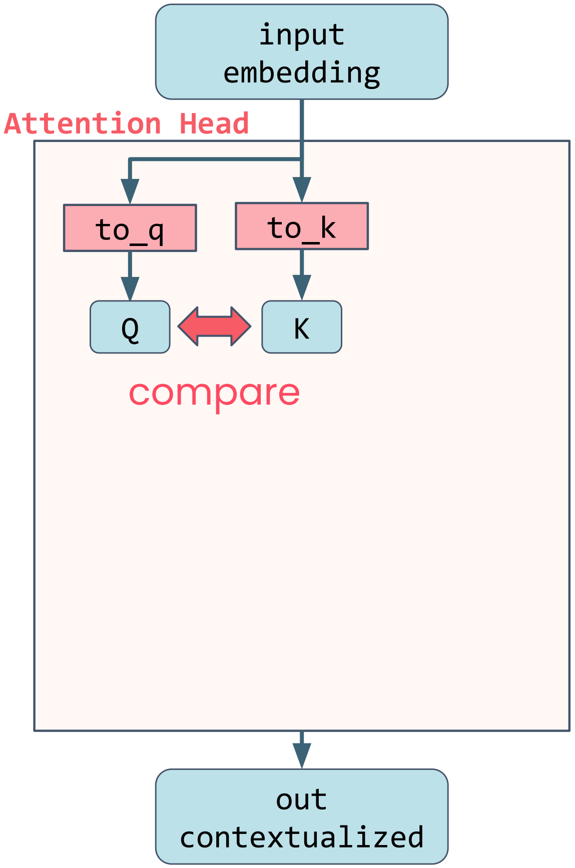


Query
What I'm looking for

Key
What I look like in this space

Attention Heads

```
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)
```

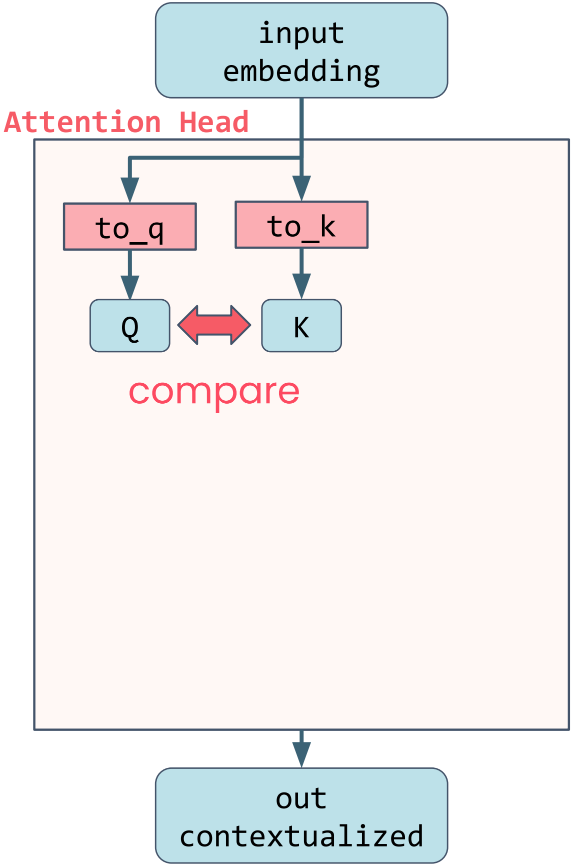


Query
What I'm looking for

Key
What I look like in this space

Attention Heads

```
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)
```

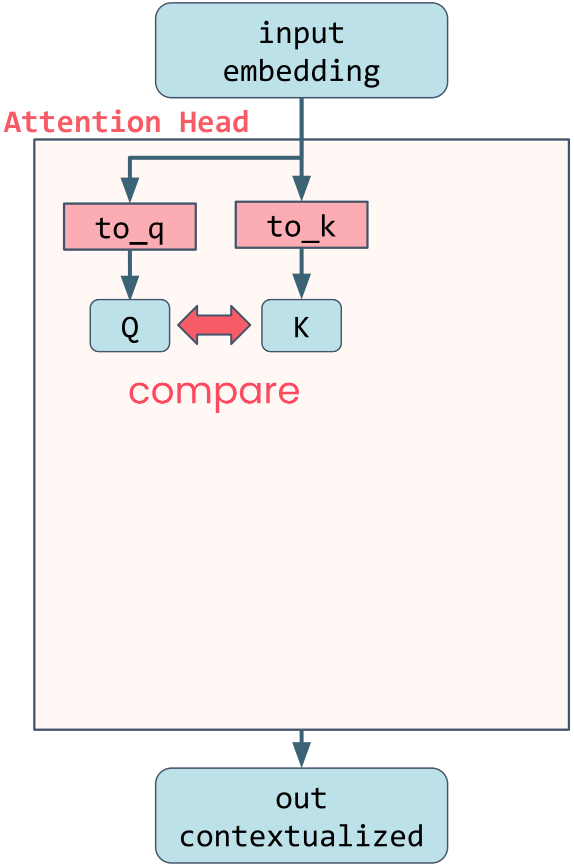


Query
What I'm looking for

Key
What I look like in this space

Attention Heads

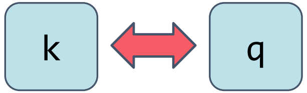
```python
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)
```



Query
What I’m looking for

Key
What I look like in this space

Calculating Attention Weights



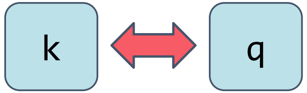
compare

scaled dot product

“the **cat** chased the dog”

“cat” `Query = [ 0.2, −0.1,  0.5, 0.3]` what ‘cat’ is looking for

Calculating Attention Weights



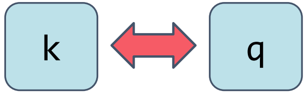
compare

scaled dot product

“**the cat** chased the dog”

```
“cat”  Query = [ 0.2, −0.1,  0.5, 0.3]   what ‘cat’ is looking for
                  *     *     *    *
“the”    Key = [ 0.1,  0.5,  0.4, 0.2]   what ‘the’ looks like

                0.02  -0.05  0.2  0.06
```

Calculating Attention Weights



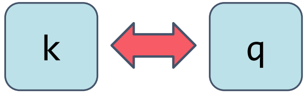
compare

scaled dot product

“**the cat** chased the dog”

```
“cat”  Query = [ 0.2, −0.1,  0.5, 0.3]   what ‘cat’ is looking for
                   *     *     *    *
“the”    Key = [ 0.1,  0.5,  0.4, 0.2]   what ‘the’ looks like


         sum([0.02, -0.05, 0.2, 0.06])
```

Calculating Attention Weights



compare

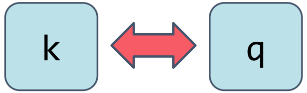
scaled dot product

“**the cat** chased the dog”

```
“cat”  Query = [ 0.2, -0.1,  0.5, 0.3]   what ‘cat’ is looking for
                  *     *     *     *
“the”    Key = [ 0.1,  0.5,  0.4, 0.2]   what ‘the’ looks like
```

0.23 attention score

Calculating Attention Weights



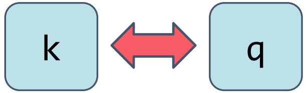
compare

scaled dot product

```
K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))

# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale
```

`0.23` attention score

Calculating Attention Weights



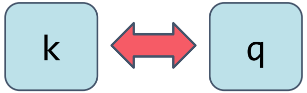
compare

scaled dot product

```
K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))

# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale
```

$$\frac{0.23}{\texttt{scale}}$$ attention score

Calculating Attention Weights



compare

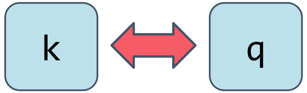
scaled dot product

```
K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))

# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale
```

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Calculating Attention Weights



compare

scaled dot product

```
K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))

# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale
```

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Calculating Attention Weights

compare

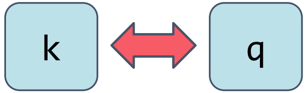
scaled dot product

```
K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))

# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

attn = scores.softmax(dim=-1)
```

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Calculating Attention Weights



compare

scaled dot product

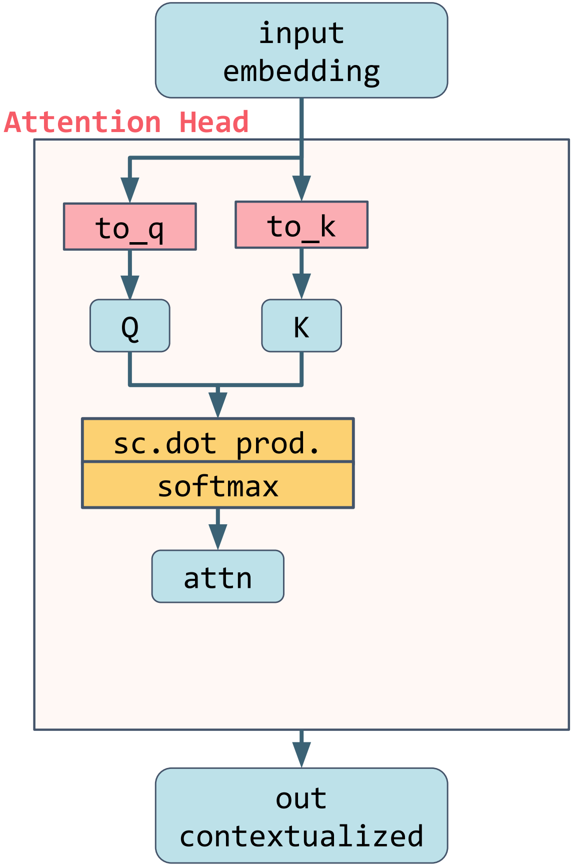
```
K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))

# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

attn = scores.softmax(dim=-1)
```

Attention Weights

	the	cat	chased	the	dog	
the	.557	.129	.077	.061	.175	Sum=1
cat	.083	.189	.020	.036	.673	Sum=1
chased	.540	.083	.046	.047	.285	Sum=1
the	.524	.036	.225	.019	.196	Sum=1
dog	.329	.205	.227	.073	.167	Sum=1

Attention Heads

```
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)
```



Query
What I'm looking for

Key
What I look like in this space

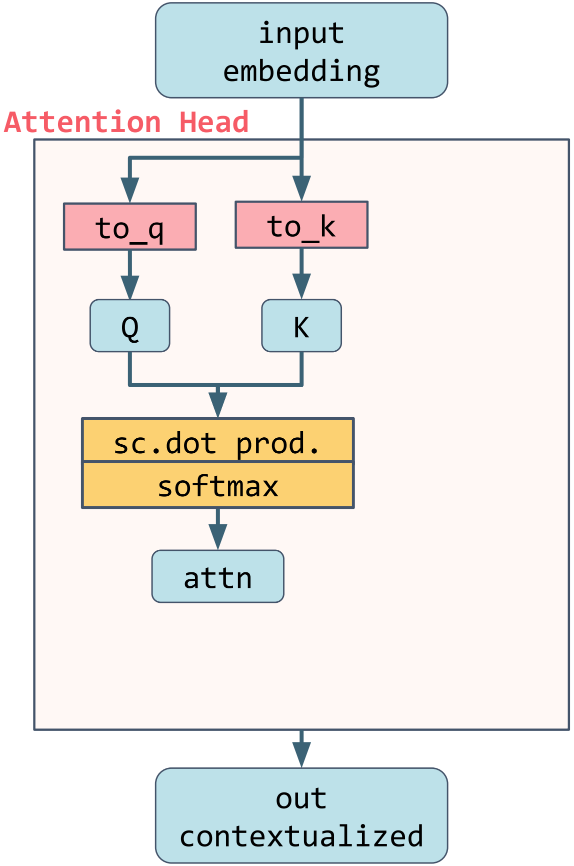
But what do tokens contribute?

DeepLearning.AI

Laurence Moroney

Attention Heads

```
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)
```



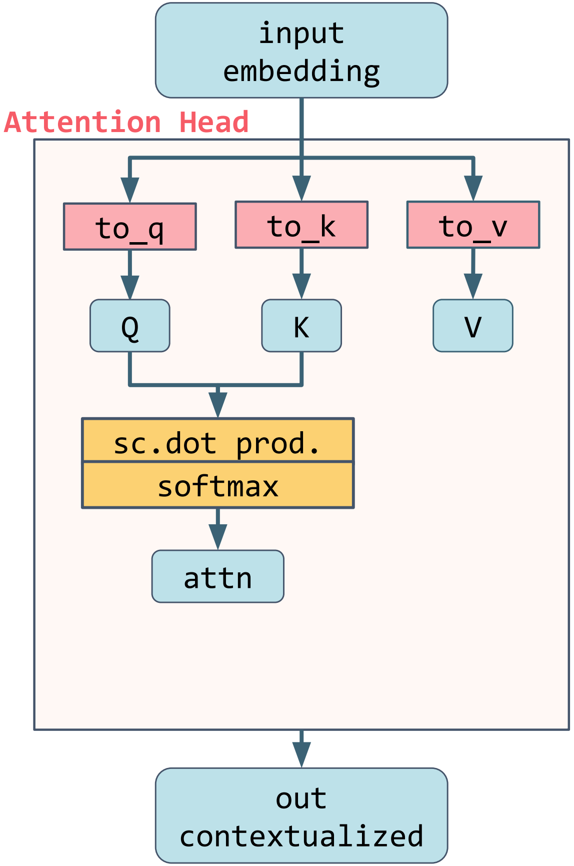
Query
What I'm looking for

Key
What I look like in this space

Value
If you attend to me,
here's what I'll give you

Attention Heads

```
to_q = nn.Linear(4, 4)
to_k = nn.Linear(4, 4)
to_v = nn.Linear(4, 4)

Q = to_q(input_embedding)
K = to_k(input_embedding)
V = to_v(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)
```

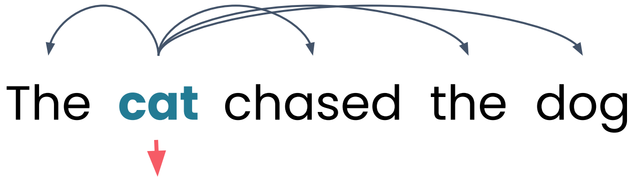


Query
What I'm looking for

Key
What I look like in this space

Value
If you attend to me,
here's what I'll give you

Attention Output



“cat” **attn.** to others

the	.083
cat	.189
chased	.020
the	.036
dog	.673

Value (v) vector of other tokens

“the” **value**

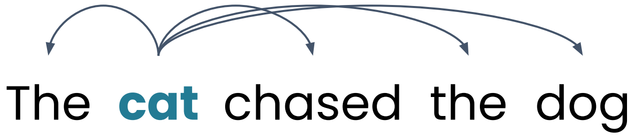
“cat” **value**

“chased” **value**

“the” **value**

“dog” **value**

Attention Output



"Cat" **attn.** * **Value** vector of other tokens

.083 * "the" **value**

.189 * "cat" **value**

.020 * "chased" **value**

.036 * "the" **value**

.673 * "dog" **value**

SUM → [0.27, -0.35, 0.80, -0.58]

"cat" contextualized output

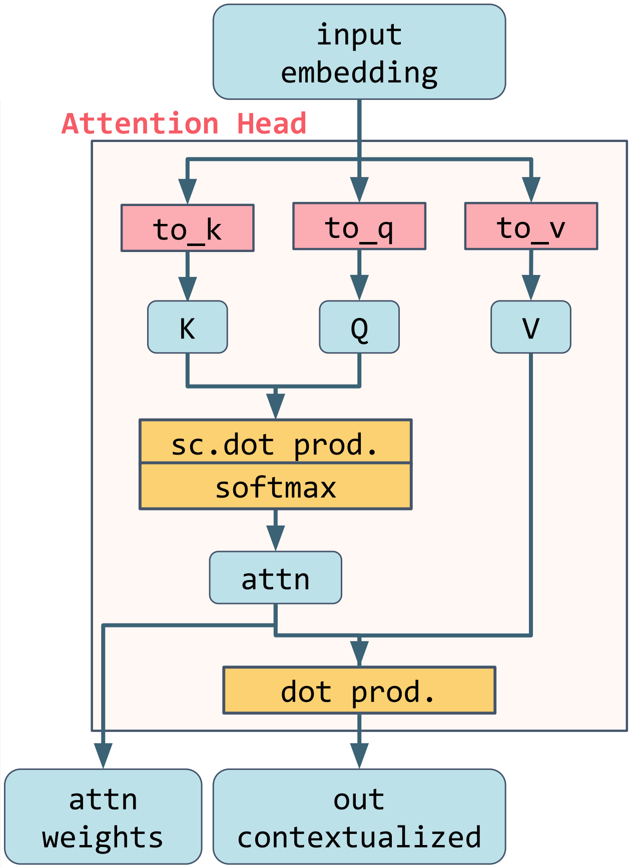
```
# contextualized output: dot product
out = torch.matmul(attn, V)
```

DeepLearning.AI

Laurence Moroney

Attention Heads

```
to_k = nn.Linear(4, 4)
to_q = nn.Linear(4, 4)
to_v = nn.Linear(4, 4)

K = to_k(input_embedding)
Q = to_q(input_embedding)
V = to_v(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)

# contextualized output: dot product
out = torch.matmul(attn, V)
return out, attn
```



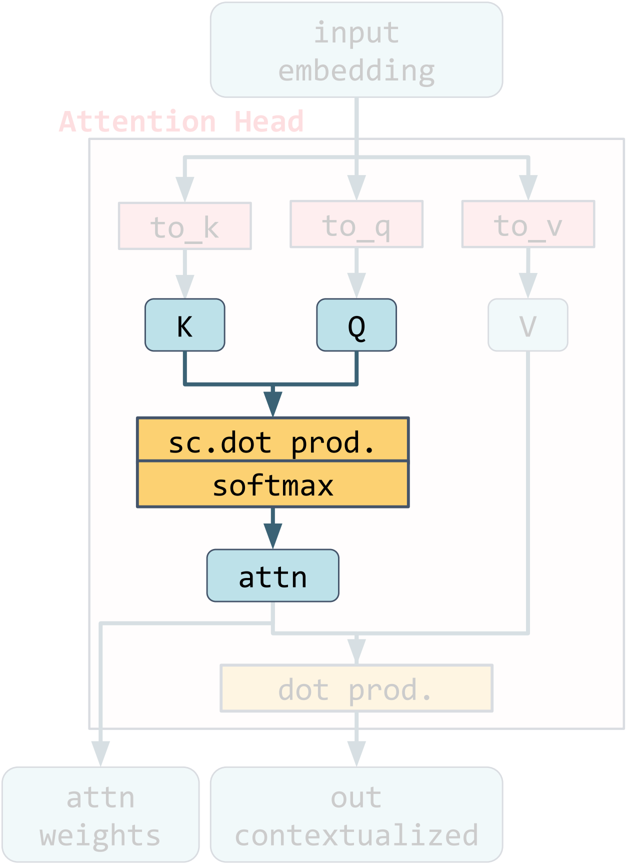
Query
What I'm looking for

Key
What I look like in this space

Value
If you attend to me,
here's what I'll give you

Attention Heads

```
to_k = nn.Linear(4, 4)
to_q = nn.Linear(4, 4)
to_v = nn.Linear(4, 4)

K = to_k(input_embedding)
Q = to_q(input_embedding)
V = to_v(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)

# contextualized output: dot product
out = torch.matmul(attn, V)
return out, attn
```



Query
What I'm looking for

Key
What I look like in this space

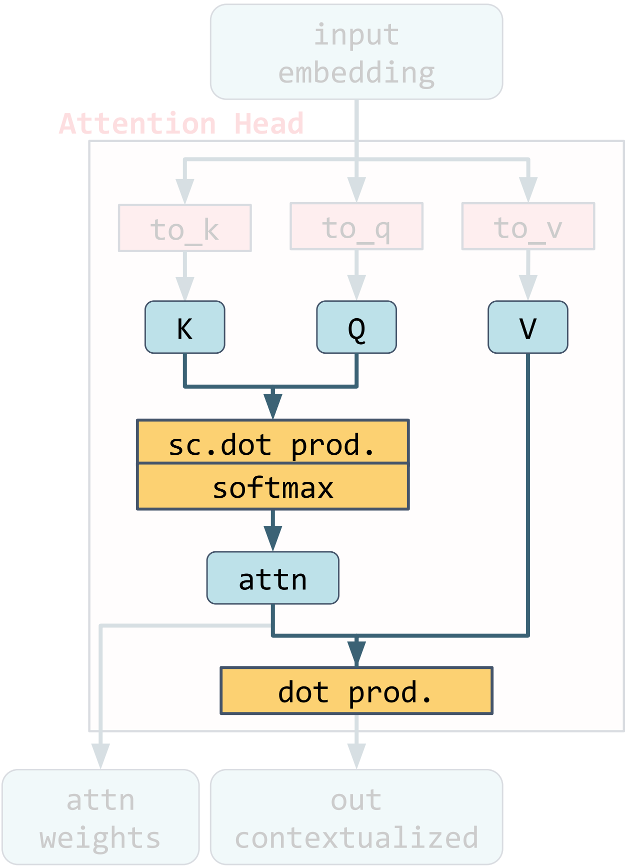
Value
If you attend to me,
here's what I'll give you

DeepLearning.AI

Laurence Moroney

Attention Heads

```
to_k = nn.Linear(4, 4)
to_q = nn.Linear(4, 4)
to_v = nn.Linear(4, 4)

K = to_k(input_embedding)
Q = to_q(input_embedding)
V = to_v(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)

# contextualized output: dot product
out = torch.matmul(attn, V)
return out, attn
```



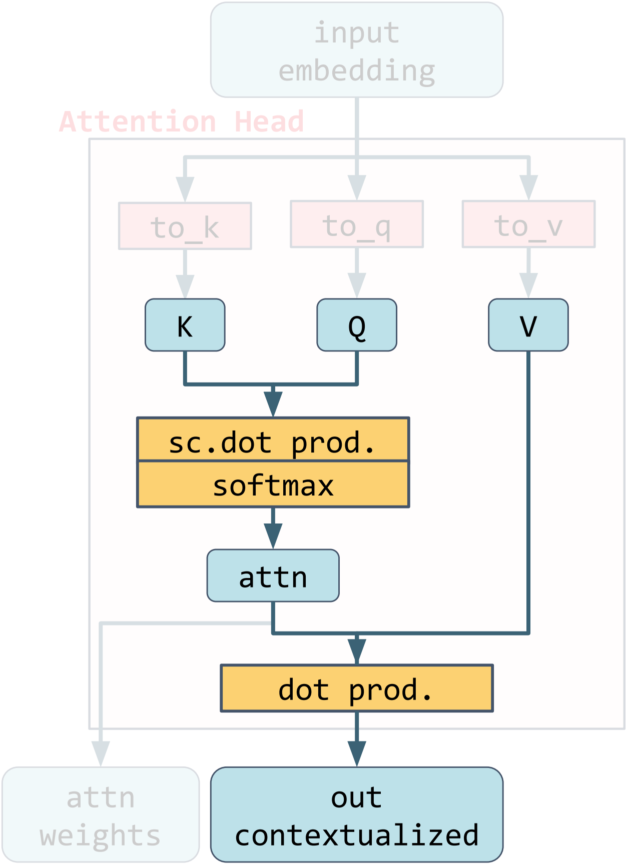
Query
What I'm looking for

Key
What I look like in this space

Value
If you attend to me,
here's what I'll give you

Attention Heads

```
to_k = nn.Linear(4, 4)
to_q = nn.Linear(4, 4)
to_v = nn.Linear(4, 4)

K = to_k(input_embedding)
Q = to_q(input_embedding)
V = to_v(input_embedding)

K_t = K.transpose(-2, 1)
scale = math.sqrt(Q.size(-1))
# KQ compare: scaled dot product
scores = torch.matmul(Q, K_t)/scale

# attn weights: softmax on scaled dp
attn = F.softmax(scores, dim=-1)

# contextualized output: dot product
out = torch.matmul(attn, V)
return out, attn
```



Query
What I'm looking for

Key
What I look like in this space

Value
If you attend to me,
here's what I'll give you

DeepLearning.AI

Laurence Moroney

nn.MultiheadAttention

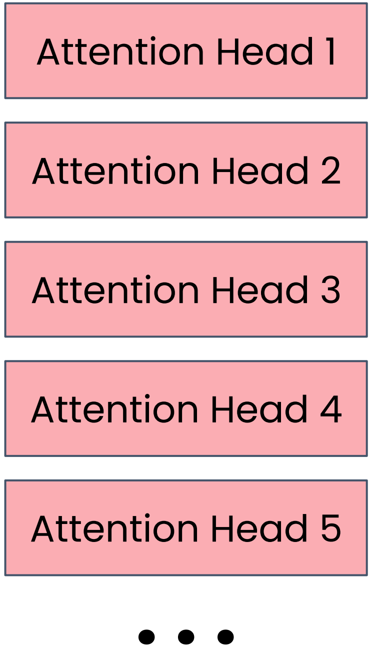
```python
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)
```

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)


torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],    # the
               [-0.44,  0.91, -0.12, -0.77],    # cat
               [ 0.48,  0.02,  0.05,  0.39],    # chased
               [ 0.12, -0.55,  0.33,  0.10],    # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)
```



nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)
```

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)
```

(batch_size, seq_len, embed_dim)

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)


                                    Q  K  V
attn_output, attn_weights = mha(x, x, x)
```

self attention

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)


                                    Q  K  V
attn_output, attn_weights = mha(x, x, x)  cross attention
```

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)


                                Q  K  V
attn_output, attn_weights = mha(x, x, x)  cross attention
```

nn.MultiheadAttention

```
import torch
import torch.nn as nn

mha = nn.MultiheadAttention(embed_dim=4,
                            num_heads=2,
                            dropout=0.1,
                            batch_first=True)


                                Q  K  V
attn_output, attn_weights = mha(x, y, y) cross attention
```

DeepLearning.AI

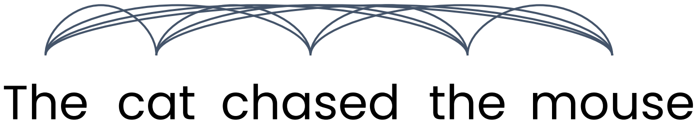
Laurence Moroney

DeepLearning.AI

Encoders

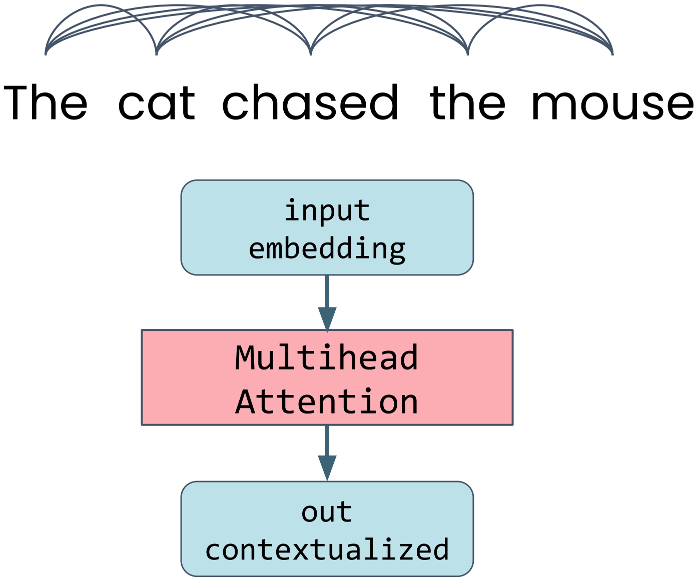
Specialized Approaches to
Natural Language Processing in Pytorch

Attention



The cat chased the mouse

Attention



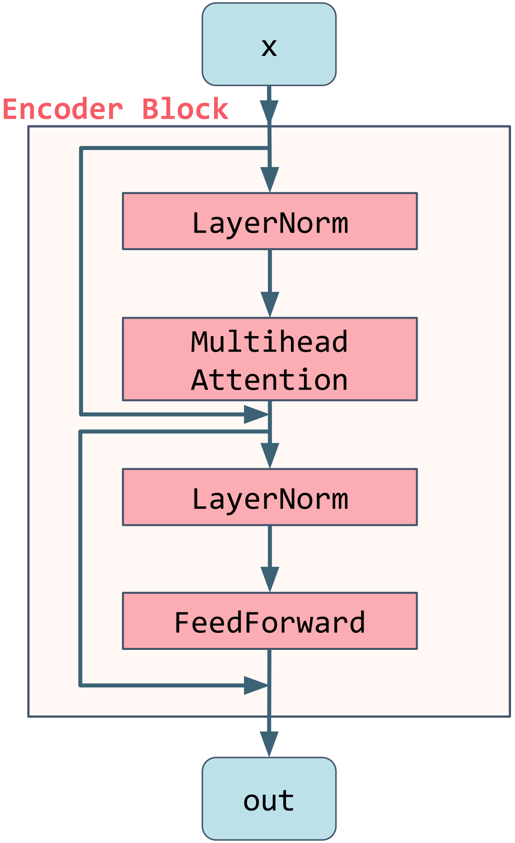
The cat chased the mouse

input embedding

Multihead Attention

out contextualized

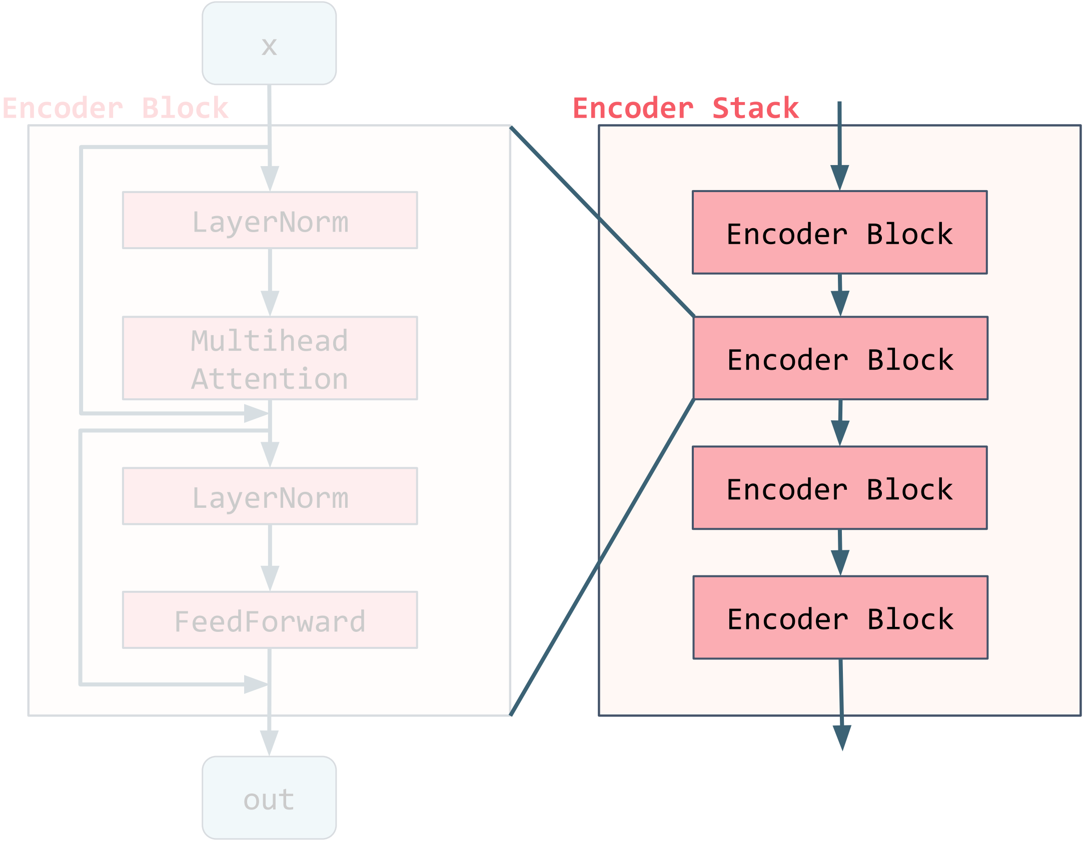
Encoder Block



DeepLearning.AI

Laurence Moroney

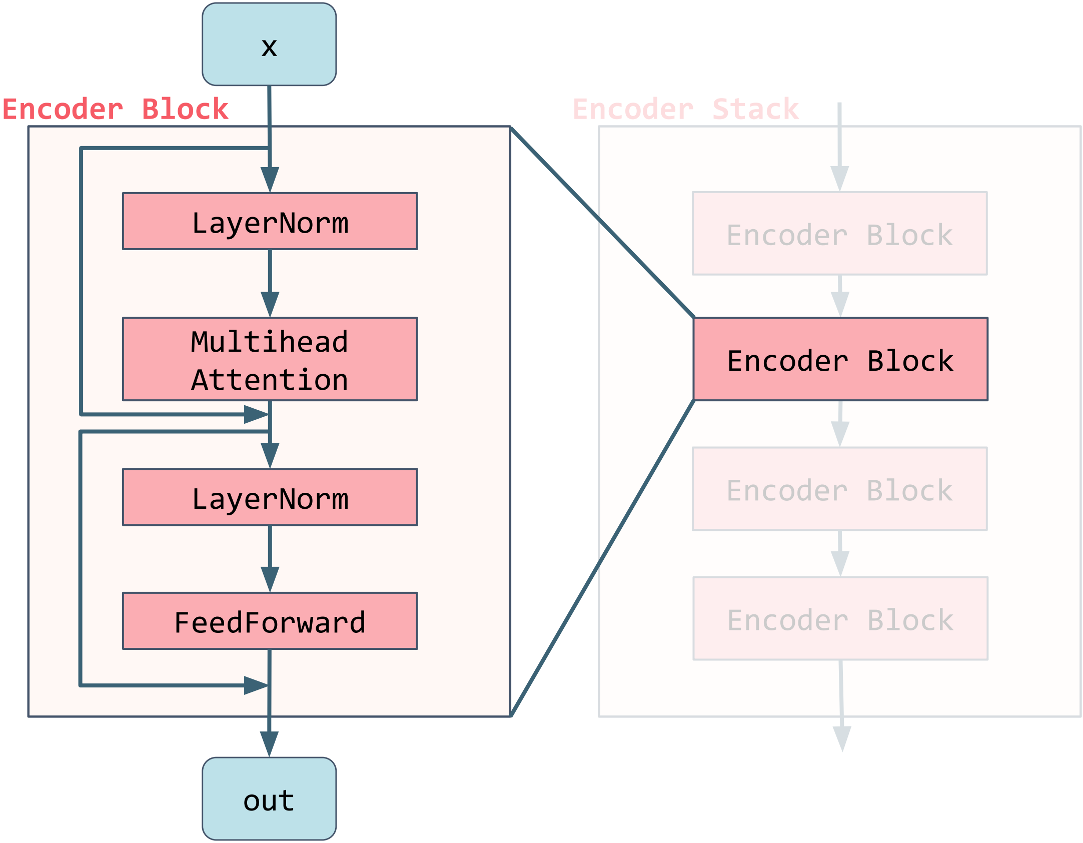
Encoder Block



DeepLearning.AI

Laurence Moroney

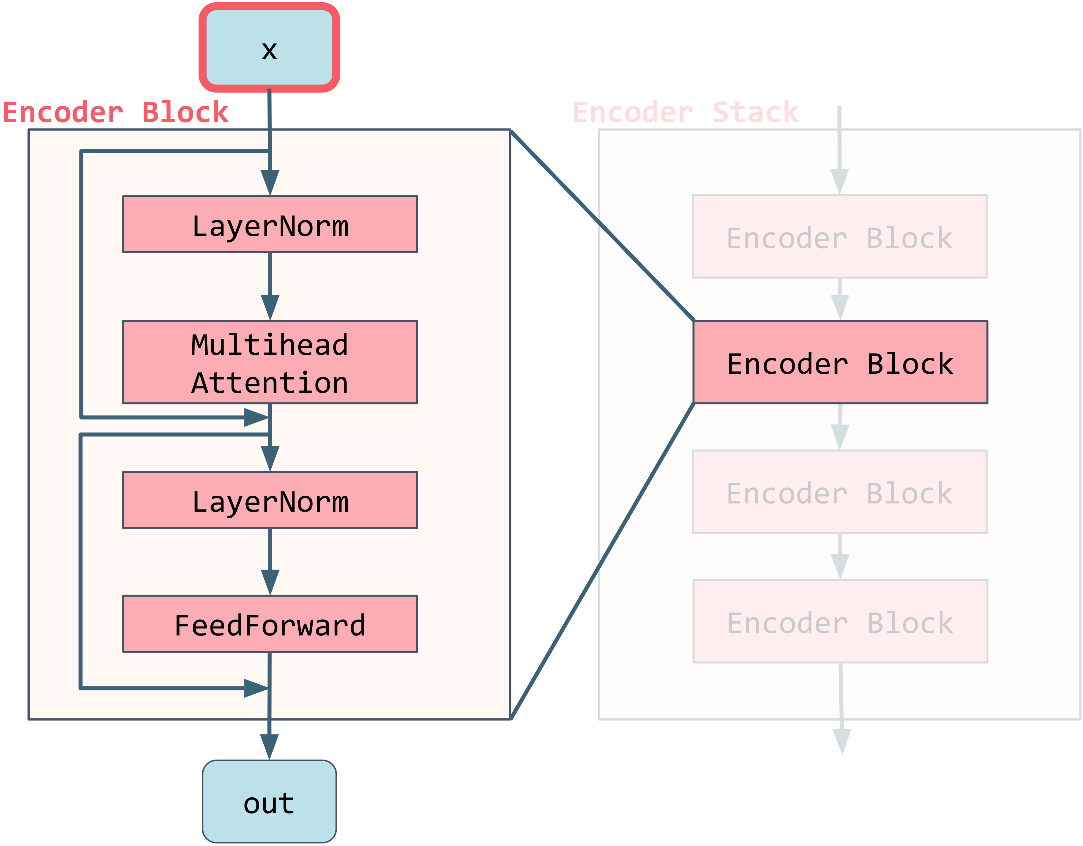
Encoder Block



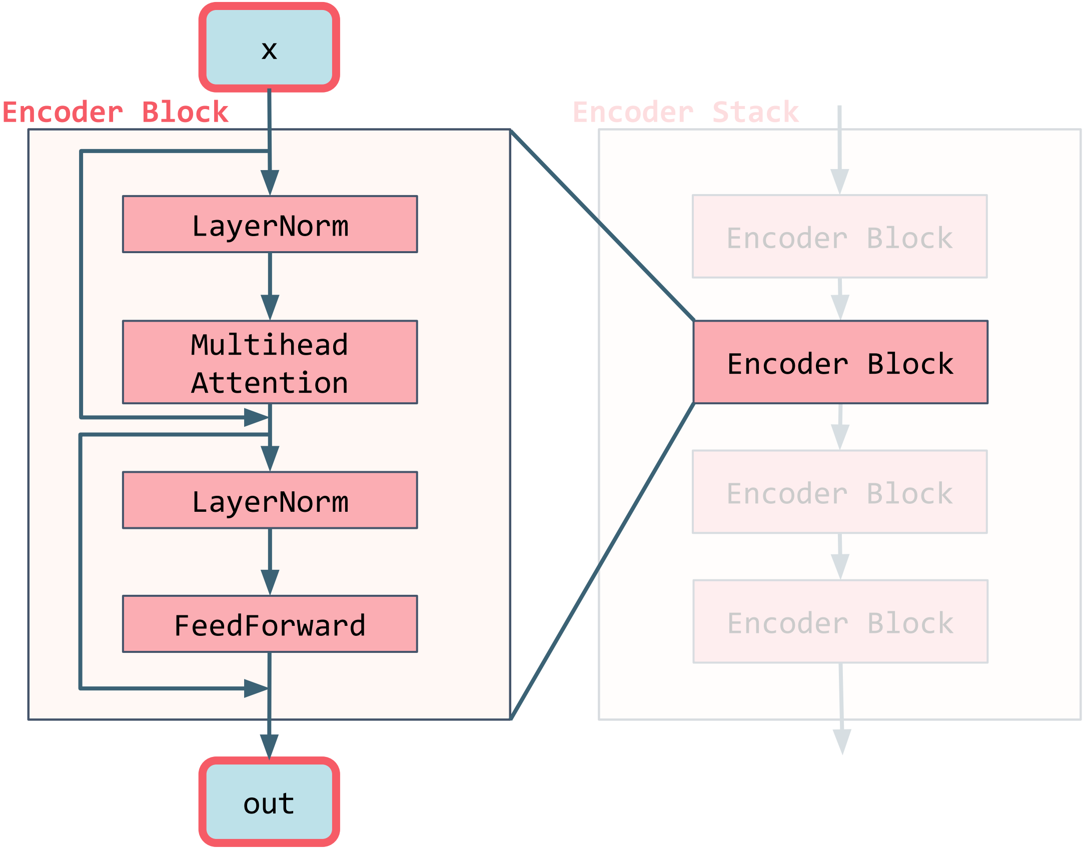
DeepLearning.AI

Laurence Moroney

Encoder Block



Encoder Block



DeepLearning.AI

Laurence Moroney

nn.TransformerEncoder

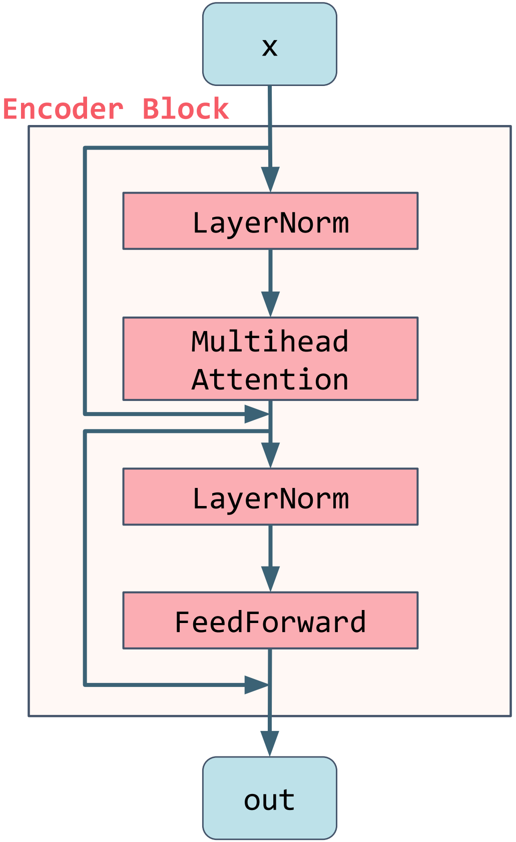
```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)
```

nn.TransformerEncoder

```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)


layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

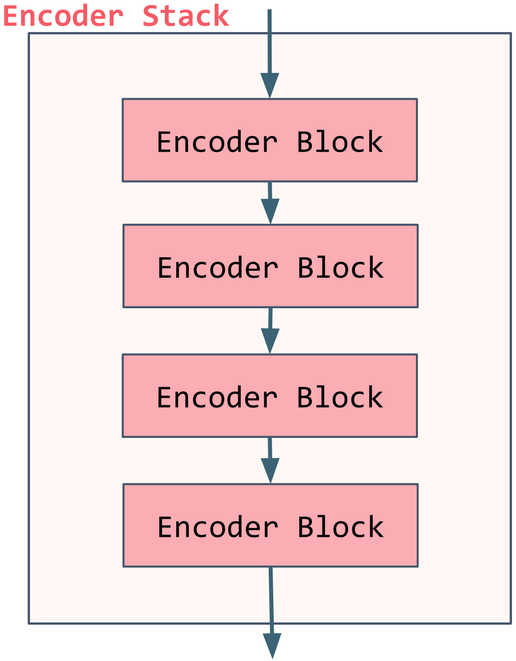
nn.TransformerEncoder

```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)


layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)



stack = nn.TransformerEncoder(layer, num_layers=4)
```



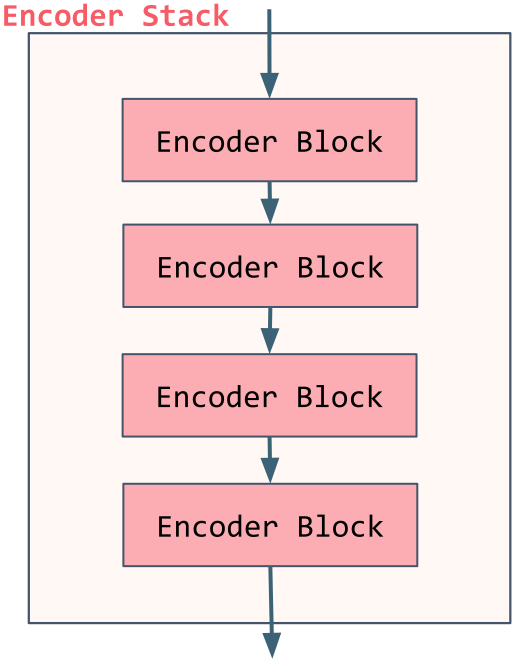
nn.TransformerEncoder

```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)


layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



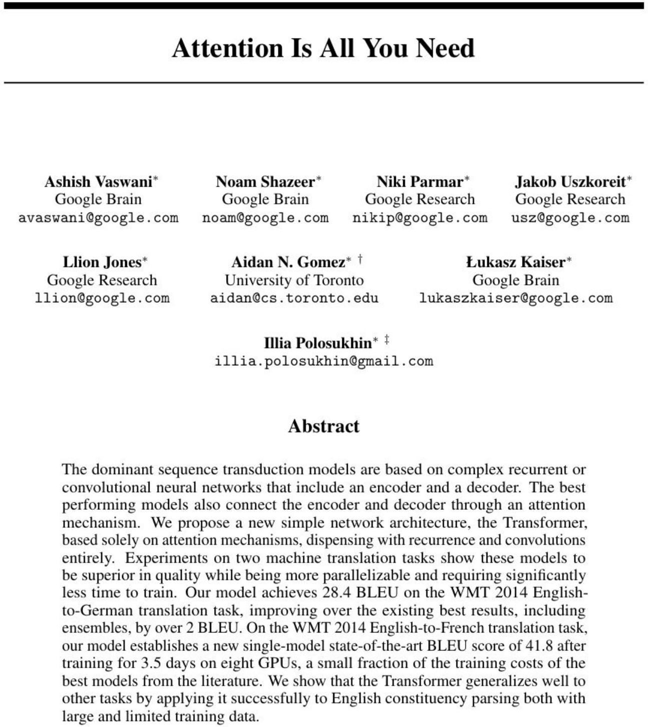
nn.TransformerEncoder

```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)


layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



nn.TransformerEncoder

```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .


layer = nn.TransformerEncoderLaye




stack = nn.TransformerEncoder(lay
```



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

Abstract

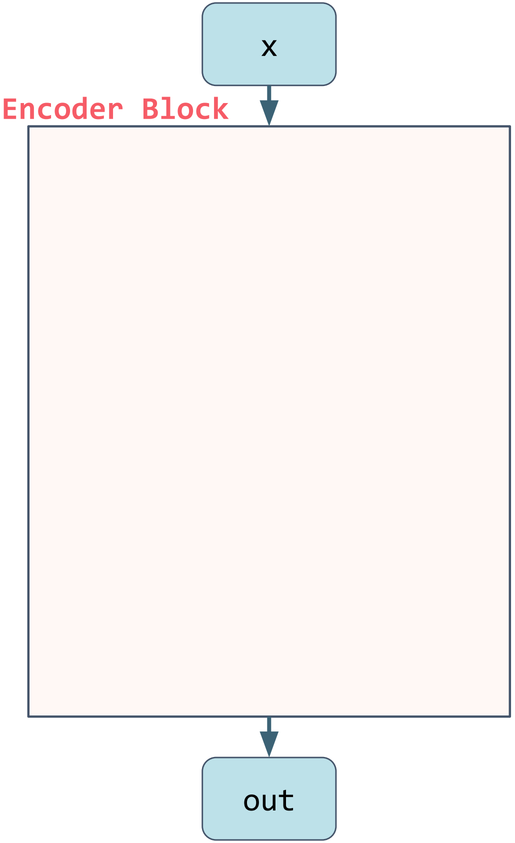
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

nn.TransformerEncoder

```
import torch.nn as nn

mha = nn.MultiheadAttention(. . .)


layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

Encoder Block

```python
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()







    def forward(self, x):
```



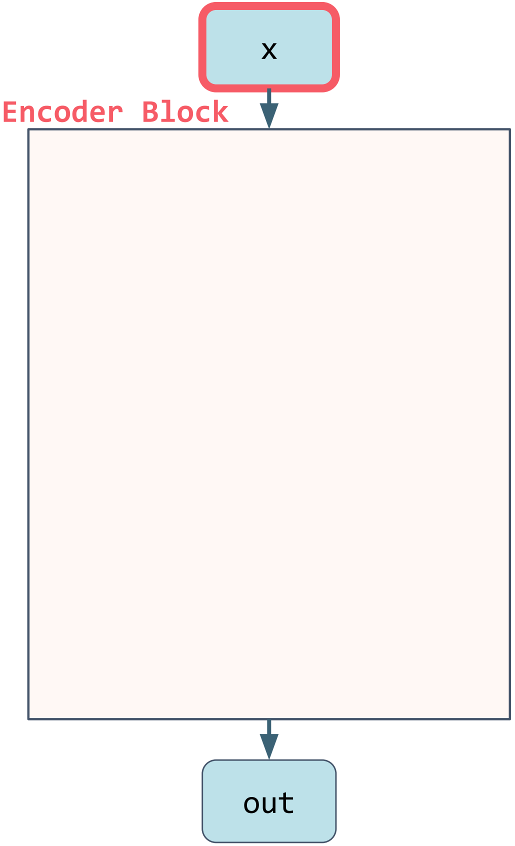
x

Encoder Block

out

Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()







    def forward(self, x):
```

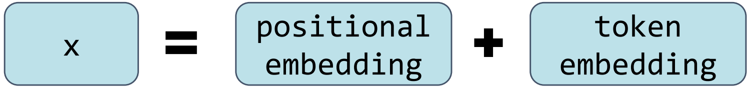


Input

x

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

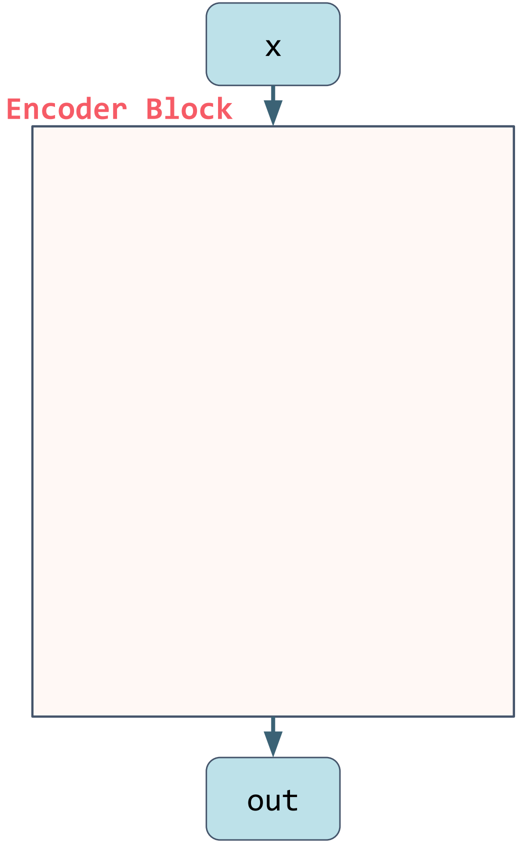
Input



```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
                [-0.44,  0.91, -0.12, -0.77],   # cat
                [ 0.48,  0.02,  0.05,  0.39],   # chased
                [ 0.12, -0.55,  0.33,  0.10],   # the
                [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()






    def forward(self, x):
```

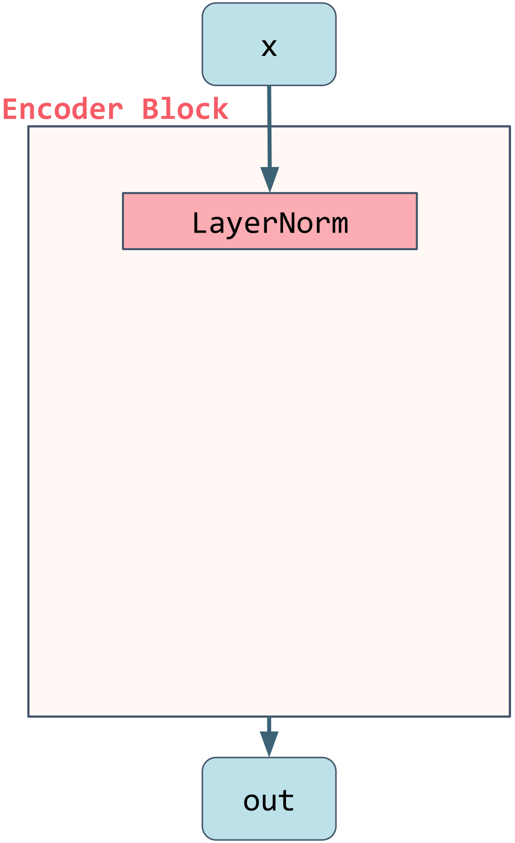


DeepLearning.AI

Laurence Moroney

Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)




    def forward(self, x):
        x_norm = self.ln1(x)
```



DeepLearning.AI

Laurence Moroney

LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.53,  1.61, -0.02, -1.05],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.30,  0.14, -0.70,  0.81]]]) # dog
```

LayerNorm

LayerNorm

Tokens normalized independently

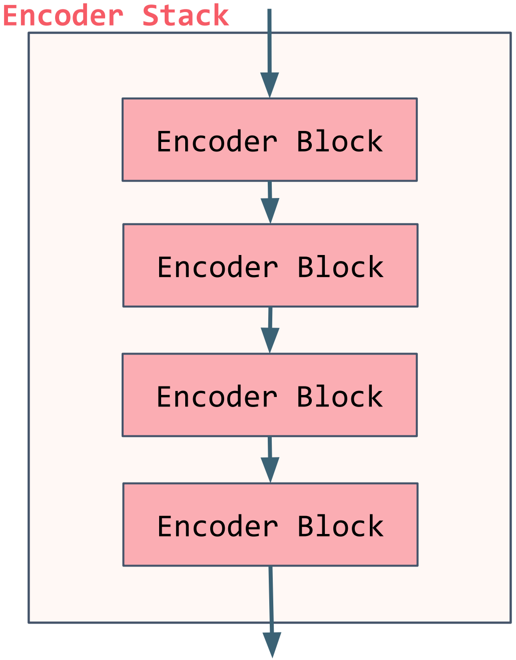
```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.53,  1.61, -0.02, -1.05],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.53,  1.61, -0.02, -1.05],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

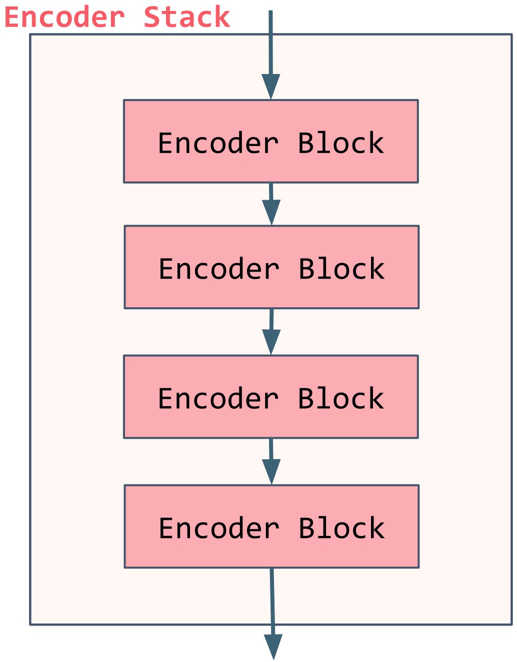


LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.53,  1.61, -0.02, -1.05],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

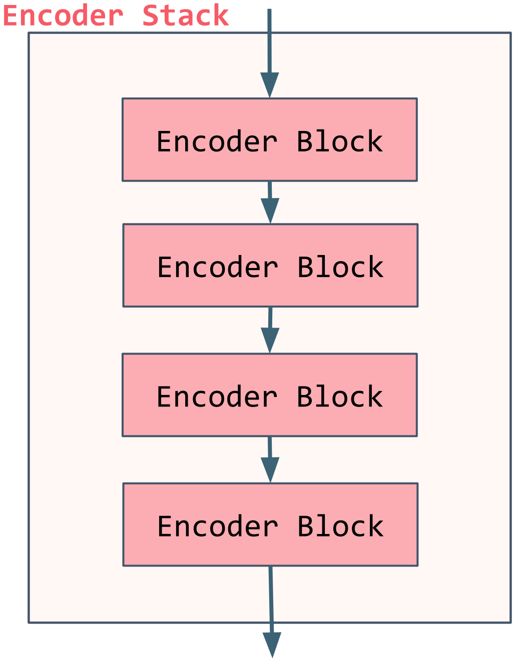


LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [ 12.4,  -57.8,  30.2,  8.9],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

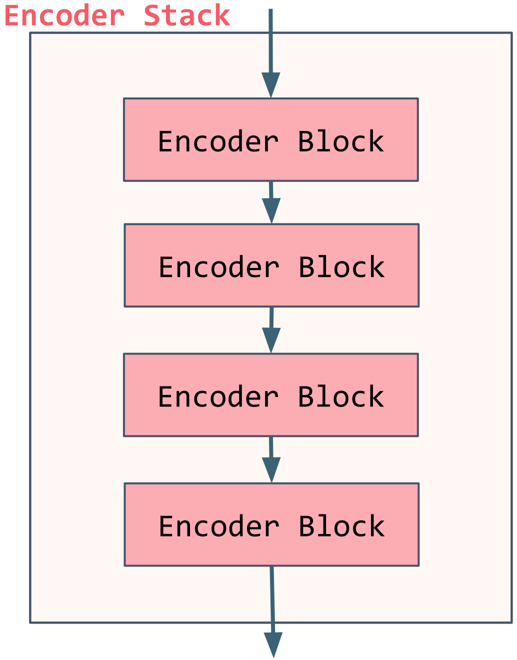


LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [ .001, -.004,   .002,  0.009],  # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

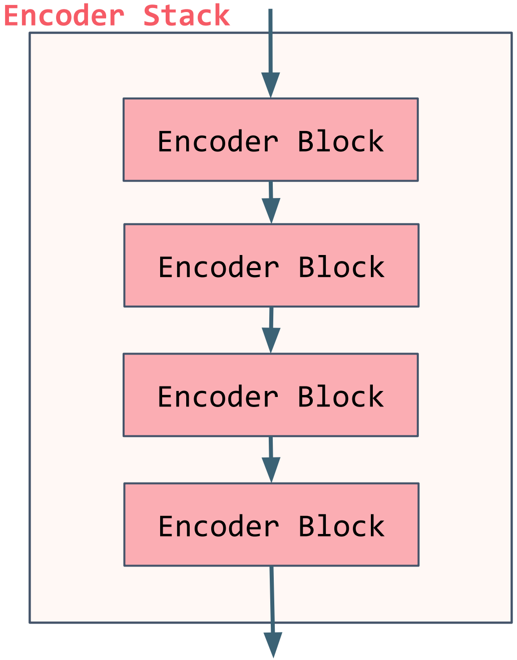


LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

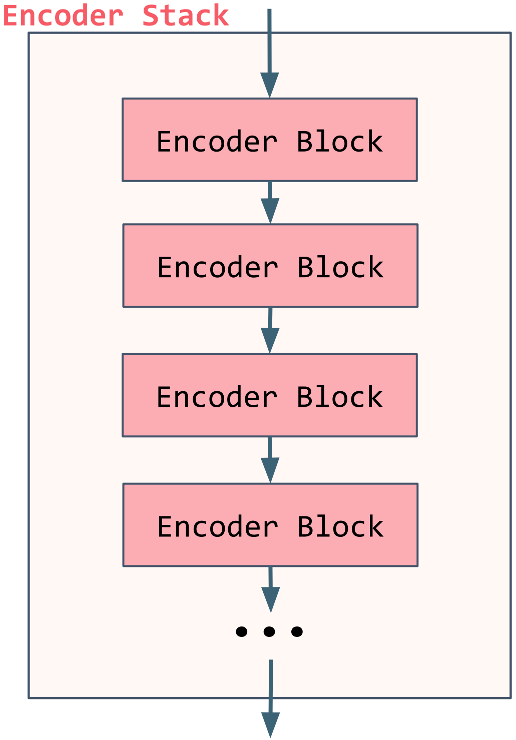


LayerNorm

LayerNorm

Tokens normalized independently

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
                [-0.44,  0.91, -0.12, -0.77],   # cat
                [ 0.48,  0.02,  0.05,  0.39],   # chased
                [ 0.12, -0.55,  0.33,  0.10],   # the
                [-0.51,  0.27, -1.23,  1.47]]]) # dog
```



LayerNorm

LayerNorm

mean: 0 **variance:** 1

```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.44,  0.91, -0.12, -0.77],   # cat
               [ 0.48,  0.02,  0.05,  0.39],   # chased
               [ 0.12, -0.55,  0.33,  0.10],   # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

LayerNorm

LayerNorm

mean: 0 **variance:** 1

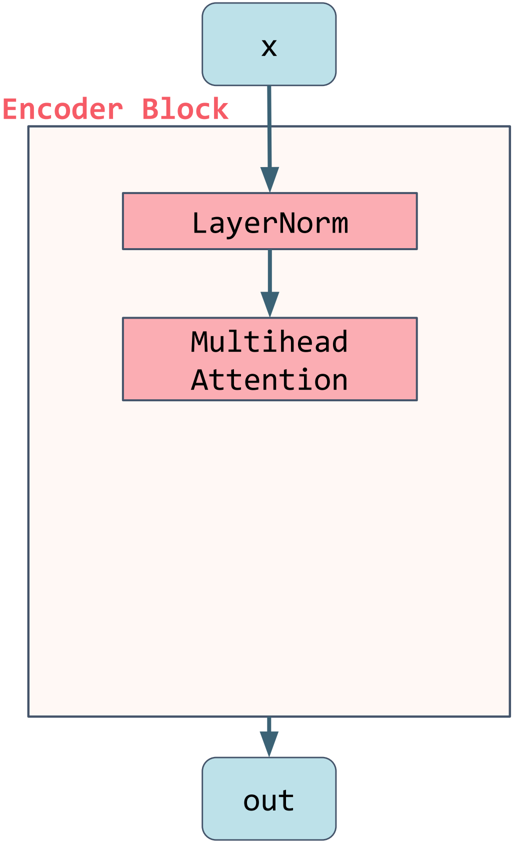
```
torch.tensor([[[ 0.12, -0.55,  0.33,  0.10],    # the
               [-0.44,  0.91, -0.12, -0.77],    # cat
               [ 0.48,  0.02,  0.05,  0.39],    # chased
               [ 0.12, -0.55,  0.33,  0.10],    # the
               [-0.51,  0.27, -1.23,  1.47]]]) # dog
```

scale	γ_1	γ_2	γ_3	γ_4
shift	β_1	β_2	β_3	β_4

$$\gamma_1 \, x_{t,1} + \beta_1$$

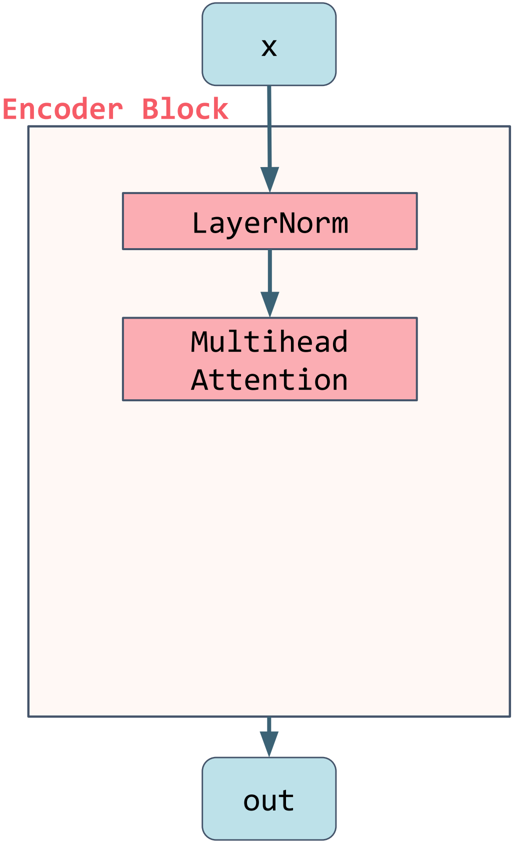
Encoder Block

```python
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)






    def forward(self, x):
        x_norm = self.ln1(x)
```



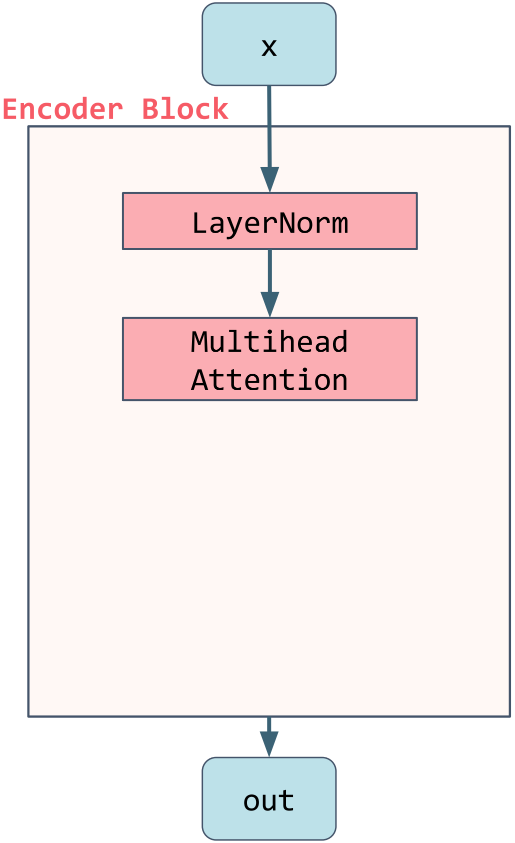
Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)



    def forward(self, x):
        x_norm = self.ln1(x)
```



Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)




    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
```

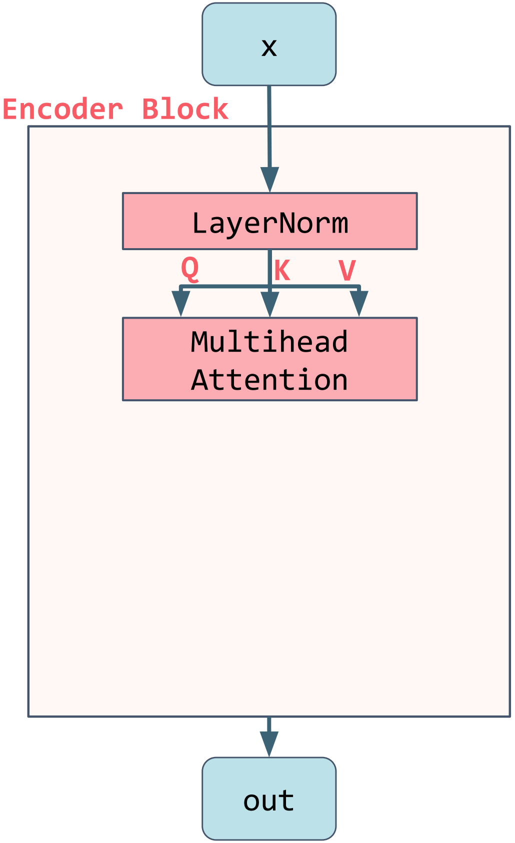


DeepLearning.AI

Laurence Moroney

Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)



    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
```

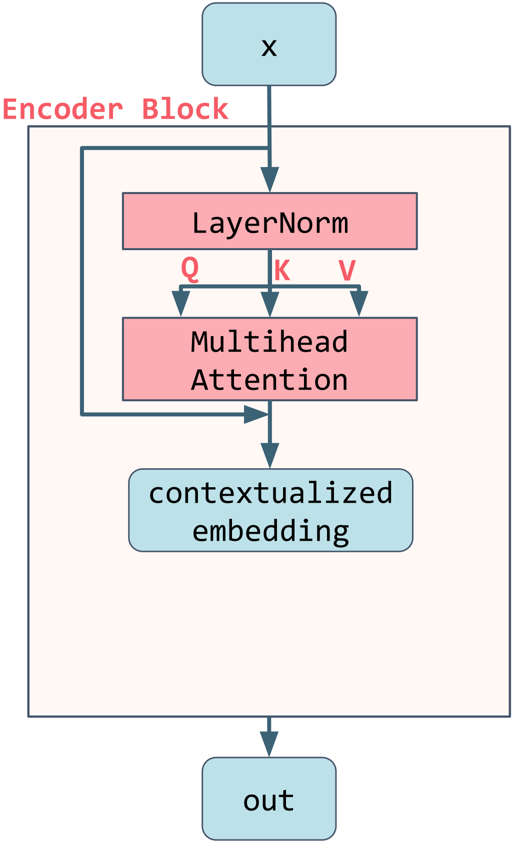


DeepLearning.AI

Laurence Moroney

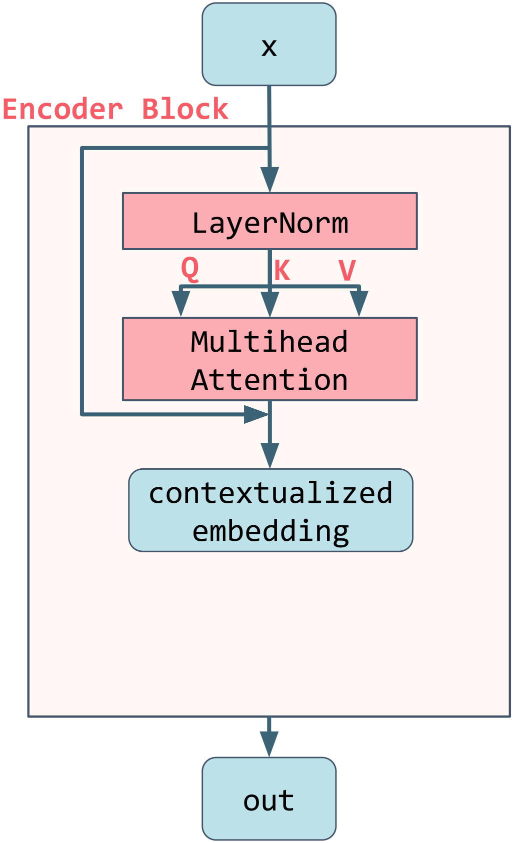
Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)




    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
```



Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)



    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
```

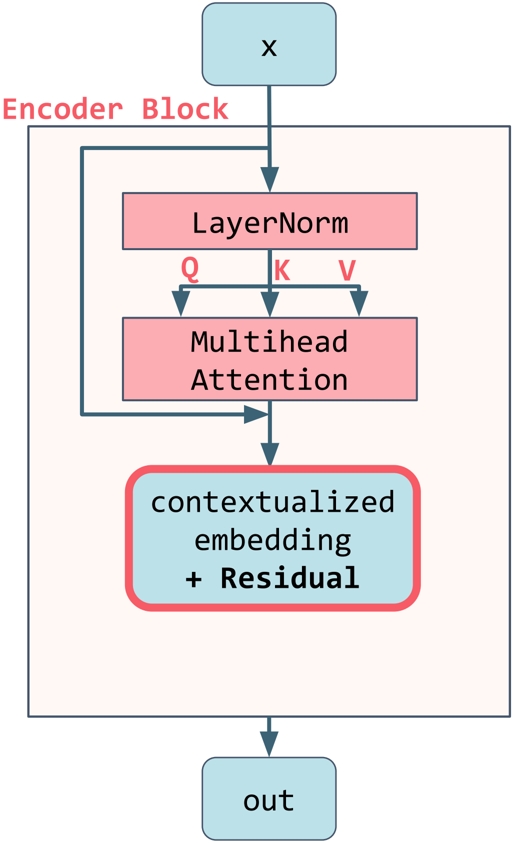


DeepLearning.AI

Laurence Moroney

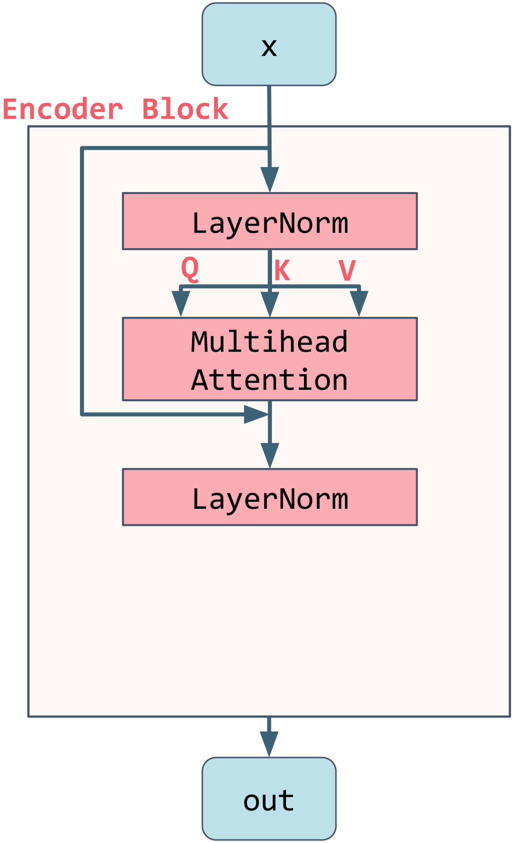
Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)




    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
```



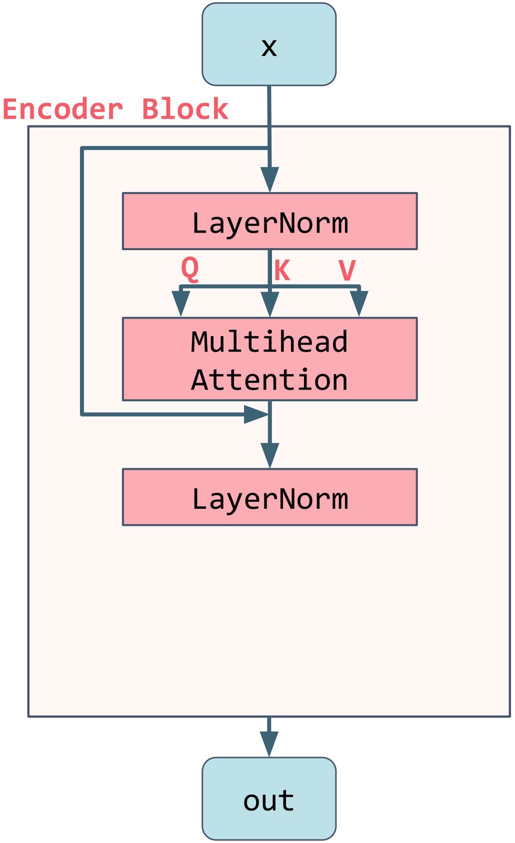
Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)



    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
```



Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)


    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
```

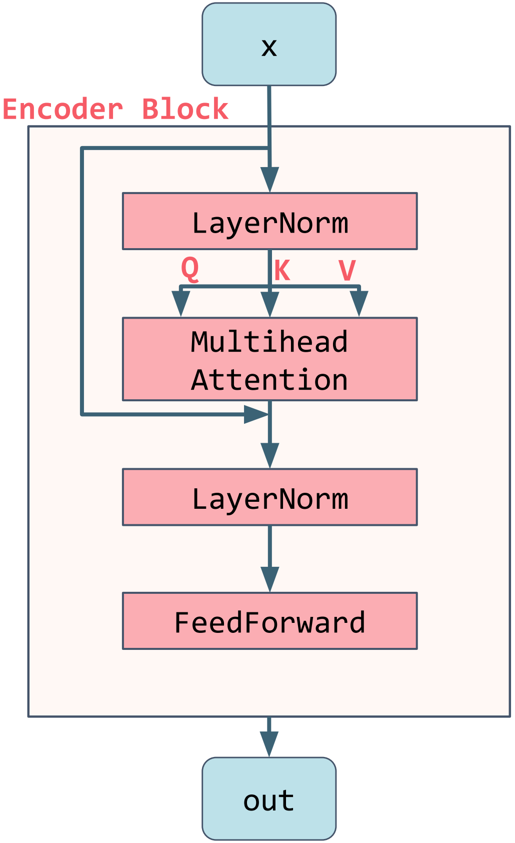


DeepLearning.AI

Laurence Moroney

Encoder Block

```python
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)



    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
```

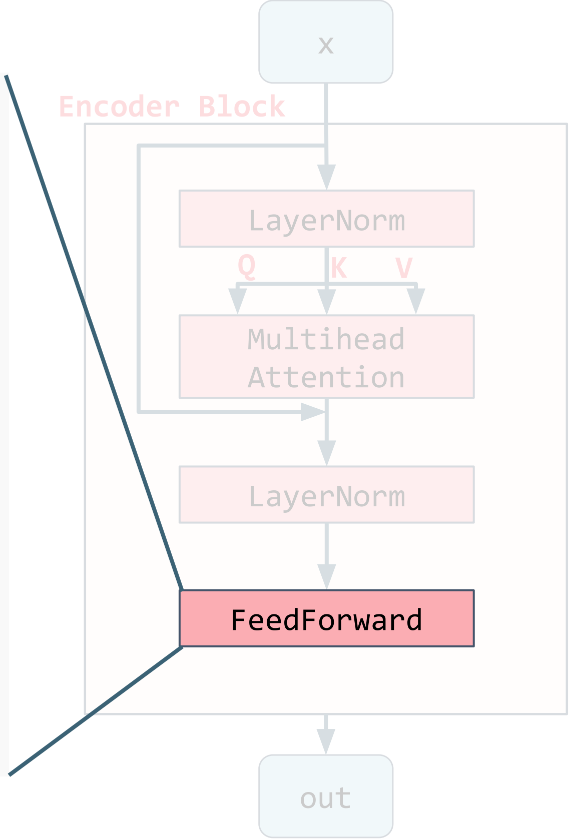


DeepLearning.AI

Laurence Moroney

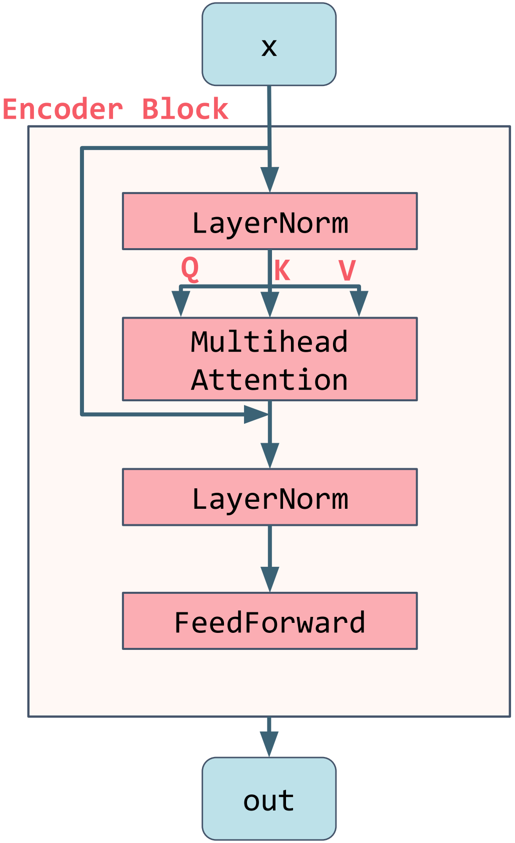
Encoder Block

```
nn.Sequential(
            nn.Linear(d_model, hidden),
            nn.ReLU(),
            nn.Linear(hidden, d_model))
```



Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
```

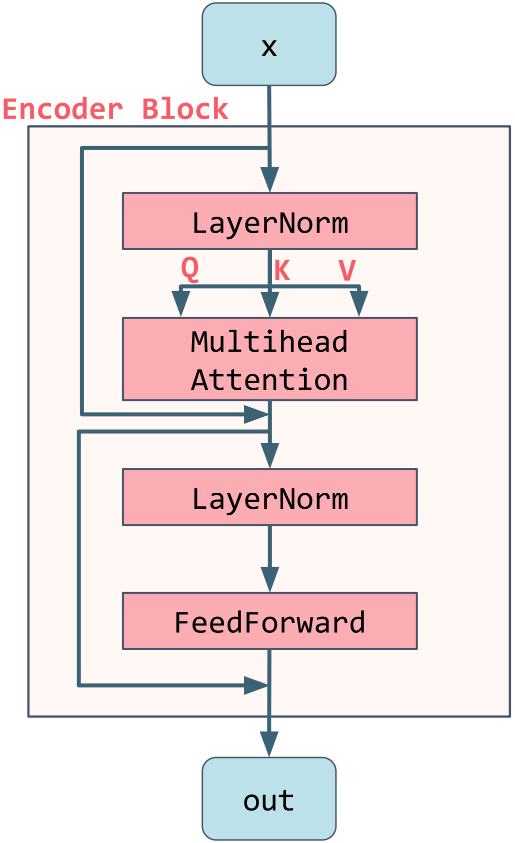


DeepLearning.AI

Laurence Moroney

Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```

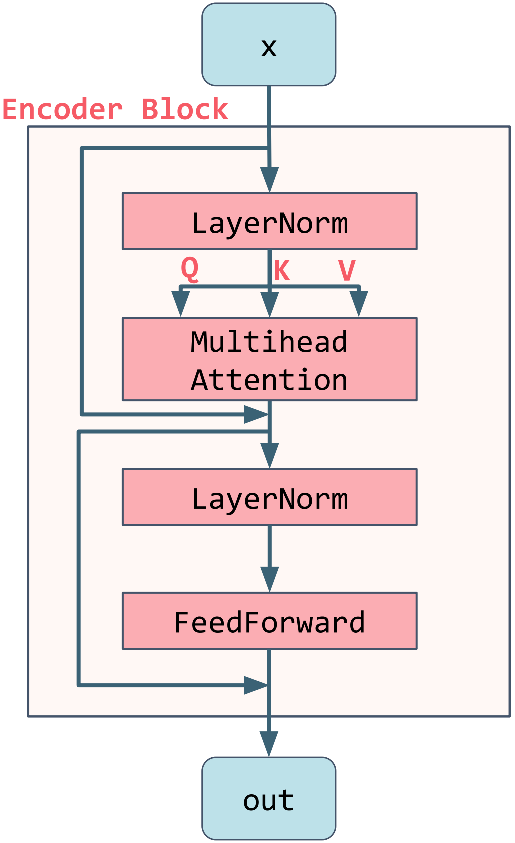


DeepLearning.AI

Laurence Moroney

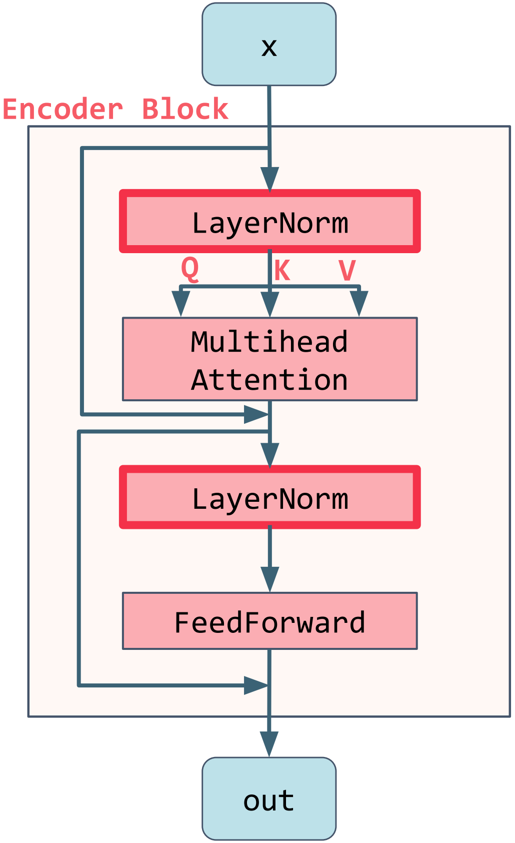
Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```



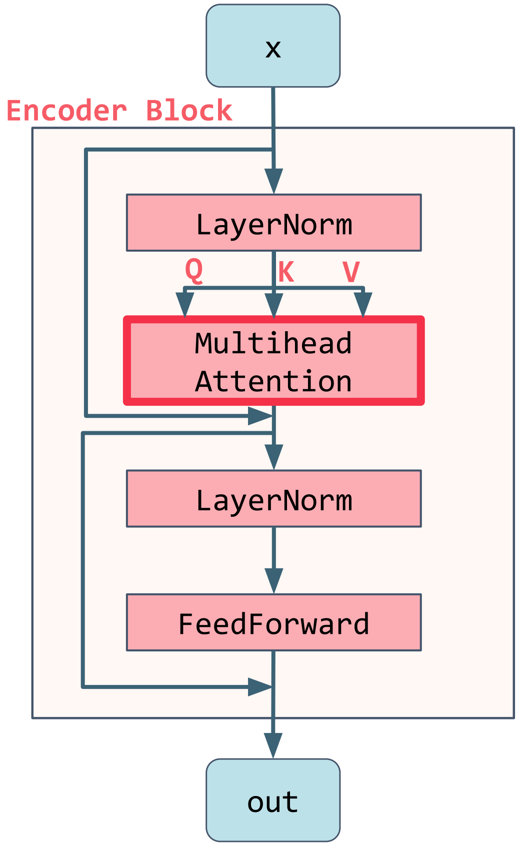
Encoder Block

```python
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```



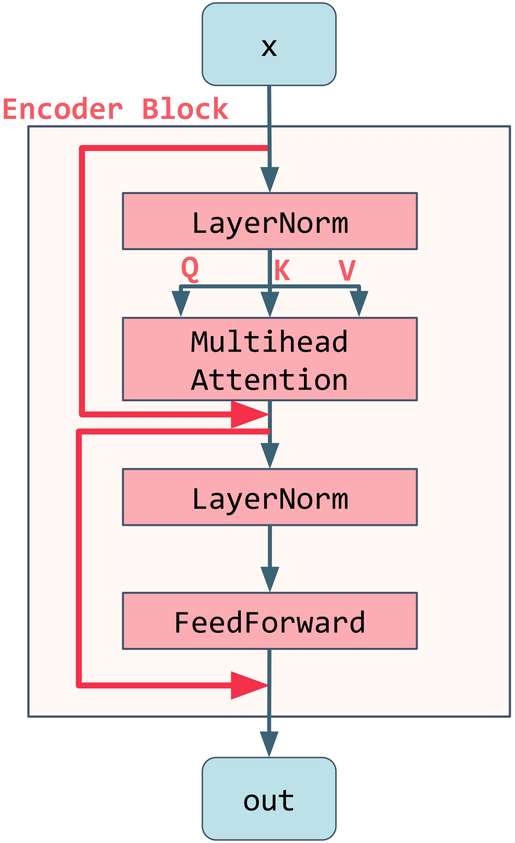
Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```



Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```

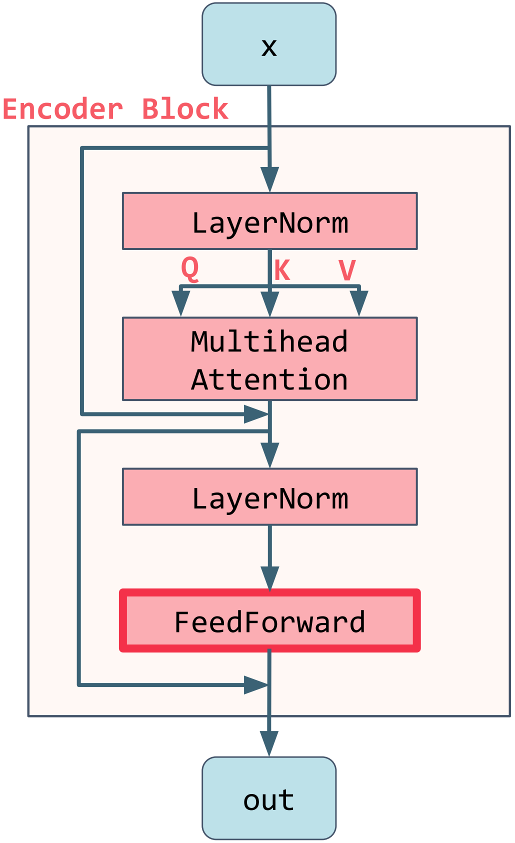


DeepLearning.AI

Laurence Moroney

Encoder Block

```
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```

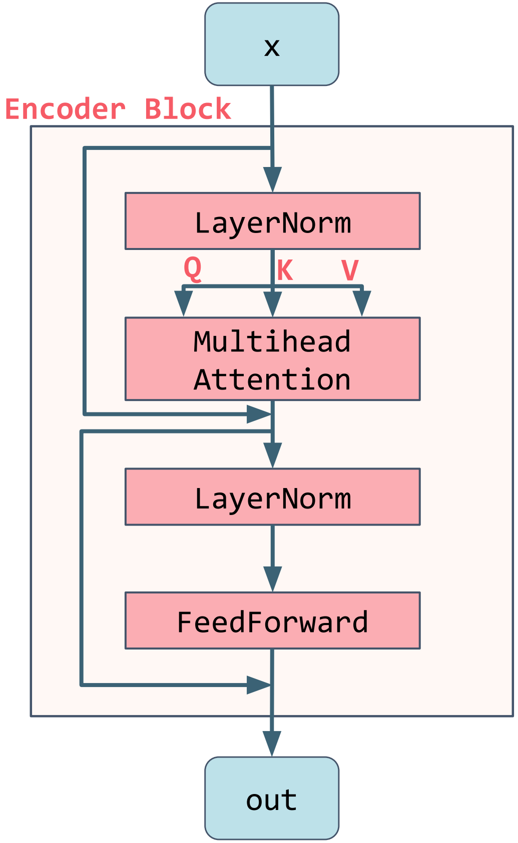


DeepLearning.AI

Laurence Moroney

Encoder Block

```python
class EncoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```



DeepLearning.AI

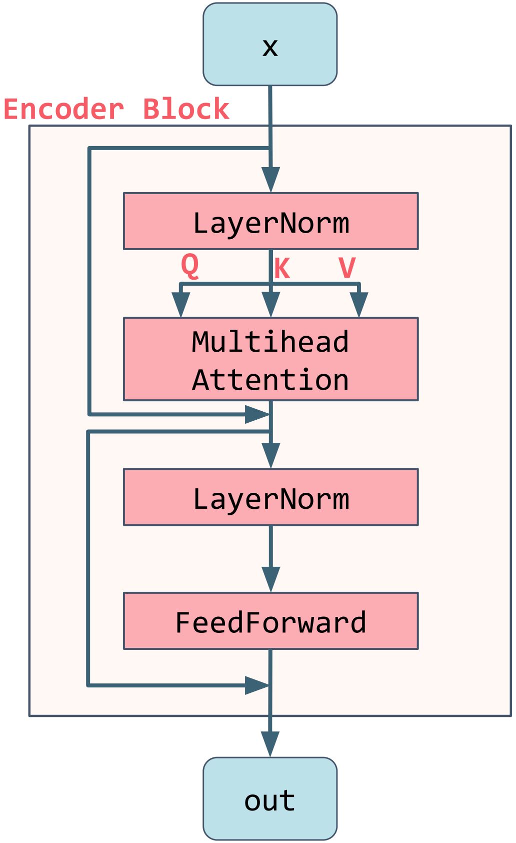
Laurence Moroney

nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

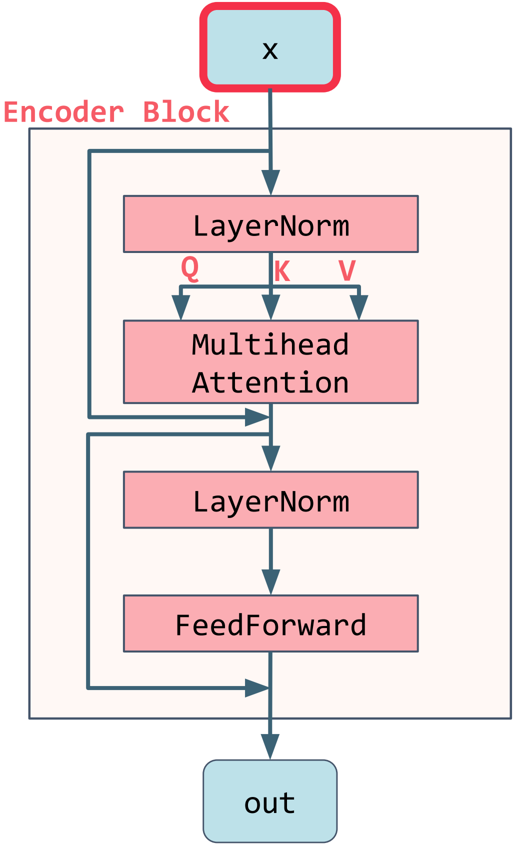


DeepLearning.AI

Laurence Moroney

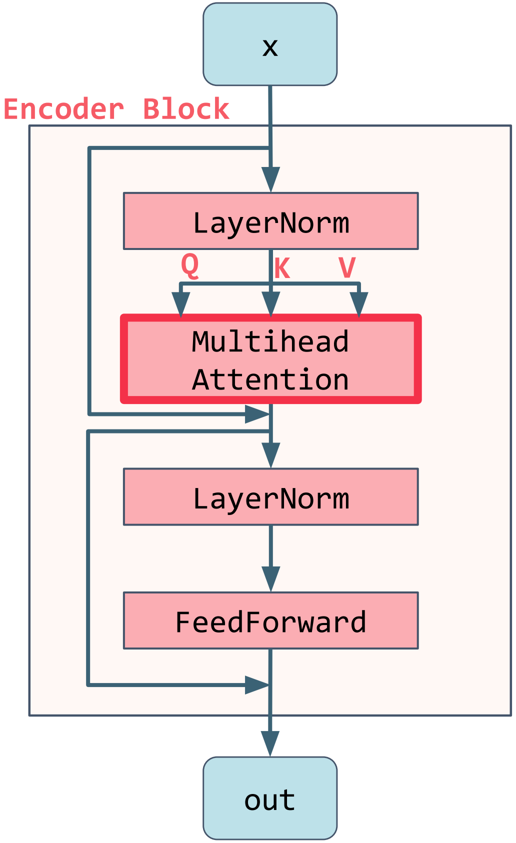
nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



nn.TransformerEncoder

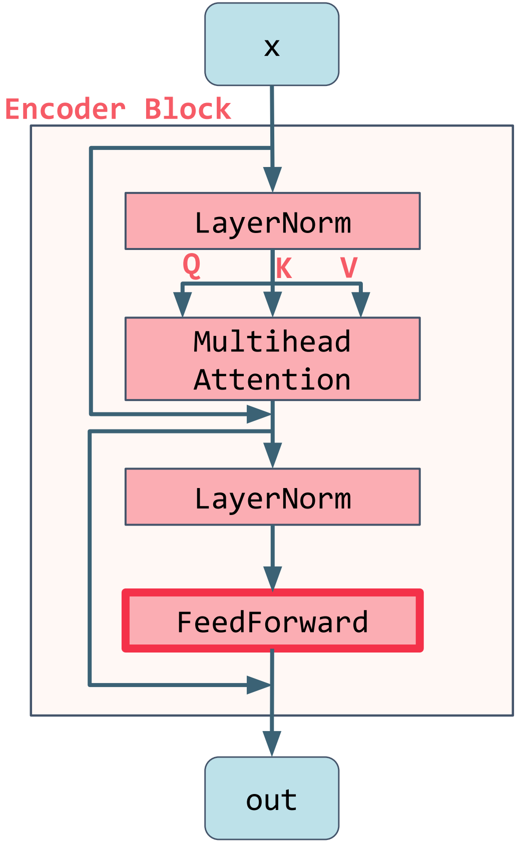
```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



Laurence Moroney

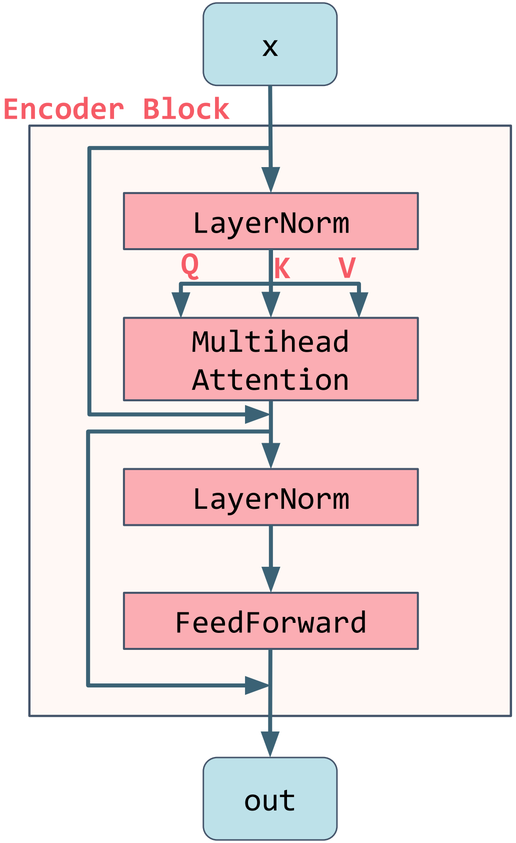
nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



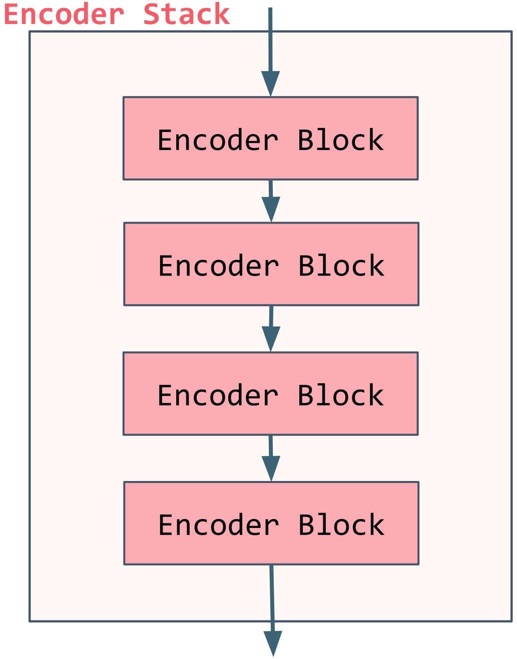
nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

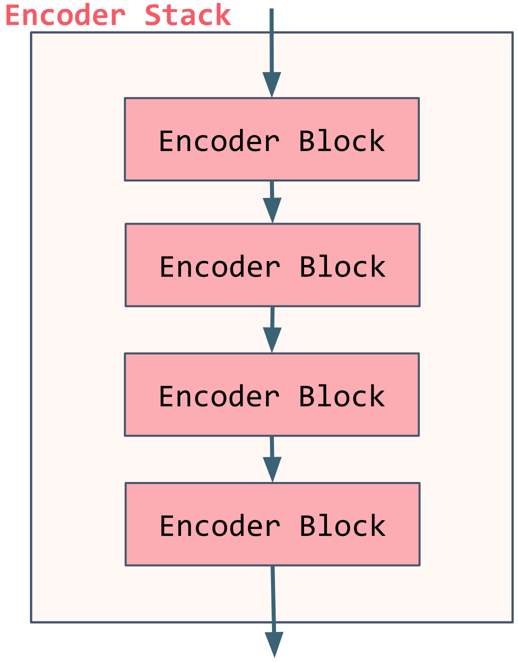


DeepLearning.AI

Laurence Moroney

nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

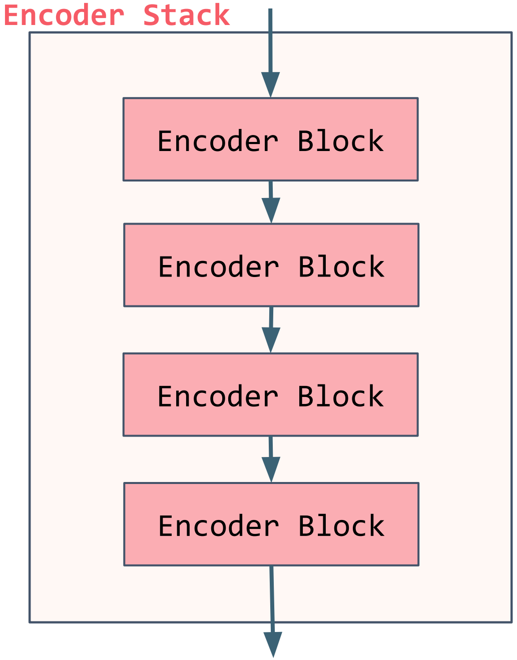


DeepLearning.AI

Laurence Moroney

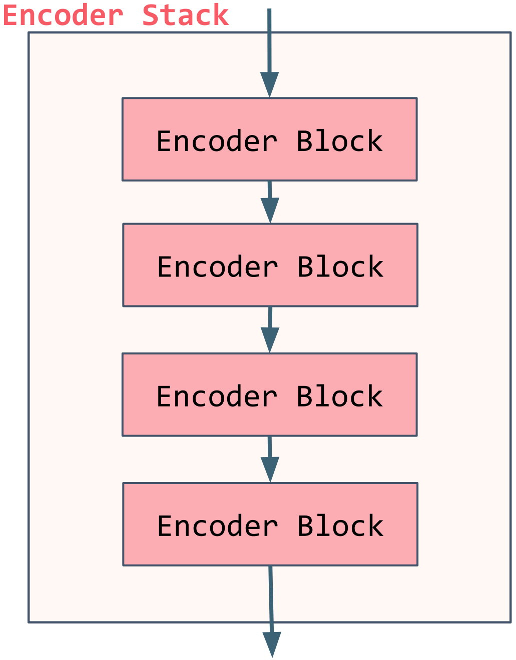
nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```

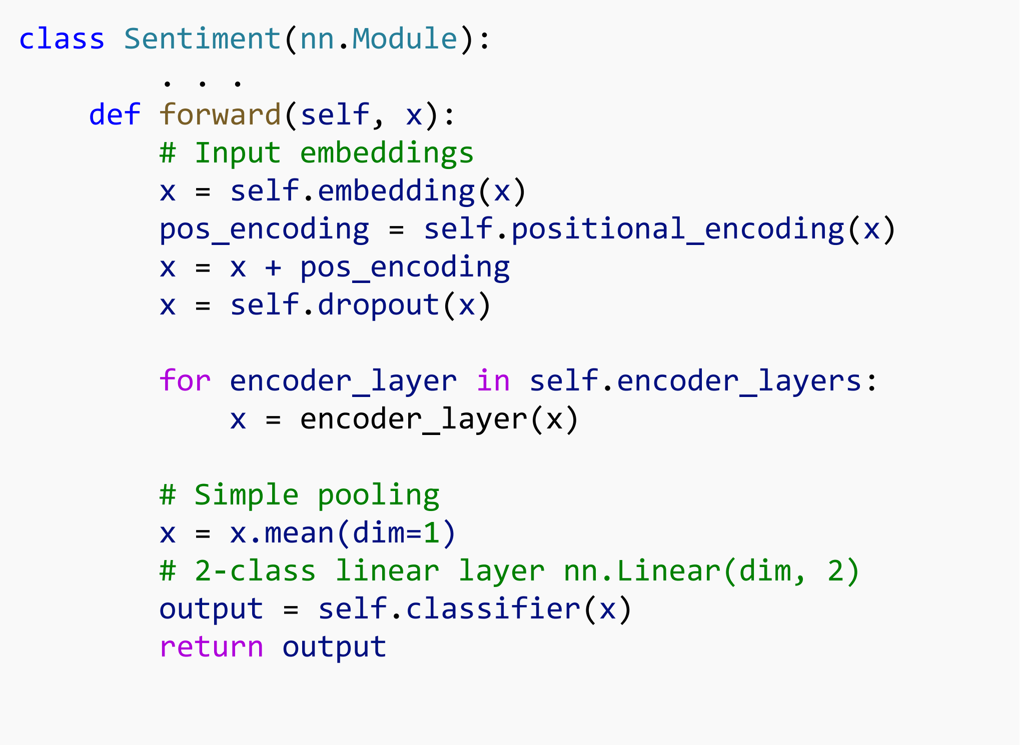


nn.TransformerEncoder

```
layer = nn.TransformerEncoderLayer(d_model=768,
                                   nhead=12,
                                   dim_feedforward=3072,
                                   dropout=0.1,
                                   batch_first=True)


stack = nn.TransformerEncoder(layer, num_layers=4)
```



Sentiment Analysis



```
class Sentiment(nn.Module):
        . . .
    def forward(self, x):
        # Input embeddings
        x = self.embedding(x)
        pos_encoding = self.positional_encoding(x)
        x = x + pos_encoding
        x = self.dropout(x)

        for encoder_layer in self.encoder_layers:
            x = encoder_layer(x)

        # Simple pooling
        x = x.mean(dim=1)
        # 2-class linear layer nn.Linear(dim, 2)
        output = self.classifier(x)
        return output
```

Sentiment Analysis

```
class Sentiment(nn.Module):
        . . .
    def forward(self, x):
        # Input embeddings
        x = self.embedding(x)
        pos_encoding = self.positional_encoding(x)
        x = x + pos_encoding
        x = self.dropout(x)

        for encoder_layer in self.encoder_layers:
            x = encoder_layer(x)

        # Simple pooling
        x = x.mean(dim=1)
        # 2-class linear layer nn.Linear(dim, 2)
        output = self.classifier(x)
        return output
```

Sentiment Analysis

```
class Sentiment(nn.Module):
        . . .
    def forward(self, x):
        # Input embeddings
        x = self.embedding(x)
        pos_encoding = self.positional_encoding(x)
        x = x + pos_encoding
        x = self.dropout(x)

        for encoder_layer in self.encoder_layers:
            x = encoder_layer(x)

        # Simple pooling
        x = x.mean(dim=1)
        # 2-class linear layer nn.Linear(dim, 2)
        output = self.classifier(x)
        return output
```

“I really loved this movie — the acting was fantastic!”

POSITIVE

Transformer Architectures

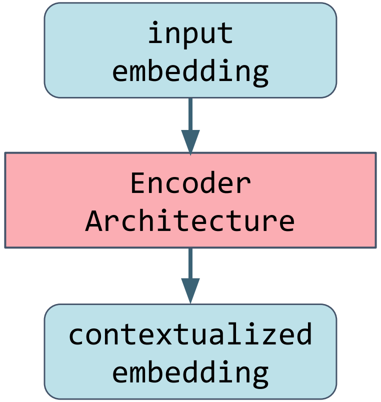
- **Encoders:**
  - Understanding text
  - Classification
  - Named entity recognition
  - Question answering

DeepLearning.AI

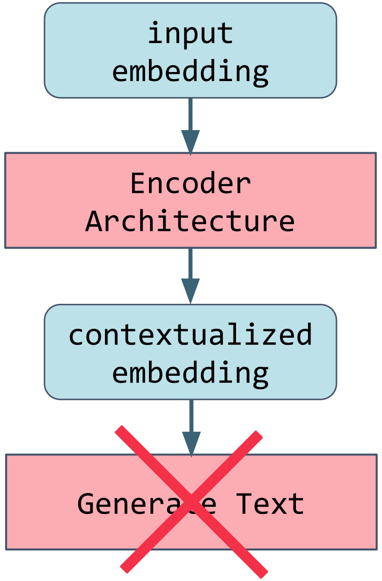
Decoders

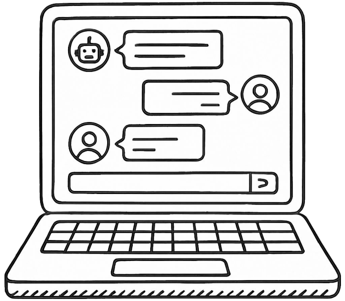
Specialized Approaches to
Natural Language Processing in Pytorch

Encoders


input embedding
Encoder Architecture
contextualized embedding

Encoders



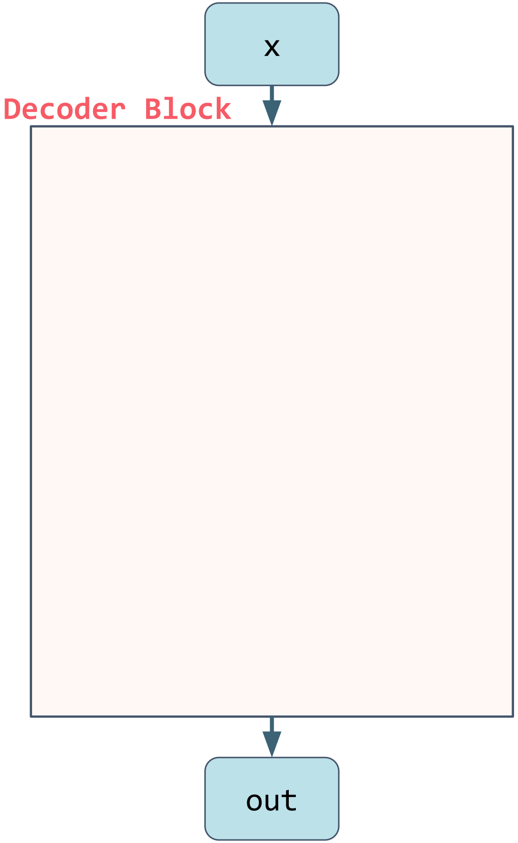


DeepLearning.AI

Laurence Moroney

Decoder Block

```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()







    def forward(self, x):
```



Predicting Tokens

The cat chased the _____

Predicting Tokens

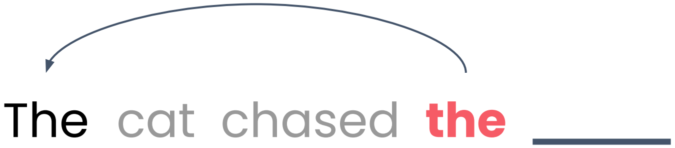
?

The cat chased **the** _____

Predicting Tokens

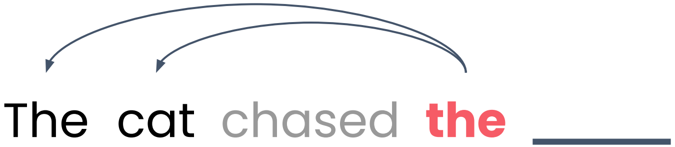
The cat chased **the** ______

Predicting Tokens

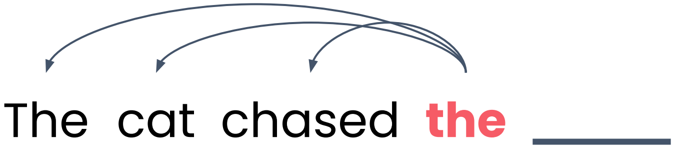


The cat chased **the** _____

Predicting Tokens



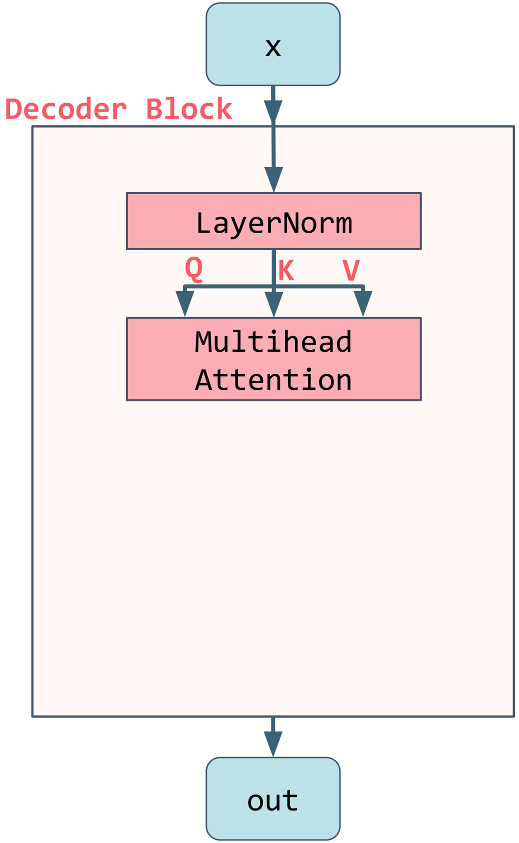
Predicting Tokens



The cat chased **the** ______

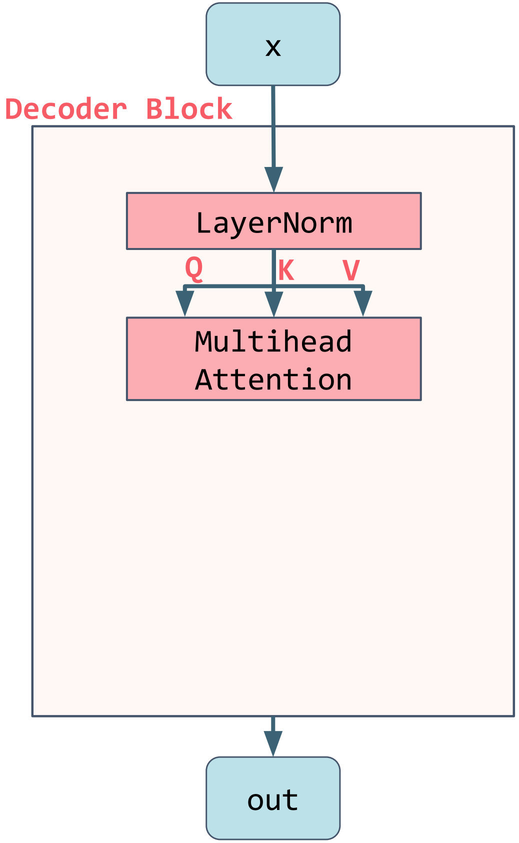
Decoder Block

```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()







    def forward(self, x):
```



Decoder Block

```python
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)


    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
```

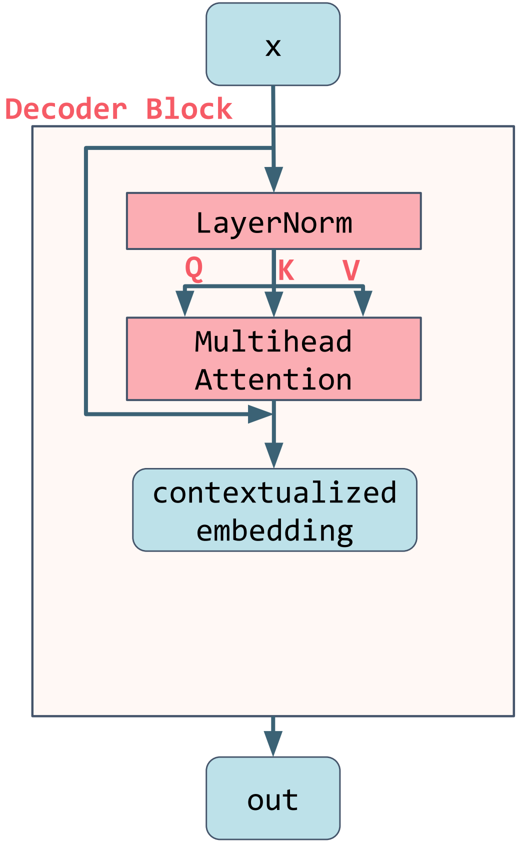


DeepLearning.AI

Laurence Moroney

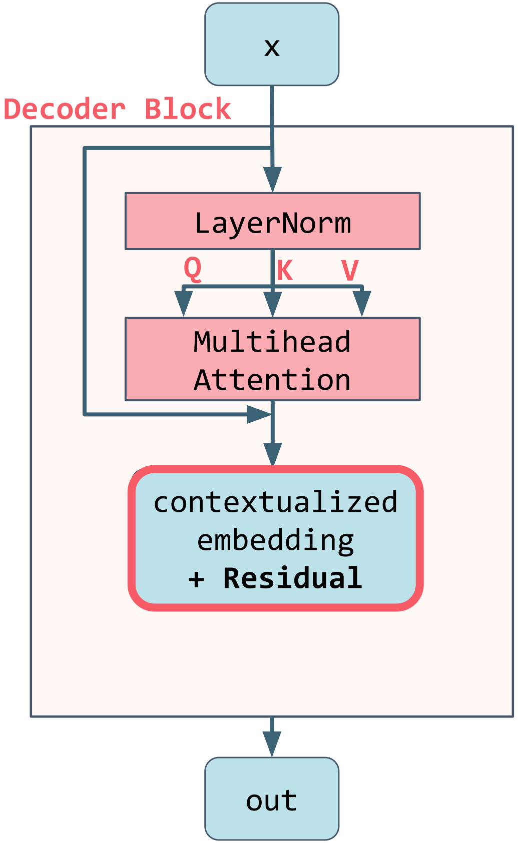
Decoder Block

```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)




    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
```



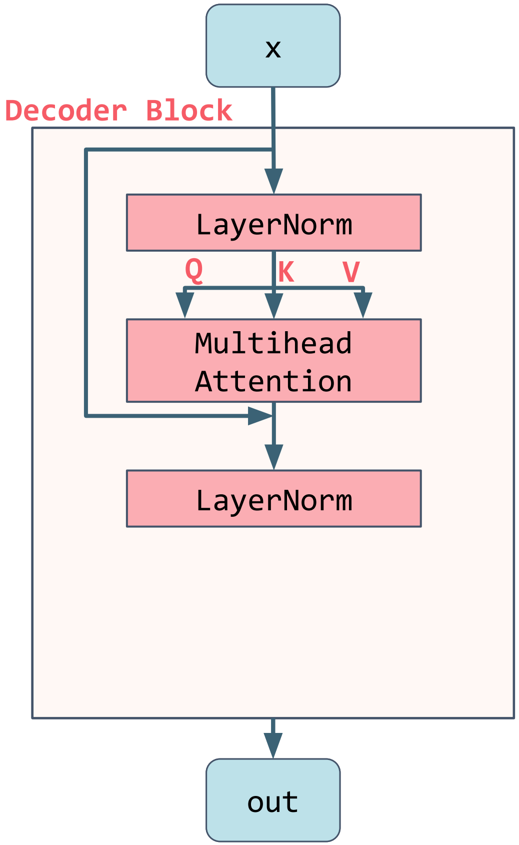
Decoder Block

```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)




    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
```



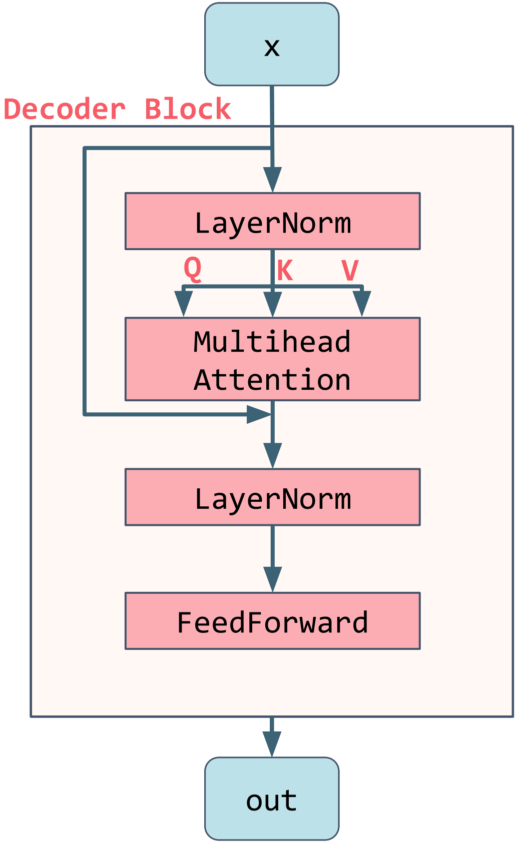
Decoder Block

```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)



    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
```



Decoder Block

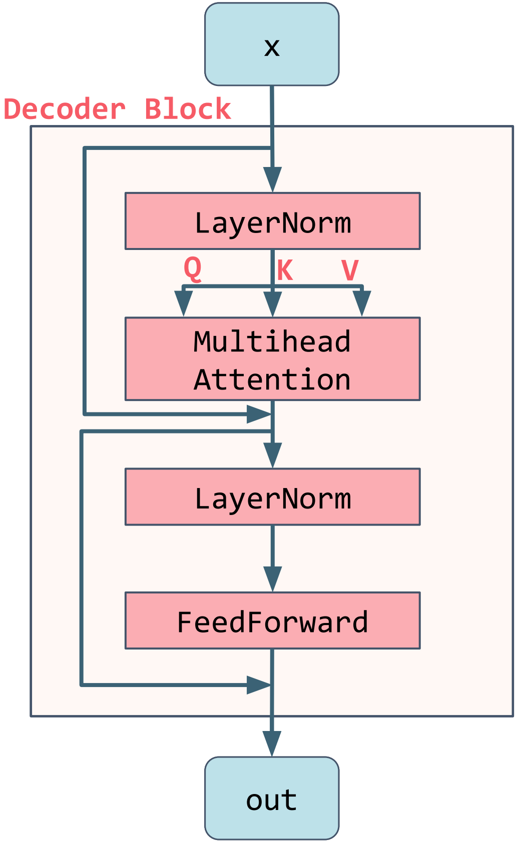
```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
```



Laurence Moroney

Decoder Block

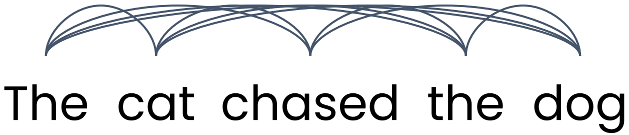
```
class DecoderBlock(nn.Module):
    def __init__(self, d_model=4, nhead=1, ffn_mult=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.mha = nn.MultiheadAttention(. . .)
        self.ln2 = nn.LayerNorm(d_model)
        hidden = ffn_mult * d_model
        self.ffn = nn.Sequential(. . .)

    def forward(self, x):
        x_norm = self.ln1(x)
        attn_out, _ = self.mha(x_norm, x_norm, x_norm)
        x = x + attn_out
        ffn_in = self.ln2(x)
        ffn_out = self.ffn(ffn_in)
        x = x + ffn_out
        return x
```



DeepLearning.AI

Laurence Moroney

Predicting Tokens

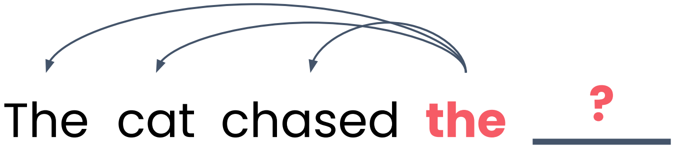


The cat chased the dog

Predicting Tokens

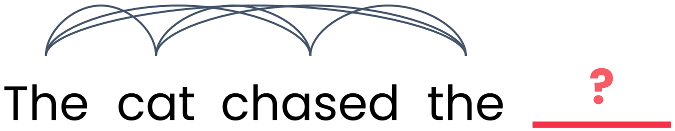
The cat chased **the** ___?___

Predicting Tokens



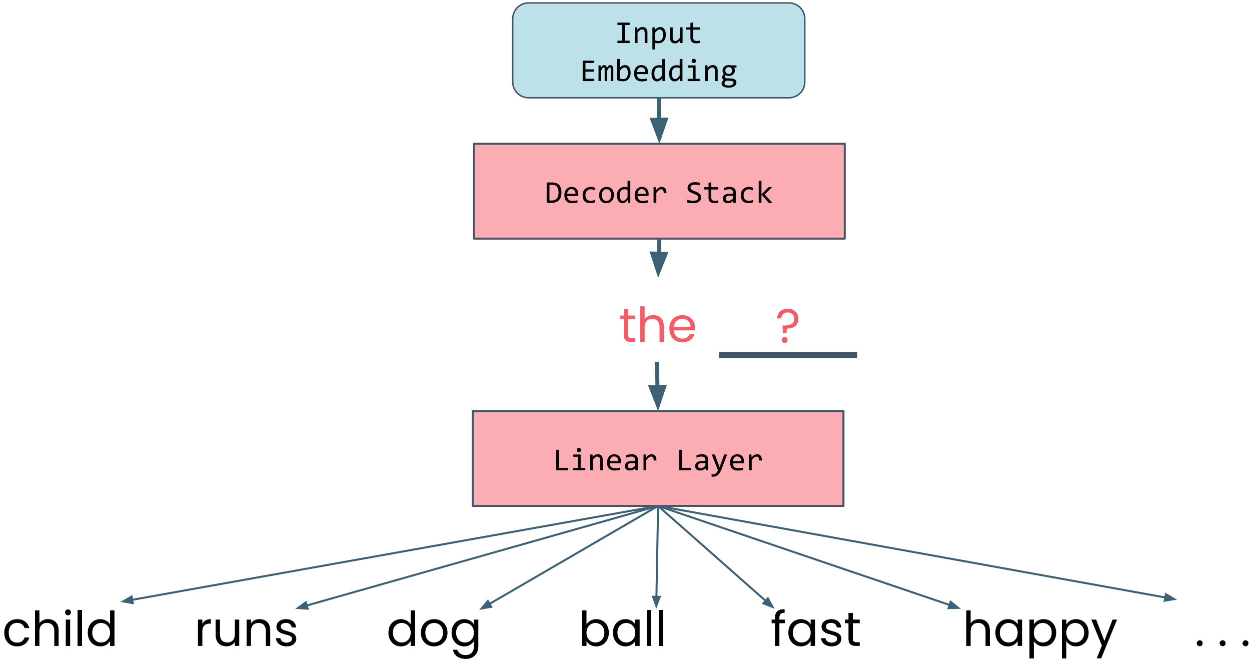
The cat chased **the** ___?___

Predicting Tokens



The cat chased the ___ ?

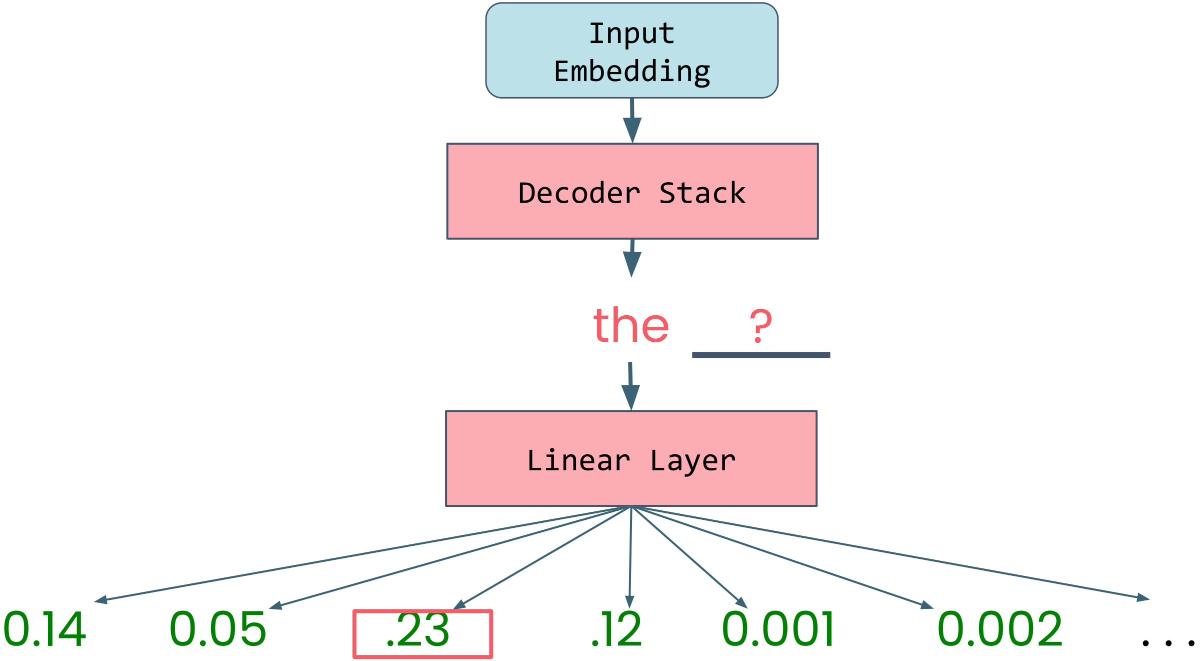
Predicting Tokens



DeepLearning.AI

Laurence Moroney

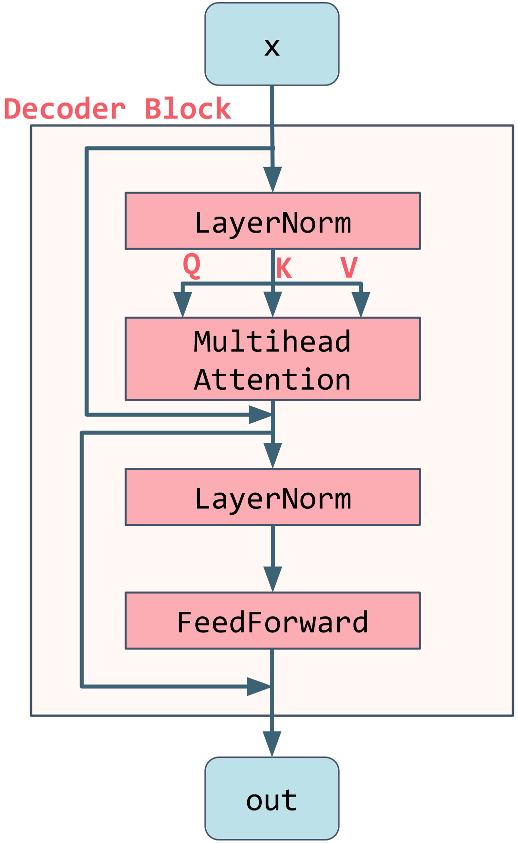
Predicting Tokens



DeepLearning.AI

Laurence Moroney

Decoder Block



DeepLearning.AI

Laurence Moroney

How to Train a Decoder

Input	Targets
The cat chased the ____	dog
When the lights went ____	out
I would have ____	left
She saw a ____	stranger

How to Train a Decoder

The cat chased the dog

Input	Targets
The ____	**cat**
The cat ____	**chased**
The cat chased ____	**the**
The cat chased the ____	**dog**

Causal Masks

```
mha = nn.MultiheadAttention(. . .)
```

Causal Masks

```
mha = nn.MultiheadAttention(. . .)


def forward(self, x):

        # Custom masks
        out, _ = mha(attn_mask=custom_mask, . . .)
```

Causal Masks

```
mha = nn.MultiheadAttention(. . .)


def forward(self, x):

        # Custom masks
        out, _ = mha(attn_mask=custom_mask, . . .)

        # is_causal
        out, _ = mha(is_causal=True, . . .)
```

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

“the

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

		the	cat	chased	the	dog
	the	1.21	-0.259	-0.762	-0.995	0.054
“the	cat	-0.216	0.604	-1.650	-1.061	1.876
“the cat	chased	1.462	-0.410	-1.009	-0.990	0.822
	the	1.506	-1.180	0.659	-1.810	0.521
	dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Attention Scores

		the	cat	chased	the	dog
	the	1.21	■	■	■	■
“the	cat	-0.216	0.604	■	■	■
“the cat	chased	1.462	-0.410	-1.009	■	■
“the cat chased	the	1.506	-1.180	0.659	-1.810	■
“the cat chased the	dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

```python
def make_causal_mask(sz: int):
    mask = torch.full((sz, sz), float("-inf"))
    mask = torch.triu(mask, diagonal=1)
    return mask
```

0.	-inf	-inf	-inf	-inf
0.	0.	-inf	-inf	-inf
0.	0.	0.	-inf	-inf
0.	0.	0.	0.	-inf
0.	0.	0.	0.	0.

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-0.259	-0.762	-0.995	0.054
cat	-0.216	0.604	-1.650	-1.061	1.876
chased	1.462	-0.410	-1.009	-0.990	0.822
the	1.506	-1.180	0.659	-1.810	0.521
dog	0.995	0.521	0.627	-0.511	0.315

+

0.	-inf	-inf	-inf	-inf
0.	0.	-inf	-inf	-inf
0.	0.	0.	-inf	-inf
0.	0.	0.	0.	-inf
0.	0.	0.	0.	0.

Causal Masks

Attention Scores

	the	cat	chased	the	dog
the	1.21	-inf	-inf	-inf	-inf
cat	-0.216	0.604	-inf	-inf	-inf
chased	1.462	-0.410	-1.009	-inf	-inf
the	1.506	-1.180	0.659	-1.810	-inf
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Apply softmax

	the	cat	chased	the	dog
the	1.21	-inf	-inf	-inf	-inf
cat	-0.216	0.604	-inf	-inf	-inf
chased	1.462	-0.410	-1.009	-inf	-inf
the	1.506	-1.180	0.659	-1.810	-inf
dog	0.995	0.521	0.627	-0.511	0.315

Causal Masks

Apply softmax

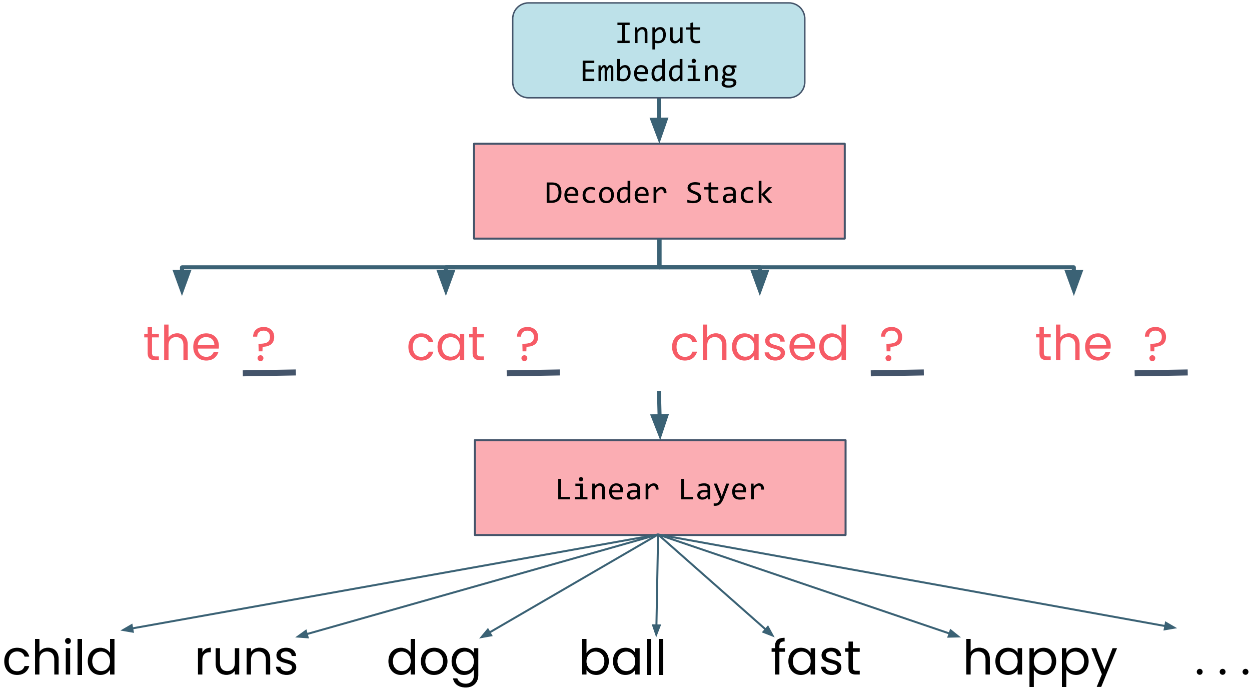
	the	cat	chased	the	dog
the	1.	0.	0.	0.	0.
cat	0.3058	0.6942	0.	0.	0.
chased	0.8075	0.1242	0.0682	0.	0.
the	0.6523	0.0445	0.2796	0.0237	0.
dog	0.3286	0.2046	0.2274	0.0729	0.1665

Causal Masks

Attention Weights

	the	cat	chased	the	dog
the	1.	0.	0.	0.	0.
cat	0.3058	0.6942	0.	0.	0.
chased	0.8075	0.1242	0.0682	0.	0.
the	0.6523	0.0445	0.2796	0.0237	0.
dog	0.3286	0.2046	0.2274	0.0729	0.1665

Predicting Tokens



DeepLearning.AI

Laurence Moroney

Training Data

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']] # Convert to token IDs
```

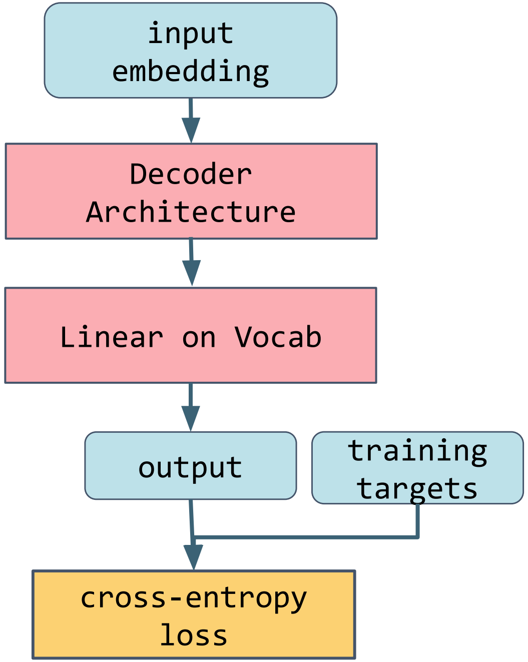
Training Data

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']] # Convert to token IDs


# Inputs
inputs = token_sequences[:, :-1] # Inputs as strings ['the', 'cat', 'chased', 'the']
```

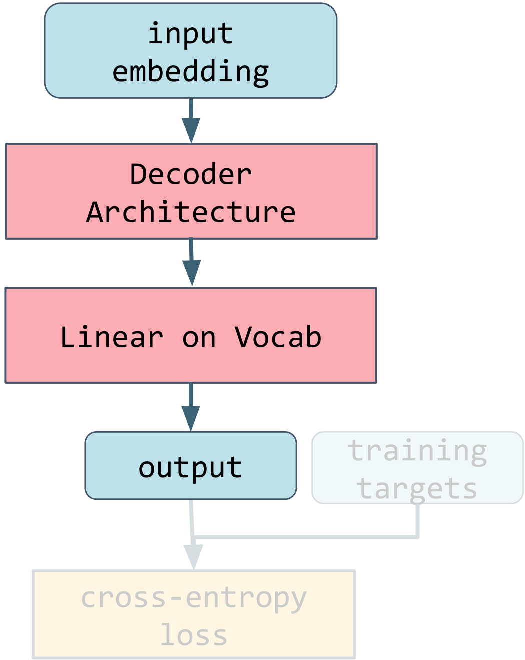
Training Data

```
token_sequences = [['the', 'cat', 'chased', 'the', 'dog']] # Convert to token IDs


# Inputs
inputs = token_sequences[:, :-1] # Inputs as strings ['the', 'cat', 'chased', 'the']


# Targets
targets = token_sequences[:, 1:] # Targets as strings ['cat', 'chased', 'the', 'dog']
```

Training Data



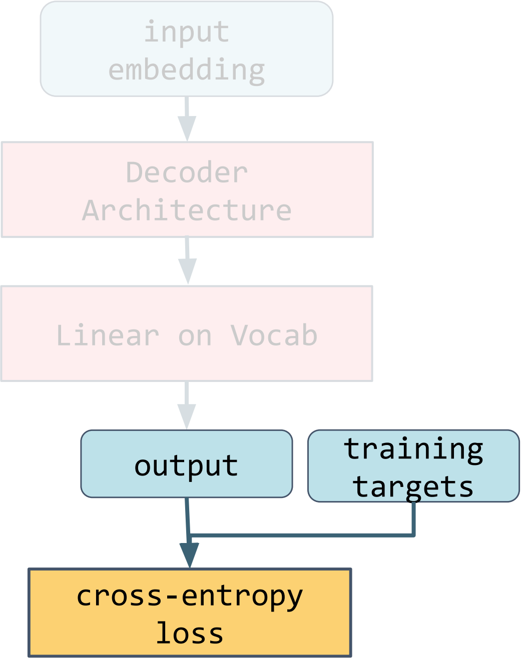
Training Data



DeepLearning.AI

Laurence Moroney

Training Data



DeepLearning.AI

Laurence Moroney

Decoder Only Architectures

```python
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder
```

Decoder Only Architectures

```
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder

nn.TransformerDecoderLayer
nn.TransformerDecoder
```

For encoder decoder models

Decoder Only Architectures

```
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder

nn.TransformerDecoderLayer
nn.TransformerDecoder
```

Decoder Only Architectures

```
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder
```

+ causal mask

Decoder Only Architectures

```
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder


out = encoder_layer (is_causal=True, . . .)
out = encoder_layer (src_mask=custom_mask, . . .)

out = encoder_stack (is_causal=True, . . .)
out = encoder_stack (mask=custom_mask, . . .)
```

Decoder Only Architectures

```
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder


out = encoder_layer (is_causal=True, . . .)
out = encoder_layer (src_mask=custom_mask, . . .)

out = encoder_stack (is_causal=True, . . .)
out = encoder_stack (mask=custom_mask, . . .)
```

Decoder Only Architectures

```
import torch.nn as nn

nn.TransformerEncoderLayer
nn.TransformerEncoder


out = encoder_layer (is_causal=True, . . .)
out = encoder_layer (src_mask=custom_mask, . . .)

out = encoder_stack (is_causal=True, . . .)
out = encoder_stack (mask=custom_mask, . . .)
```

DeepLearning.AI

Encoder - Decoder

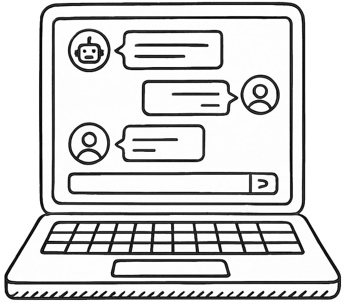
Specialized Approaches to
Natural Language Processing in Pytorch

Transformer Architectures

- **Encoders:**
  - Analyze using rich contextual representations
  - Does **not** generate new tokens

- **Decoders:**
  - Can generate new text from old
  - But only by continuing from whatever it was given

Decoders as Chatbots



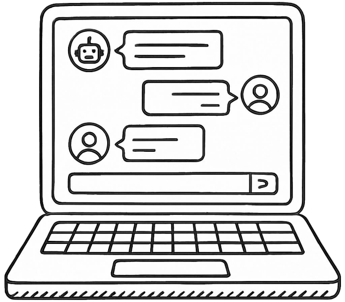
<user> </user>

<assistant> </assistant>

<s> </s>

<user> *did the cat chase the dog?* </user><assistant> . . .

Decoders as Chatbots



<user> </user>

<assistant> </assistant>

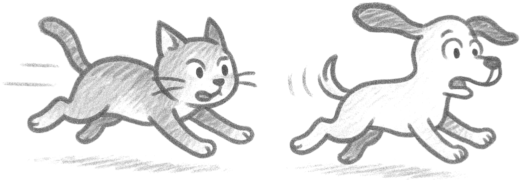
<s> </s>

<user> *did the cat chase the dog?* </user><assistant> *it depends on who you ask* </assistant>

DeepLearning.AI

Laurence Moroney

Translation

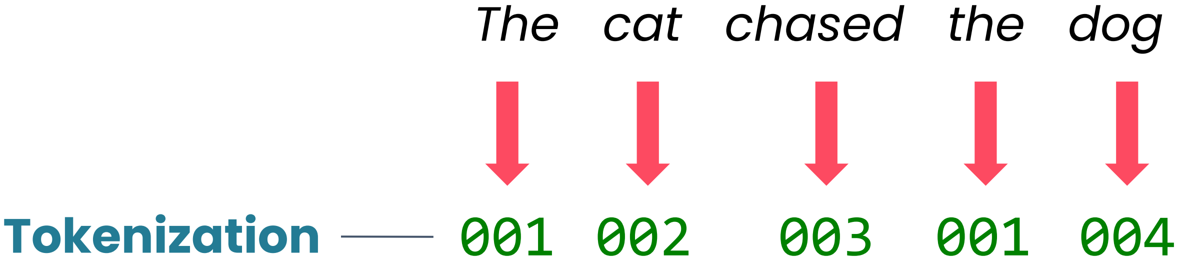


🇮🇹

The cat chased the dog ~~. . .~~ → *Il gatto ha inseguito il cane*

<user> *Translate "the cat chased the dog" into Italian*</user><assistant> . . .

Input



The cat chased the dog

Tokenization

001 002 003 001 004

Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()







    def forward(self, src):
```

Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()







    def forward(self, src):
```

Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()








    def forward(self, src):
        padding_mask = create_padding_mask(src)
```

src

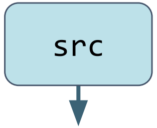
Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        self.token_emb = nn.Embedding(. . .)
        self.pos_enc = PositionalEncoding(. . .)




    def forward(self, src):
       padding_mask = create_padding_mask(src)
```

src

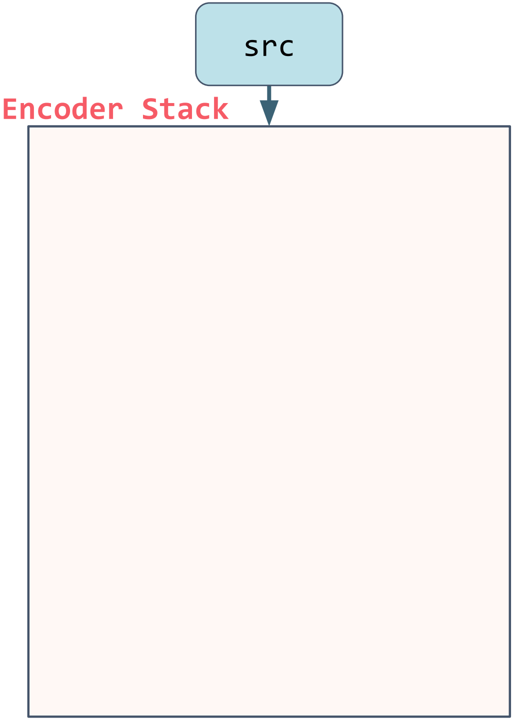
Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        self.token_emb = nn.Embedding(. . .)
        self.pos_enc = PositionalEncoding(. . .)




    def forward(self, src):
       padding_mask = create_padding_mask(src)
       src = self.token_emb(src) + self.pos_enc(src)
```



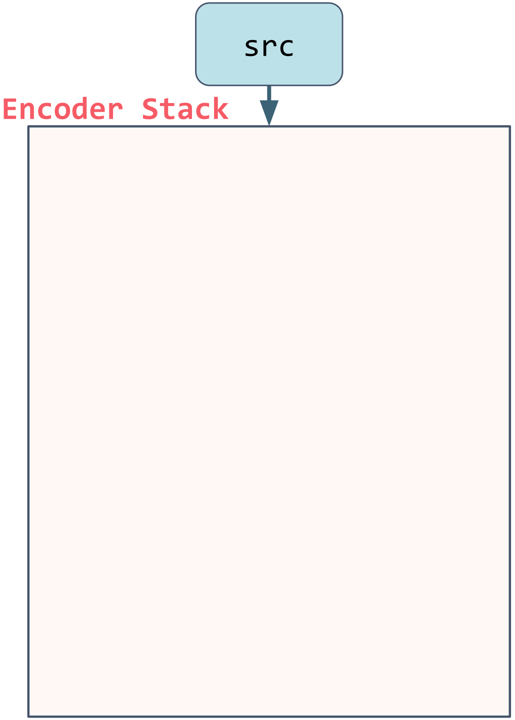
Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        . . .




    def forward(self, src):
        . . .
```



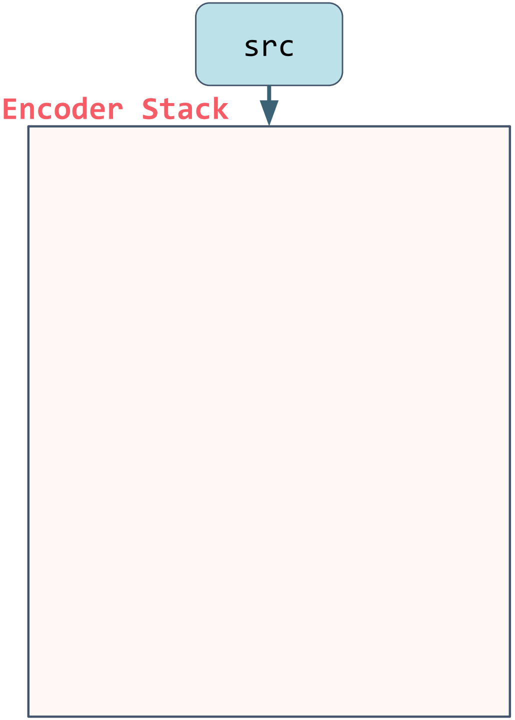
Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        . . .
        enc_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout, batch_first=True)



    def forward(self, src):
        . . .
```



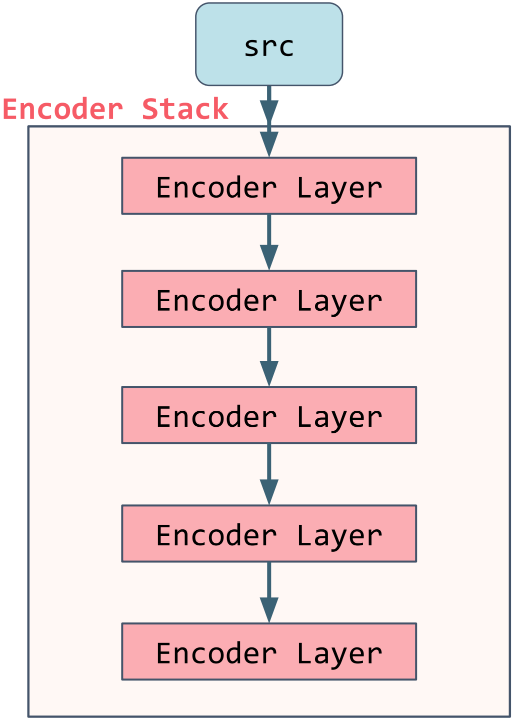
Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        . . .
        enc_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout, batch_first=True)

        self.transformer_encoder = nn.TransformerEncoder(
            enc_layer,
            num_layers=num_layers)
    def forward(self, src):
        . . .
```



Encoder

```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        . . .
        enc_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout, batch_first=True)

        self.transformer_encoder = nn.TransformerEncoder(
            enc_layer,
            num_layers=num_layers)
    def forward(self, src):
        . . .
```

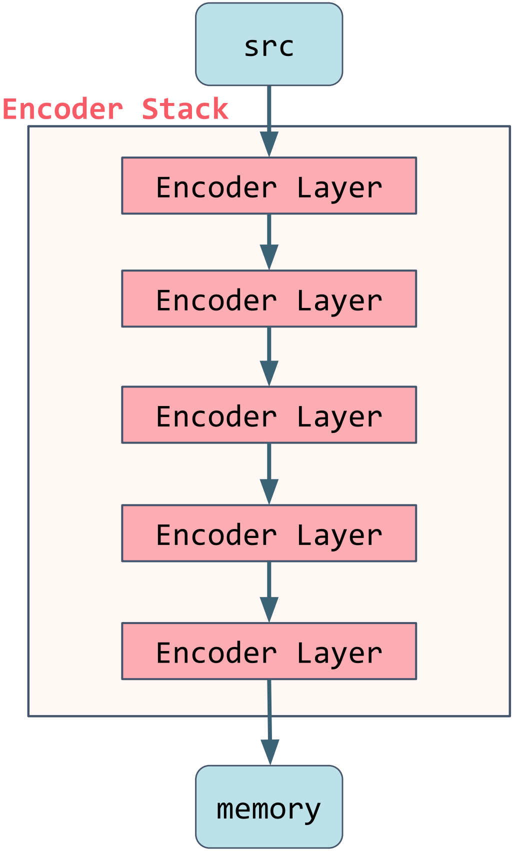


DeepLearning.AI

Laurence Moroney

Encoder

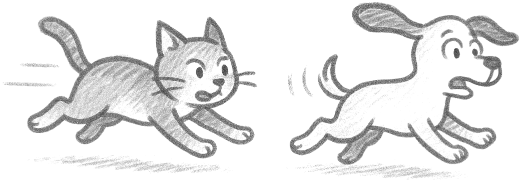
```
class Encoder(nn.Module):
    def __init__():
        super().__init__()
        . . .
        enc_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout, batch_first=True)

        self.transformer_encoder = nn.TransformerEncoder(
            enc_layer,
            num_layers=num_layers)
    def forward(self, src):
        . . .
        memory = self.transformer_encoder(src,
            src_key_padding_mask=padding_mask)
        return memory
```



DeepLearning.AI

Laurence Moroney

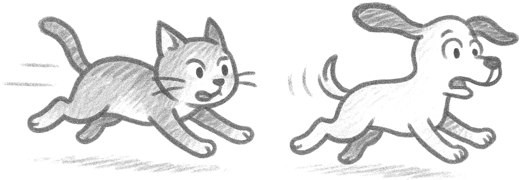
Translation



🇮🇹

The cat chased the dog → *Il gatto ha inseguito il cane*

Translation

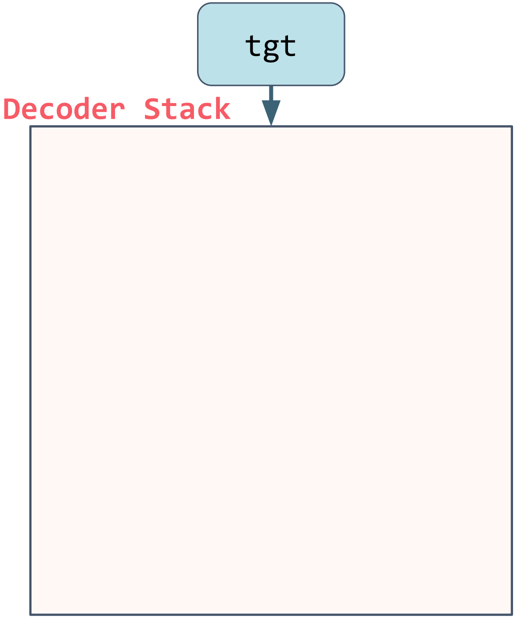


🇮🇹

The cat chased the dog → <s>

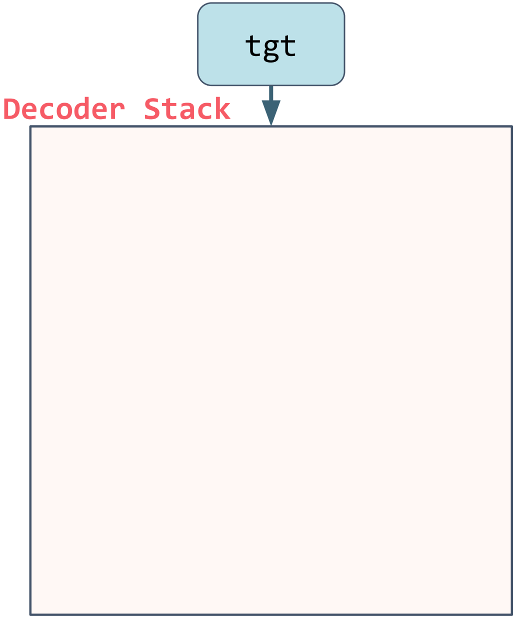
Decoder

```
class Decoder(nn.Module):
    def __init__():
        super().__init__()
        self.token_emb = nn.Embedding(. . .)
        self.pos_enc = PositionalEncoding(. . .)




    def forward(self, tgt, memory, memory_padding_mask):
       src_key_padding_mask = create_pad_mask(tgt)
       src = self.src_tok(tgt) + self.src_pos(tgt)
```



Decoder

```
class Decoder(nn.Module):
    def __init__():
        super().__init__()
        self.token_emb = nn.Embedding(. . .)
        self.pos_enc = PositionalEncoding(. . .)



    def forward(self, tgt, memory, memory_padding_mask):
        src_key_padding_mask = create_pad_mask(tgt)
        src = self.src_tok(tgt) + self.src_pos(tgt)
```

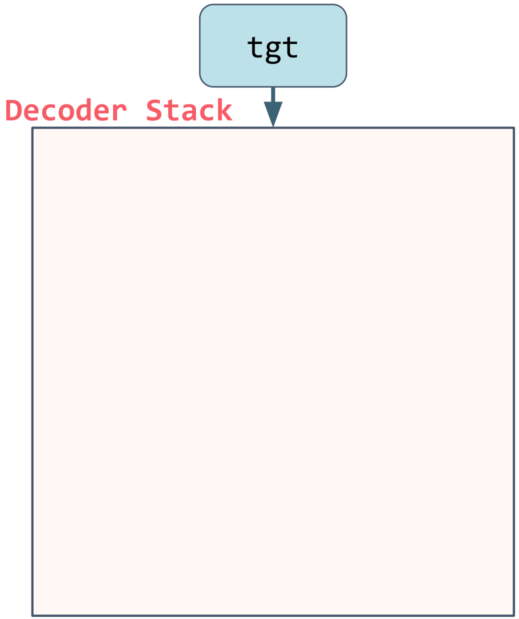


tgt

Decoder Stack

Decoder

```
class Decoder(nn.Module):
    def __init__():
        super().__init__()
        self.token_emb = nn.Embedding(. . .)
        self.pos_enc = PositionalEncoding(. . .)




    def forward(self, tgt, memory, memory_padding_mask):
       src_key_padding_mask = create_pad_mask(tgt)
       src = self.src_tok(tgt) + self.src_pos(tgt)
```



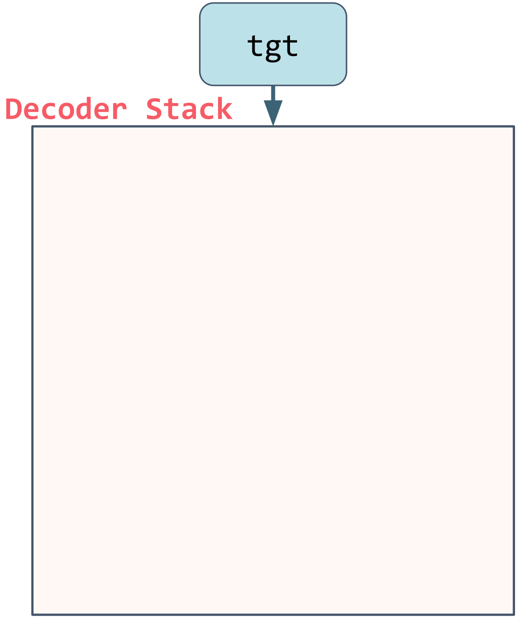
nn.DecoderLayer

```
import torch.nn as nn

nn.TransformerDecoderLayer
nn.TransformerDecoder
```

For encoder decoder models

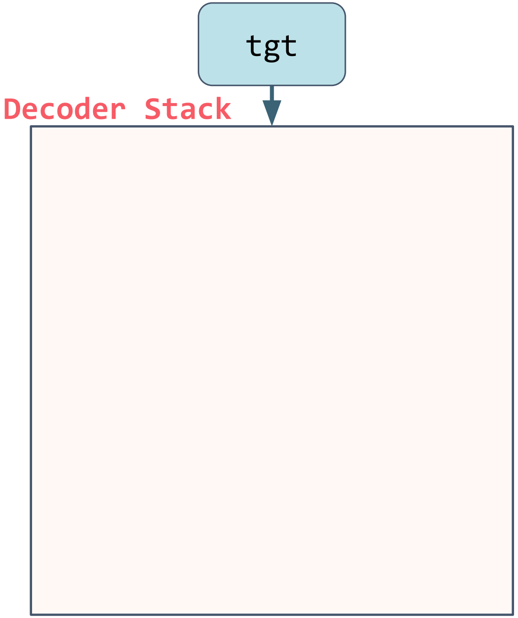
Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .







    def forward(self, tgt, memory, memory_padding_mask):
        . . .
```



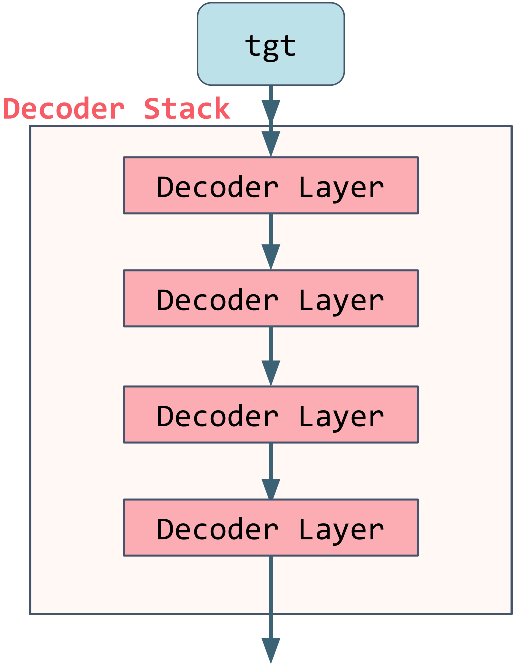
Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .
        decoder_layer = nn.TransformerDecoderLayer(
            d_model=d_model, nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout, batch_first=True)




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
```



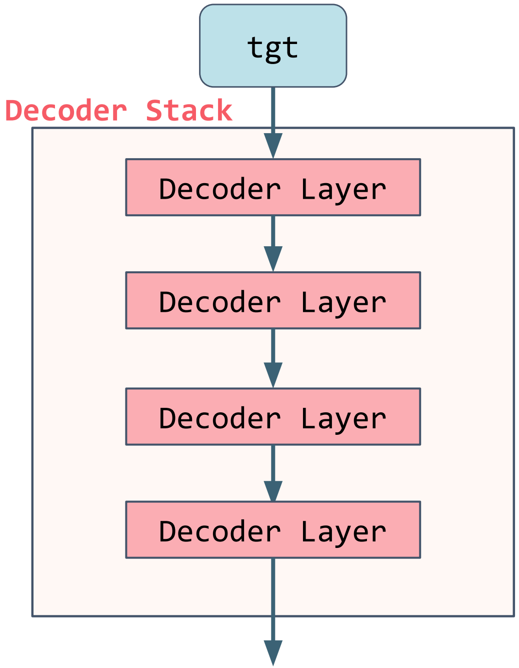
Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .
        decoder_layer = nn.TransformerDecoderLayer(
            d_model=d_model, nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout, batch_first=True)

        self.transformer_decoder = nn.TransformerDecoder(
            decoder_layer,
            num_layers=num_layers)

    def forward(self, tgt, memory, memory_padding_mask):
        . . .
```



Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .

    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(
            tgt,
            memory,
            tgt_mask=tgt_causal_mask
            memory_mask=None
            tgt_key_padding_mask=tgt_padding_mask
            memory_key_padding_mask=memory_padding_mask)
```



tgt

Decoder Stack

Decoder Layer

Decoder Layer

Decoder Layer

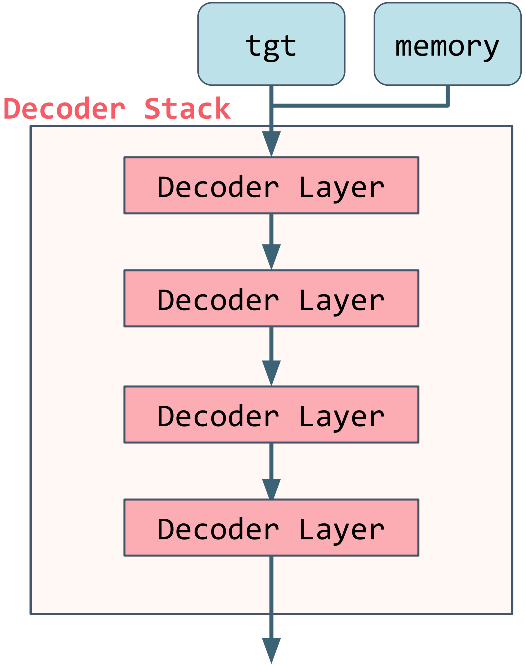
Decoder Layer

DeepLearning.AI

Laurence Moroney

Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(
            tgt,
            memory,
            tgt_mask=tgt_causal_mask
            memory_mask=None
            tgt_key_padding_mask=tgt_padding_mask
            memory_key_padding_mask=memory_padding_mask)
```

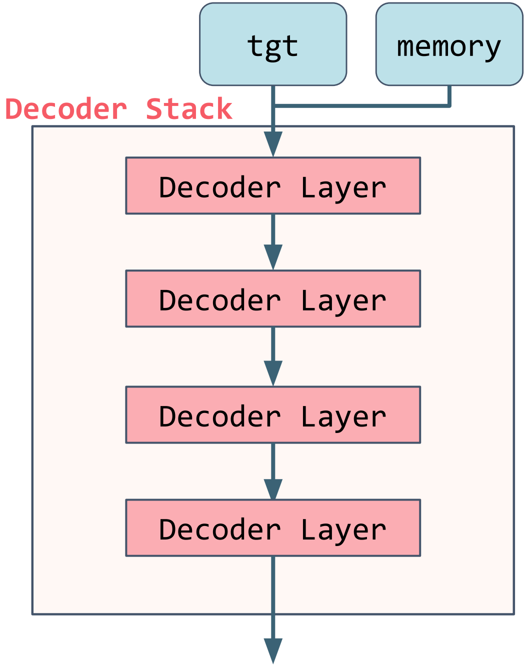


DeepLearning.AI

Laurence Moroney

Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(
            tgt,
            memory,
            tgt_mask=tgt_causal_mask
            memory_mask=None
            tgt_key_padding_mask=tgt_padding_mask
            memory_key_padding_mask=memory_padding_mask)
```



tgt

memory

Decoder Stack

Decoder Layer

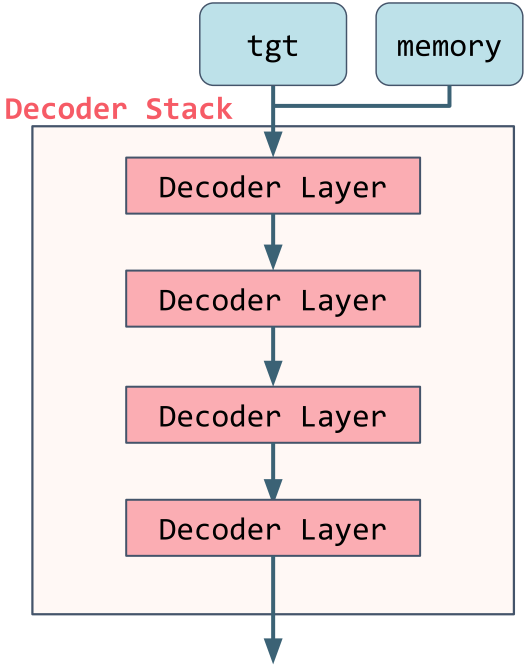
Decoder Layer

Decoder Layer

Decoder Layer

Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(
            tgt,
            memory,
            tgt_mask=tgt_causal_mask
            memory_mask=None
            tgt_key_padding_mask=tgt_padding_mask
            memory_key_padding_mask=memory_padding_mask)
```

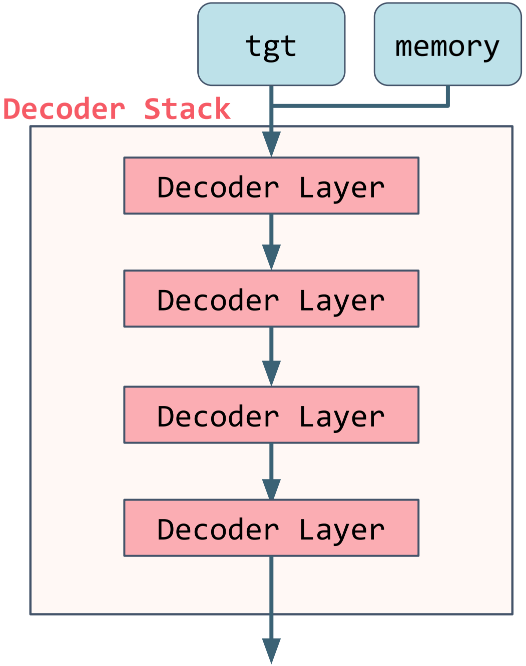


DeepLearning.AI

Laurence Moroney

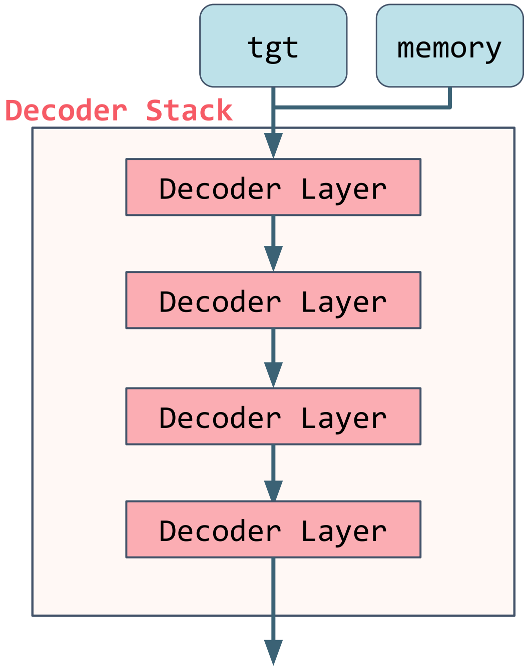
Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(
            tgt,
            memory,
            tgt_mask=tgt_causal_mask
            memory_mask=None
            tgt_key_padding_mask=tgt_padding_mask
            memory_key_padding_mask=memory_padding_mask)
```



Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(
            tgt,
            memory,
            tgt_mask=tgt_causal_mask
            memory_mask=None
            tgt_key_padding_mask=tgt_padding_mask
            memory_key_padding_mask=memory_padding_mask)
```

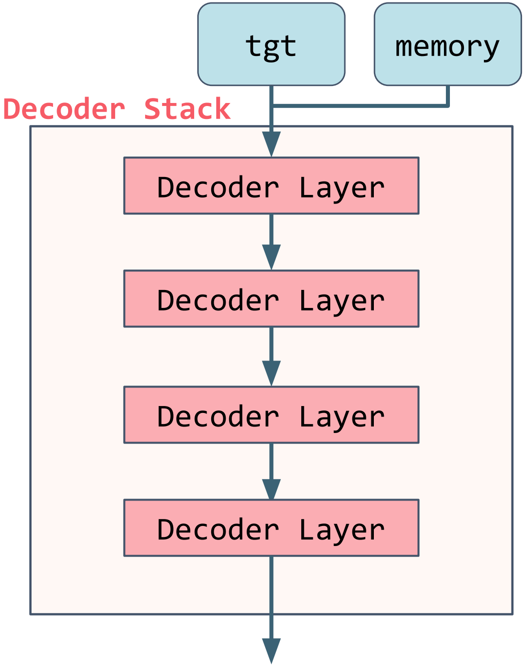


DeepLearning.AI

Laurence Moroney

Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .




    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(. . .)
```

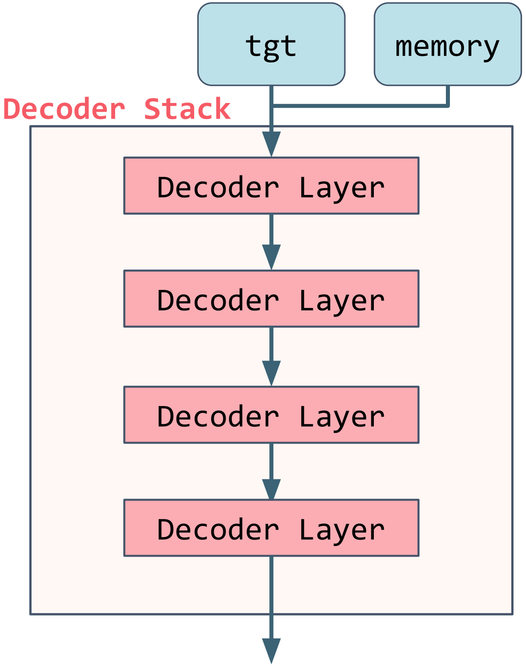


DeepLearning.AI

Laurence Moroney

Decoder

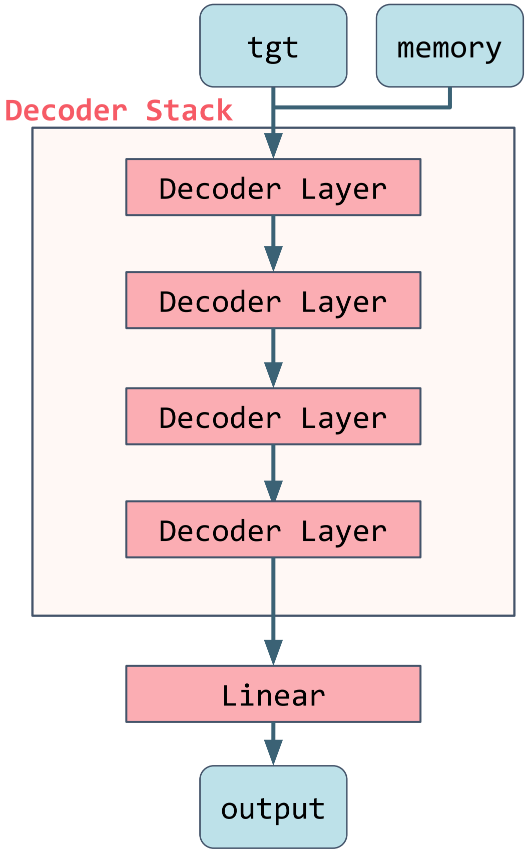
```
class Decoder(nn.Module):
    def __init__():
        . . .
        self.out_proj = nn.Linear(d_model, vocab_size)



    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(. . .)
```



Laurence Moroney

Decoder

```
class Decoder(nn.Module):
    def __init__():
        . . .
        self.out_proj = nn.Linear(d_model, vocab_size)



    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(. . .)

        output = self.output_proj(decoded)

        return output
```

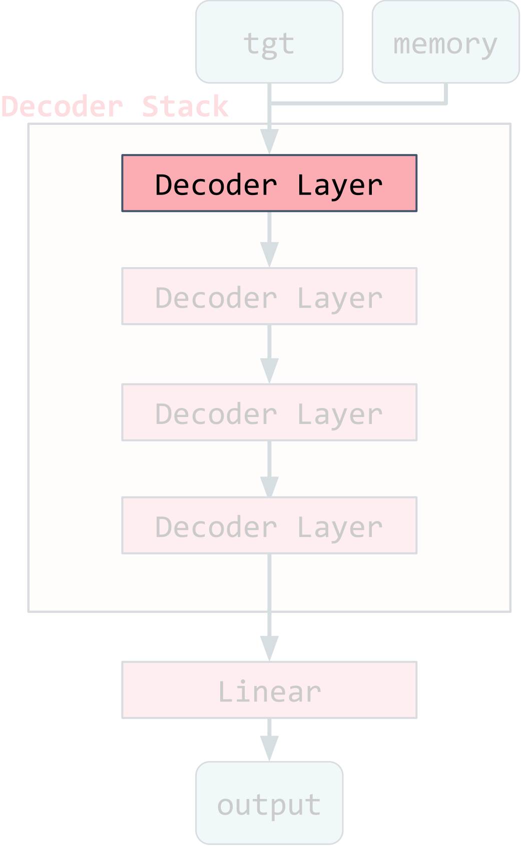


DeepLearning.AI

Laurence Moroney

Decoder

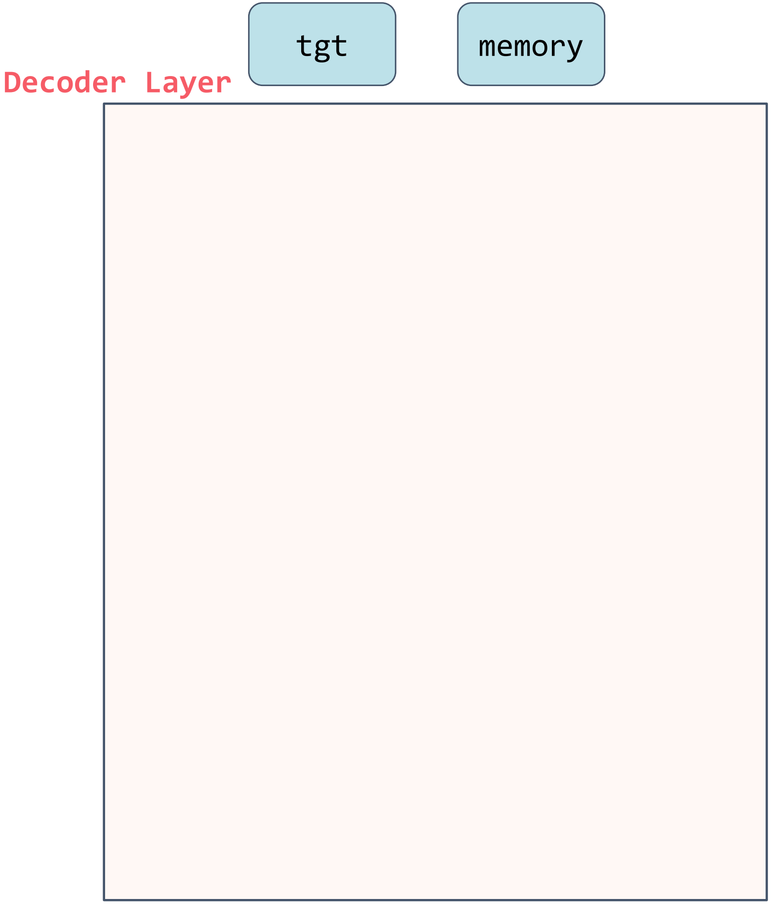
```
class Decoder(nn.Module):
    def __init__():
        . . .
        self.out_proj = nn.Linear(d_model, vocab_size)


    def forward(self, tgt, memory, memory_padding_mask):
        . . .
        decoded= self.transformer_decoder(. . .)

        output = self.output_proj(decoded)

        return output
```



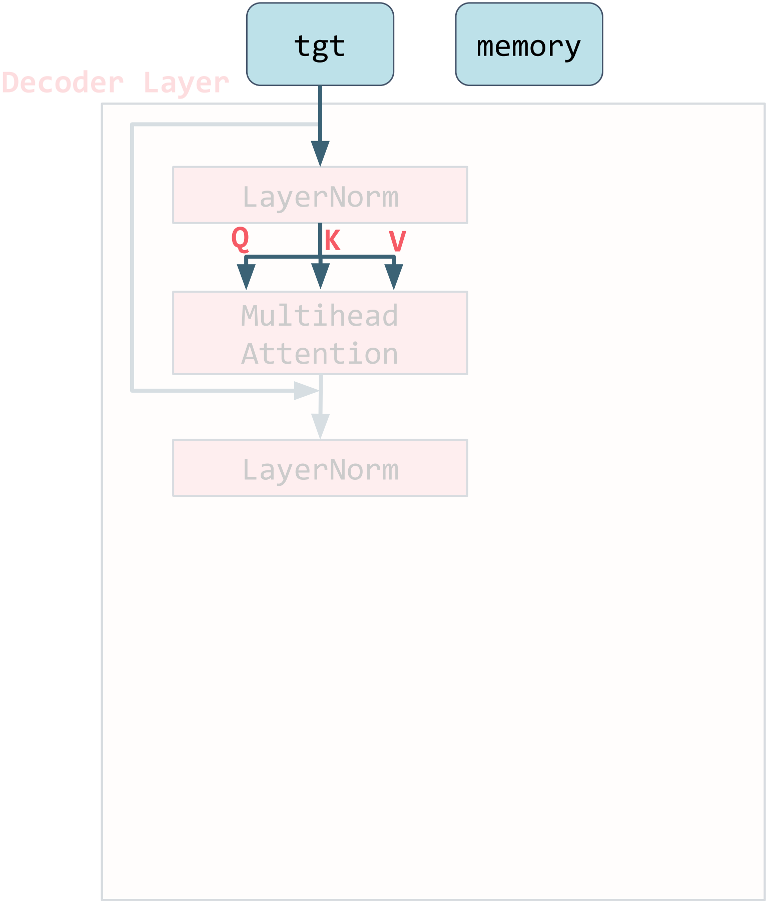
DeepLearning.AI

Laurence Moroney

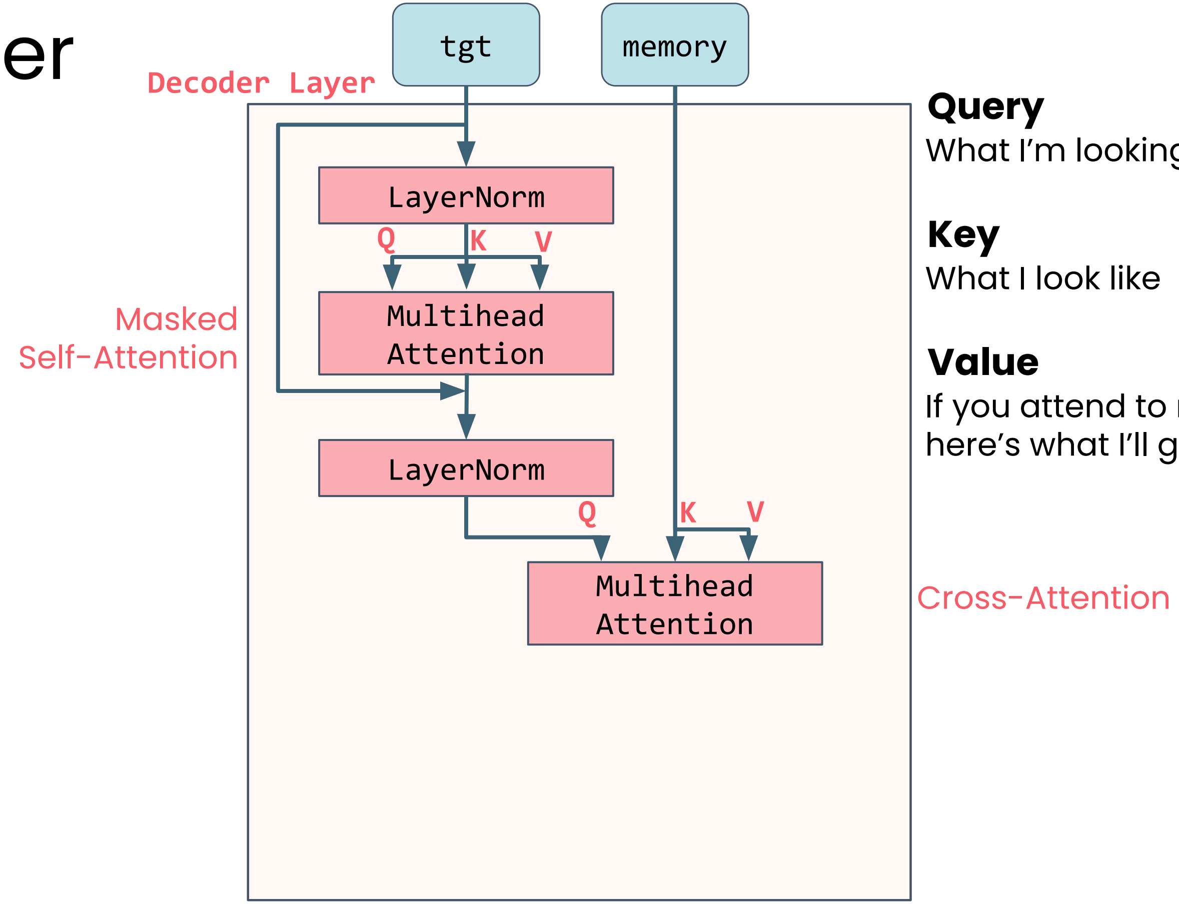
Decoder Layer



Decoder Layer



Decoder Layer



Query
What I'm looking for

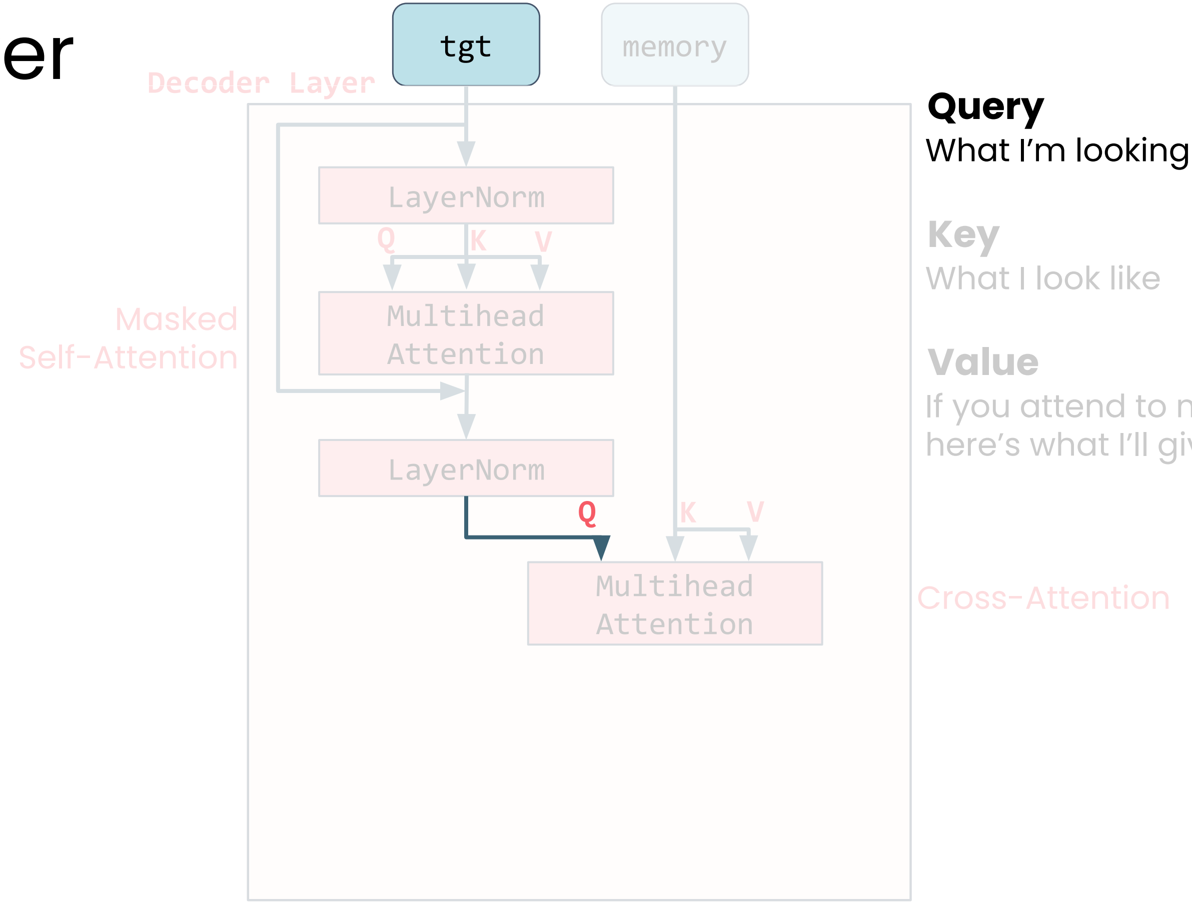
Key
What I look like

Value
If you attend to me, here's what I'll give you

DeepLearning.AI

Laurence Moroney

Decoder Layer

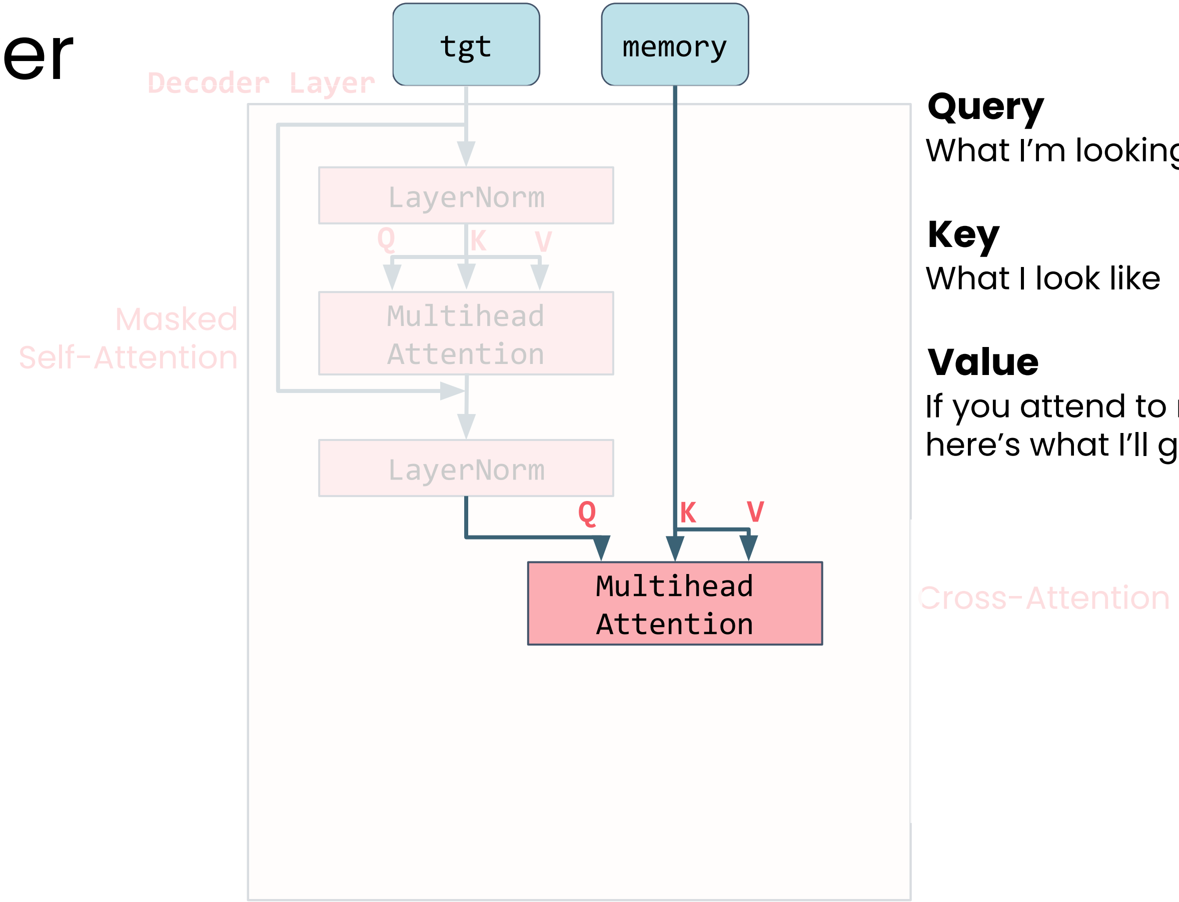


Query
What I'm looking for

Key
What I look like

Value
If you attend to me,
here's what I'll give you

Decoder Layer

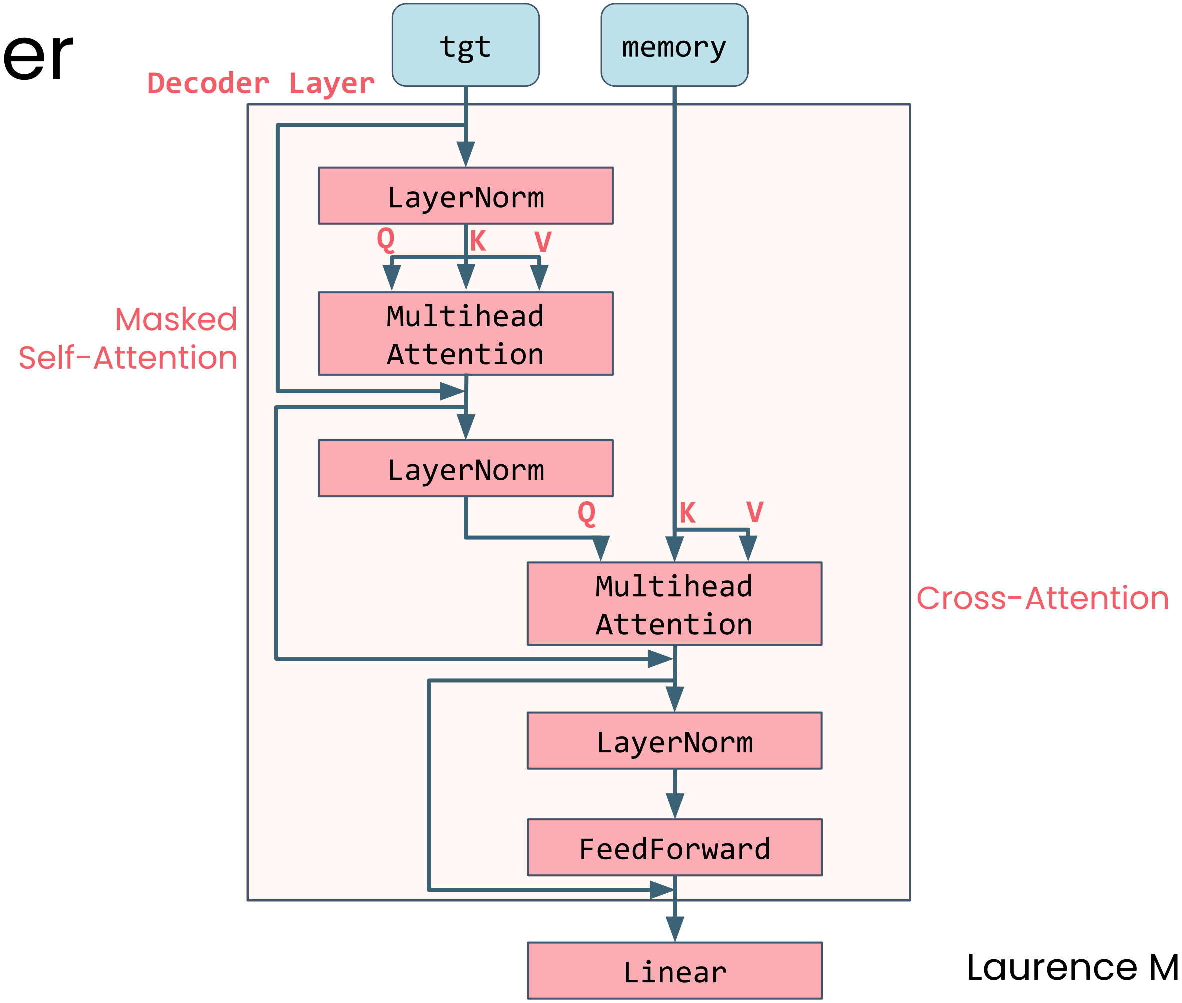


Query
What I'm looking for

Key
What I look like

Value
If you attend to me,
here's what I'll give you

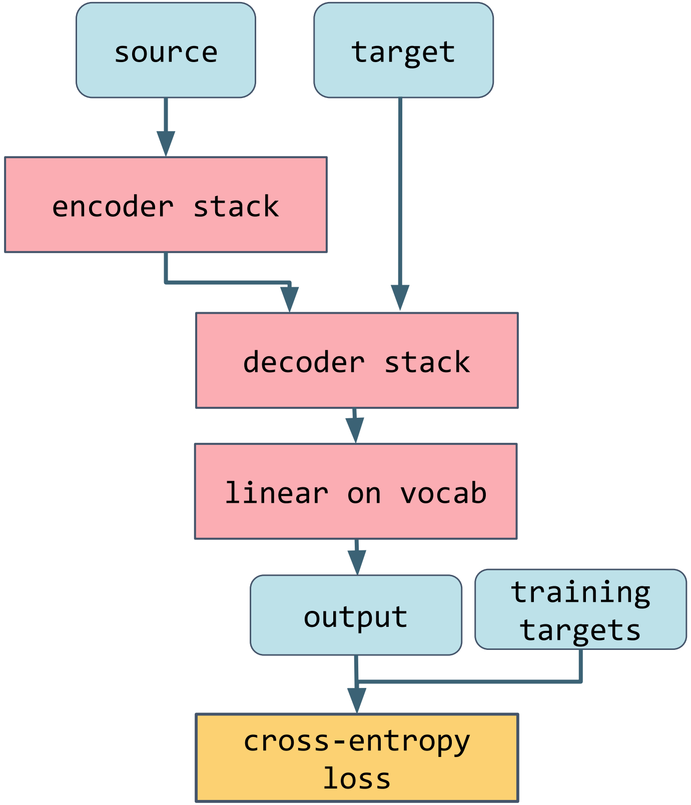
Decoder Layer



DeepLearning.AI

Laurence Moroney

Encoder Decoder



DeepLearning.AI

Laurence Moroney

Transformers

Transformer Black Box