LAB 6:

ATTENTION AND TRANSFORMER

University of Washington, Seattle

Spring 2025

OUTLINE

Part 1: Transformer motivation

- Limitation of RNNs with sequence data
- Seq2seq and attention
- Attention is all you need

Part 2: Self-attention layer

- Overview
- Key, Query and Value retrieval process
- Multi-headed attention

Part 3: Transformer architecture

- Encoder
- Decoder
- Transformer vs RNN

Part 4: Transformer example

- Text Classification on IMDB dataset

Part 5: Lab Assignment

- Text Classification on AG News dataset

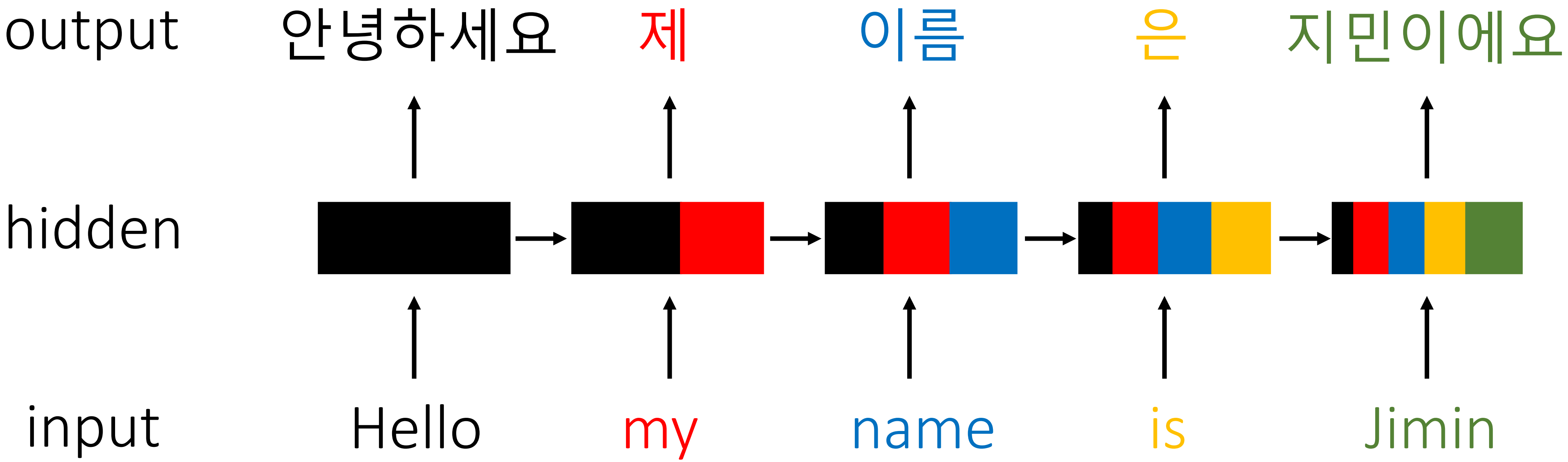
Transformer Motivation

Limitations of RNNs with sequence data

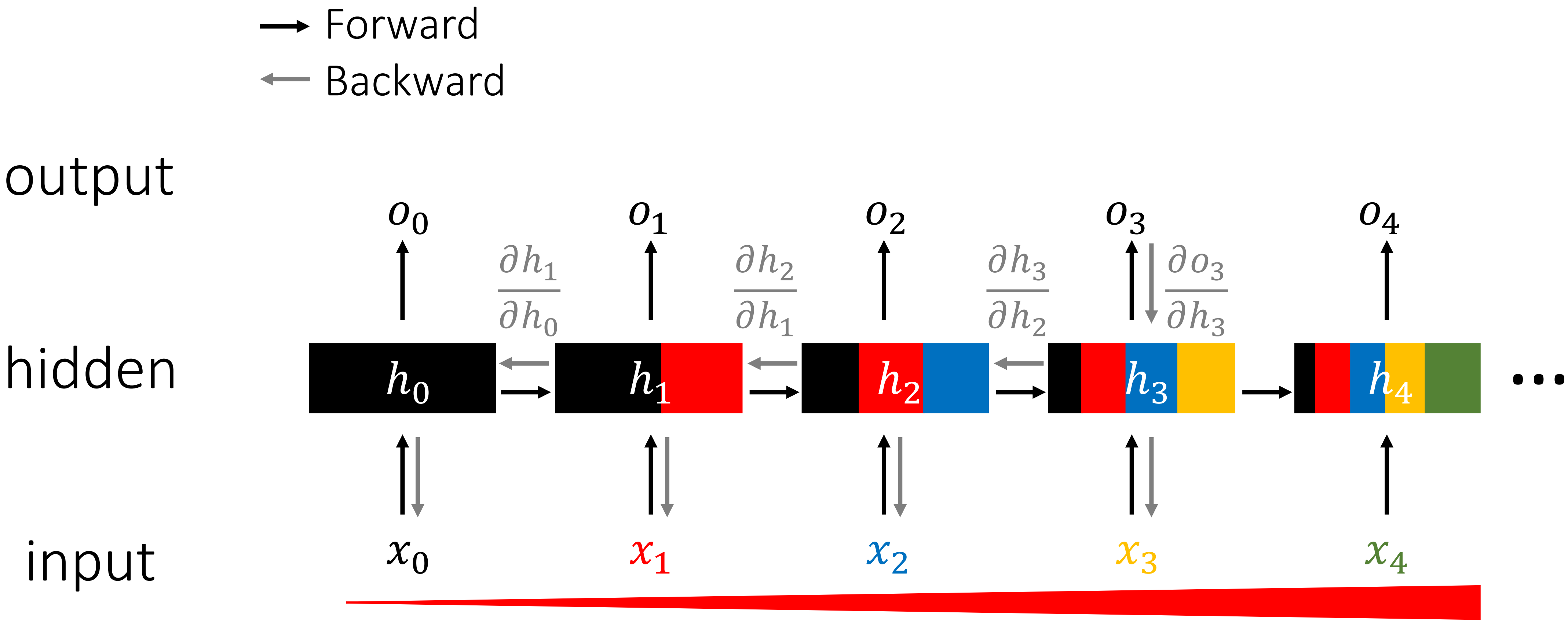
Seq2Seq and attention

Attention is all you need

Limitations of RNNs



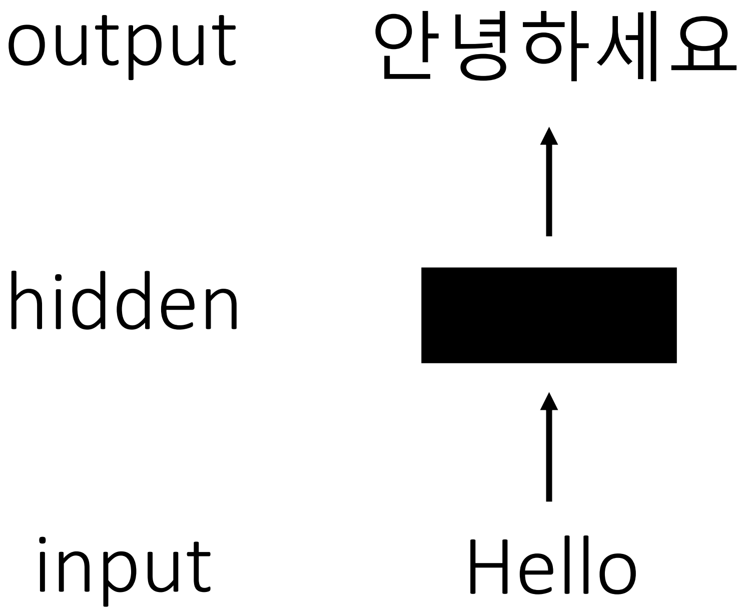
Vanishing and Exploding Gradients



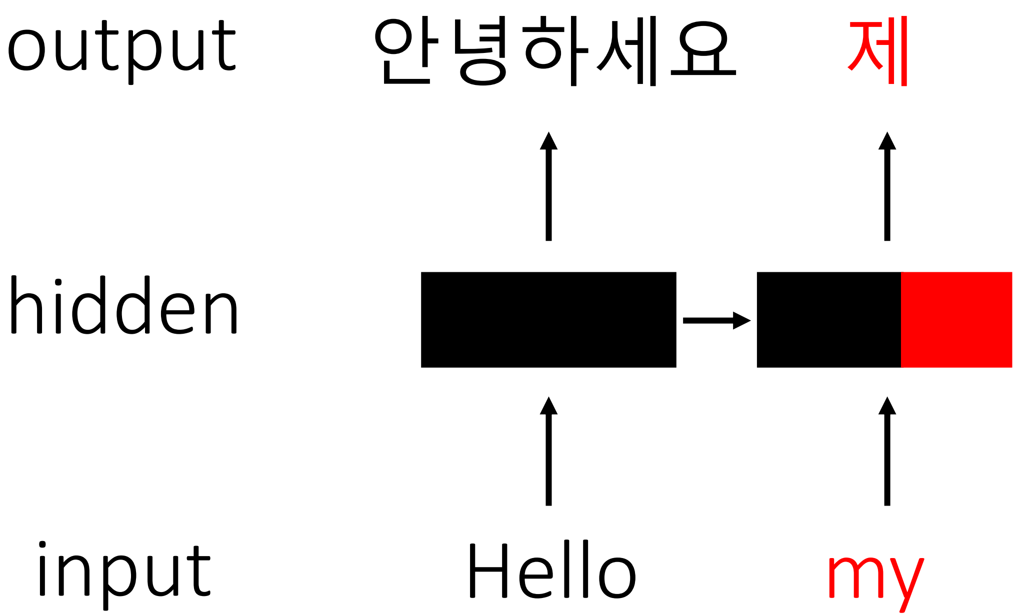
Longer input sequence →

higher risk of Vanishing/Exploding Gradients!

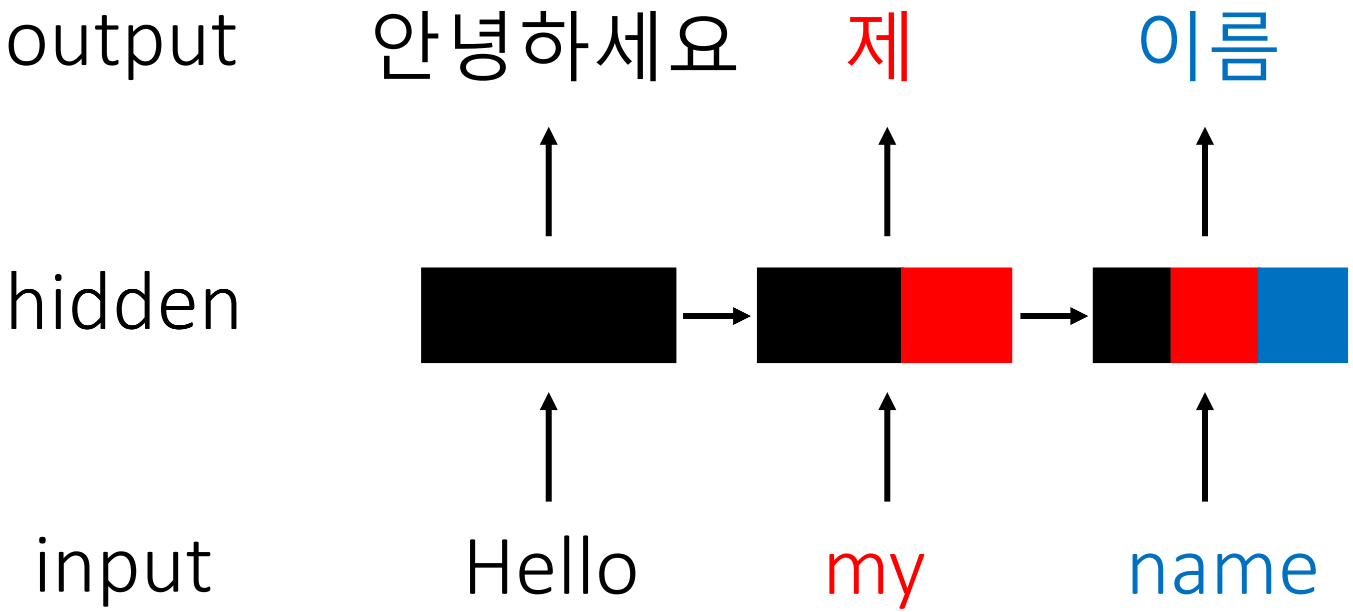
RNN Architecture



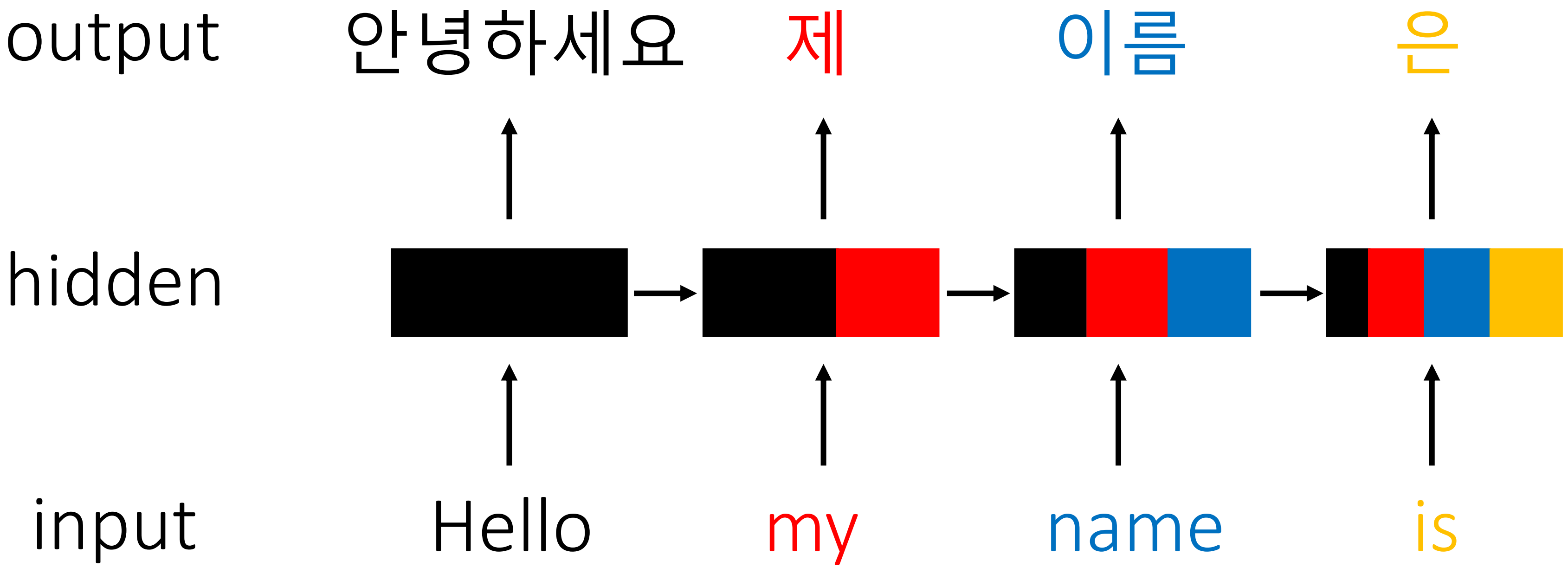
RNN Architecture



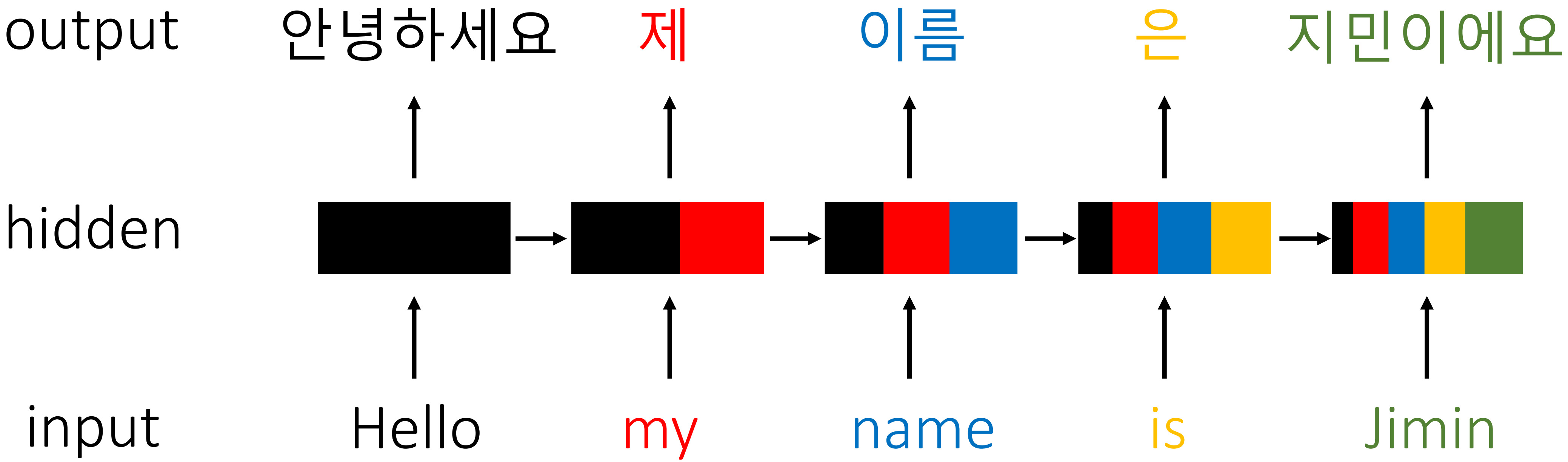
RNN Architecture



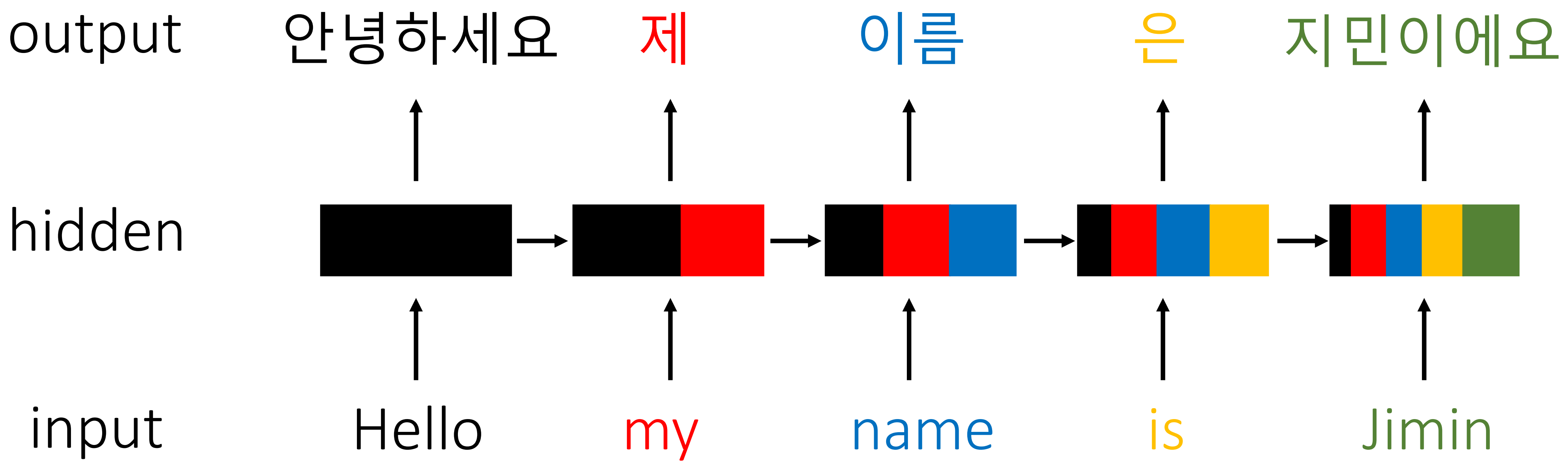
RNN Architecture



RNN Architecture

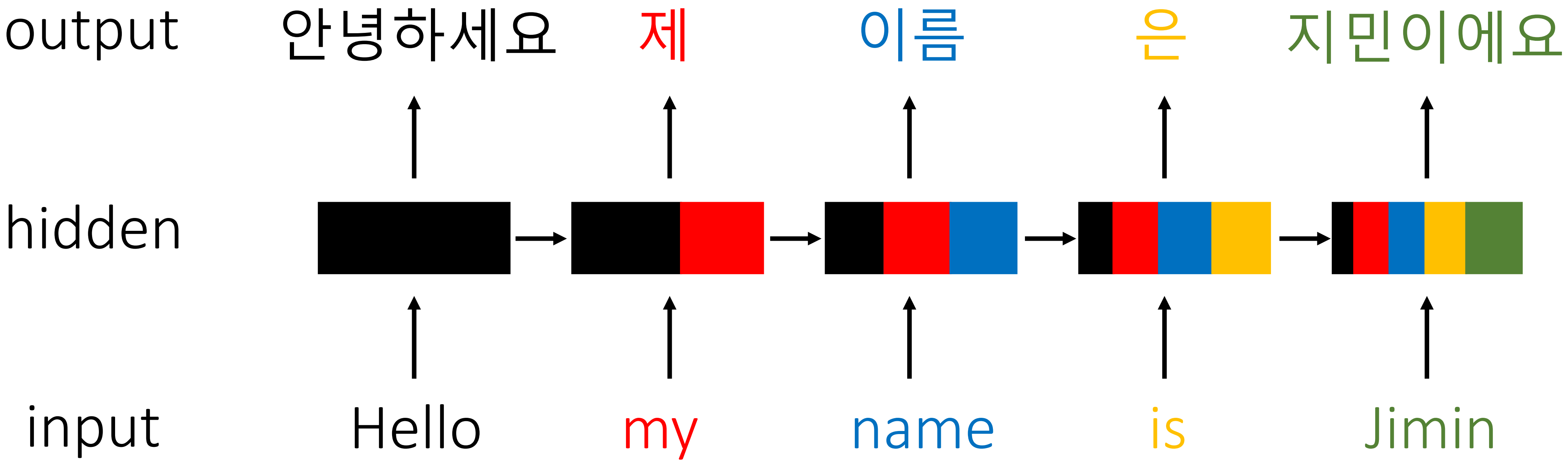


RNN Architecture



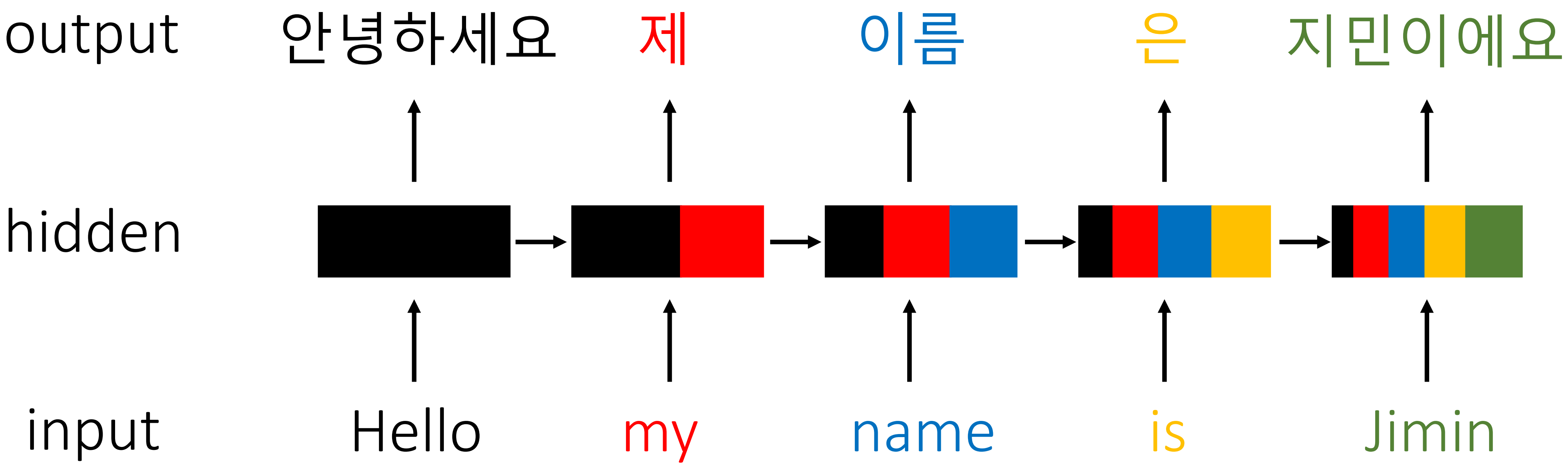
Each input (token) is fed sequentially →
No parallelization

RNN Architecture



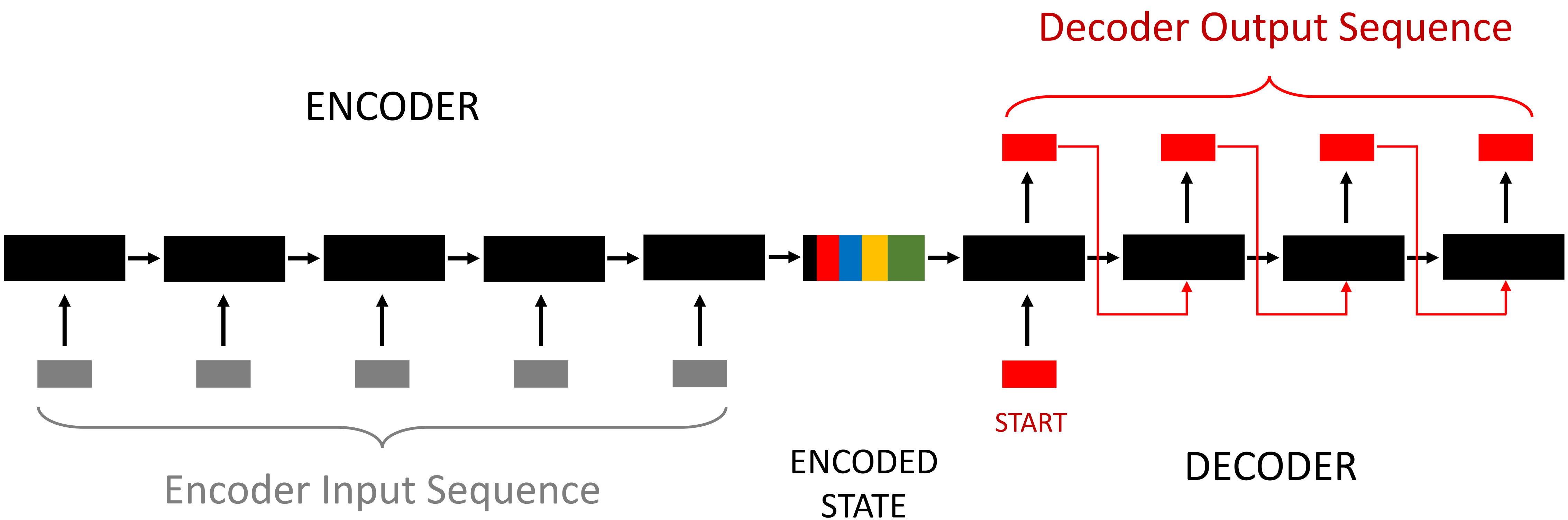
Difficult to store long-term context when sequence is long

RNN Architecture



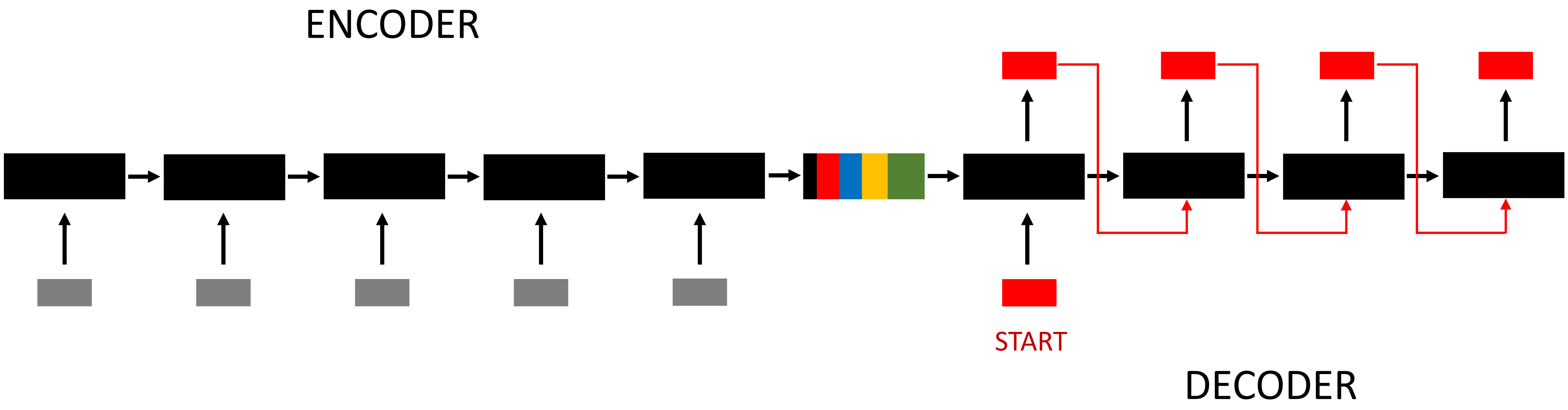
If using time-synced many-to-many →
$len(input\ seq) == len(output\ seq)$

Seq2Seq



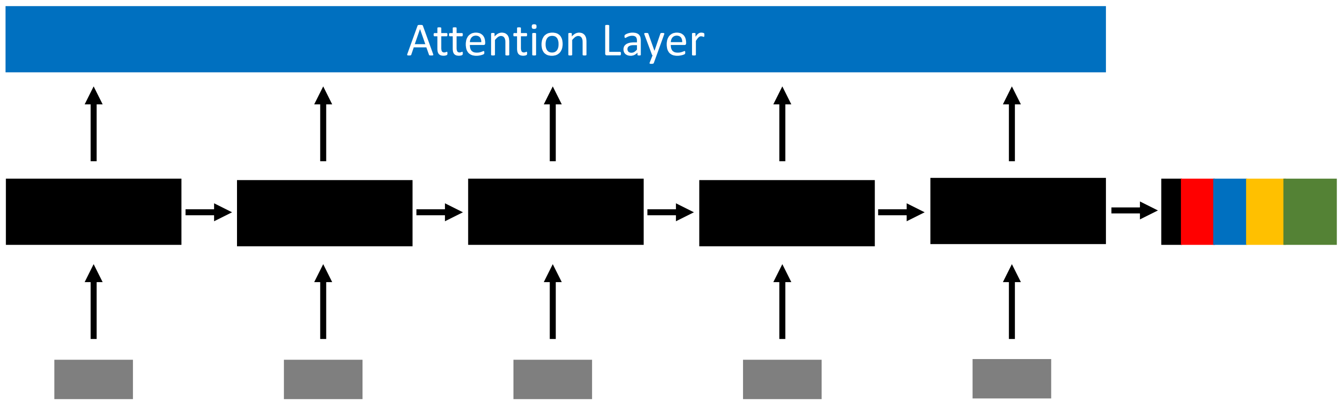
(+) Can be trained to translate input sequence to output sequence with two different lengths

Seq2Seq

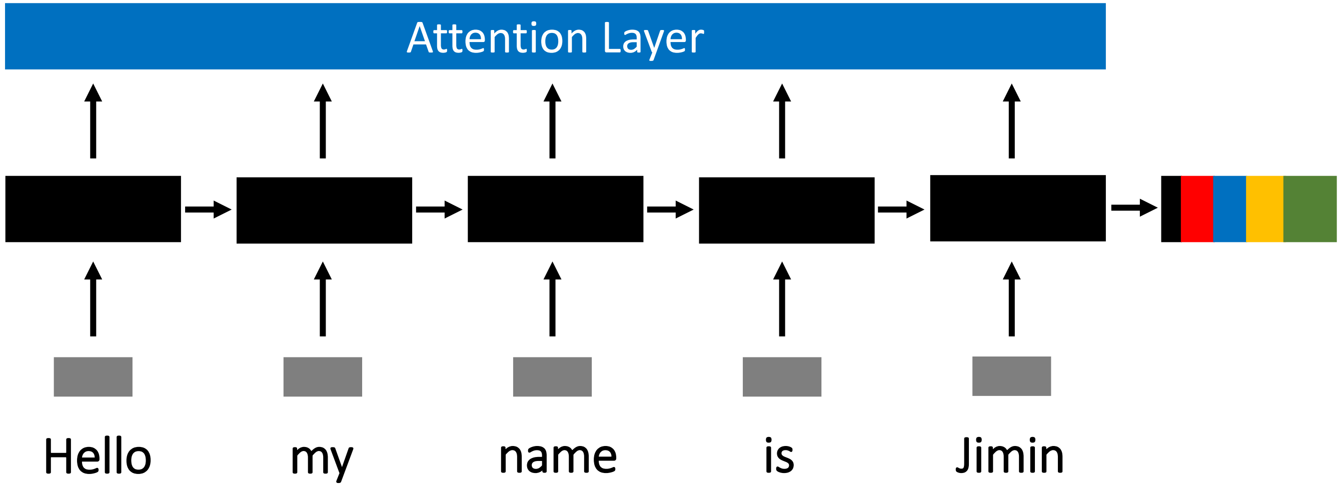


(-) Suffers from identical limitations as RNNs → Can’t process long context, Hard to parallelize

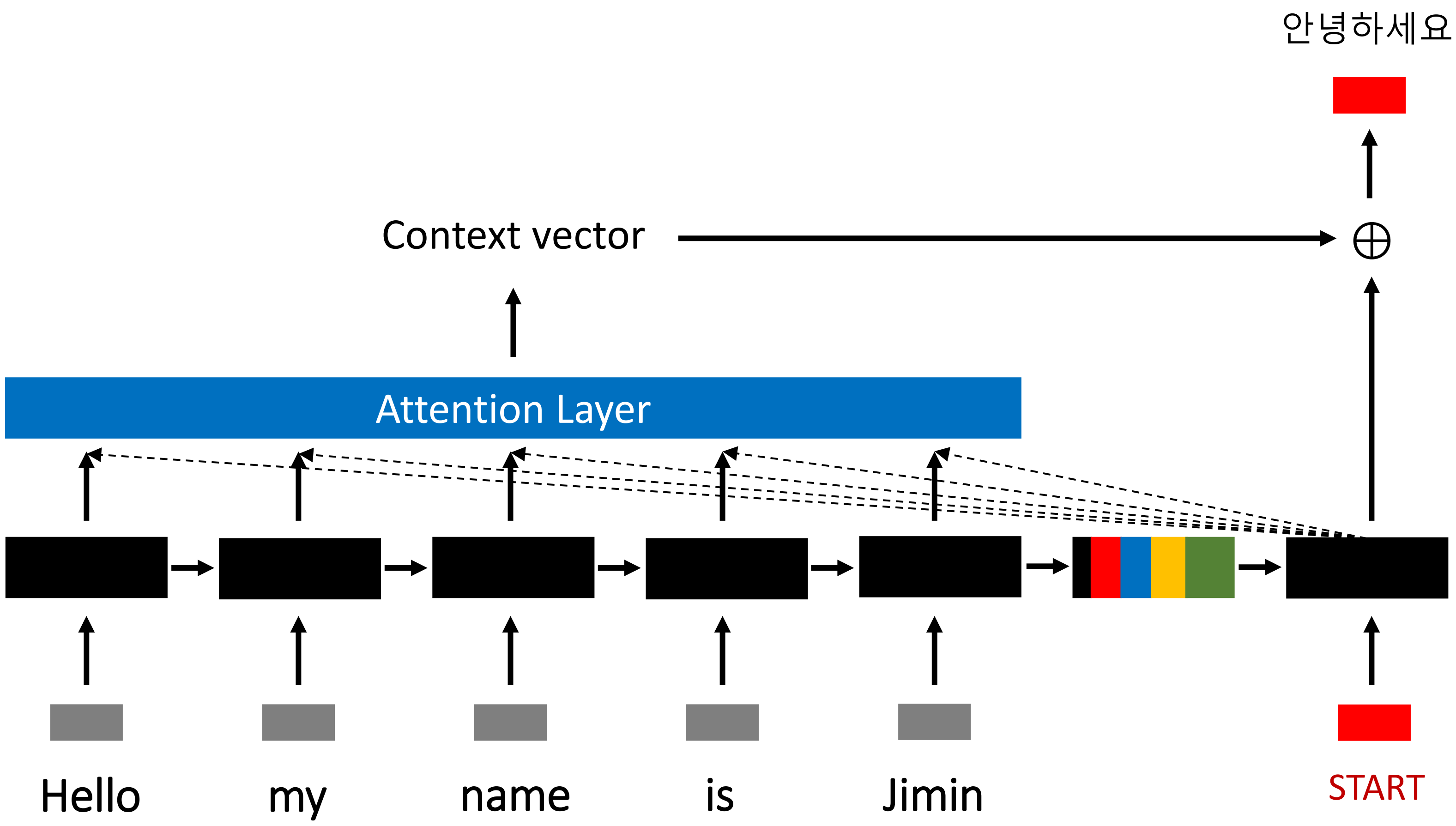
Seq2Seq with Attention



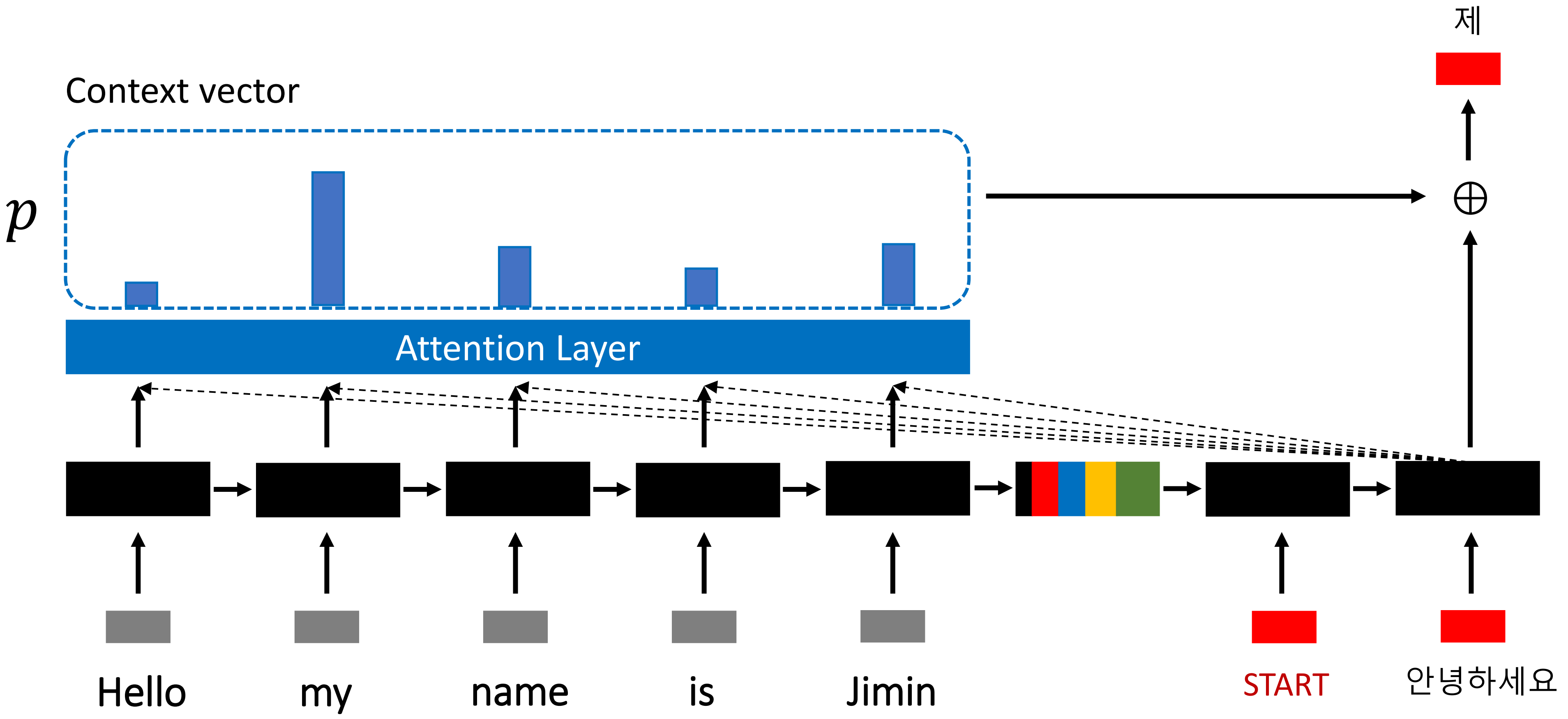
Seq2Seq with Attention



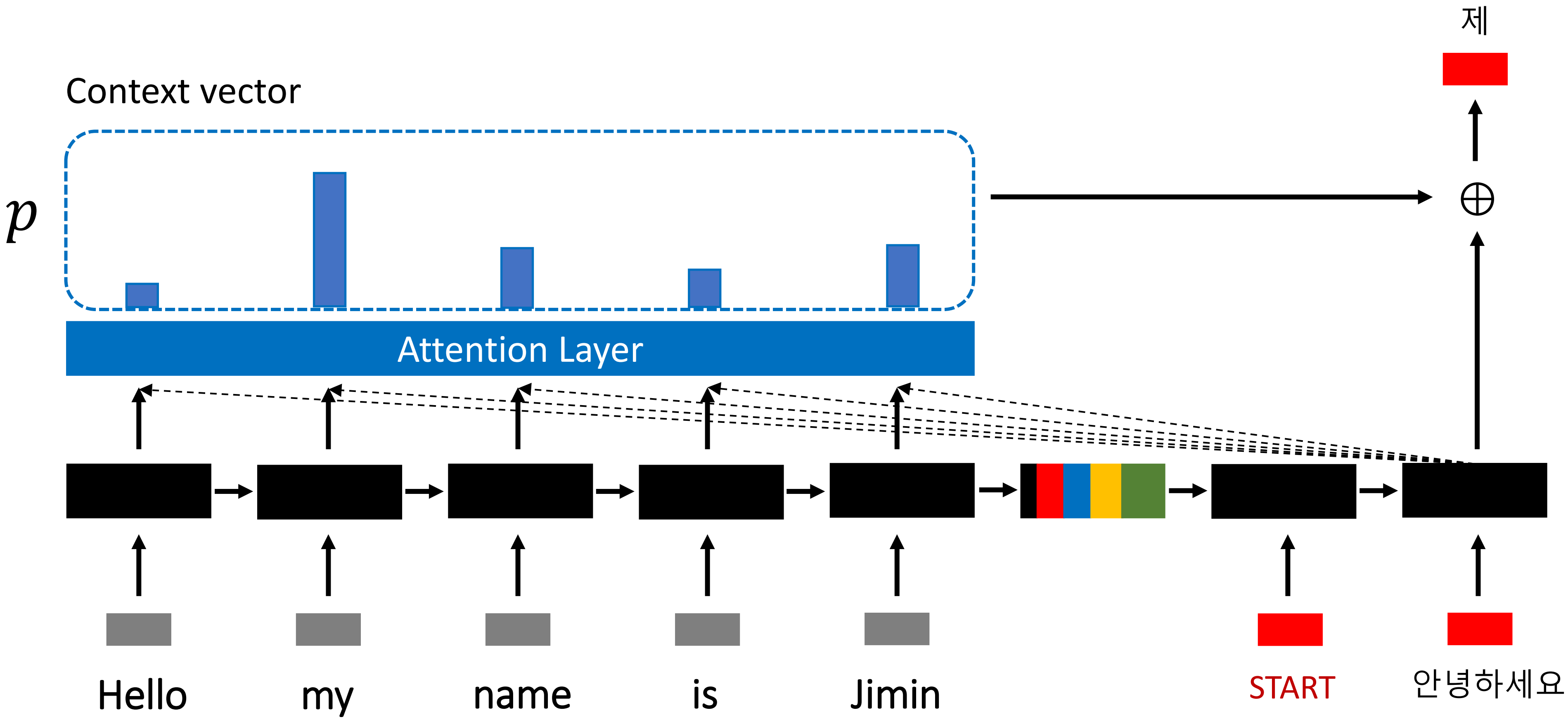
Seq2Seq with Attention



Seq2Seq with Attention

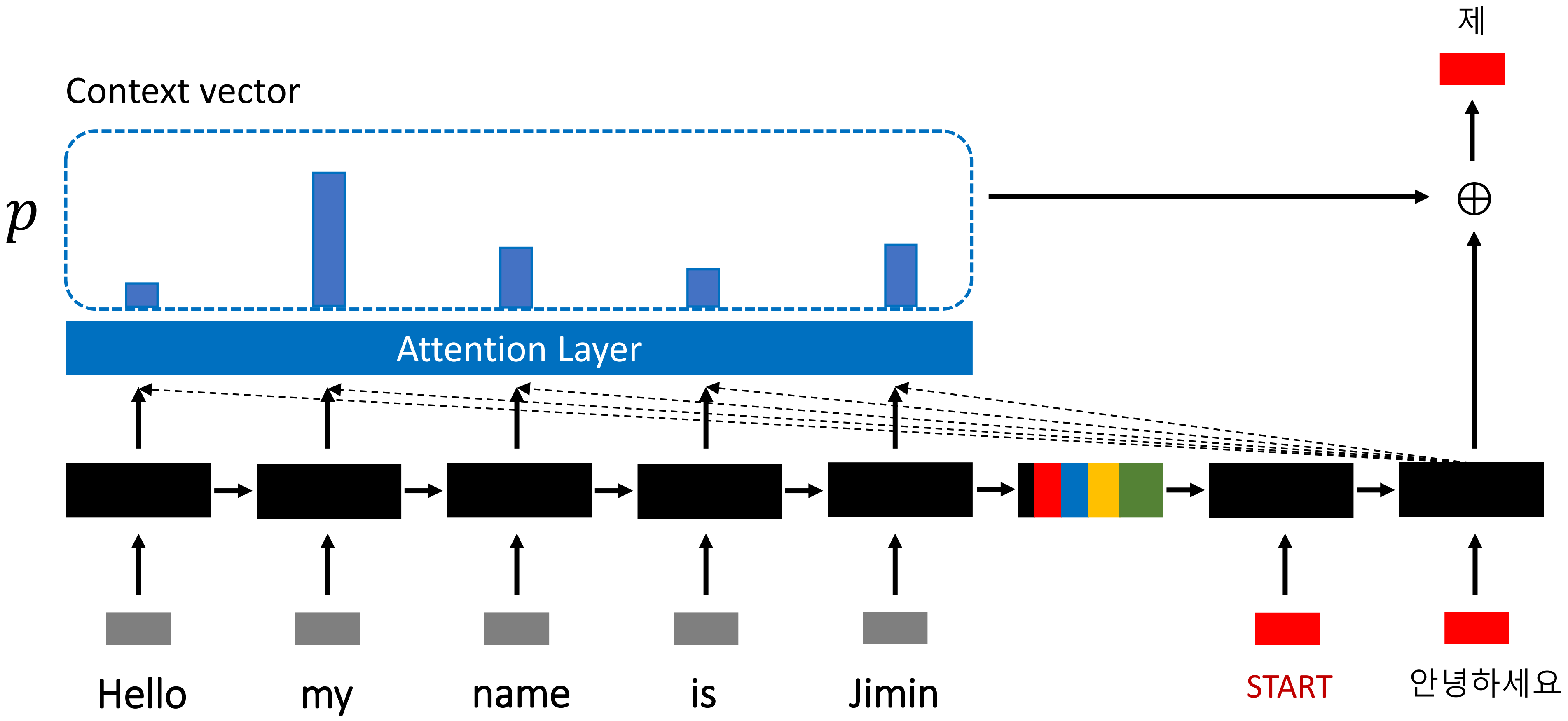


Seq2Seq with Attention



(+) Addresses long context issue

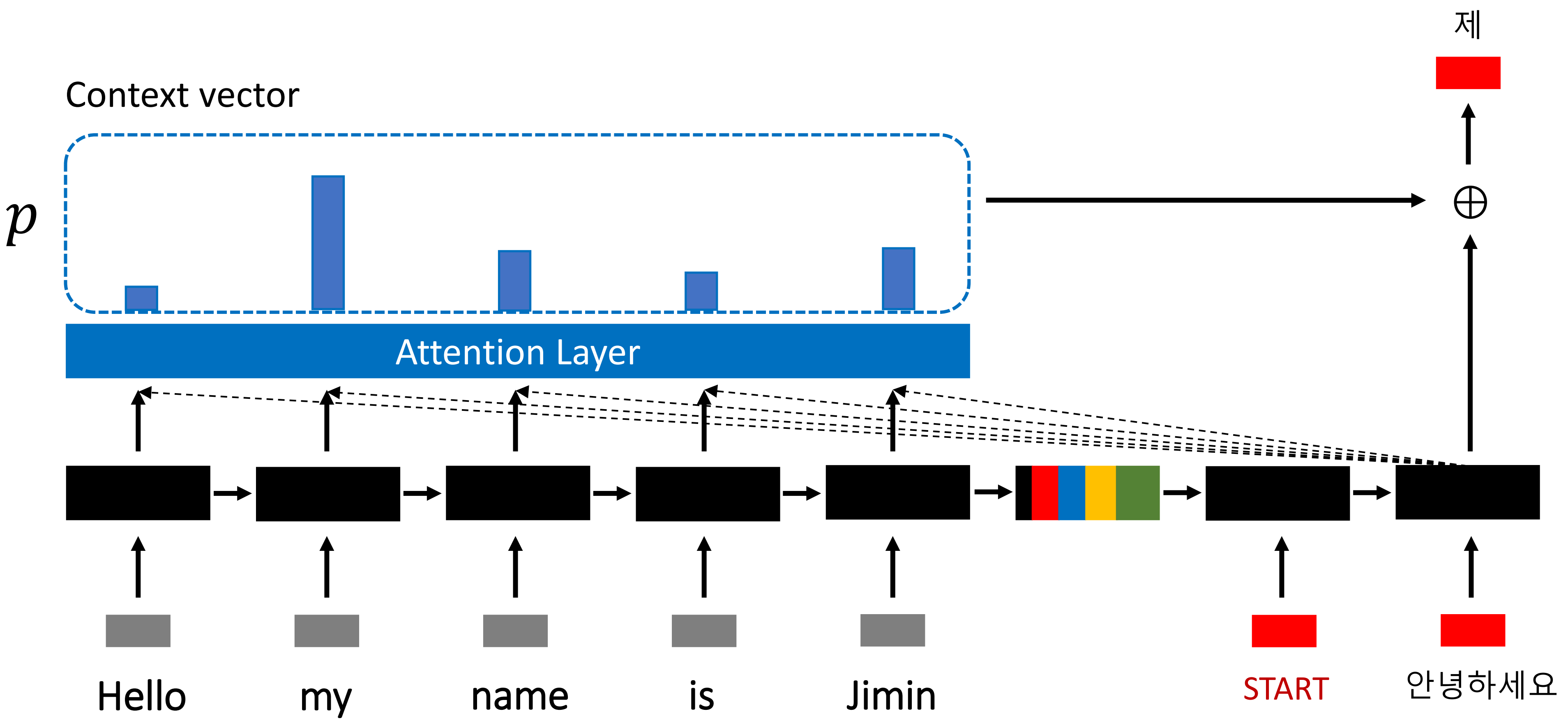
Seq2Seq with Attention



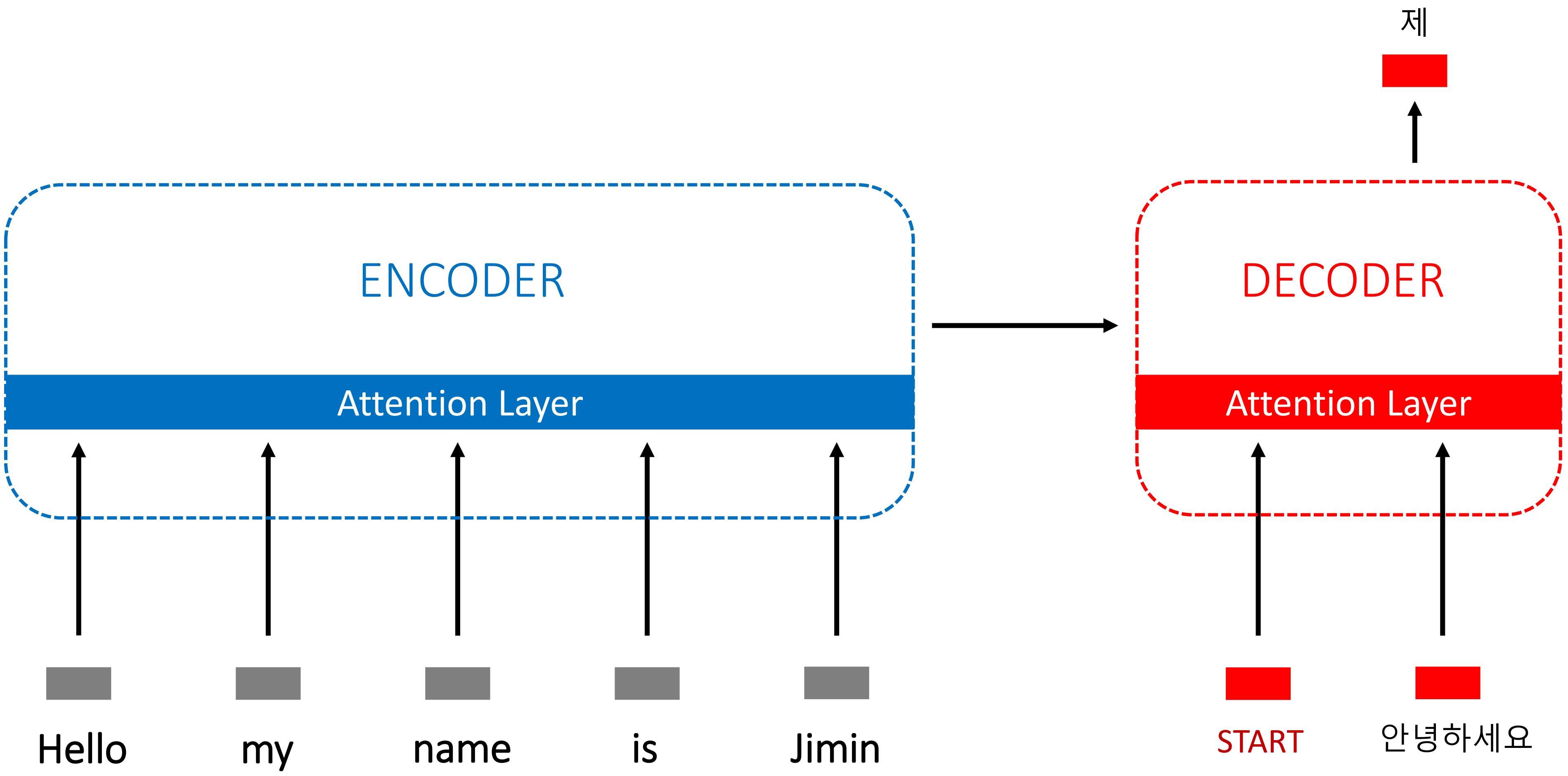
(+) Addresses long context issue

(-) Difficult to parallelize

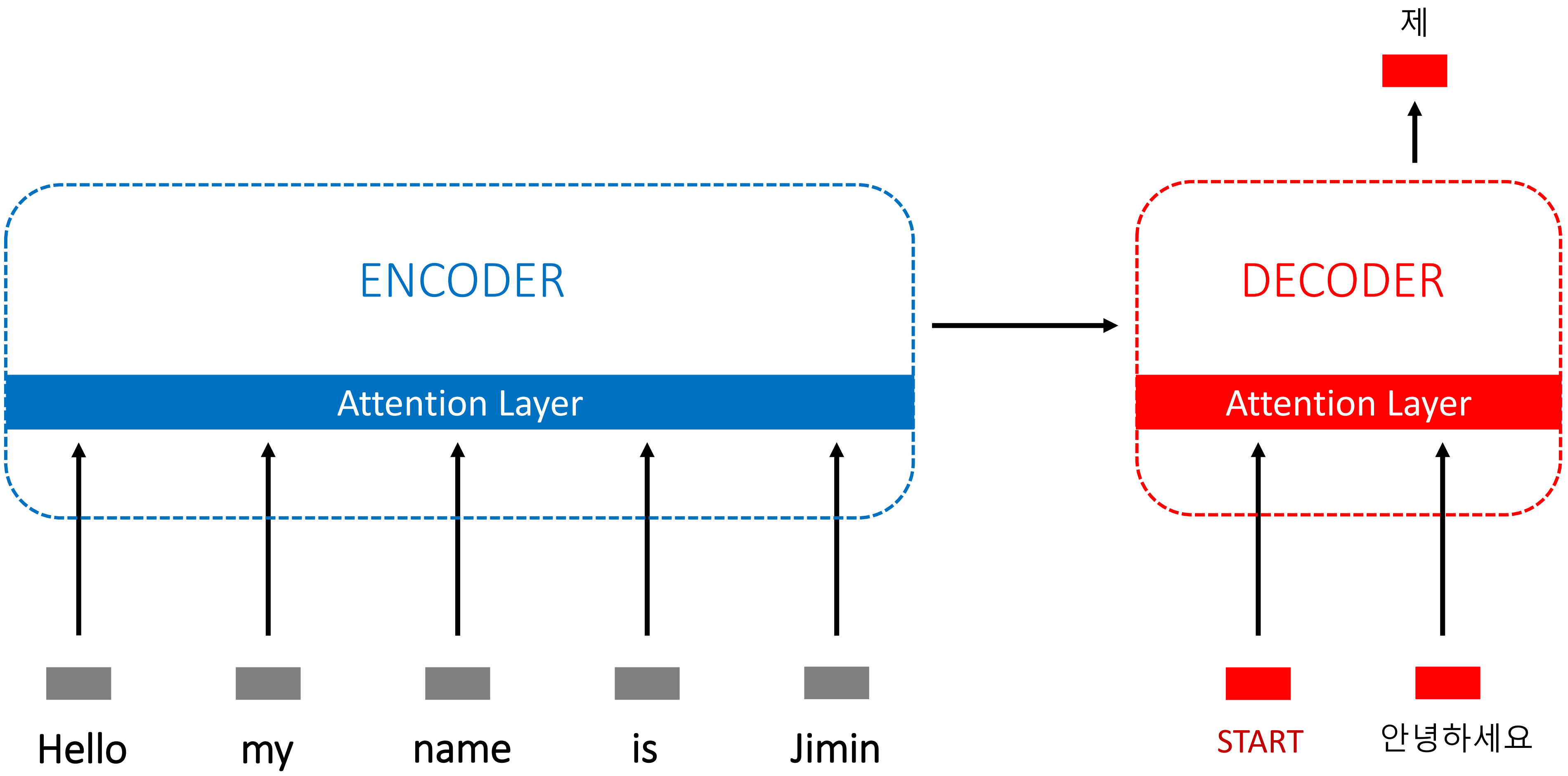
Attention is all you need (2017)



Attention is all you need (2017)



Attention is all you need (2017)



Attention without RNN is sufficient
Can utilize parallelization with GPUs

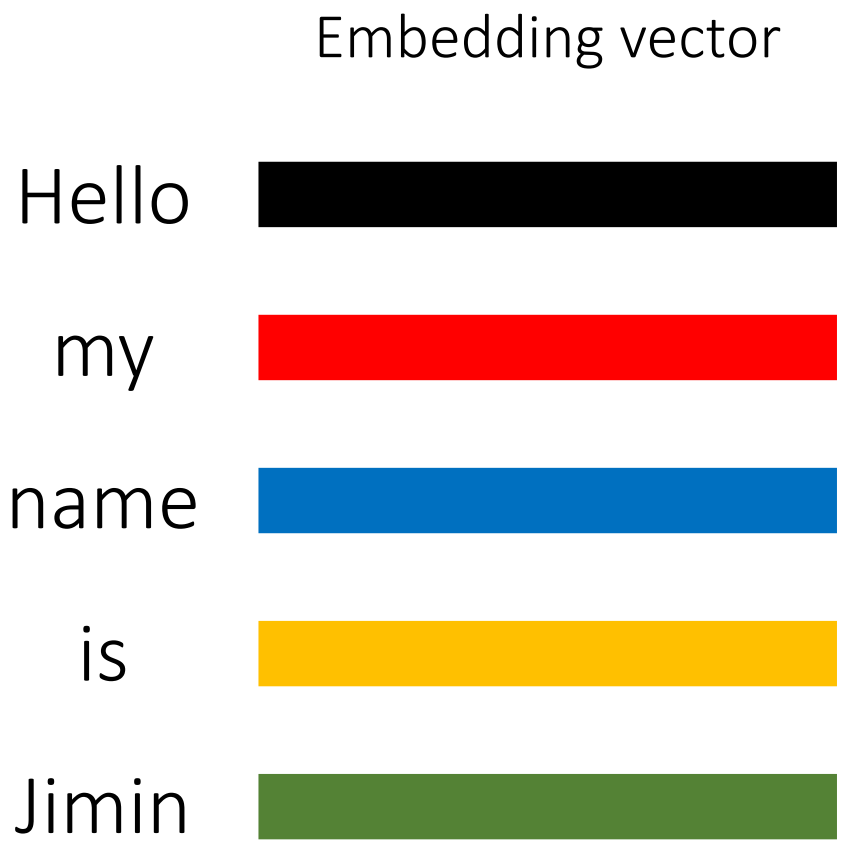
Self-attention layer

Overview

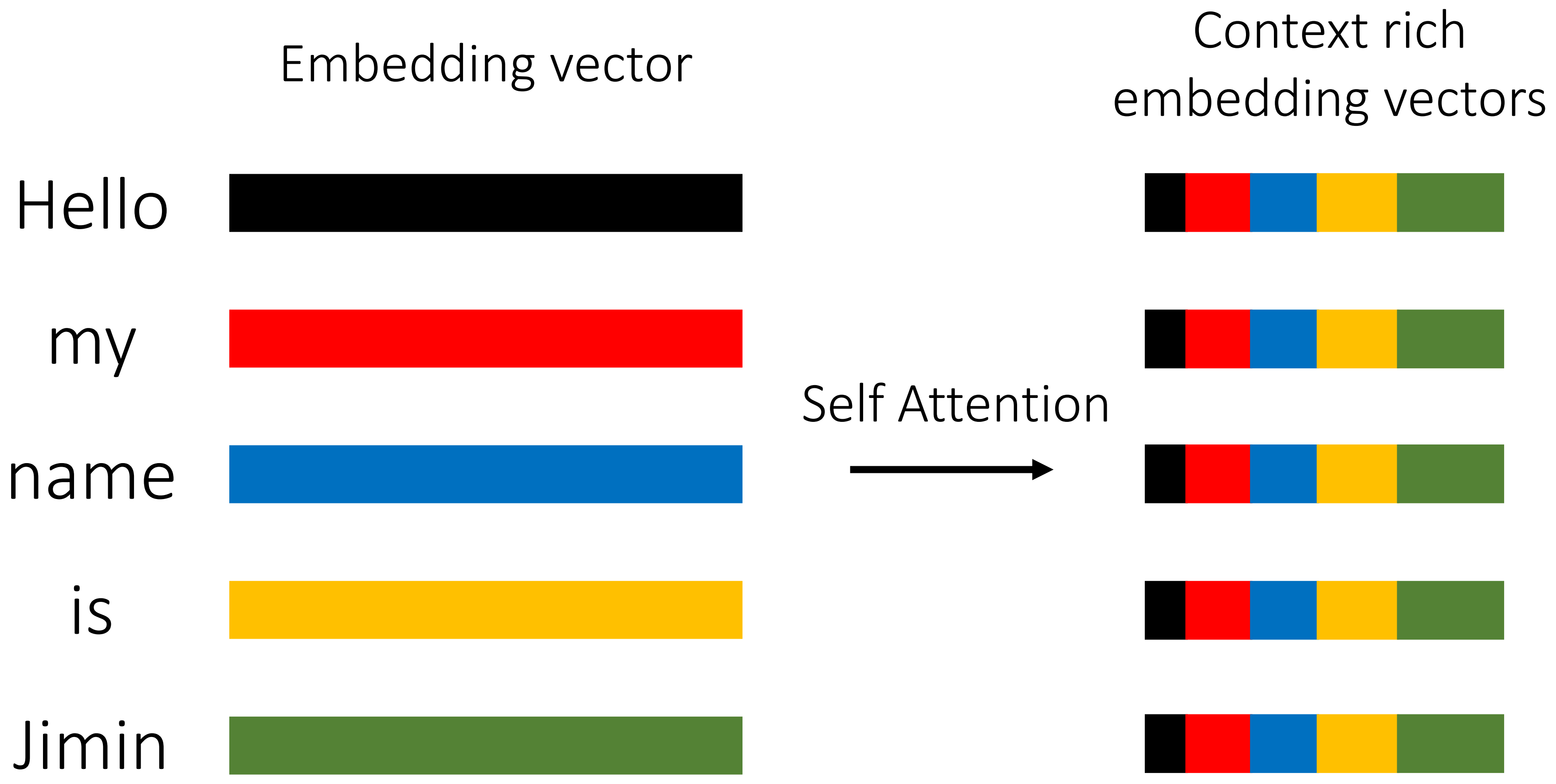
Key, Query, Value retrieval process

Multi-headed attention

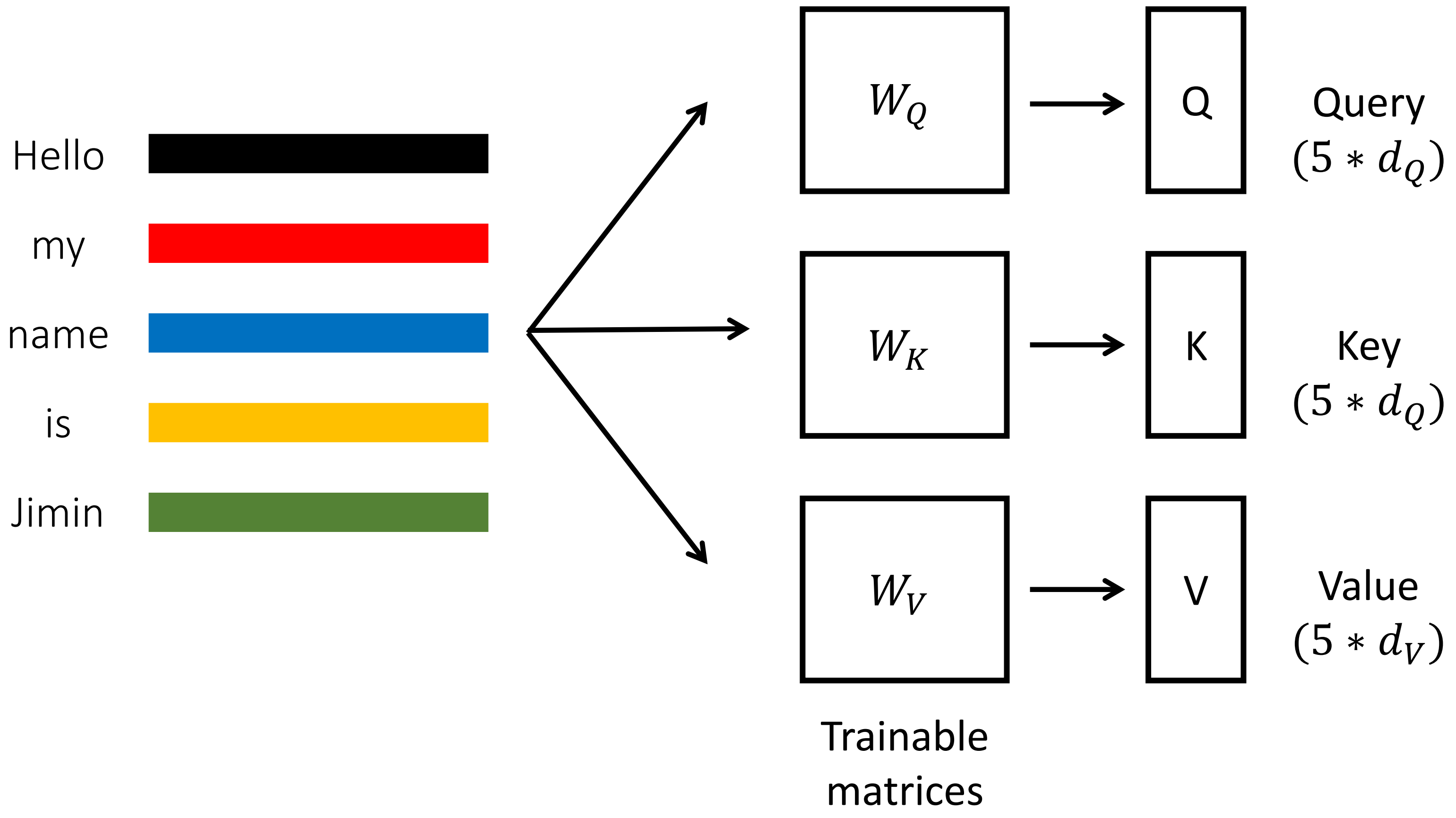
Overview of self-attention layer



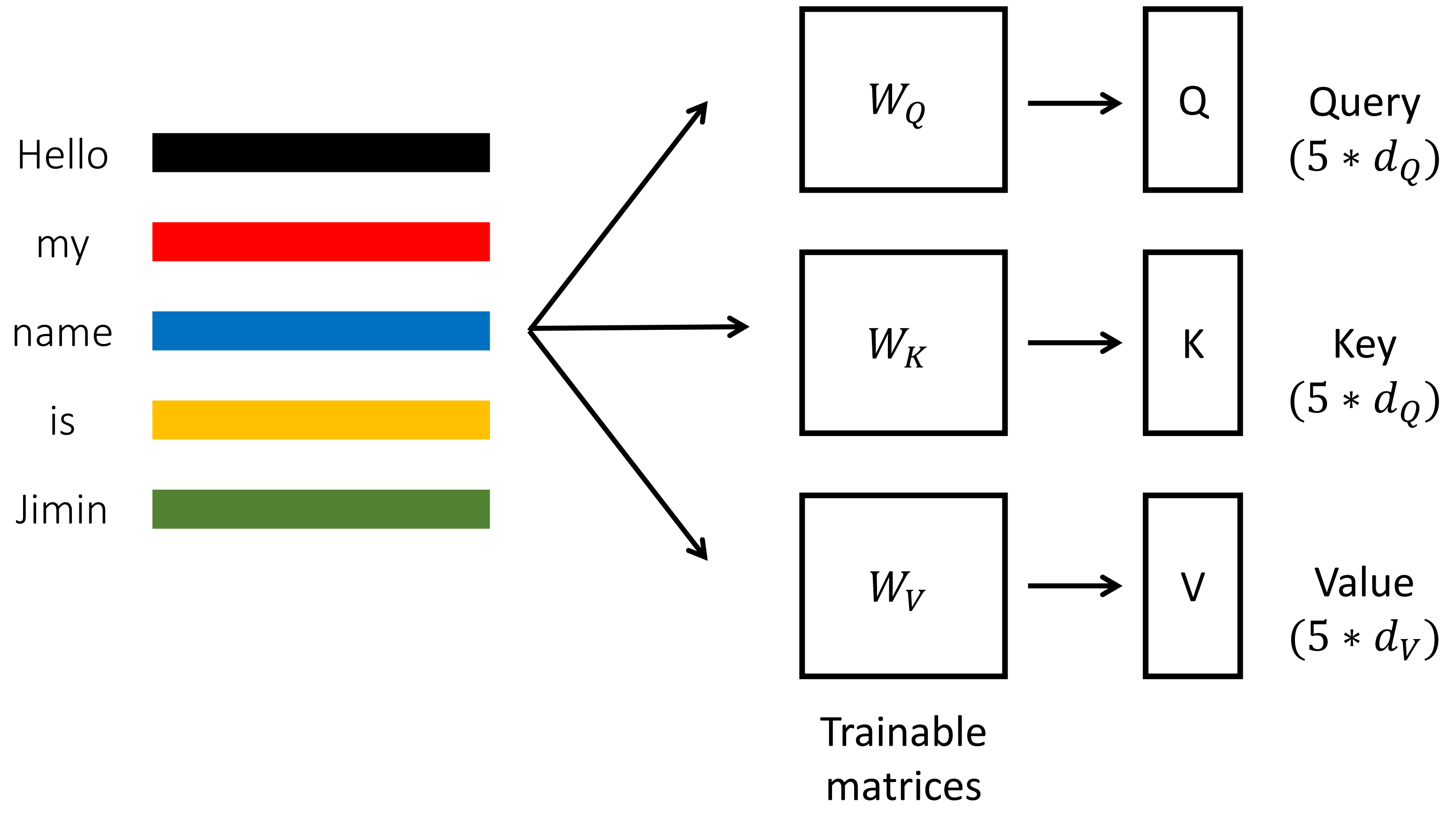
Overview of self-attention layer



Key, Query, Value retrieval

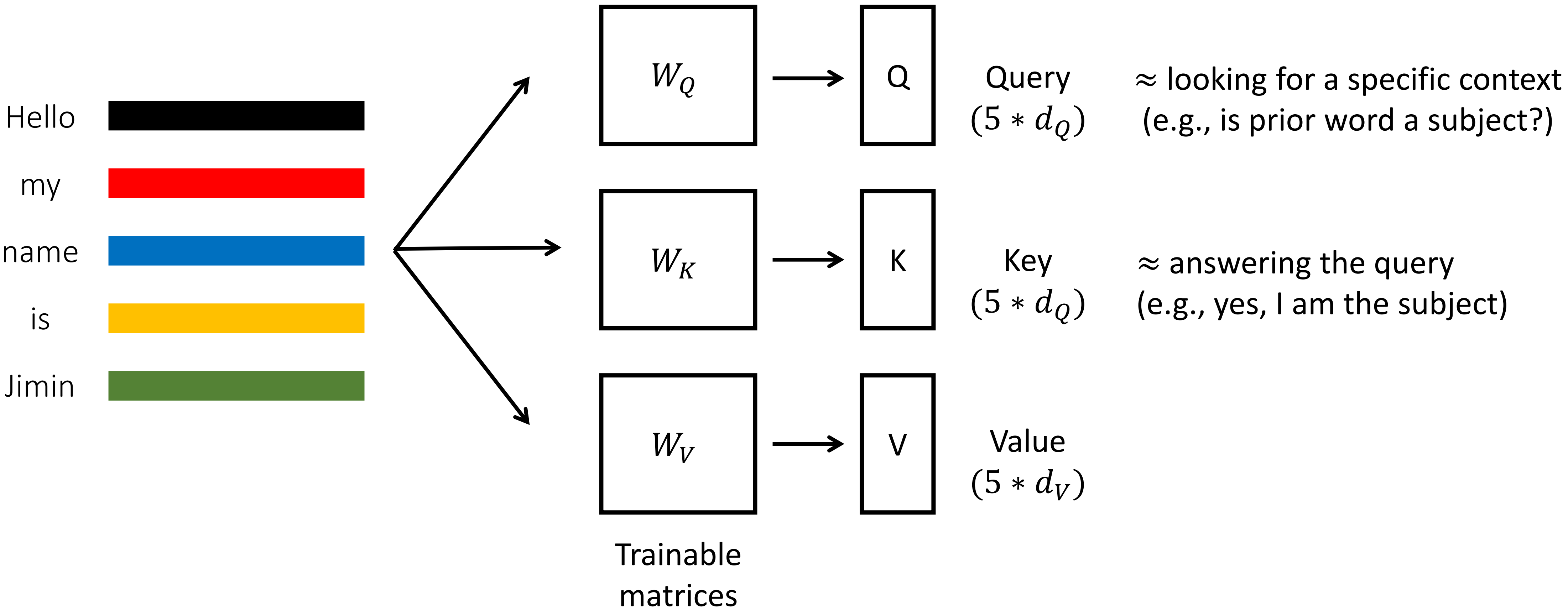


Key, Query, Value retrieval

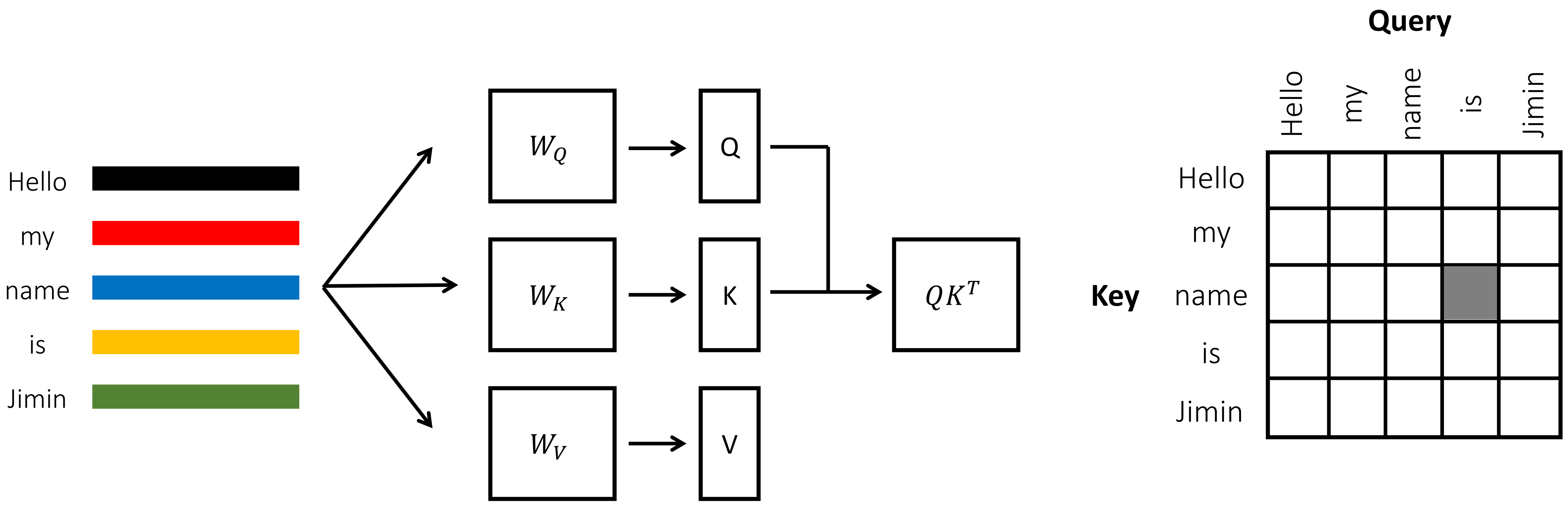


≈ looking for a specific context
(e.g., is prior word a subject?)

Key, Query, Value retrieval



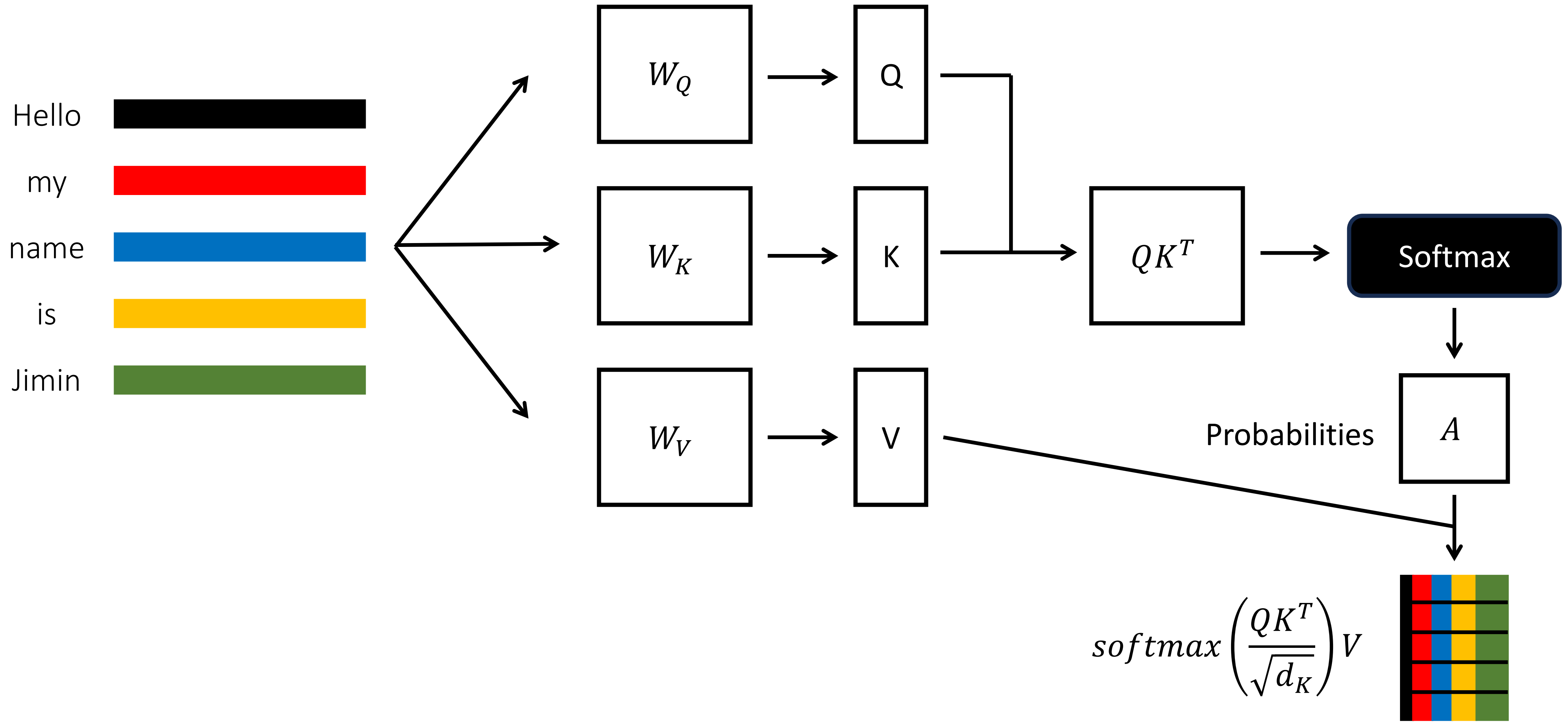
Key, Query, Value retrieval



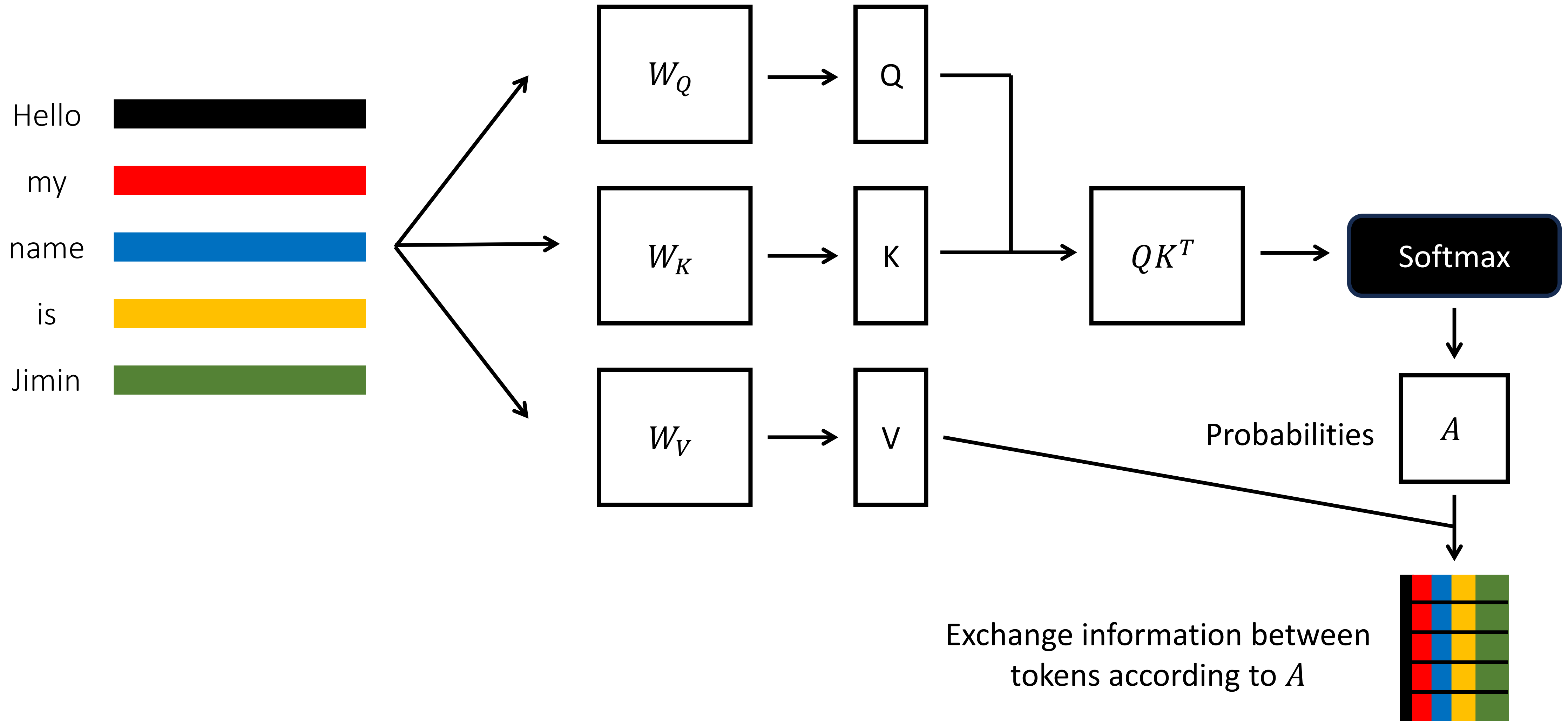
Query $\approx$ is prior word a subject?

Key $\approx$ yes, I am the subject

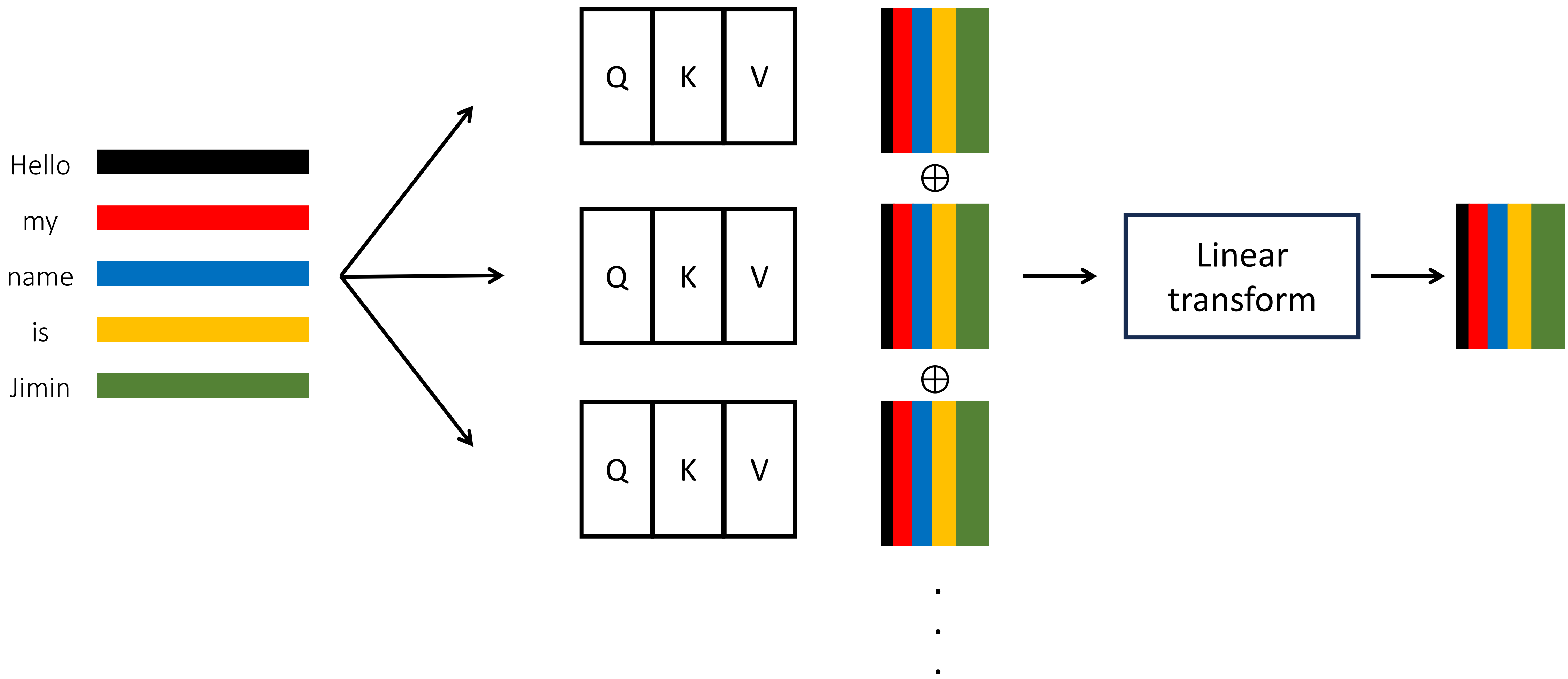
Key, Query, Value retrieval



Key, Query, Value retrieval



Multi-headed attention



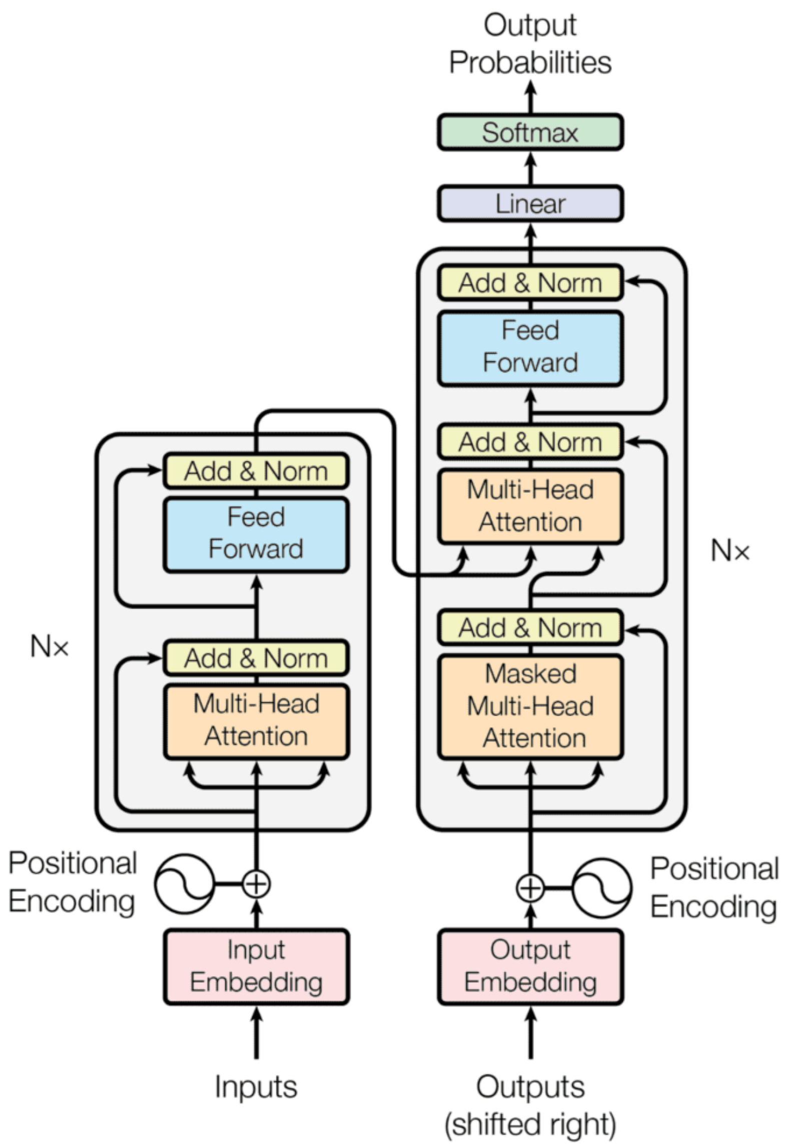
Transformer Architecture

Encoder

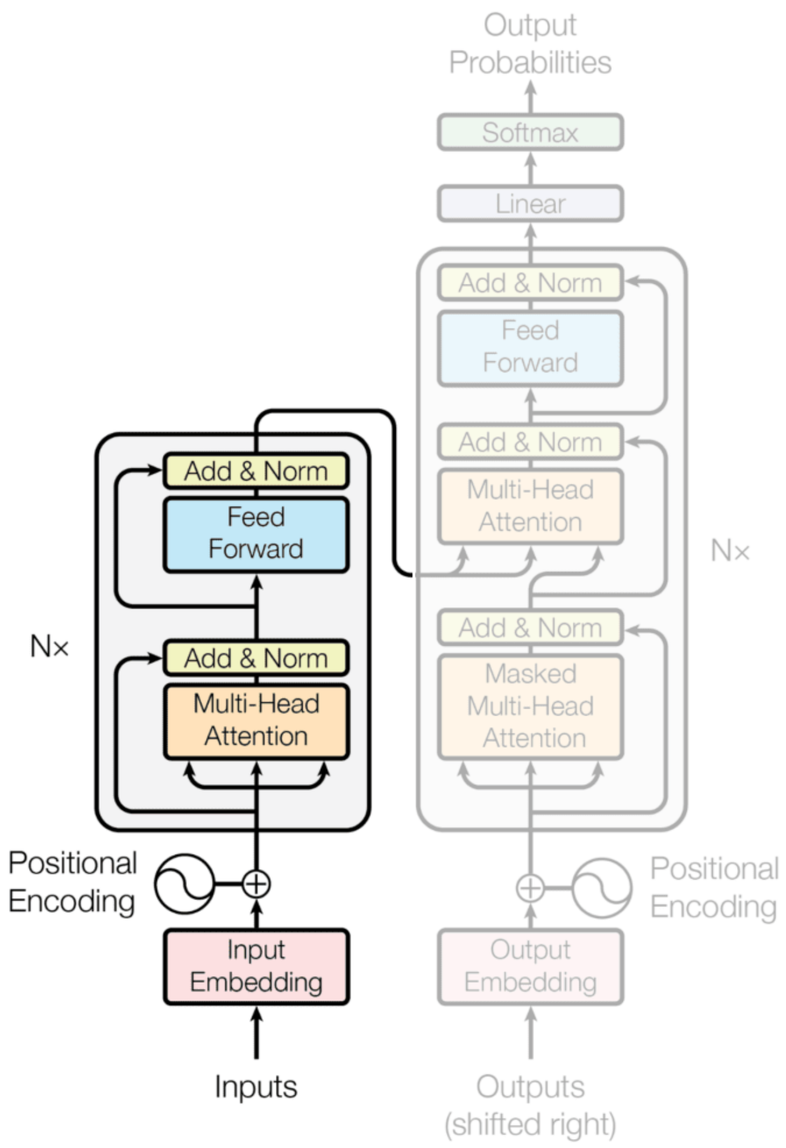
Decoder

Transformer vs RNN

Transformer Architecture



Encoder

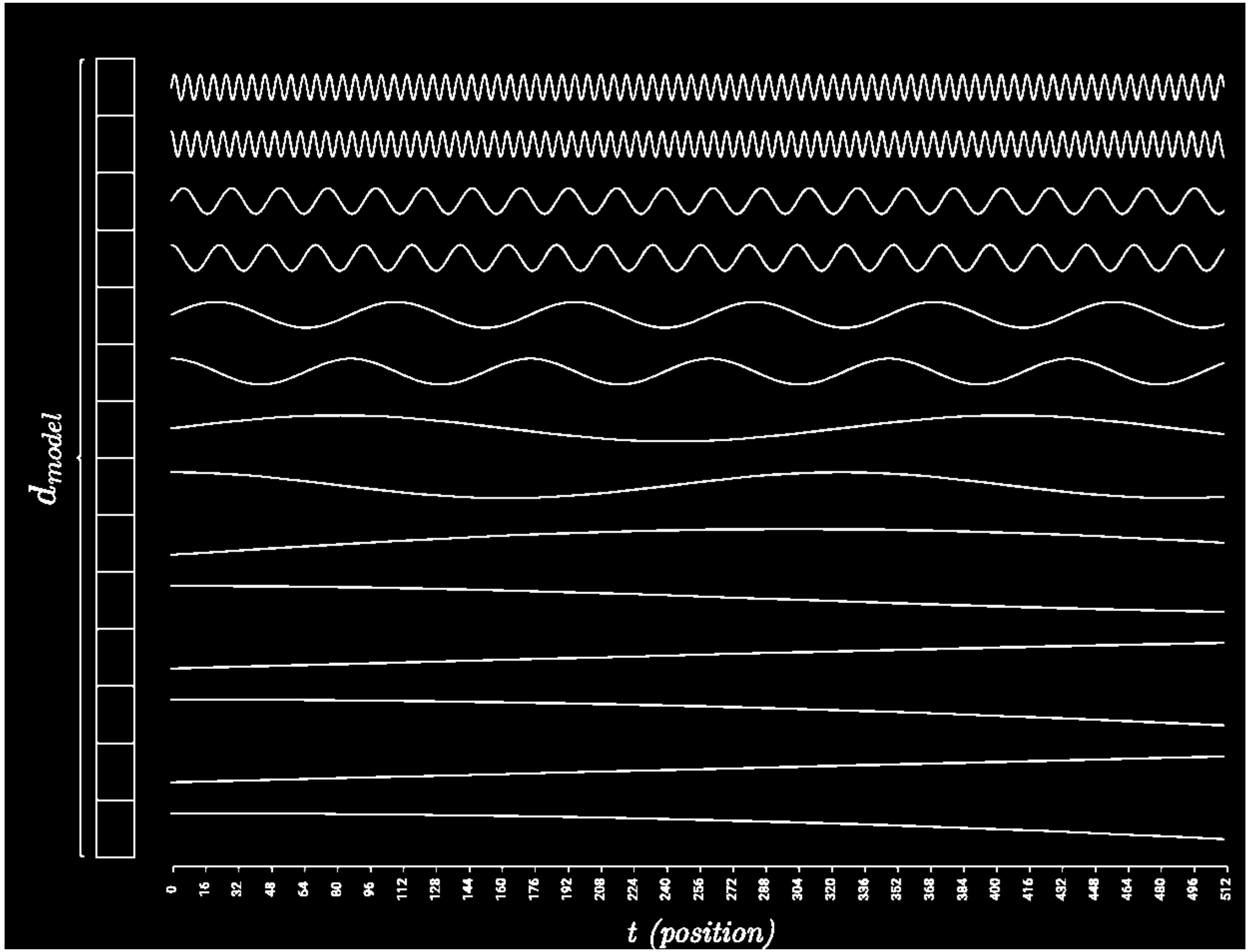


Encoder layer with

- Input embedding with positional encoding
- multi-headed self attention
- Residual connections, Layer norm & dropout

Positional Encoding

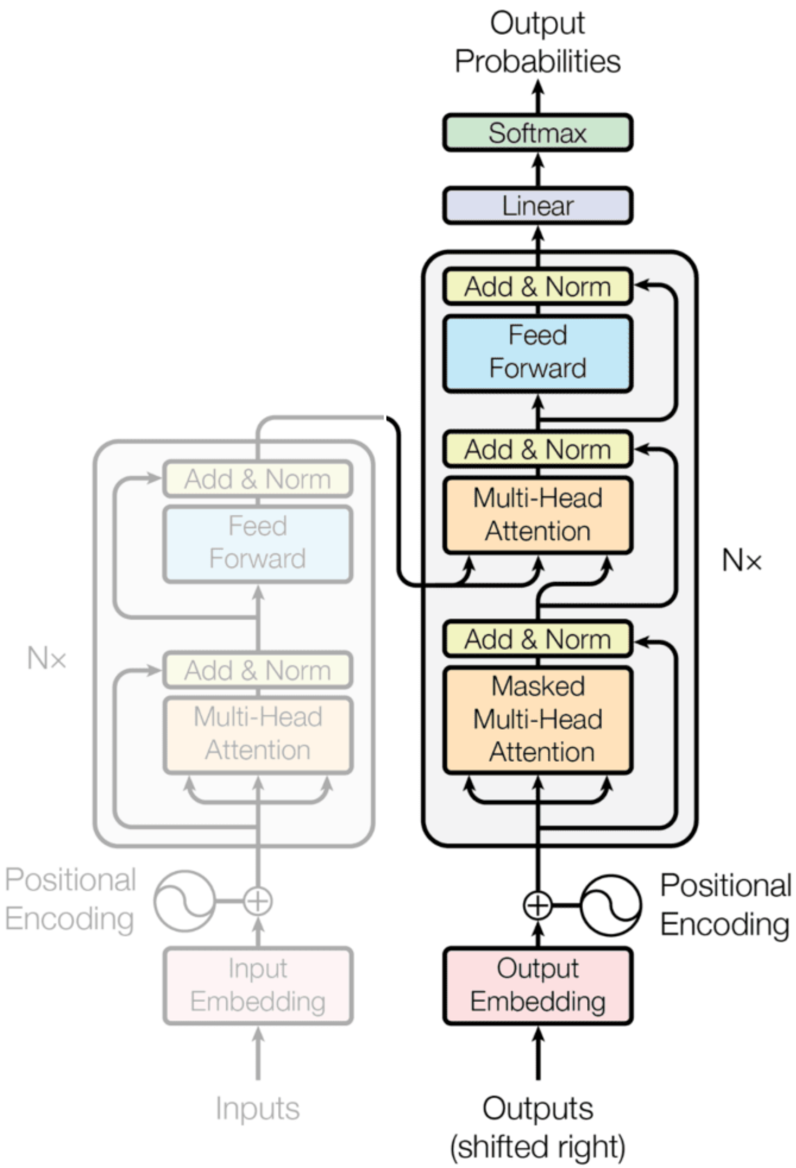
Hello my name is Jimin



$$k\ is\ even: \sin\left(\frac{t}{10000^{\frac{k}{d_{embedding}}}}\right)$$

$$k\ is\ odd: \cos\left(\frac{t}{10000^{\frac{k}{d_{embedding}}}}\right)$$

Decoder



Decoder layer with

- Masked multi-headed self attention
- Multiheaded cross attention
  - Inputs → Key, Query
  - Outputs → Value

Transformer vs RNN

	Transformers	RNNs
Sequential	No	Yes
Parallel computation	Yes	No
Long-term dependencies	Yes	Kind of
Scalability	Yes	Problematic
Fine tuning	Yes	Difficult

Transformer Example

Text classification of IMDB Dataset

Prepare Data

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
6	I sure would like to see a resurrection of a u...	positive
7	This show was an amazing, fresh & innovative i...	negative
8	Encouraged by the positive comments about this...	negative
9	If you like original gut wrenching laughter yo...	positive

Training	# samples	label
Positive	12,500	1
Negative	12,500	2

Testing	# samples	label
Positive	12,500	1
Negative	12,500	2

Prepare Data

```
import torchtext
from torch.utils.data import DataLoader
from collections import Counter

# Load dataset and initialize tokenizer
train_iter, test_iter = torchtext.datasets.IMDB(root='datasets', split=('train', 'test'))

label_counts = Counter()
for label, samples in train_iter:
    label_counts[label] += 1
print("Label distribution in train_iter:", label_counts)

label_counts = Counter()
for label, _ in test_iter:
    label_counts[label] += 1
print("Label distribution in test_iter:", label_counts)
```

```
Label distribution in train_iter: Counter({1: 12500, 2: 12500})
Label distribution in test_iter: Counter({1: 12500, 2: 12500})
```

Training	# samples	label
Positive	12,500	1
Negative	12,500	2

Testing	# samples	label
Positive	12,500	1
Negative	12,500	2

Prepare Data

```
def yield_tokens(data_iter):
    for _, text in data_iter:
        yield tokenizer(text)

# Create vocabulary with special tokens for padding and unknown words
vocab = torchtext.vocab.build_vocab_from_iterator(yield_tokens(train_iter), specials=["<unk>", "<pad>"])
vocab.set_default_index(vocab["<unk>"])
```

Example sample: This movie was fantastic!

tokenizer

['This', 'movie', 'was', 'fantastic', '!']

vocab

What the model sees: [14, 21, 17, 762, 36]

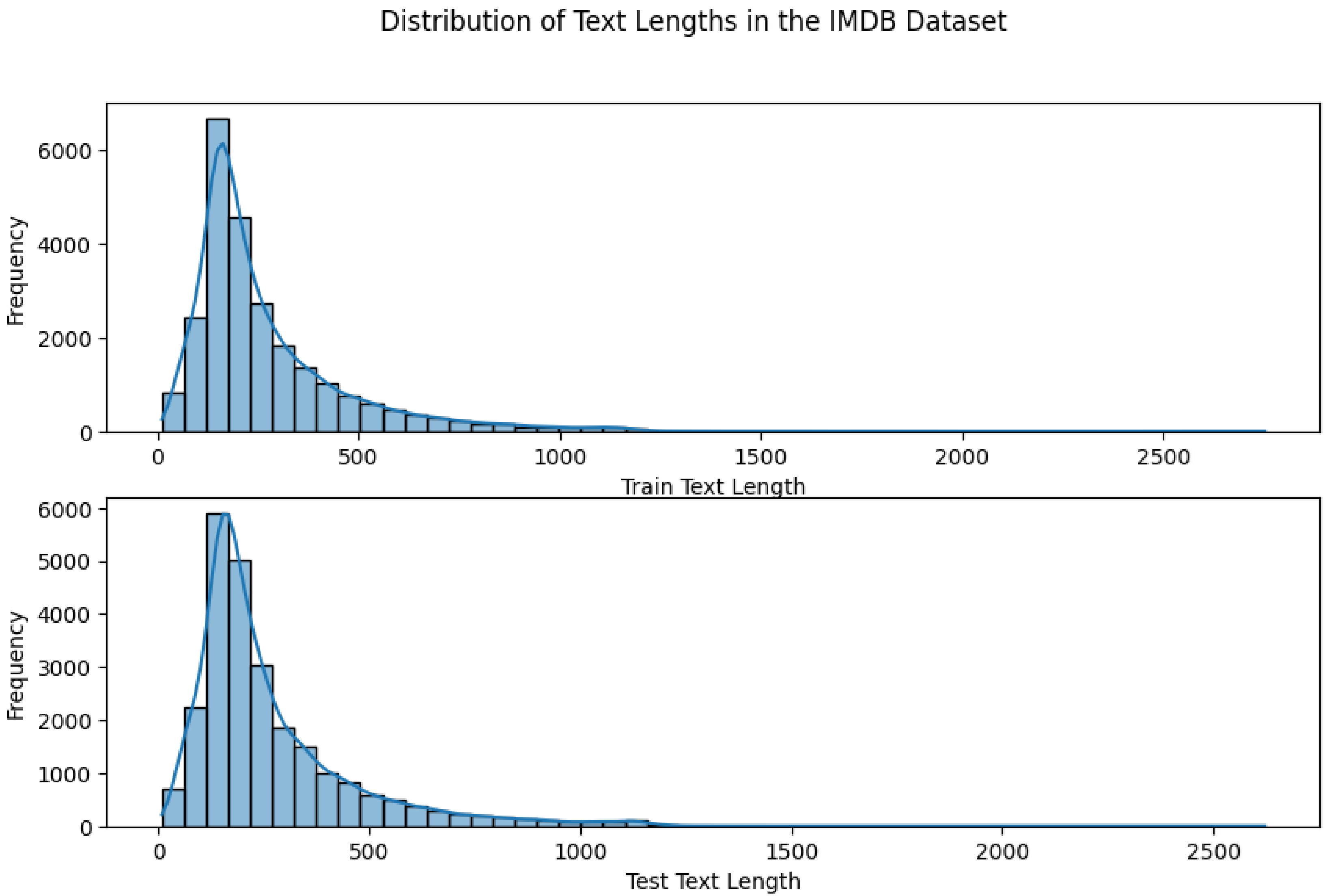
Prepare Data

```
def yield_tokens(data_iter):
    for _, text in data_iter:
        yield tokenizer(text)

# Create vocabulary with special tokens for padding and unknown words
vocab = torchtext.vocab.build_vocab_from_iterator(yield_tokens(train_iter), specials=["<unk>", "<pad>"])
vocab.set_default_index(vocab["<unk>"])
```

```
def text_pipeline(x):
    return vocab(tokenizer(x))

# Example: Test the pipeline on a sample text
sample_text = "This movie was fantastic!"
print(text_pipeline(sample_text))
```

```
[14, 21, 17, 762, 36]
```

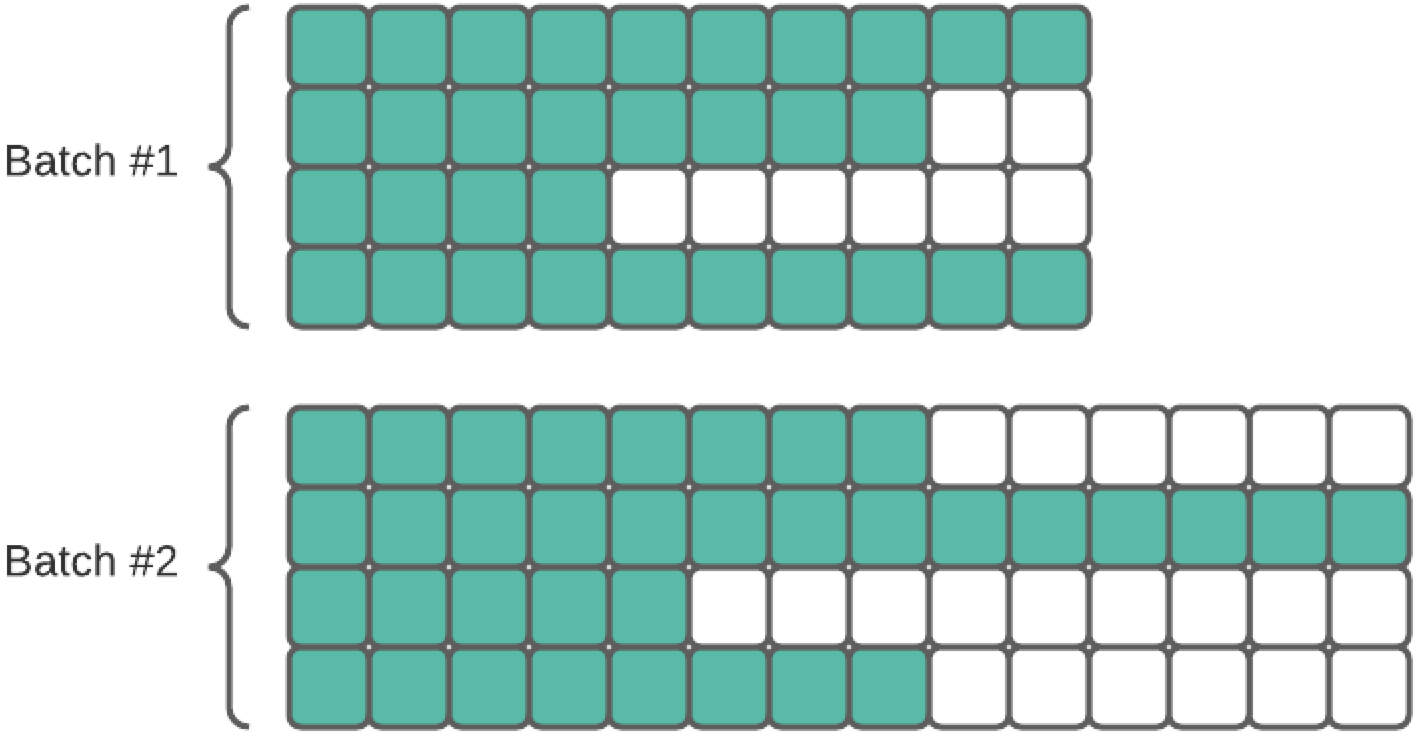
Prepare Data



Prepare Data

```
from torch.nn.utils.rnn import pad_sequence

# Define collate function for padding and batching
# setting a max_seq_len helps with estimating the max gpu memory usage
def collate_batch(batch, max_seq_len=1024):
    labels, texts = zip(*batch)
    # the labels start at 1 but predictions start at 0. To align them, we modify lables
    labels = torch.tensor(labels,dtype=torch.long)-1

    text_list = []
    for text in texts:
        # Truncate or pad to max_seq_len
        tokenized_text = text_pipeline(text)
        if len(tokenized_text) > max_seq_len:
            tokenized_text = tokenized_text[:max_seq_len]  # Truncate if longer than max_seq_len
        else:
            # Pad if shorter than max_seq_len
            tokenized_text = tokenized_text + [vocab["<pad>"]] * (max_seq_len - len(tokenized_text))

        text_list.append(torch.tensor(tokenized_text, dtype=torch.long))

    padded_texts = torch.stack(text_list)  # Stack the sequences into a tensor
    return padded_texts, labels
```

Prepare Data



Prepare Data

```
class IMDBDataset(Dataset):
    def __init__(self, data_iter):
        self.data_iter = list(data_iter)  # Converting the iterator to a list for easier access

    def __len__(self):
        return len(self.data_iter)

    def __getitem__(self, idx):
        label, text = self.data_iter[idx]
        return label, text

train_iter, test_iter = torchtext.datasets.IMDB(root='datasets', split=('train', 'test'))
train_dataset = IMDBDataset(train_iter)
test_dataset = IMDBDataset(test_iter)

batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, collate_fn=collate_batch)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, collate_fn=collate_batch)

# Train_loader and test_loader sanity check
# Initialize counter
label_counter = Counter()

# Iterate through batches in train_loader
for texts, labels in train_loader:
    label_counter.update(labels.tolist())
print("Label counts:", label_counter)
```

Define Pytorch Dataloader

Use collate_batch() function earlier to Dataloader.

Sanity check by counting the labels for train dataloader.

Define Model

```python
class TransformerModel(nn.Module):
    def __init__(self, vocab_size, embed_size, num_heads, num_encoder_layers, num_classes, dropout=0.1):

        super(TransformerModel, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.transformer = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(embed_size, num_heads, embed_size * 2, dropout),
            num_encoder_layers
        )
        self.fc = nn.Linear(embed_size, num_classes)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        x = self.embedding(x)  # Embedding layer
        x = x.permute(1, 0, 2)  # Transformer expects (seq_len, batch_size, embedding_size)
        x = self.transformer(x)  # Apply transformer
        x = x.mean(dim=0)  # Pooling (take the mean of all tokens in the sequence)
        x = self.dropout(x)
        x = self.fc(x)  # Final classification layer
        return x
```

Embedding layer

Transformer encoder

Classification layer

1. Input sequence
2. Embedding layer
3. Encoder
4. Pooling
5. Classification layer

Define Hyperparameters

```python
# Check for device compatibility, prioritizing CUDA, then MPS for MacBooks with Apple Silicon, and defaulting to CPU
if torch.cuda.is_available():
    device = torch.device("cuda")
elif torch.backends.mps.is_available():
    device = torch.device("mps")
else:
    device = torch.device("cpu")

print(f"Using device: {device}")

# Initialize the model,
embed_size = 32
num_heads = 4
num_encoder_layers = 2
num_classes = 2  # Positive or negative sentiment
model = TransformerModel(len(vocab), embed_size, num_heads, num_encoder_layers, num_classes)

# Initialize loss function, and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
model.to(device)
```

Identify Tracked Values

```
num_epochs = 5
train_losses = np.zeros(num_epochs)
train_accuracies = np.zeros(num_epochs)

test_losses = np.zeros(num_epochs)
test_accuracies = np.zeros(num_epochs)
```

Placeholders for training and testing losses/accuracies

Train Model

```python
def train_epoch(model, train_loader, loss_fn, optimizer):
    model.train()
    epoch_loss = 0
    epoch_accuracy = 0
    # total_batches = 0
    total_batches = len(train_loader)

    for texts, labels in tqdm(train_loader):
        texts, labels = texts.to(device), labels.to(device)
        optimizer.zero_grad()

        # Forward pass
        outputs = model(texts)

        # Compute loss and gradients
        loss = loss_fn(outputs, labels)
        loss.backward()

        # Update model parameters
        optimizer.step()

        # Calculate accuracy
        preds = torch.argmax(outputs, dim=1)
        correct = (preds == labels).sum().item()
        accuracy = correct / labels.size(0)

        epoch_loss += loss.item()
        epoch_accuracy += accuracy

    return epoch_loss / total_batches, epoch_accuracy / total_batches
```

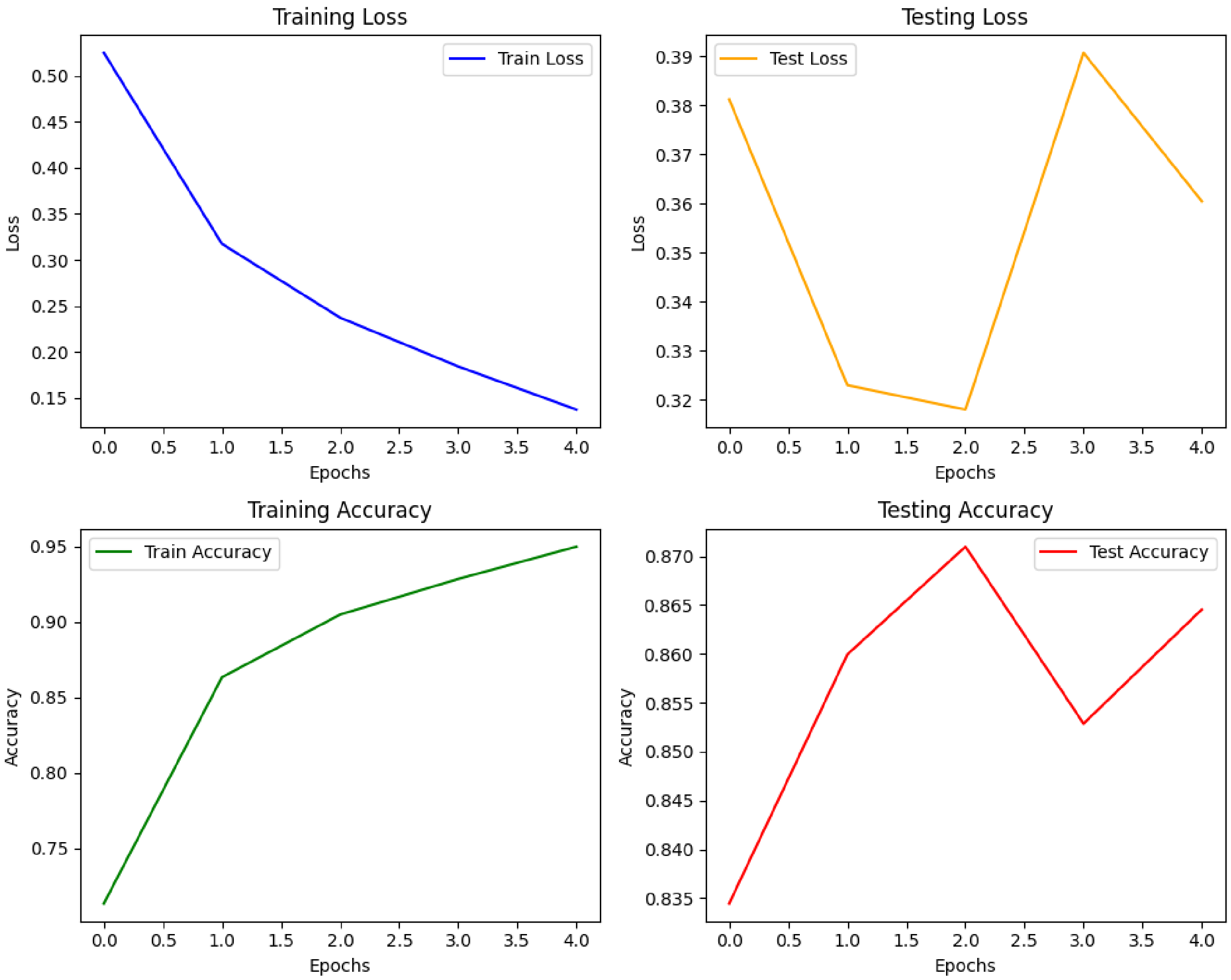
```python
def evaluate(model, test_loader, loss_fn):
    model.eval()
    epoch_loss = 0
    epoch_accuracy = 0
    total_batches = len(test_loader)

    with torch.no_grad():
        for texts, labels in tqdm(test_loader):
            texts, labels = texts.to(device), labels.to(device)
            # Forward pass
            outputs = model(texts)

            # Compute loss
            loss = loss_fn(outputs, labels)

            # Calculate accuracy
            preds = torch.argmax(outputs, dim=1)
            correct = (preds == labels).sum().item()
            accuracy = correct / labels.size(0)

            epoch_loss += loss.item()
            epoch_accuracy += accuracy

    return epoch_loss / total_batches, epoch_accuracy / total_batches
```

Train Model

```
for epoch in range(num_epochs):
    start_time = time.time()

    # Train for one epoch
    train_loss, train_accuracy = train_epoch(model, train_loader, loss_fn, optimizer)
    train_losses[epoch] = train_loss
    train_accuracies[epoch] = train_accuracy

    # Evaluate on the test set
    test_loss, test_accuracy = evaluate(model, test_loader, loss_fn)
    test_losses[epoch] = test_loss
    test_accuracies[epoch] = test_accuracy
    end_time = time.time()

    print(f"Epoch [{epoch+1}/{num_epochs}] | Time: {end_time - start_time:.2f}s")
    print(f"Train Loss: {train_loss:.4f} | Train Accuracy: {train_accuracy:.4f}")
    print(f"Test Loss: {test_loss:.4f} | Test Accuracy: {test_accuracy:.4f}")
```

```
100%|██████████████████████████████████████████████| 782/782 [00:54<00:00, 14.38it/s]
100%|██████████████████████████████████████████████| 782/782 [00:10<00:00, 77.67it/s]
Epoch [1/5] | Time: 64.46s
Train Loss: 0.5248 | Train Accuracy: 0.7138
Test Loss: 0.3812 | Test Accuracy: 0.8345
```

Visualize and Evaluate Model



Lab Assignment

Text classification on AG News Dataset

AG News Dataset

Topic:
Sci/Tech

Title:
Your PC May Be Less Secure Than You Think

Description:
Most users think their computer is safe from adware and spyware--but they're wrong. A survey conducted by Internet service provider America Online found that 20 percent of home computers were infected by

Train: 120000

Test: 7600

1: World
2: Sports
3: Business
4: Sci/Tech

AG News Dataset

Topic:
Sci/Tech

Title:
Your PC May Be Less Secure Than You Think

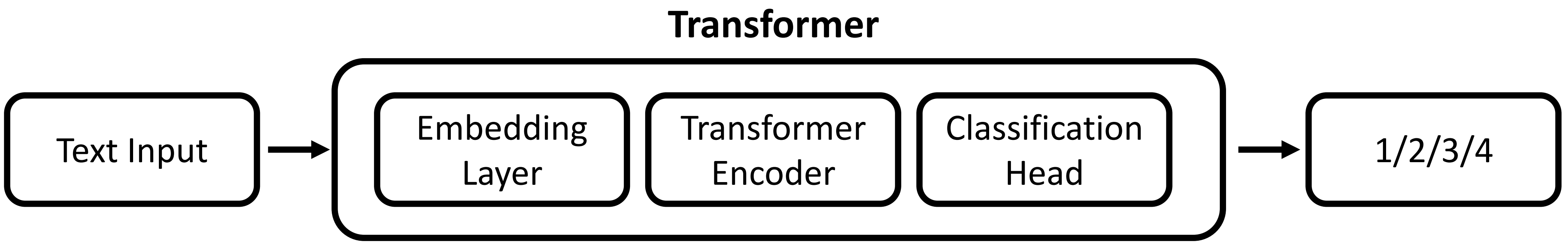
Description:
Most users think their computer is safe from adware and spyware--but they're wrong. A survey conducted by Internet service provider America Online found that 20 percent of home computers were infected by

Train: 120000

Test: 7600

1: World
2: Sports
3: Business
4: Sci/Tech

AG News Text Classification



In this exercise, you will use Transformer encoder to perform text classification on **AG News dataset**

Before training, make sure to **tokenize** and **pre-process data** into trainable formats (e.g., batching, data loading)

You are free to design architectures such as # of encoder layers, activation functions, etc.

You are also free to pick your hyperparameters e.g., total epochs, batch size, learning rate, optimizer, etc.

After training,

1. Plot **training/testing loss** and **training/testing accuracy**
2. Print out snippets of **3 examples that were successfully classified** and **3 examples that were incorrectly classified.**