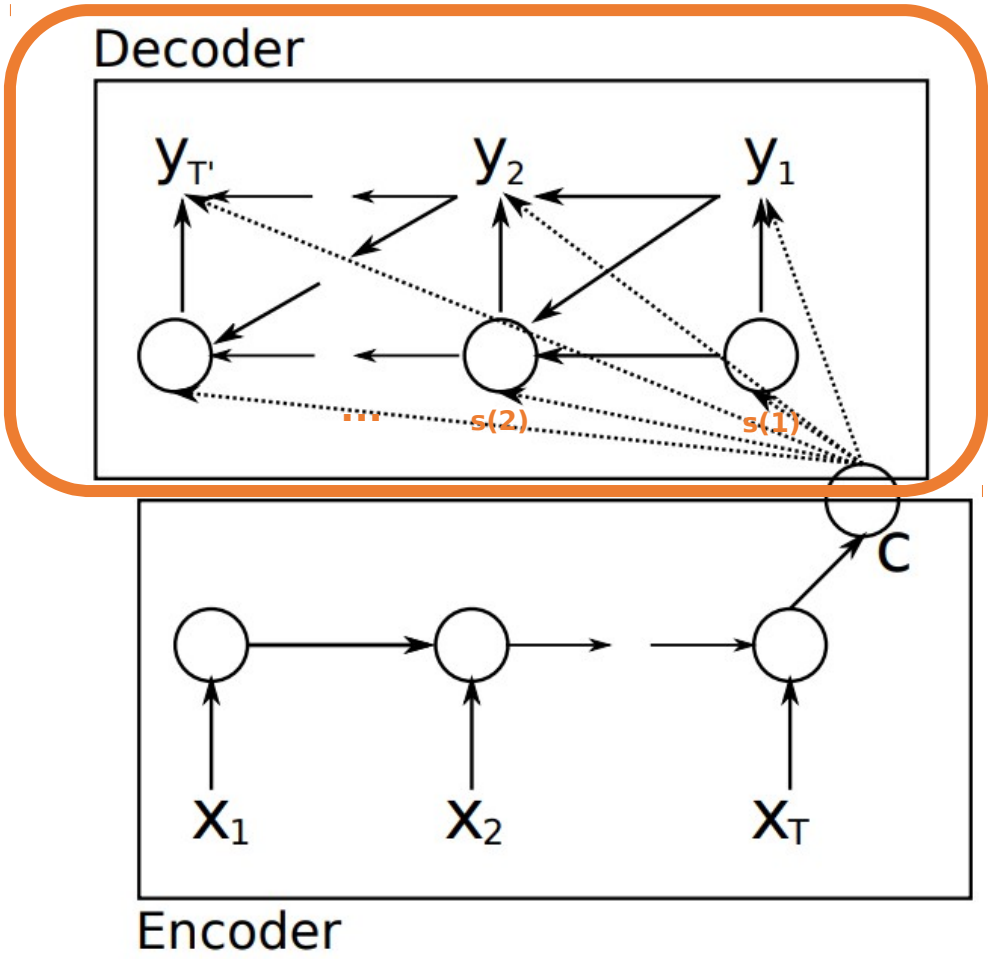
NMT(Neural Machine Translator) Decoder Sample Slides

By Mohit Kumar

NMT:Encoder-Architecture:Decoder

- **Output:**
  - word sequence (target) $y = (y_1, \cdots, y_{T'})$
- **Decoder Transition:**
  - $s_t = RNN_{dec}(s_{t-1}, y_{t-1}, \boldsymbol{c})$
- **Conditional recurrent language model**
  - $p(\vec{\boldsymbol{y}}|\vec{\boldsymbol{x}}) = \prod_{i=1}^{T'} p(y_t|y_1 \cdots, y_{t-1}; \boldsymbol{c})$
  - $p(y_t|y_1 \cdots, y_{t-1}; \boldsymbol{c}) = g(s_t, y_{t-1}, \boldsymbol{c})$



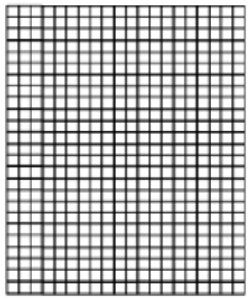
g(·): Function that produce valid probs, e.g. softmax

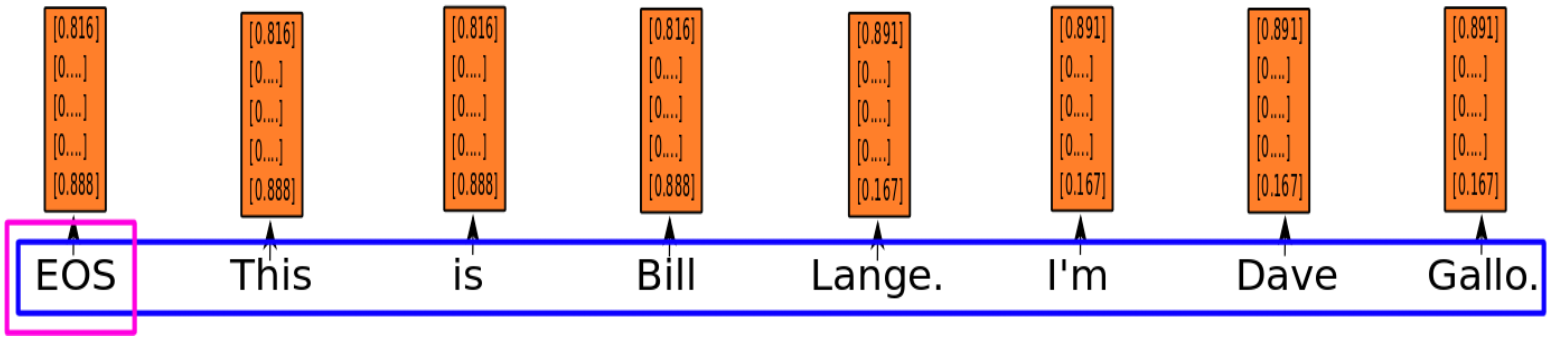
NMT:Detailed:Encoder-Decoder:Decoder:Graph(step-3)

```python
def process_encoding_input(target_data, word2int, batch_size):
    ending = tf.strided_slice(target_data, [0, 0], [batch_size, -1], [1, 1])
    decoding_input = tf.concat([tf.fill([batch_size, 1], word2int['TOKEN_GO']), ending], 1)
    return decoding_input
```

```
[[37 16 37 37 37 37 37 39 38 38 38 38 38 38 38]
 [40 37 16 37 37 37 37 37 39 38 38 38 38 38 38]
```

- Pick up the corresponding English sentence.
- End of statement marked.
- GO marker added.

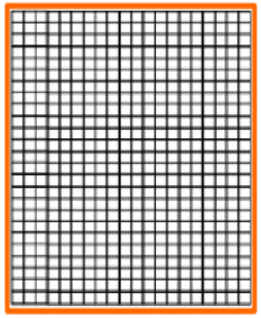




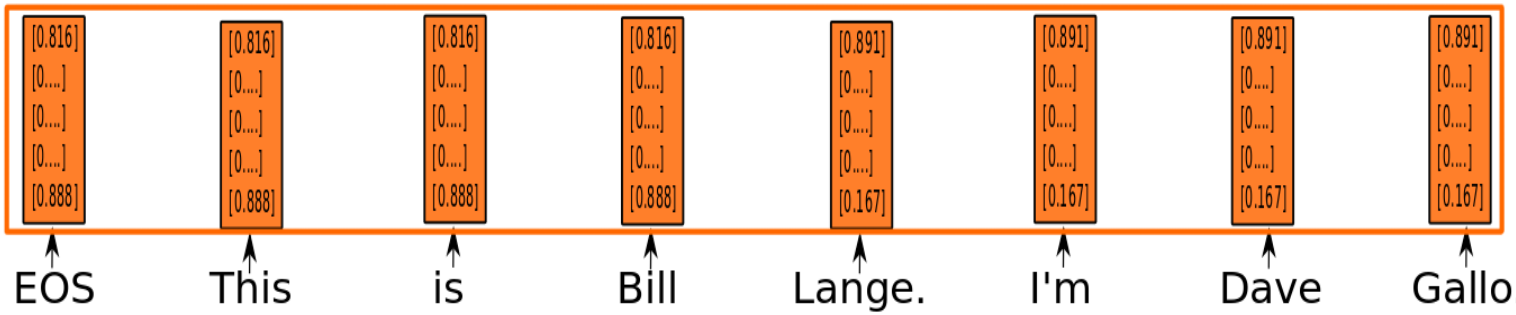
NMT:Detailed:Encoder-Decoder:Decoder:Graph(step-4)

```
decoding_input = process_encoding_input(target_en_data, word2int_en, batch_size)

decoding_embed_input = tf.nn.embedding_lookup(en_embeddings_matrix, decoding_input)
```

- Pick up the embeddings from english embeddings.
- Not a "tf" variable so no learning english embeddings while training.



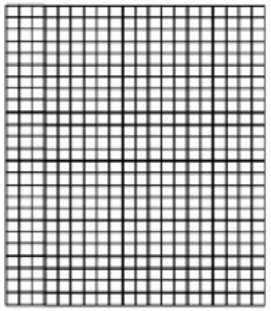
English Embeddings

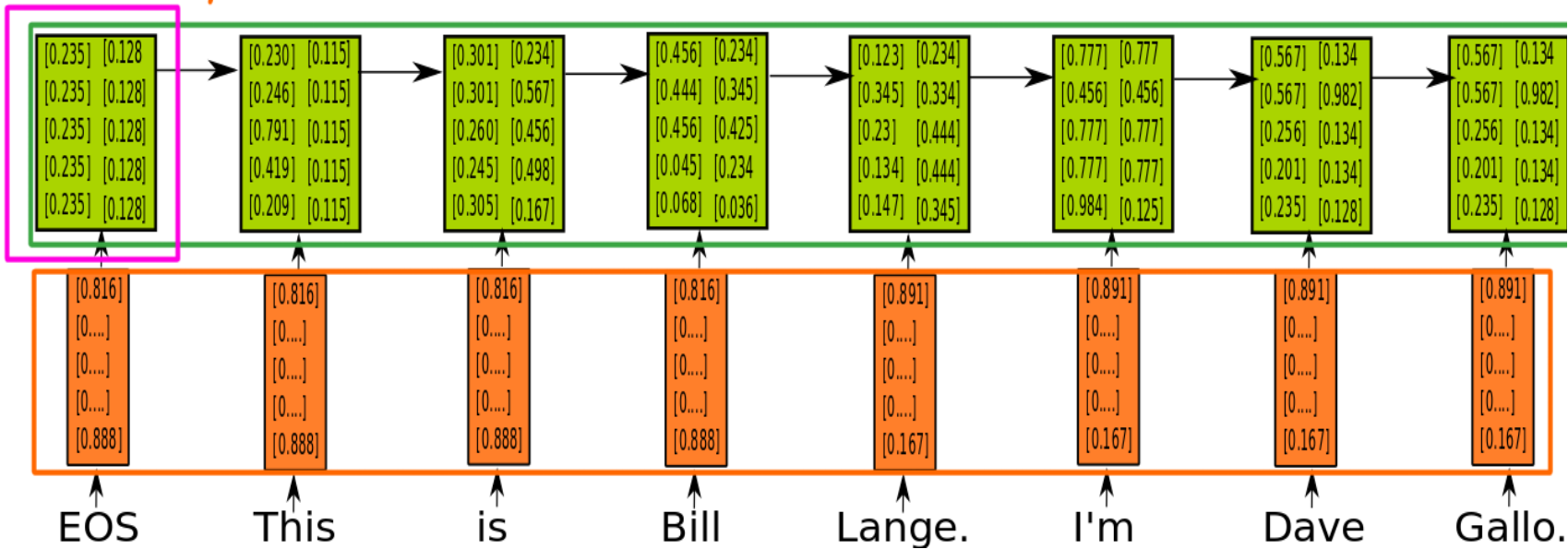


NMT:Detailed:Encoder-Decoder:Decoder:Graph(step-5)

```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initializer(0.1)) as vs:
    helper = TrainingHelper(inputs=decoding_embed_input, sequence_length=en_len, time_major=False)
    dec = BasicDecoder(decoding_cell, helper, encoding_st  op_layer )
    logits, _, _ = dynamic_decode(dec, output_time_major=False, impute_finished=True,
                                  maximum_iterations=max_en_len)
    return logits
```

- Similar to dynamic_rnn, except allows more control over c and h states after every step.
- Last c and h state of the encoder.
- Embeddings as input.





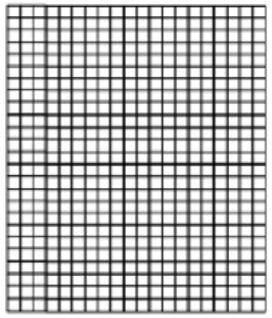
NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-5)

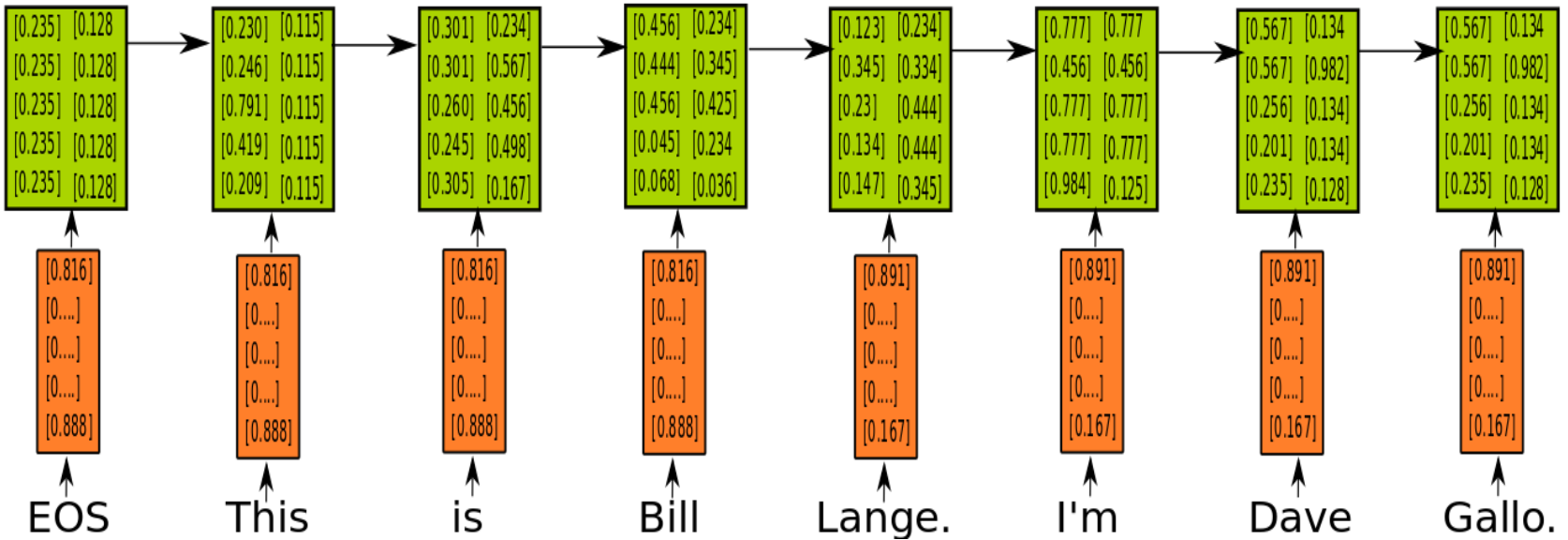
```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initializer(0.1)) as vs:
    helper = TrainingHelper(inputs=decoding_embed_input, sequence_length=en_len, time_major=False)
    dec = BasicDecoder(decoding_cell, helper, encoding_st, op_layer )
    logits, _, _ = dynamic_decode(dec, output_time_major=False, impute_finished=True,
                                  maximum_iterations=max_en_len)
    return logits
```

- Will be more useful with attention.

- Training helper's next input function returns the next input.
- For pre and post processing at every step.

```
def next_inputs(self, time, outputs, state, name=None):

    self.time=time+1
    finished=False
    if(self.time == self.sequence):
        finished=True
    if(not finished):
        next_inputs=self.inputs[0:self.batch,self.time].T
    else:
        next_inputs=None
    return (finished, next_inputs, state)
```



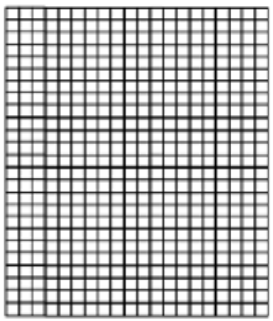


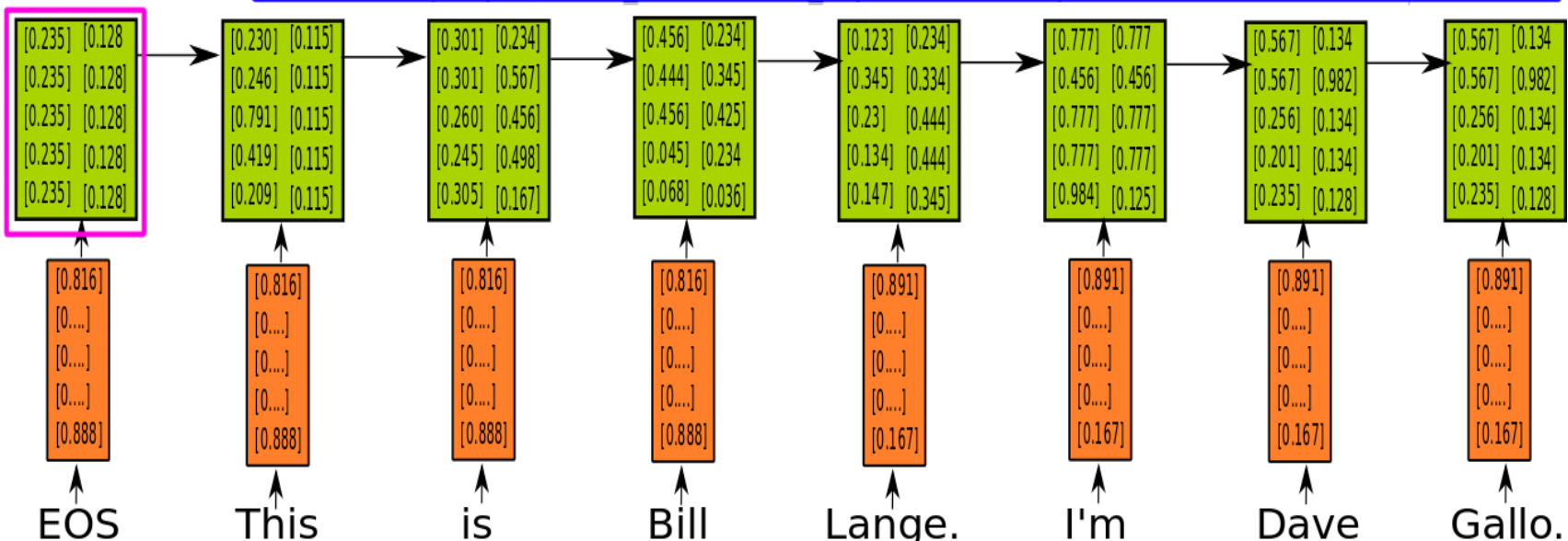
NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-5)

```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initializer(0.1)) as vs:
    helper = TrainingHelper(inputs=decoding_embed_input, sequence_length=en_len, time_major=False)
    dec = BasicDecoder(decoding_cell, helper, encoding_st, op_layer )
    logits, _, _ = dynamic_decode(dec, output_ti
                                  maximum_iterati
    return logits
```

- BasicDecoder's step function
  1. accepts current input and initial state.

```
def step(self, time, inputs, initial_state, name=None):
    #print("Decoder.step.initial_state:",initial_state,self._cell)
    if(initial_state is None):
        output,newstate = self._cell(inputs)
    else:
        output,newstate = self._cell(inputs,initial_state)

    if (self.output_layer is not None):
        output=self.output_layer(output[-1])
    else:
        output=output[-1]
    self.result[time]=output

    finished, next_input, next_state=self._helper.next_inputs(time, output, newstate)
    if self.attention is False:
        next_state=None
    return (output, next_state, next_input, finished)
```



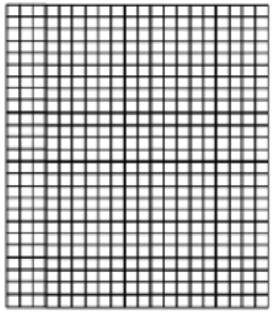


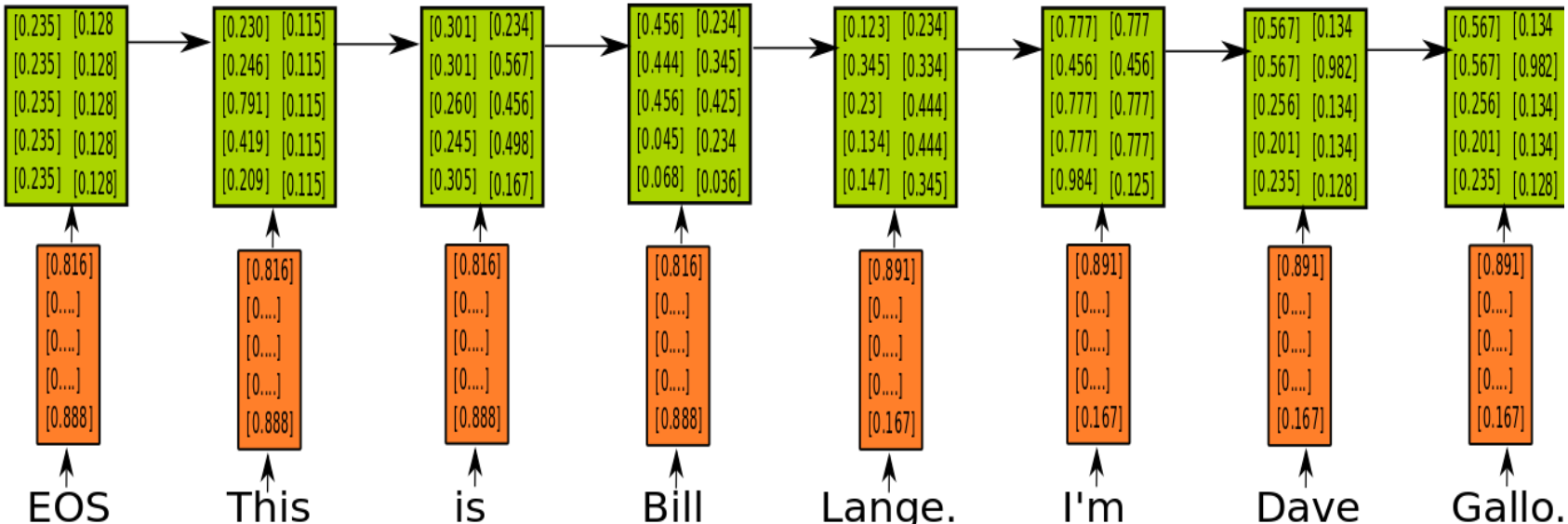
NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-5)

```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initializer(0.1)) as vs:
    helper = TrainingHelper(inputs=decoding_embed_input, sequence_length=en_len, time_major=False)
    dec = BasicDecoder(decoding_cell, helper, encoding_st, op_layer )
    logits, _, _ = dynamic_decode(dec, output_ti
                                  maximum_iterati
    return logits
```

```
def step(self, time, inputs, initial_state, name=None):
    #print("Decoder.step.initial_state:",initial_state,self._cell)
    if(initial_state is None):
        output,newstate = self._cell(inputs)
    else:
        output,newstate = self._cell(inputs,initial_state)

    if (self.output_layer is not None):
        output=self.output_layer(output[-1])
    else:
        output=output[-1]
    self.result[time]=output

    finished, next_input, next_state=self._helper.next_inputs(time, output, newstate)
    if self.attention is False:
        next_state=None
    return (output, next_state, next_input, finished)
```

- BasicDecoder's step function
  2. Transform the RNN output with a FF multiplier, to "Y" shape.



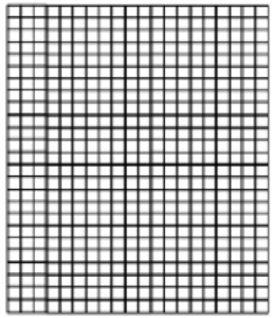


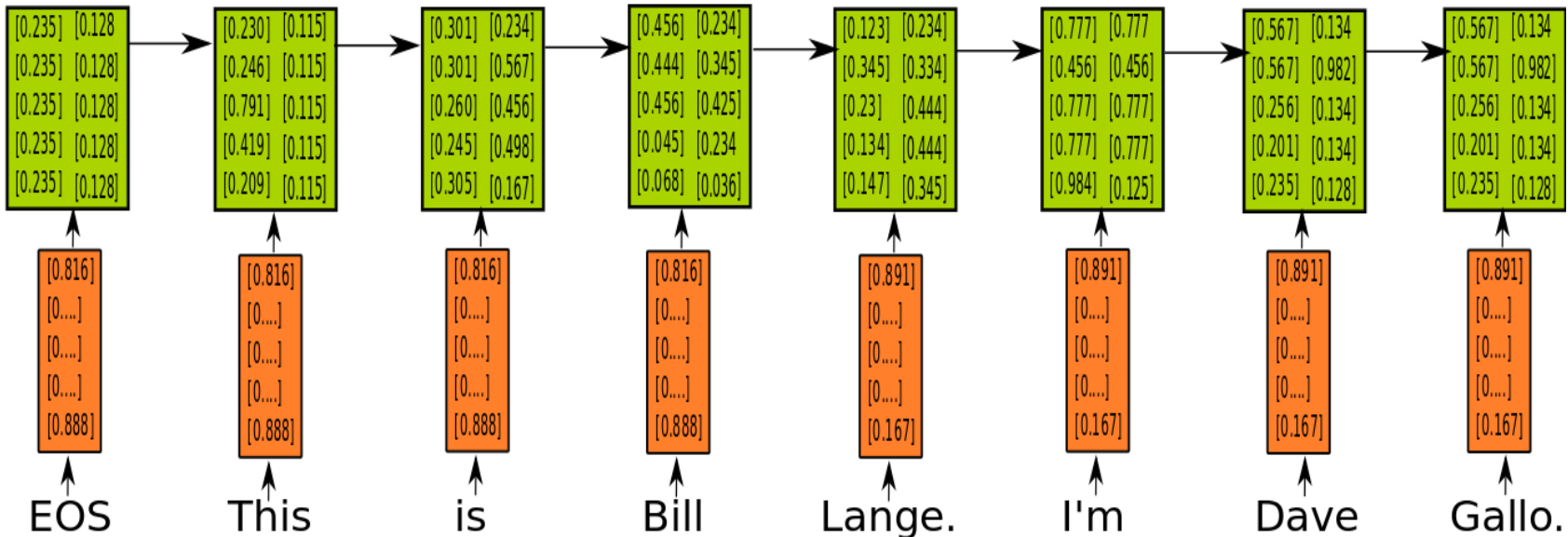
NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-5)

```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initializer(0.1)) as vs:
    helper = TrainingHelper(inputs=decoding_embed_input, sequence_length=en_len, time_major=False)
    dec = BasicDecoder(decoding_cell, helper, encoding_st, op_layer )
    logits, _, _ = dynamic_decode(dec, output_ti
                                  maximum_iterati
    return logits
```

- BasicDecoder's step function
  3. Request for the next input and return.

```
def step(self, time, inputs, initial_state, name=None):
    #print("Decoder.step.initial_state:",initial_state,self._cell)
    if(initial_state is None):
        output,newstate = self._cell(inputs)
    else:
        output,newstate = self._cell(inputs,initial_state)

    if (self.output_layer is not None):
        output=self.output_layer(output[-1])
    else:
        output=output[-1]
    self.result[time]=output

    finished, next_input, next_state=self._helper.next_inputs(time, output, newstate,
    if self.attention is False:
        next_state=None
    return (output, next_state, next_input, finished)
```





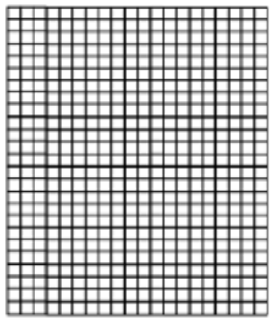
NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-5)

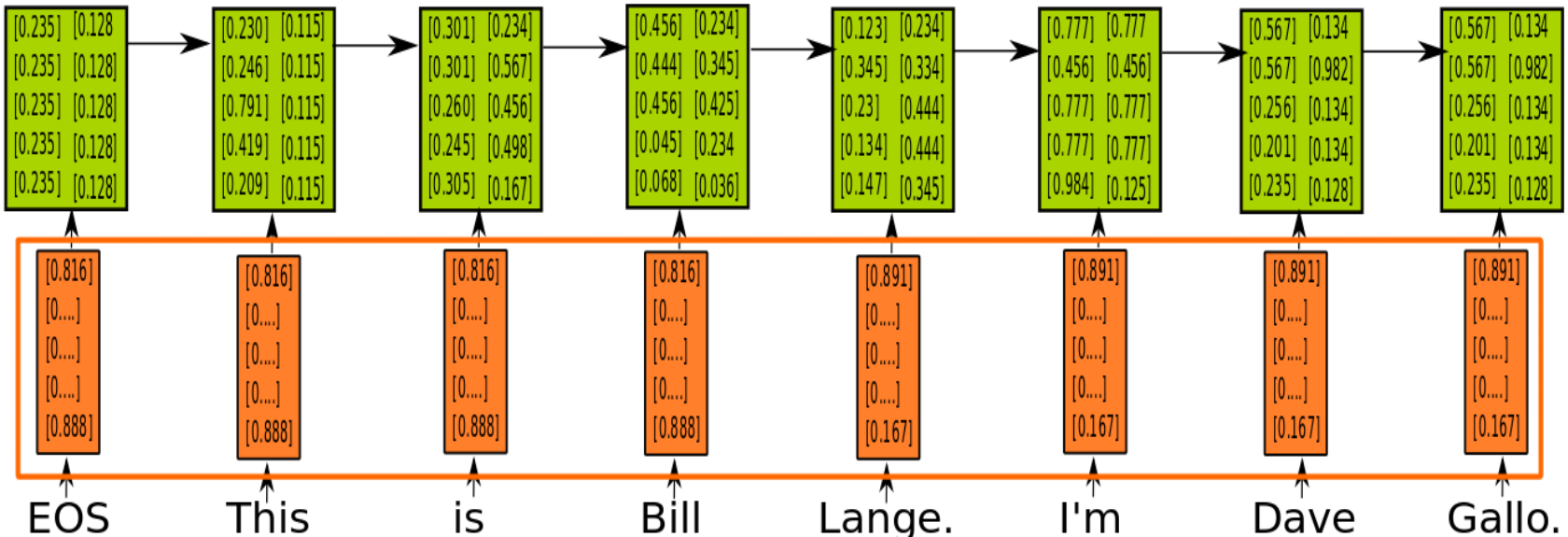
```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initiali
    helper = TrainingHelper(inputs=decoding_embed_input,
    dec = BasicDecoder(decoding_cell, helper, encoding_st
    logits, _, _ = dynamic_decode(dec, output_time_major=
                                  maximum_iterations=max_

    return logits
```

1. Controls the loop
2. internally when the input has finished it terminates the loop.

```
def dynamic_decode(decoder,
                   output_time_major=False,
                   impute_finished=False,
                   maximum_iterations=None,
                   parallel_iterations=32,
                   scope=None):

    finished,next_inputs,initial_state=decoder.initialize()
    time=0

    next_state=initial_state
    while not finished:
        outputs, next_state, next_inputs, finished=decoder.step(time,next_inputs,next_state)
        time=time+1

    decoder.finalize(outputs,next_state)

    return decoder.get_final_results(),next_state
```

3. Internally implemented using tf.while loop to optimize using GPUs.



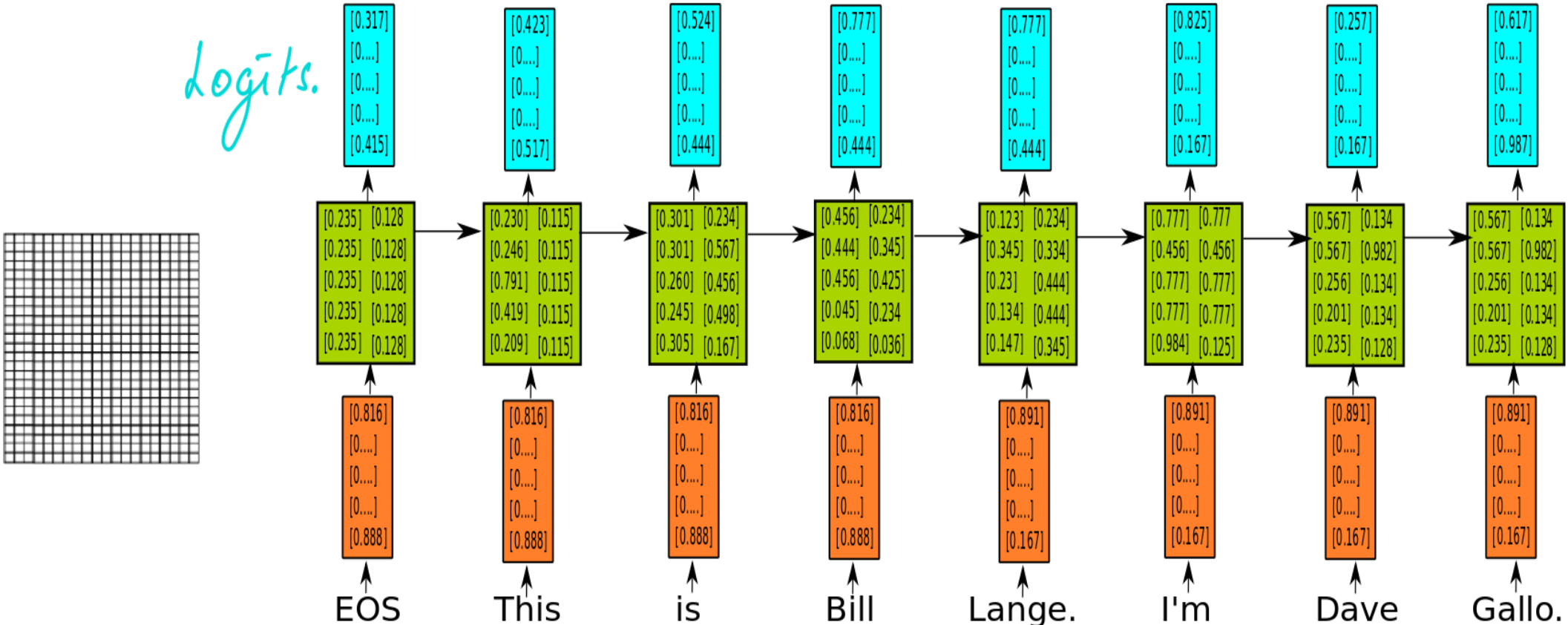


NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-5)

```
with variable_scope.variable_scope(
        "decoder", initializer=init_ops.constant_initializer(0.1)) as vs:
    helper = TrainingHelper(inputs=decoding_embed_input, sequence_length=en_len, time_major=False)
    dec = BasicDecoder(decoding_cell, helper, encoding_st, op_layer )
    logits, _, _ = dynamic_decode(dec, output_time_major=False, impute_finished=True,
                                  maximum_iterations=max_en_len)
    return logits
```

1. Feed forward layer transforms the h-state to "Y" shape.

Logits.



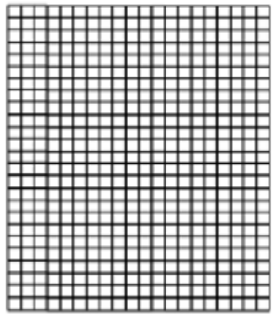
NMT:Detailed:Encoder-Decoder:Decoder:Graph(step-6)

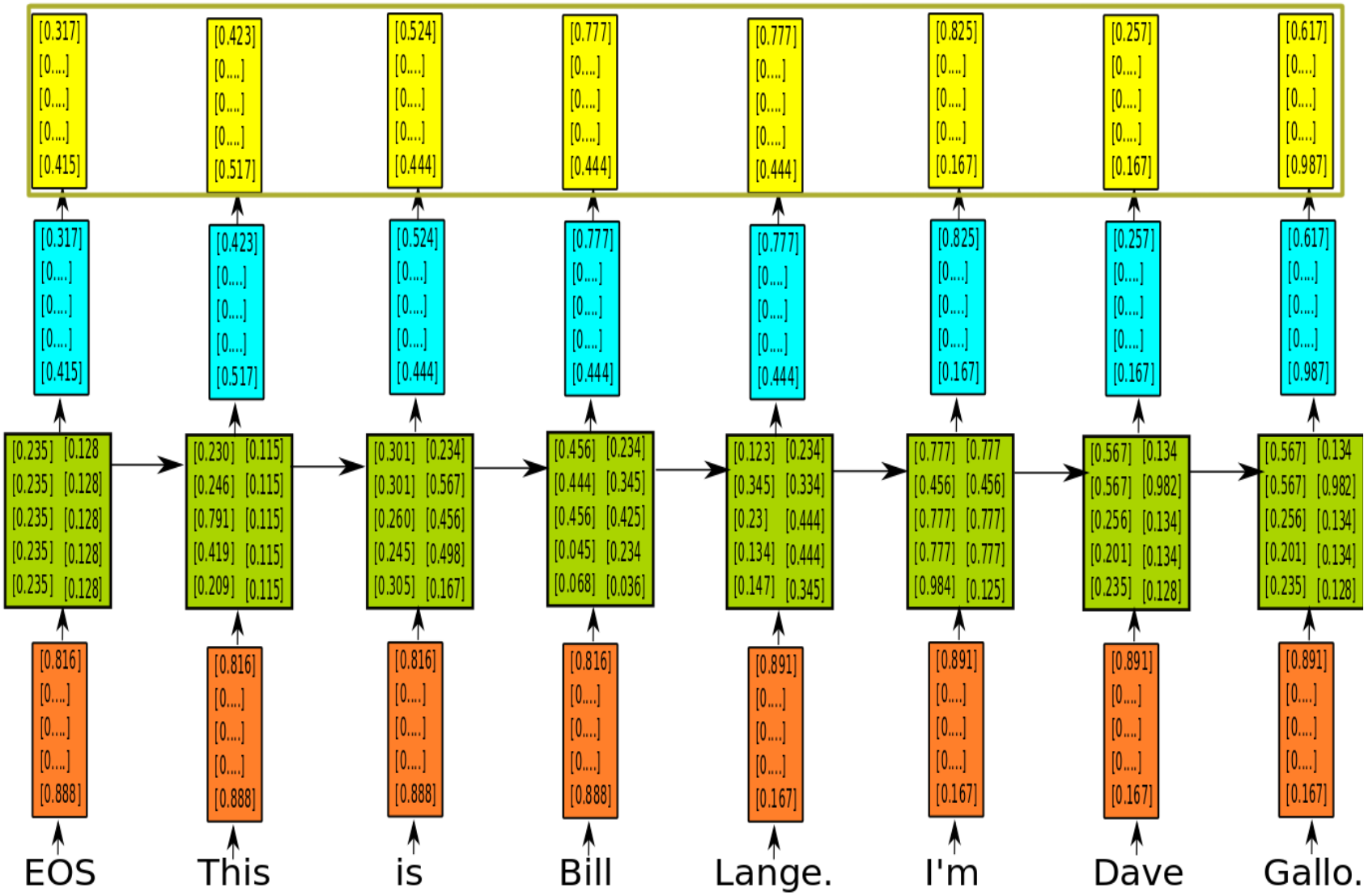
```
with tf.name_scope("optimizer"):
    tr_cost = sequence_loss(logits_tr,targets,seq_masks)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    gradients = optimizer.compute_gradients(tr_cost)
    train_op = optimizer.apply_gradients(gradients)
    tf.summary.scalar("cost", tr_cost)
```

1. Softmax($\hat{Y}$) and Loss(E) calculation are done internally. Infact this internally calls softmax_loss_function.

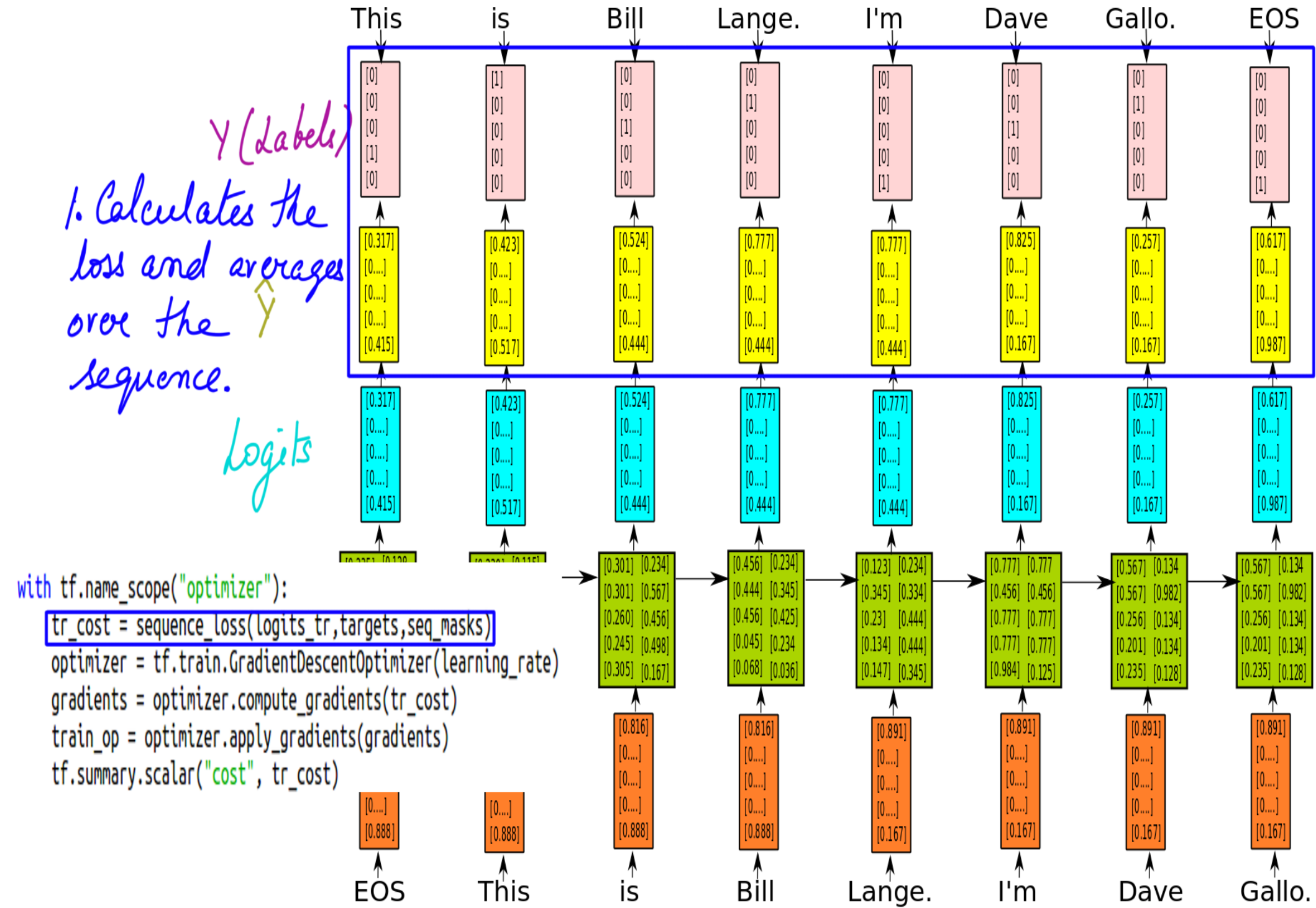
$\hat{Y}$

Logits

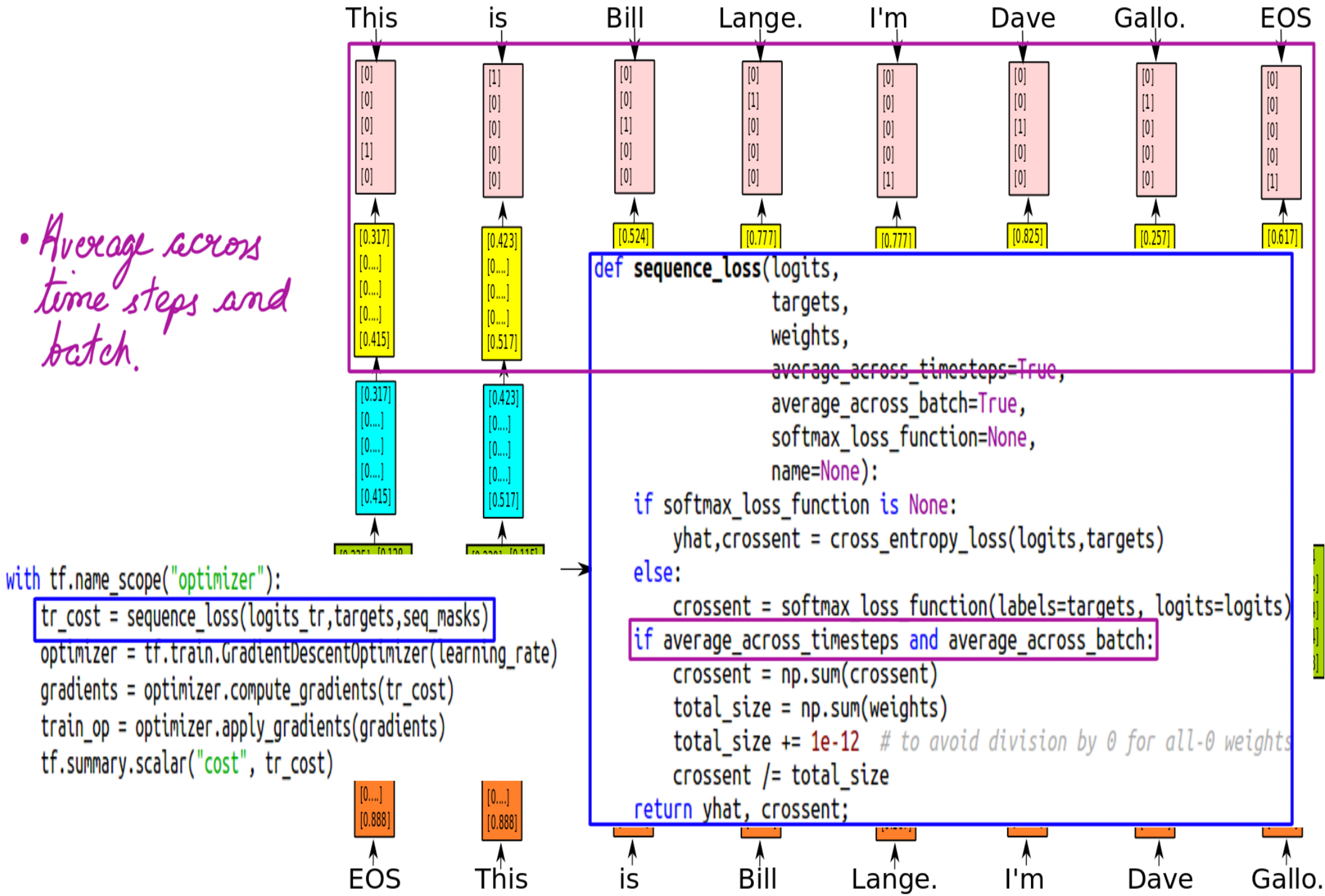




NMT:Detailed:Encoder-Decoder:Decoder:Graph(step-6)



NMT:Detailed(DB):Encoder-Decoder:Decoder:Graph(step-6)



- Average across time steps and batch.

```
with tf.name_scope("optimizer"):
    tr_cost = sequence_loss(logits_tr,targets,seq_masks)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    gradients = optimizer.compute_gradients(tr_cost)
    train_op = optimizer.apply_gradients(gradients)
    tf.summary.scalar("cost", tr_cost)
```

```
def sequence_loss(logits,
                  targets,
                  weights,
                  average_across_timesteps=True,
                  average_across_batch=True,
                  softmax_loss_function=None,
                  name=None):
    if softmax_loss_function is None:
        yhat,crossent = cross_entropy_loss(logits,targets)
    else:
        crossent = softmax_loss_function(labels=targets, logits=logits)
    if average_across_timesteps and average_across_batch:
        crossent = np.sum(crossent)
        total_size = np.sum(weights)
        total_size += 1e-12  # to avoid division by 0 for all-0 weights
        crossent /= total_size
    return yhat, crossent;
```

NMT:Detailed:Encoder-Decoder:Decoder:Graph(step-7)

```
fr_embeddings_matrix,en_embeddings_matrix,fr_word2int,en_word2int,fr_filtered,en_filtered,args=get_nmt_data()
set_modelparams(args)
train_graph = tf.Graph()
with train_graph.as_default():
    input_data, targets, learning_rate, dropout_probs, en_len, max_en_len, fr_len = model_inputs()
    encoding_embed_input, encoding_op, encoding_st, rnn_inputs, decoding_input,decoding_embed_input, logits_tr = seq2seq_model(
        tf.reverse(input_data, [-1]),
        targets,
        dropout_probs,
        fr_len,
        en_len,
        max_en_len,
        len(en_word2int) + 1,
        hidden_size,
        n_layers,
        en_word2int,
        batch_size,
        encoder_type,fr_embeddings_matrix,en_embeddings_matrix)
```

- The graph defined
- The input when served reverse increases the accuracy.