Large-Scale, Distributed Machine Learning

CSE545 - Spring 2020

Stony Brook University

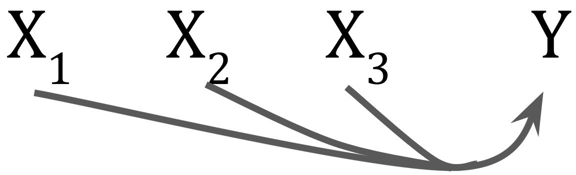
H. Andrew Schwartz

Supervised Learning

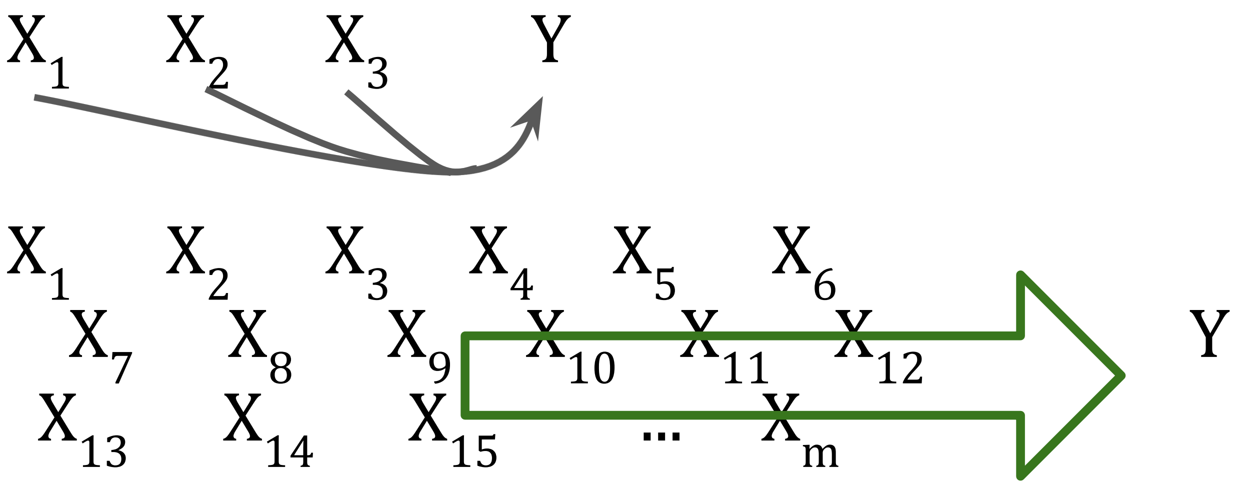
(genes) (health)

X_1 X_2 X_3 Y

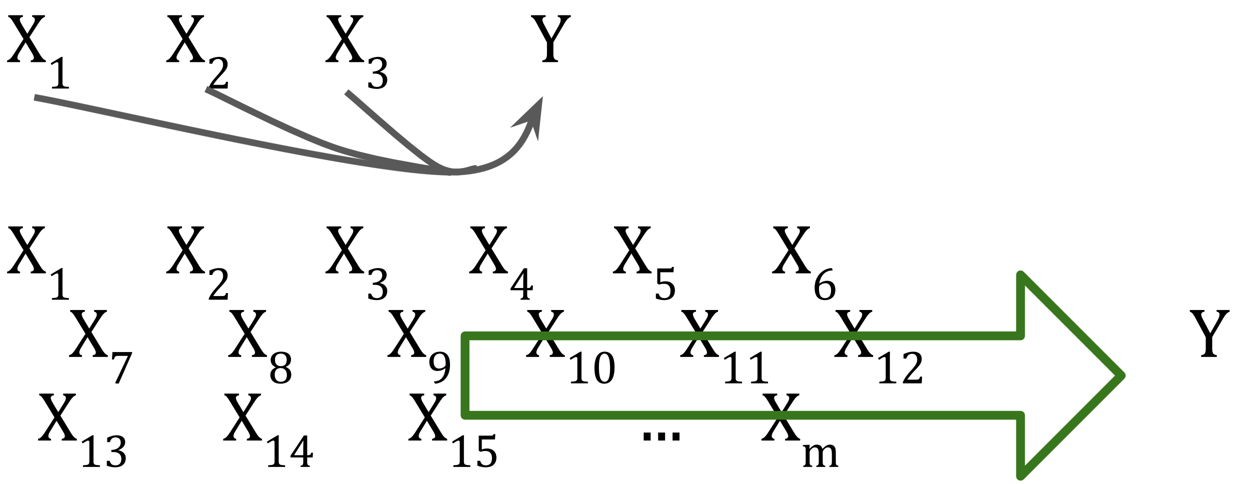
Supervised Learning



Supervised Learning

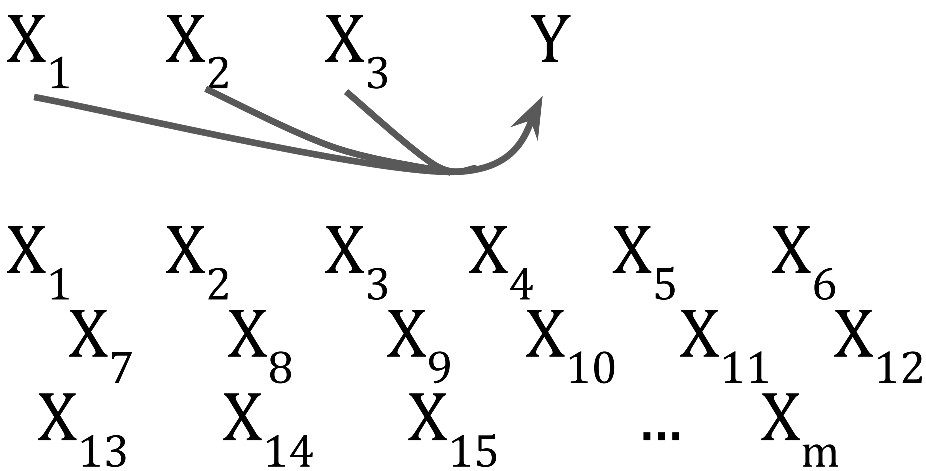


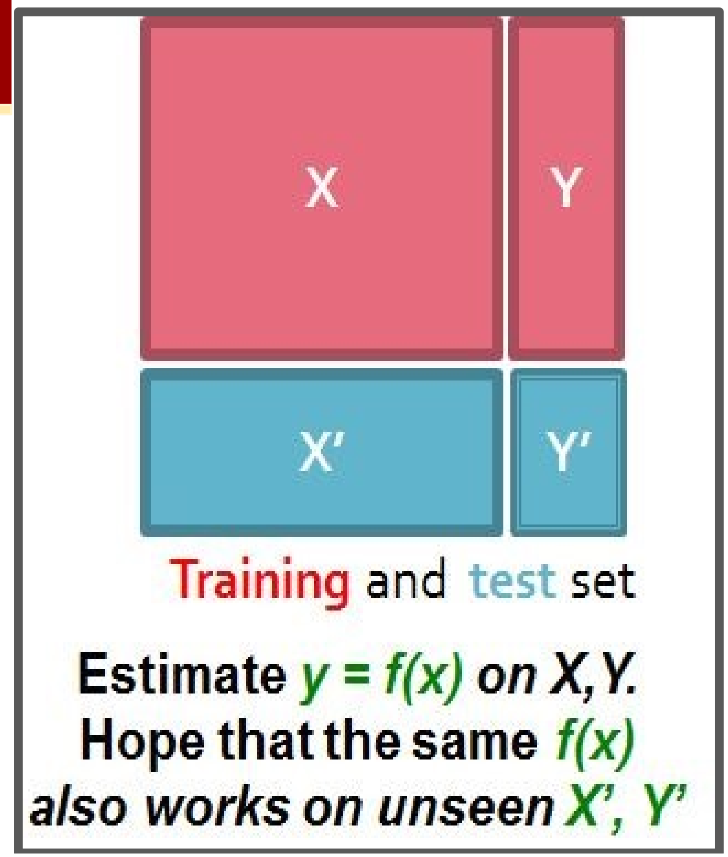
Supervised Learning



Task: Determine a function, f (or parameters to a function) such that $f(X) = Y$

Supervised Learning

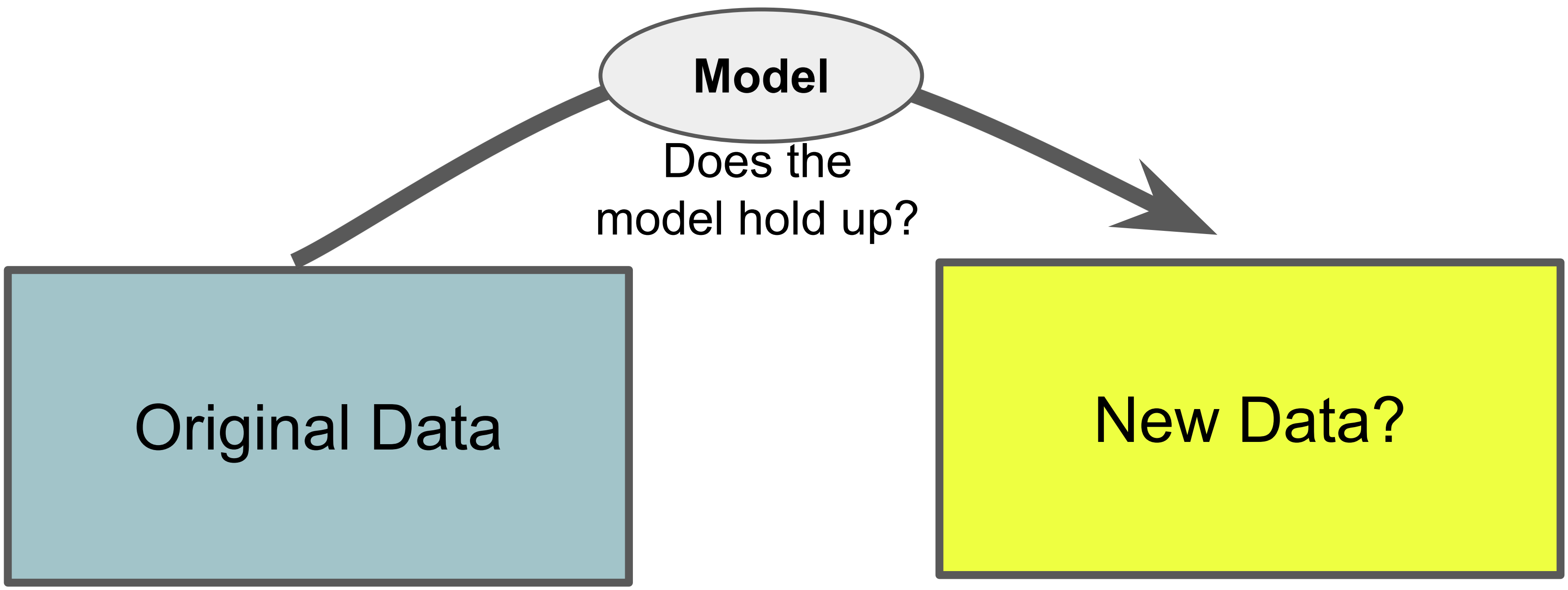




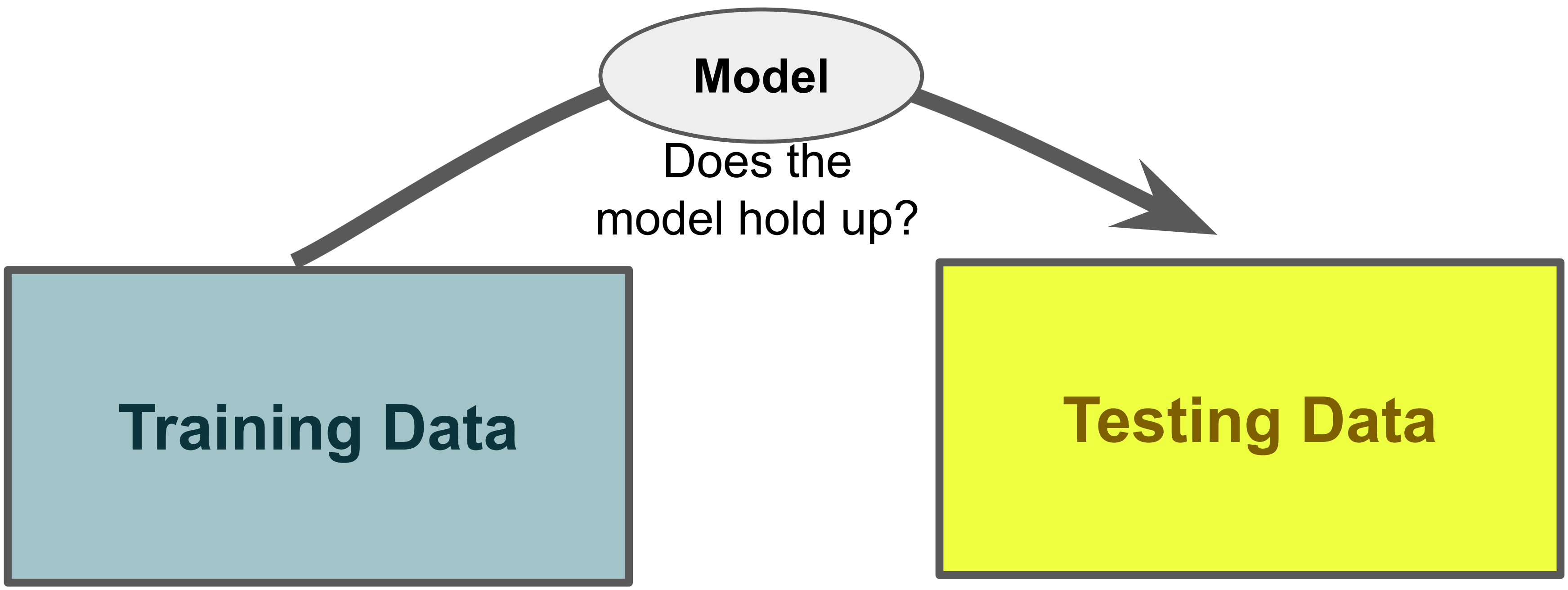
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Task: Determine a function, f (or parameters to a function) such that $f(X) = Y$

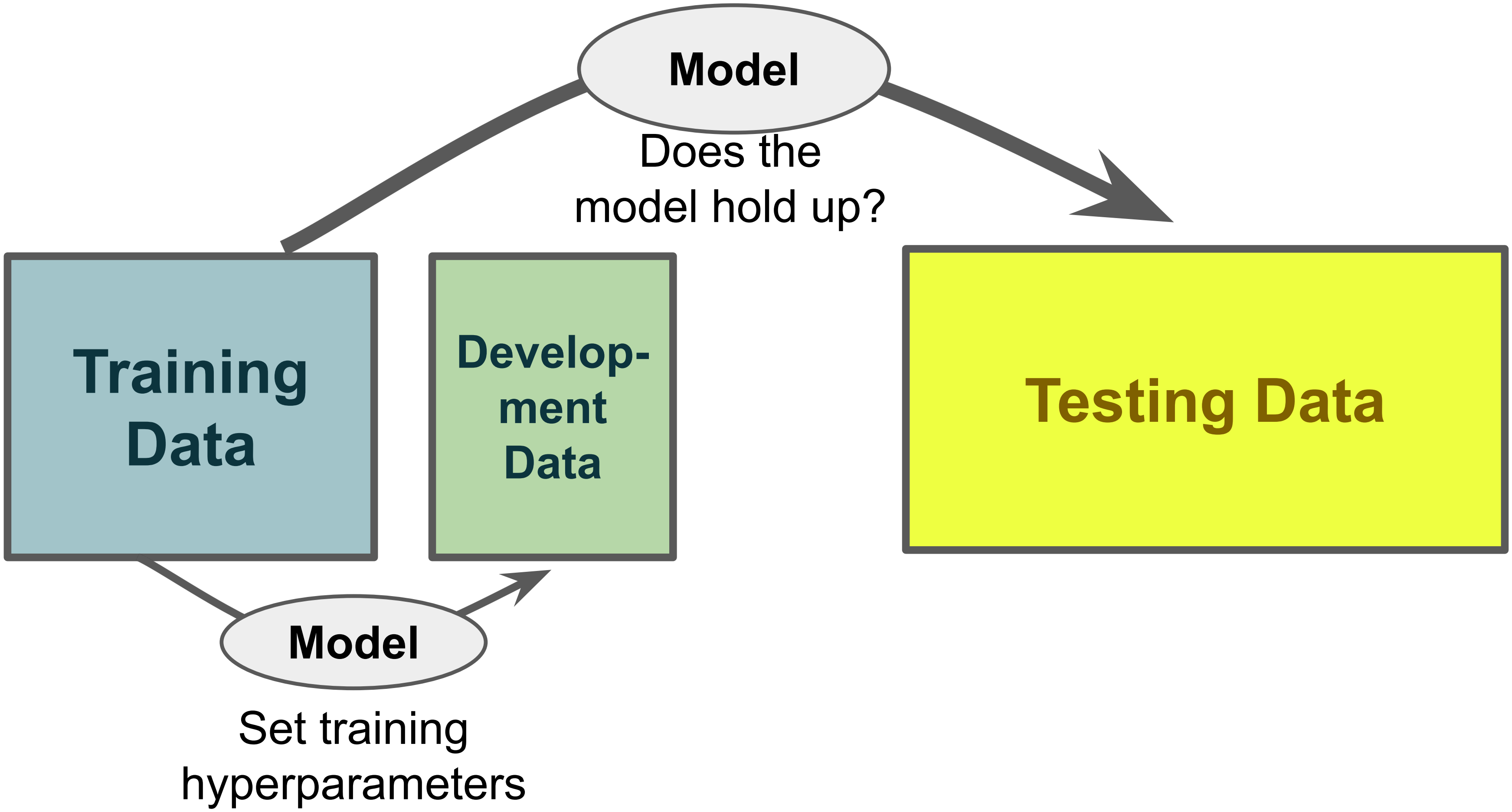
Common Goal: Generalize to new data



Common Goal: Generalize to new data

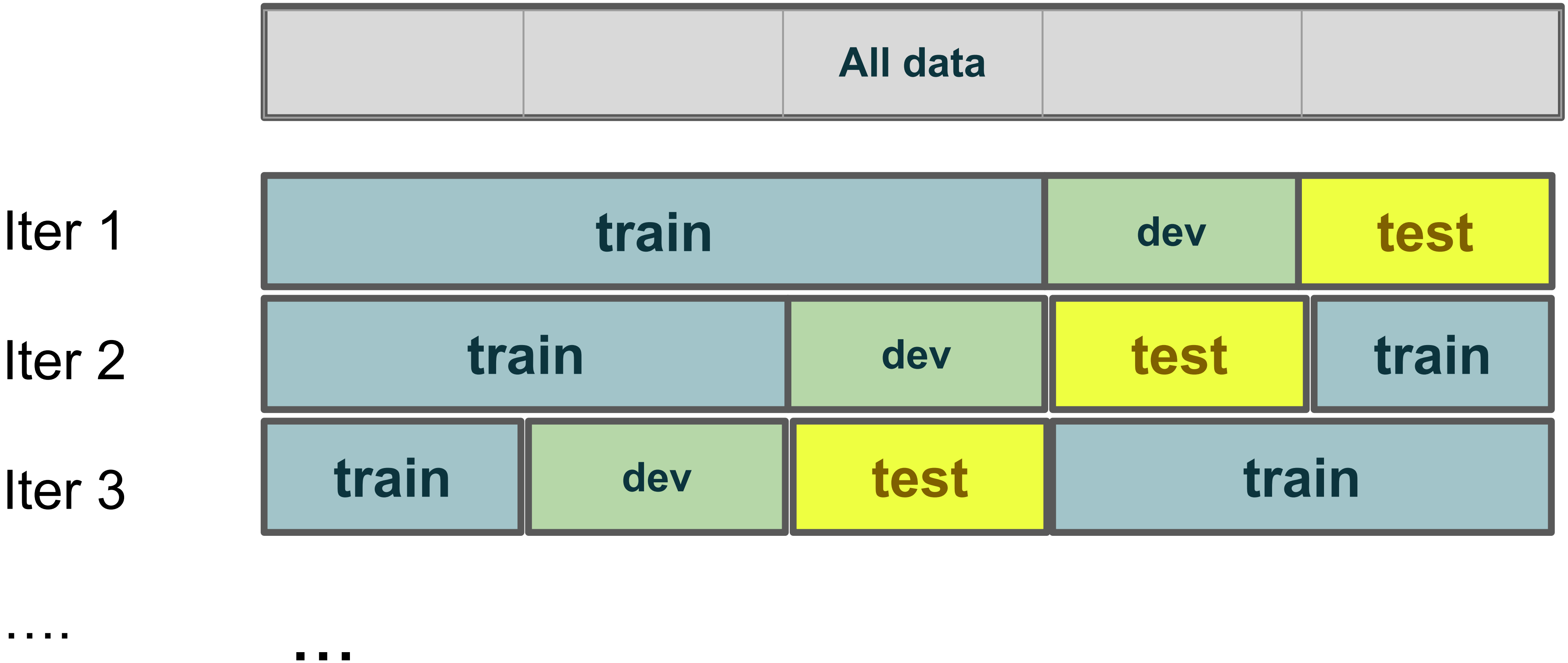


ML: GOAL


Model
Does the
model hold up?
Training
Data
Develop-
ment
Data
Testing Data
Model
Set training
hyperparameters

N-Fold Cross Validation

Goal: Decent estimate of model accuracy



Review: Distributed ML

1. **Distribute copies of entire dataset**
   a. Train over all with different parameters
   b. Train different folds per worker node.

Done very often in practice. Not talked about much because it's mostly as easy as it sounds.

Pro: Easy; Good for compute-bound; Con: Requires data fit in worker memories

2. **Distribute data**
   a. Each node finds parameters for subset of data
   b. Needs mechanism for updating parameters
      i. Centralized parameter server
      ii. Distributed All-Reduce

Data Parellelism

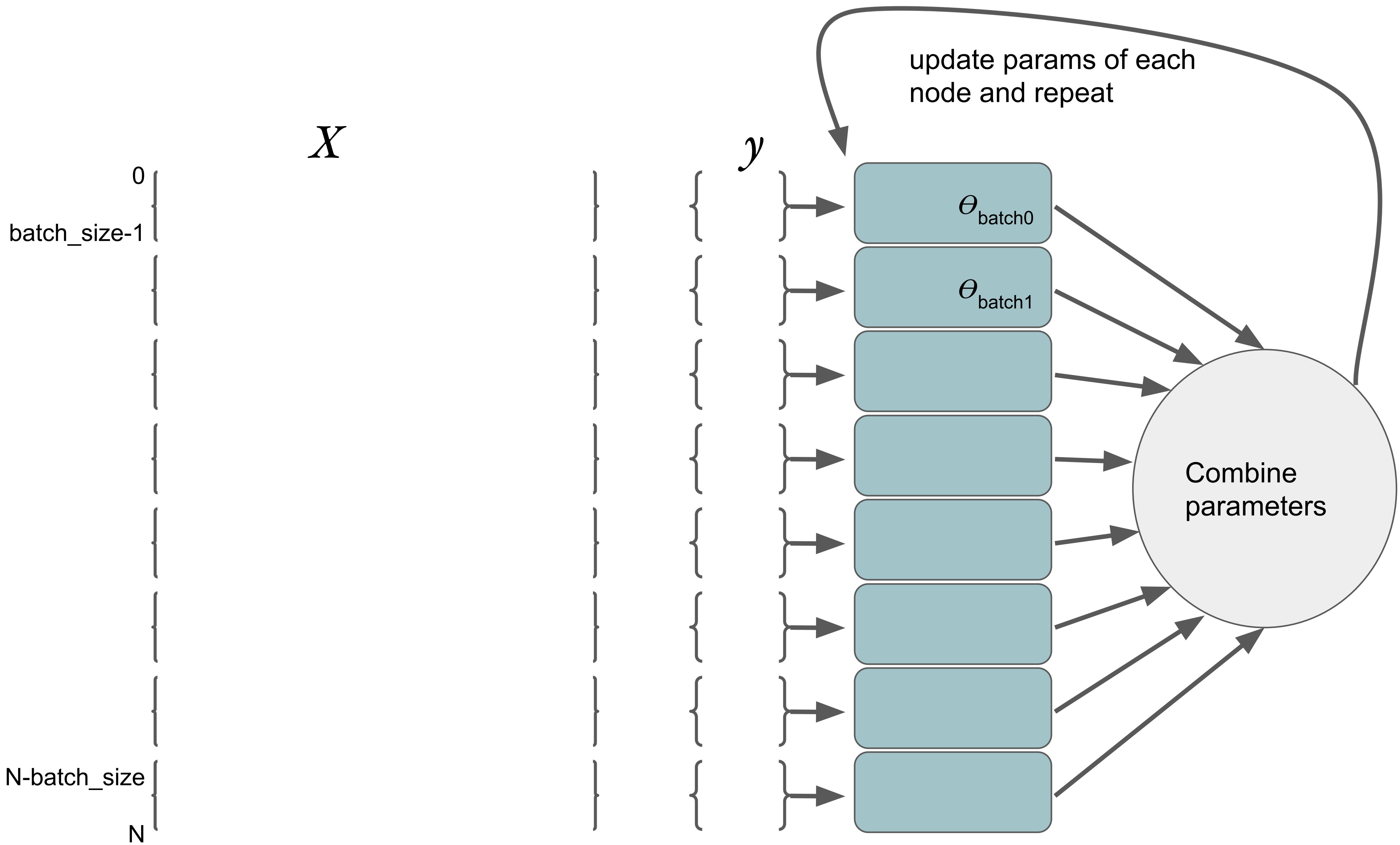
Preferred method for big data or very complex models (i.e. models with many internal parameters).

Pro: Flexible to all situations; Con: Optimizing for subset is suboptimal

3. **Distribute model or individual operations** (e.g. matrix multiply)

Model Parellelism

Pro: Parameters can be localized Con: High communication for transferring Intermediar data.


update params of each node and repeat
X
y
0
batch_size-1
N-batch_size
N
θ_batch0
θ_batch1
Combine parameters

1. Linear modeling
   (linear and logistic regression)

2. **Recurrent Neural Networks**
   Where X is a sequence of data

3. Convolutional Neural Networks
   Where X might have spatial relationships

From Linear Models to Neural Nets

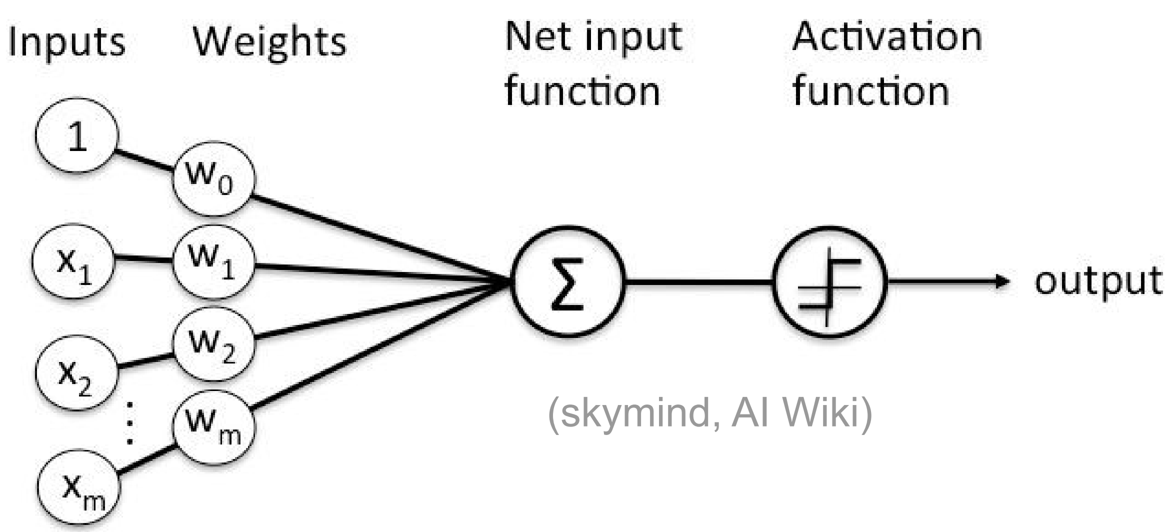
Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$

From Linear Models to Neural Nets

Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$

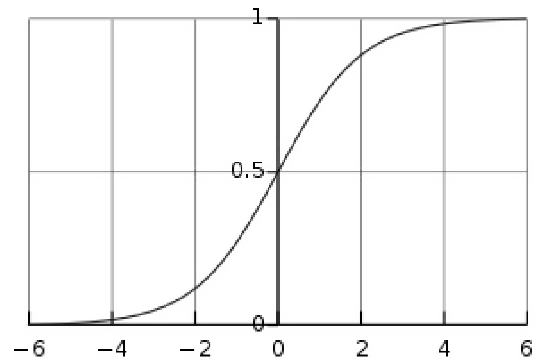


(skymind, AI Wiki)

Common Activation Functions

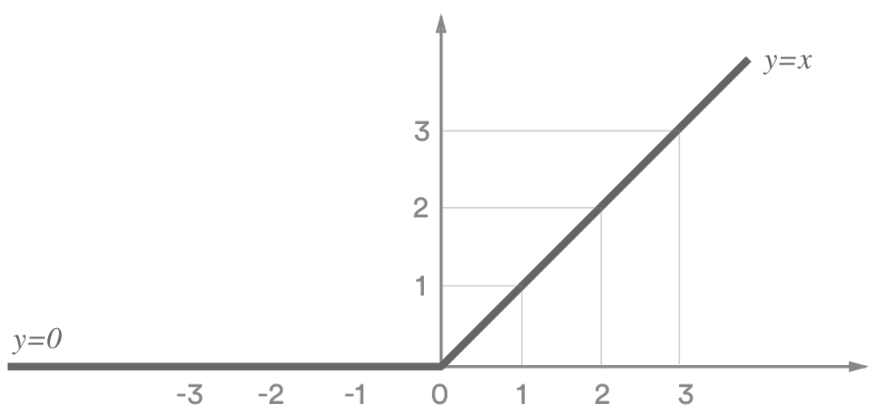
$z = wX$

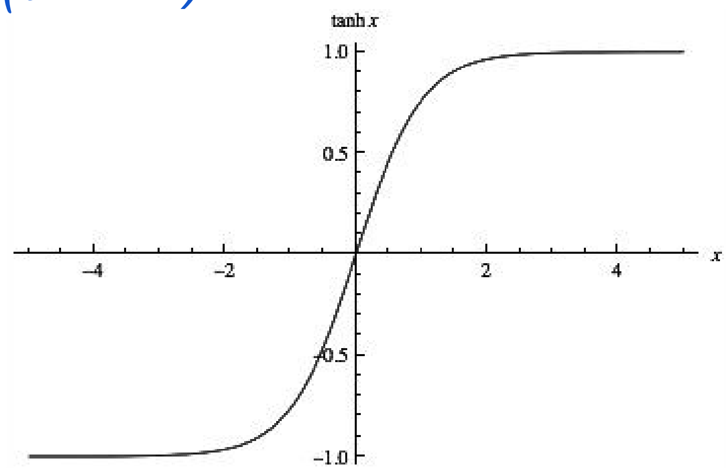
Logistic: $\sigma(z) = 1 / (1 + e^{-z})$



Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$

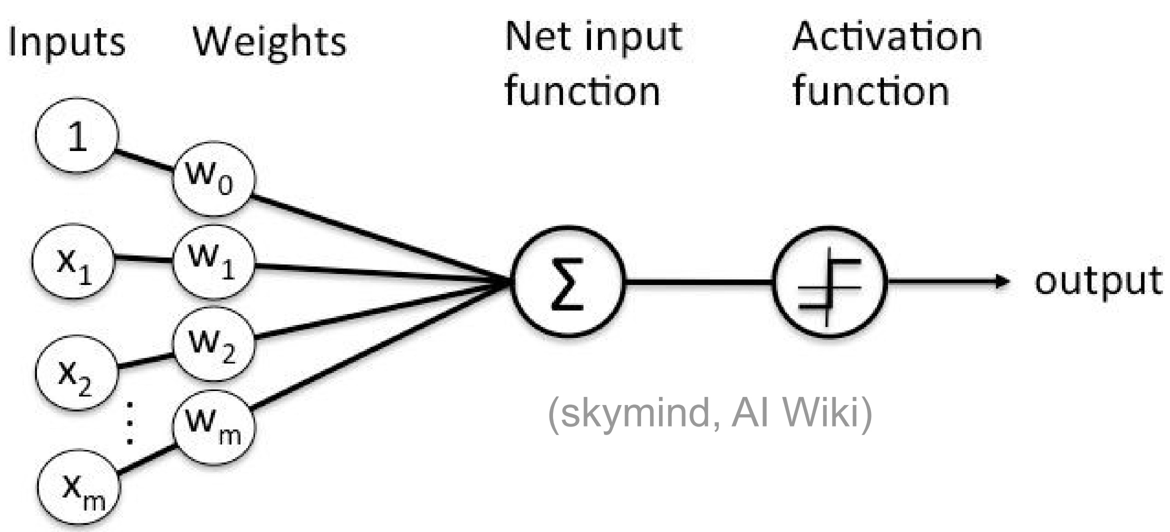




From Linear Models to Neural Nets

Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$

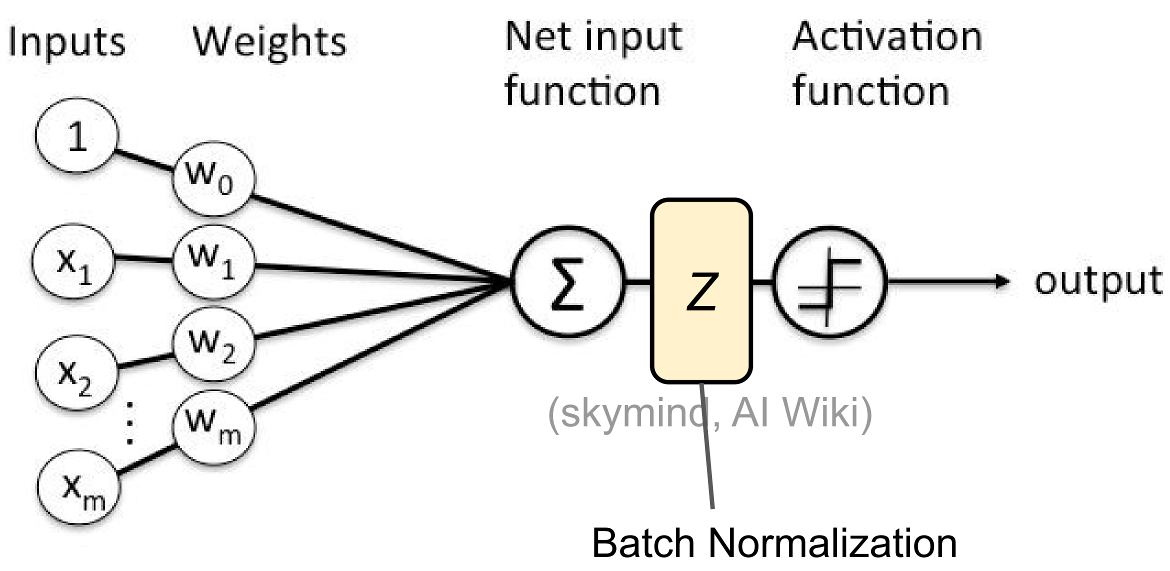


(skymind, AI Wiki)

From Linear Models to Neural Nets

Linear Regression: $y = wX$

Neural Network Nodes: $output = f(wX)$



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

Batch Normalization

This is just standardizing!
(but within the current batch of observations)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

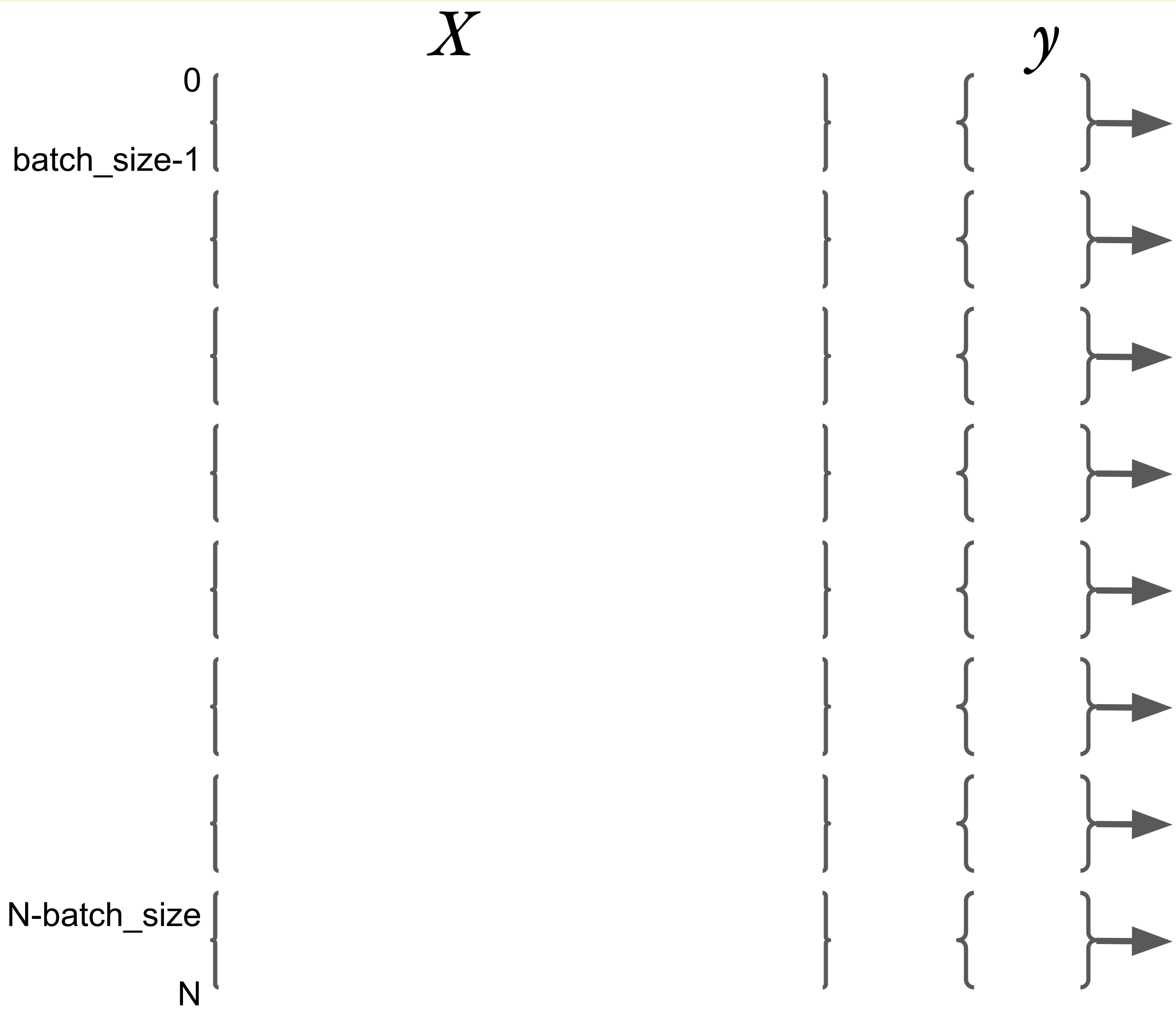
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

Batch Normalization



Batch Normalization

This is just standardizing!
(but within the current batch of observations)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

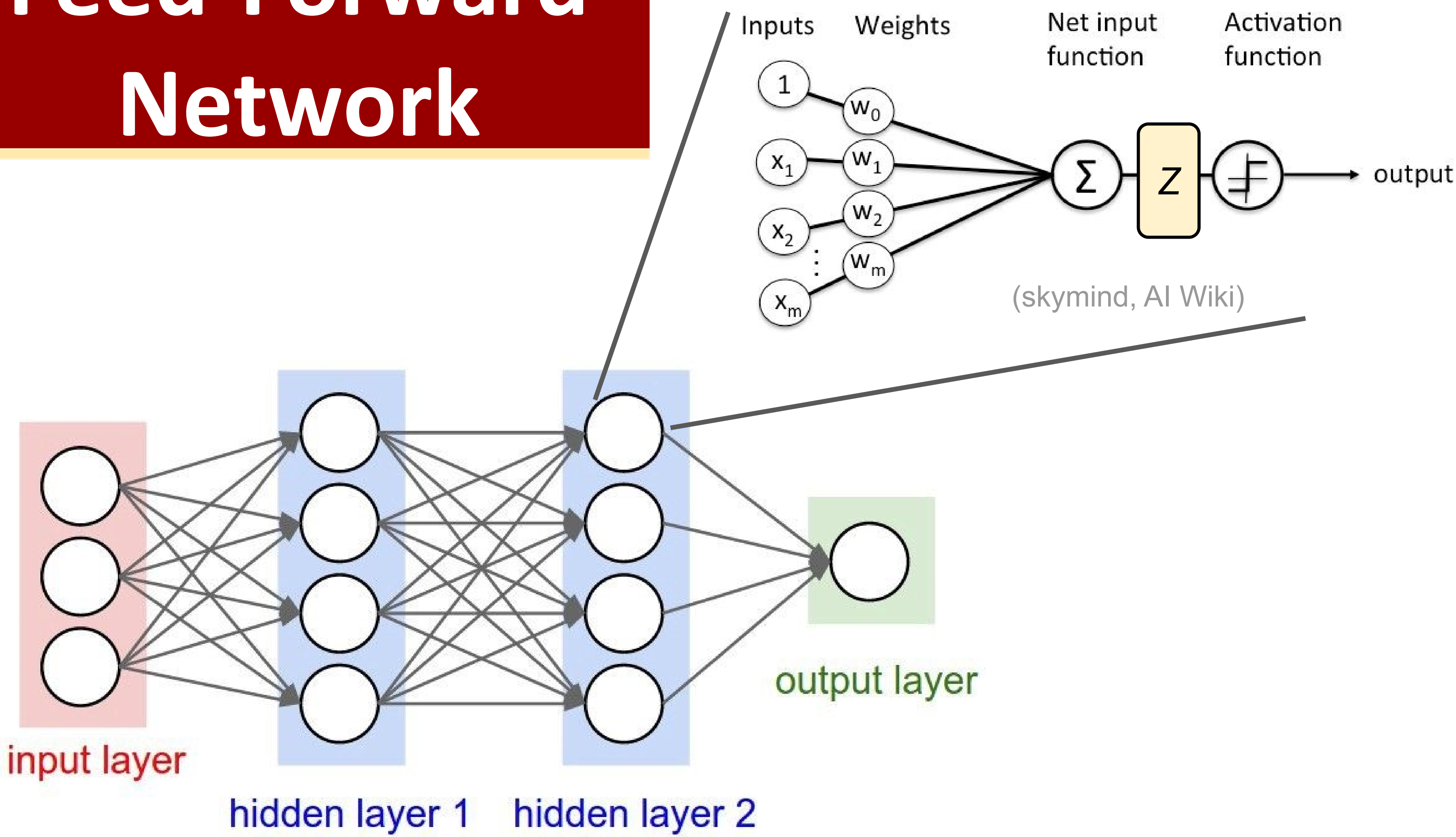
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

(Ioffe and Szegedy, 2015)

Why?

- Empirically, it works!
- Conceptually, generally good for weight optimization to keep data within a reasonable range (dividing by sigma) and such that positive weights move it up and negative down (centering).
- Small effect: When done over mini-batches, adds regularization due to differences between batches.

Feed-Forward Network



Recurrent Neural Network

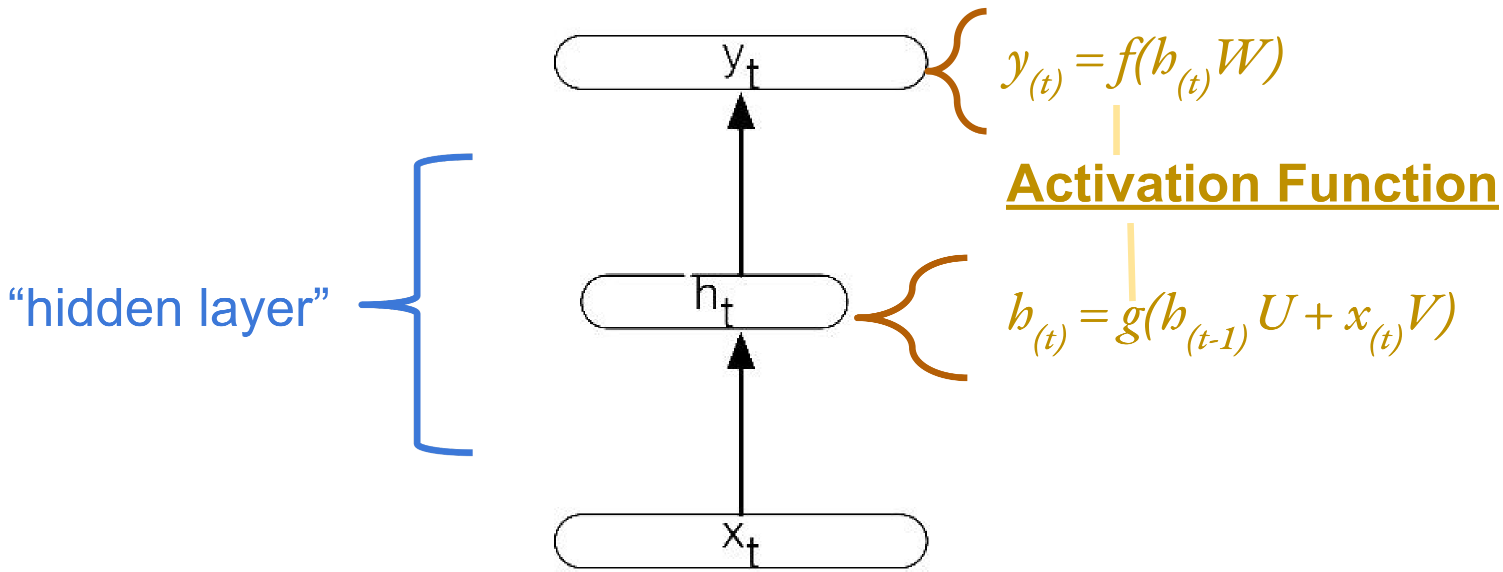
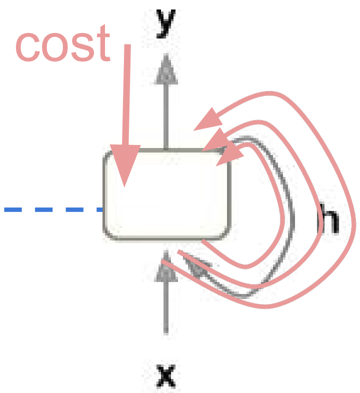


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

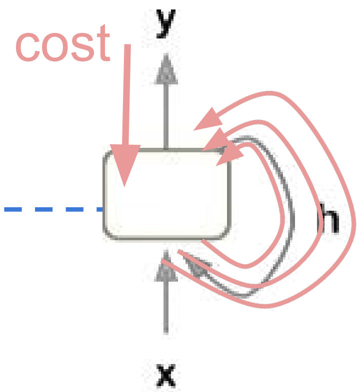
RNN: Optimization



Backward Propagation through Time

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden state
    y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
...
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

RNN: Optimization



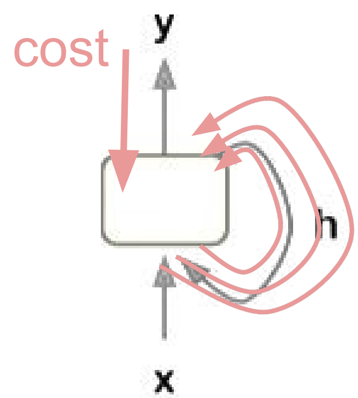
Backward Propagation through Time

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,
state
    y(i) = tf.softmax(tf.matmul
...
cost = tf.reduce_mean(-tf.redu
```

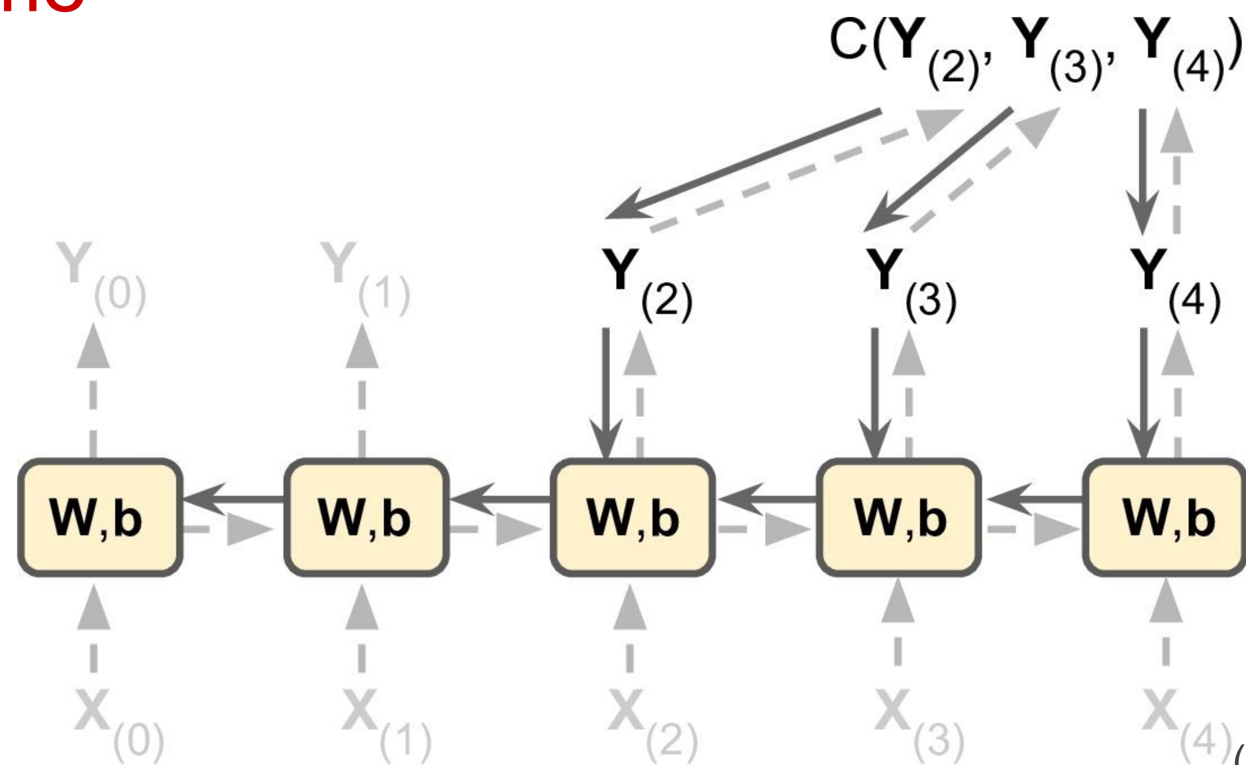
To find the gradient for the overall graph, we use **back propogation,** which *essentially* chains together the gradients for each node (function) in the graph.

With many recursions, the gradients can vanish or explode (become too large or small for floating point operations).

RNN: Optimization



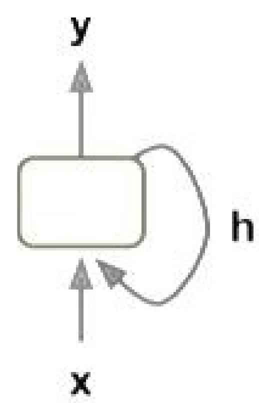
Backward Propagation through Time



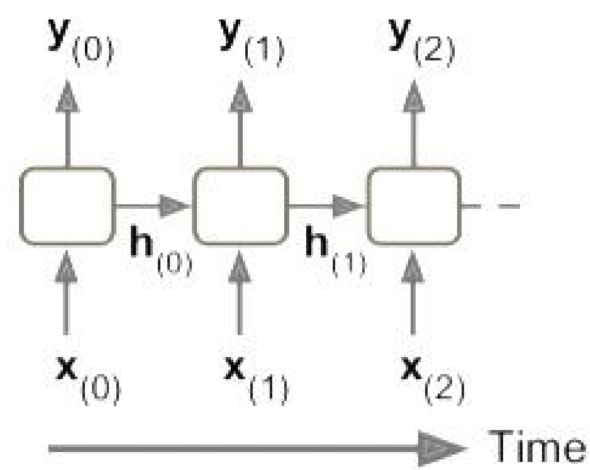
(Geron, 2017)

How to Addressing Vanishing Gradient?

Dominant approach: Use Long Short Term Memory Networks (LSTM)



RNN model

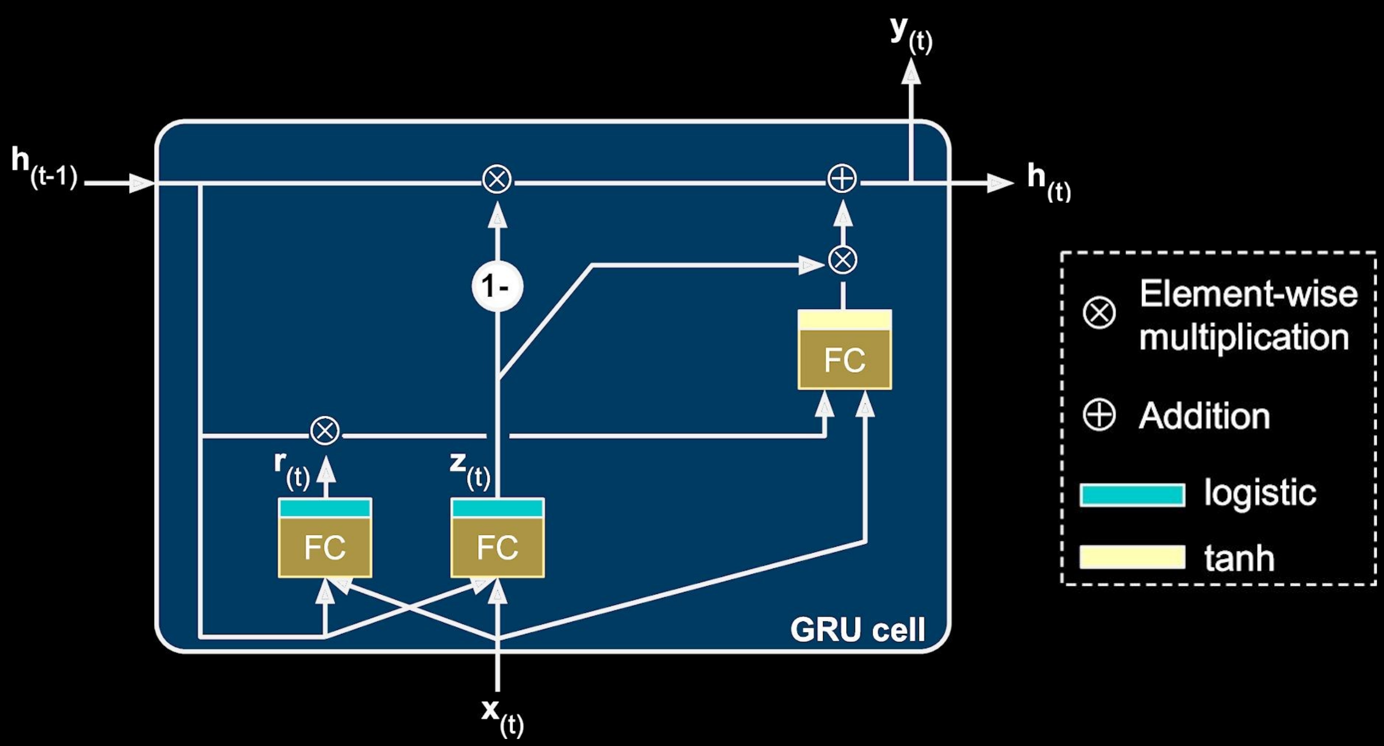


"unrolled" depiction

(Geron, 2017)

RNN: The GRU

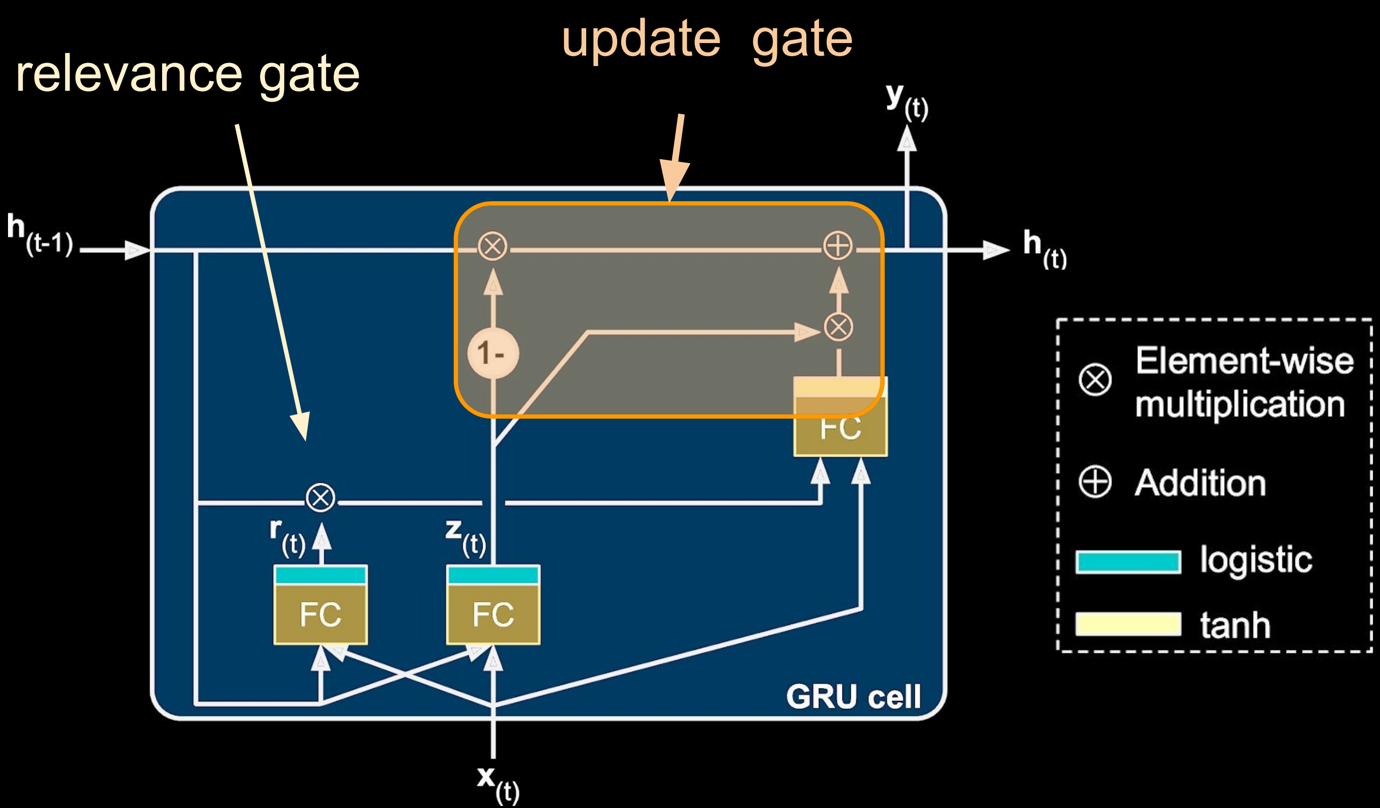
Gated Recurrent Unit



(Geron, 2017)

RNN: The GRU

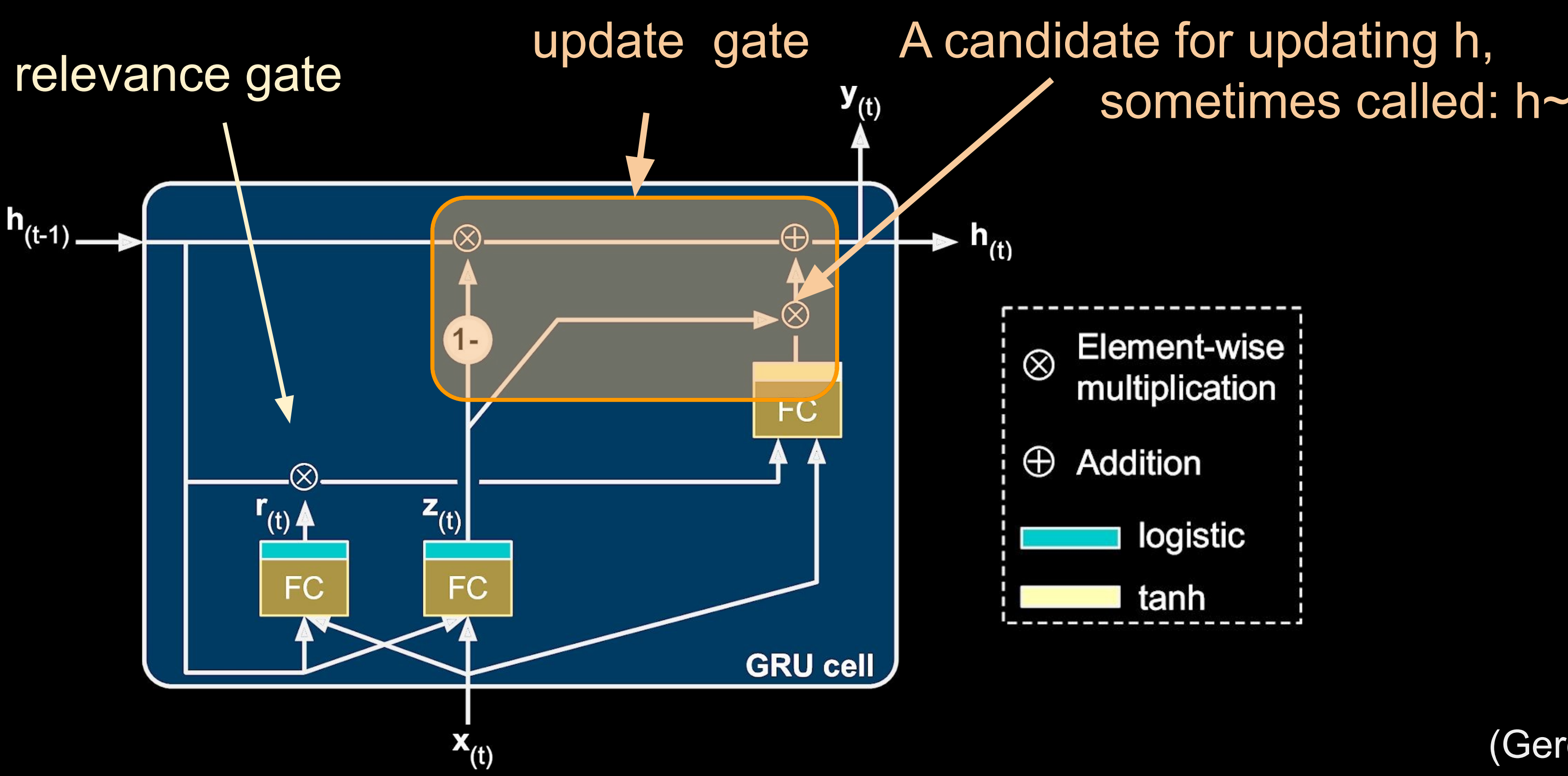
Gated Recurrent Unit



(Geron, 2017)

RNN: The GRU

Gated Recurrent Unit

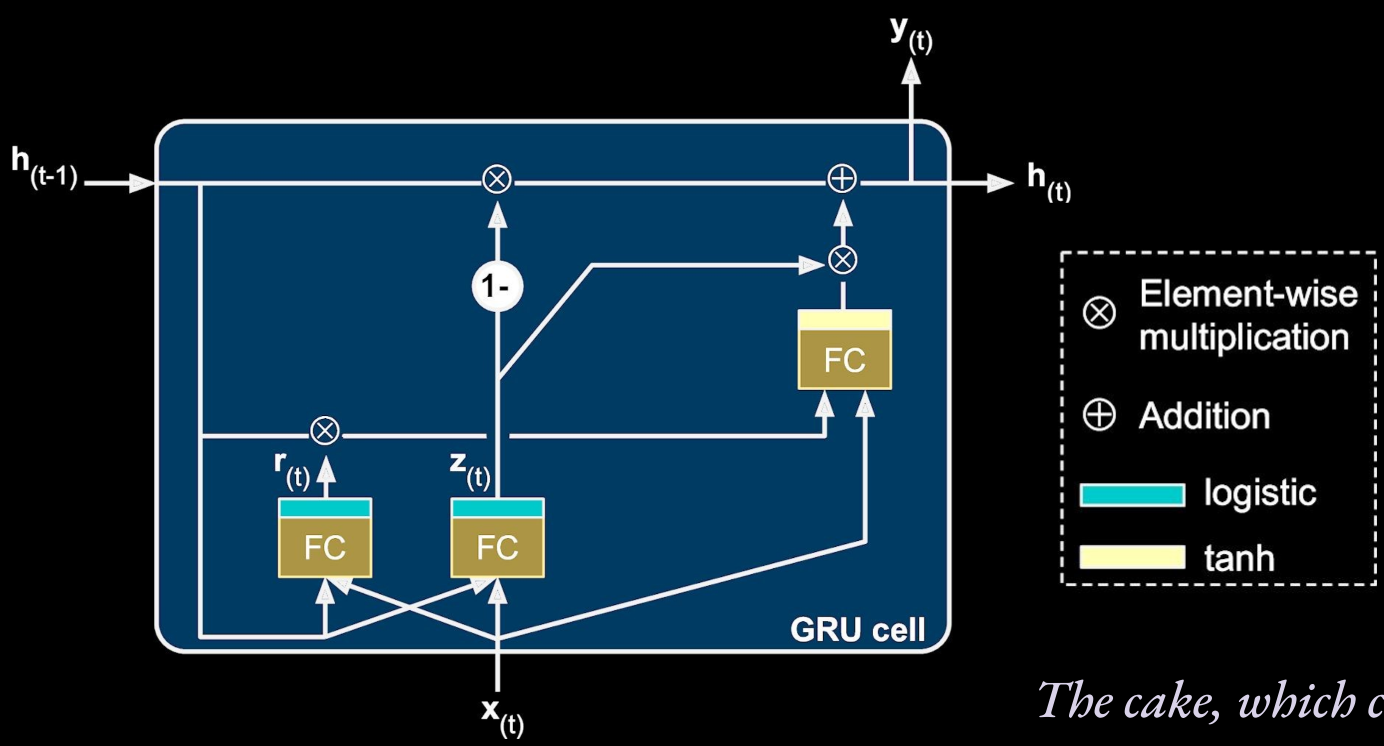


(Geron, 2017)

RNN: The GRU

Gated Recurrent Unit

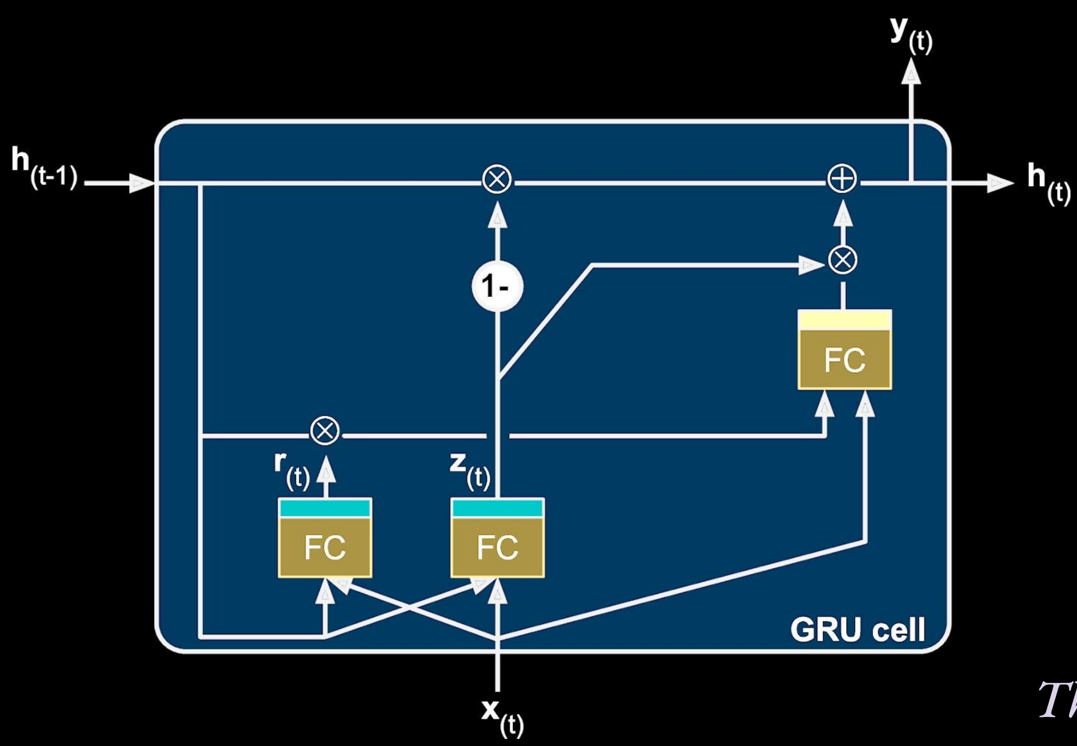
$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The cake, which contained candles, was eaten.

What about the gradient?

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



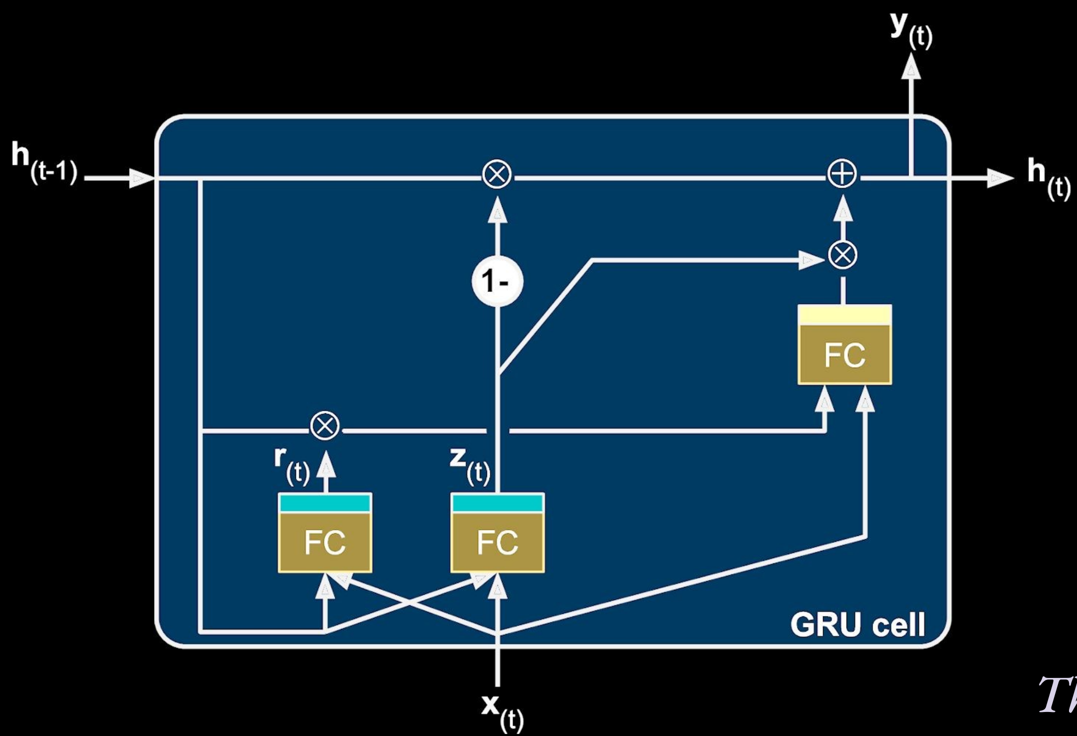
The gates (i.e. multiplications based on a logistic) often end up keeping the hidden state exactly (or nearly exactly) as it was. Thus, for most dimensions of h,

$$h_{(t)} \approx h_{(t-1)}$$

The cake, which contained candles, was eaten.

What about the gradient?

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh\,(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The gates (i.e. multiplications based on a logistic) often end up keeping the hidden state exactly (or nearly exactly) as it was. Thus, for most dimensions of h,

$$h_{(t)} \approx h_{(t-1)}$$

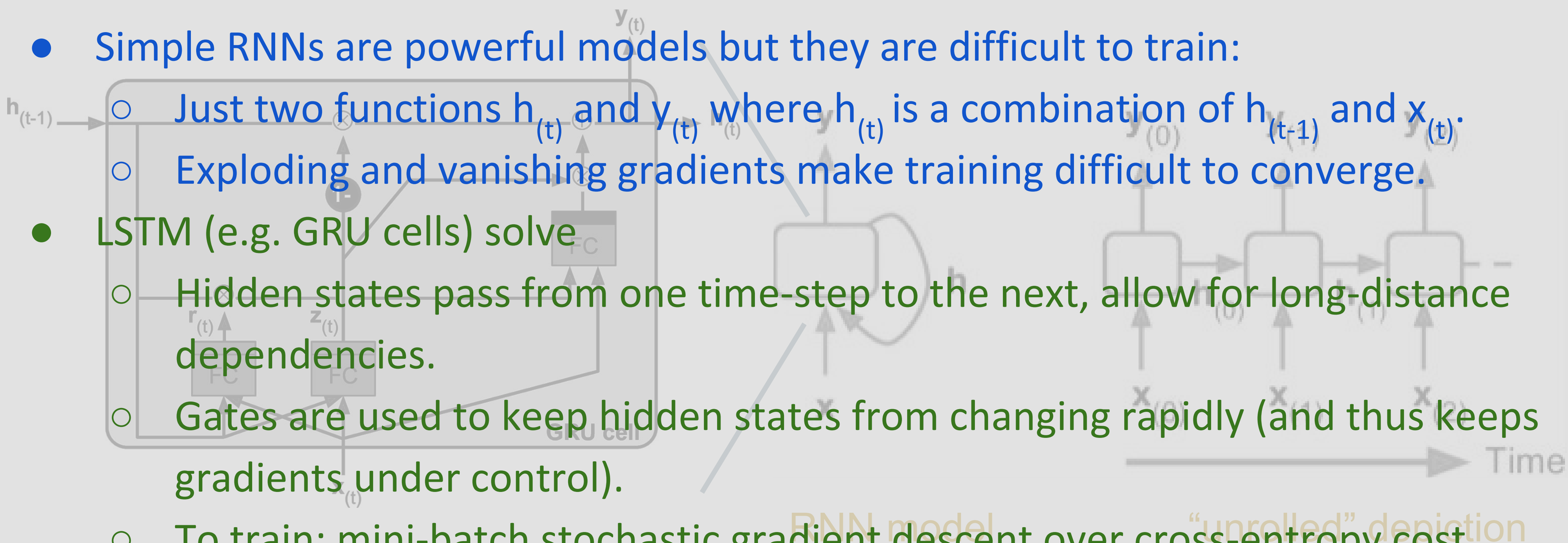
This tends to keep the gradient from vanishing since the same values will be present through multiple times in backpropagation through time. (The same idea applies to LSTMs but is easier to see here).

The cake, which contained candles, was eaten.

The GRU (LSTM): Zoomed out

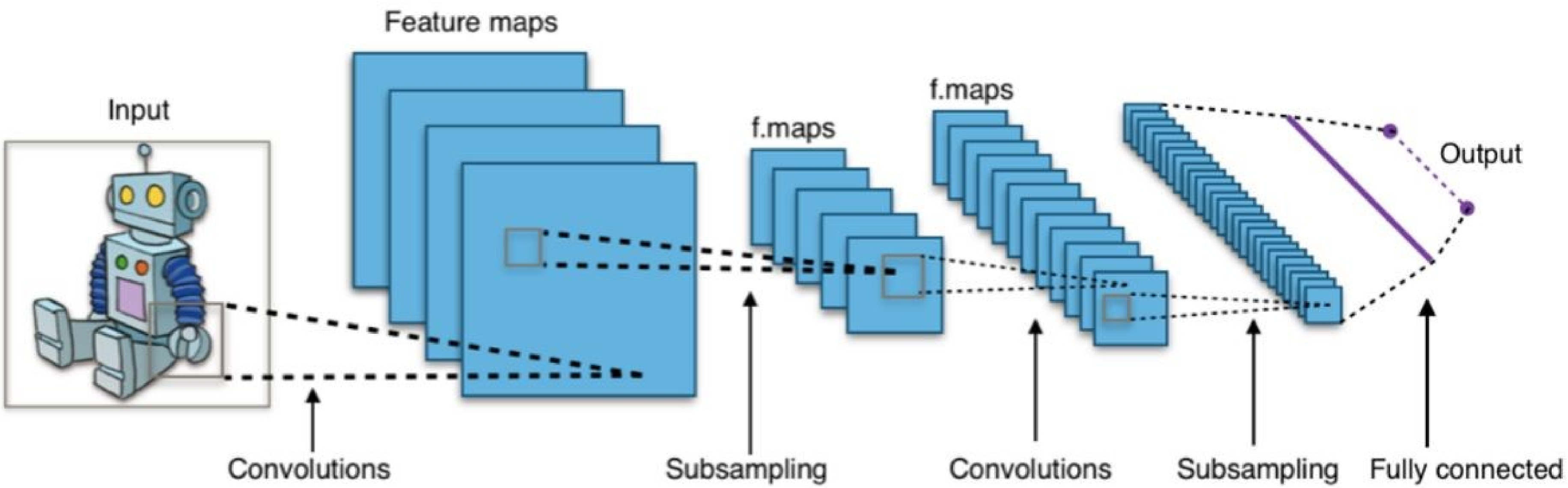
Take-Aways

- Simple RNNs are powerful models but they are difficult to train:
  - Just two functions $h_{(t)}$ and $y_{(t)}$ where $h_{(t)}$ is a combination of $h_{(t-1)}$ and $x_{(t)}$.
  - Exploding and vanishing gradients make training difficult to converge.
- LSTM (e.g. GRU cells) solve
  - Hidden states pass from one time-step to the next, allow for long-distance dependencies.
  - Gates are used to keep hidden states from changing rapidly (and thus keeps gradients under control).
  - To train: mini-batch stochastic gradient descent over cross-entropy cost



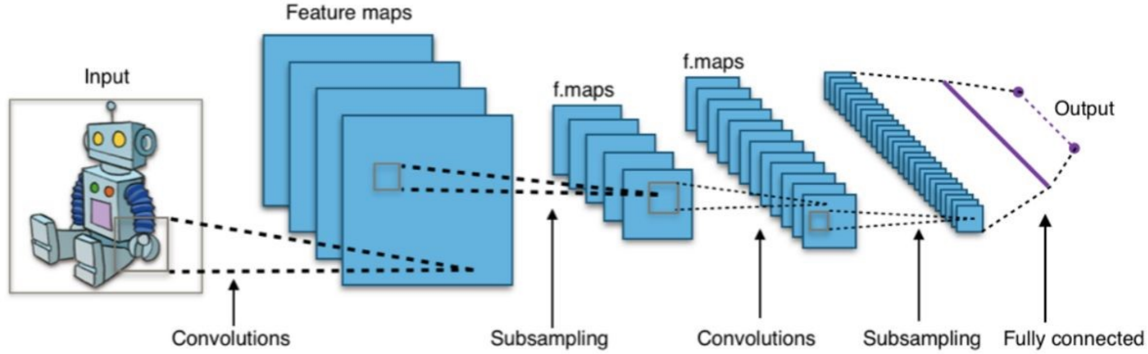
(Geron, 2017)

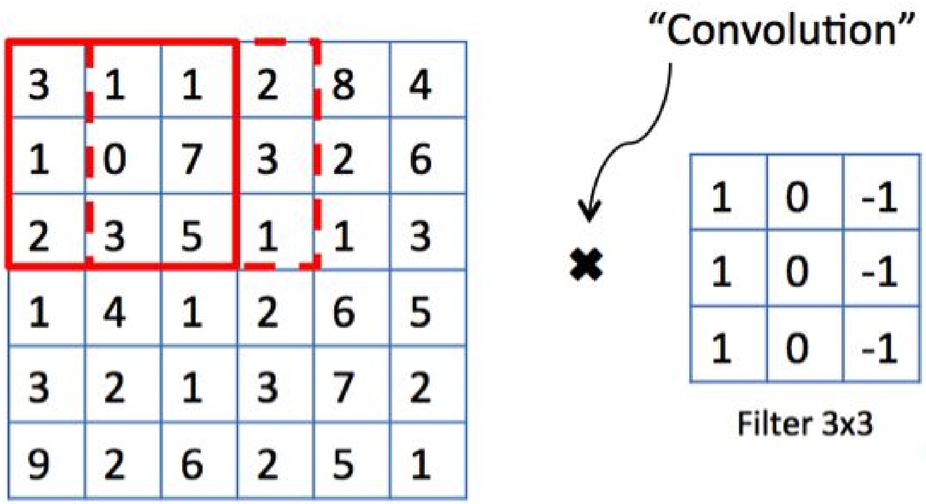
Convolutional Neural Networks



(wikipedia)

Convolution Layer





3	1	1	2	8	4
1	0	7	3	2	6
2	3	5	1	1	3
1	4	1	2	6	5
3	2	1	3	7	2
9	2	6	2	5	1

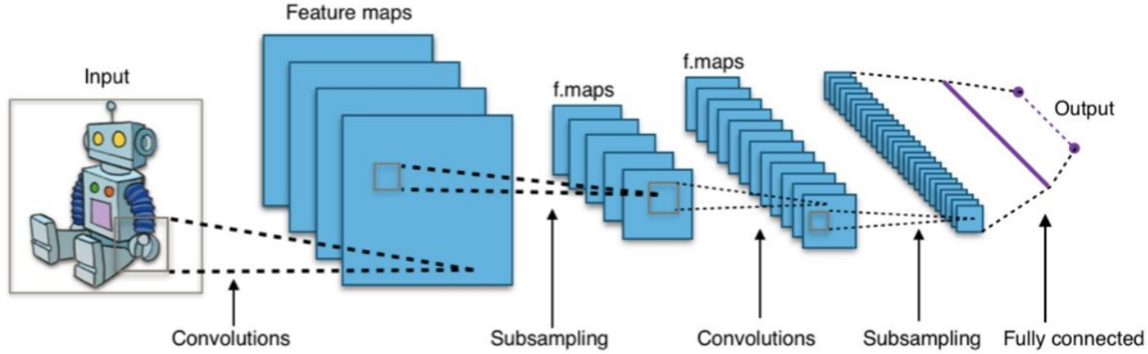
Original image 6x6

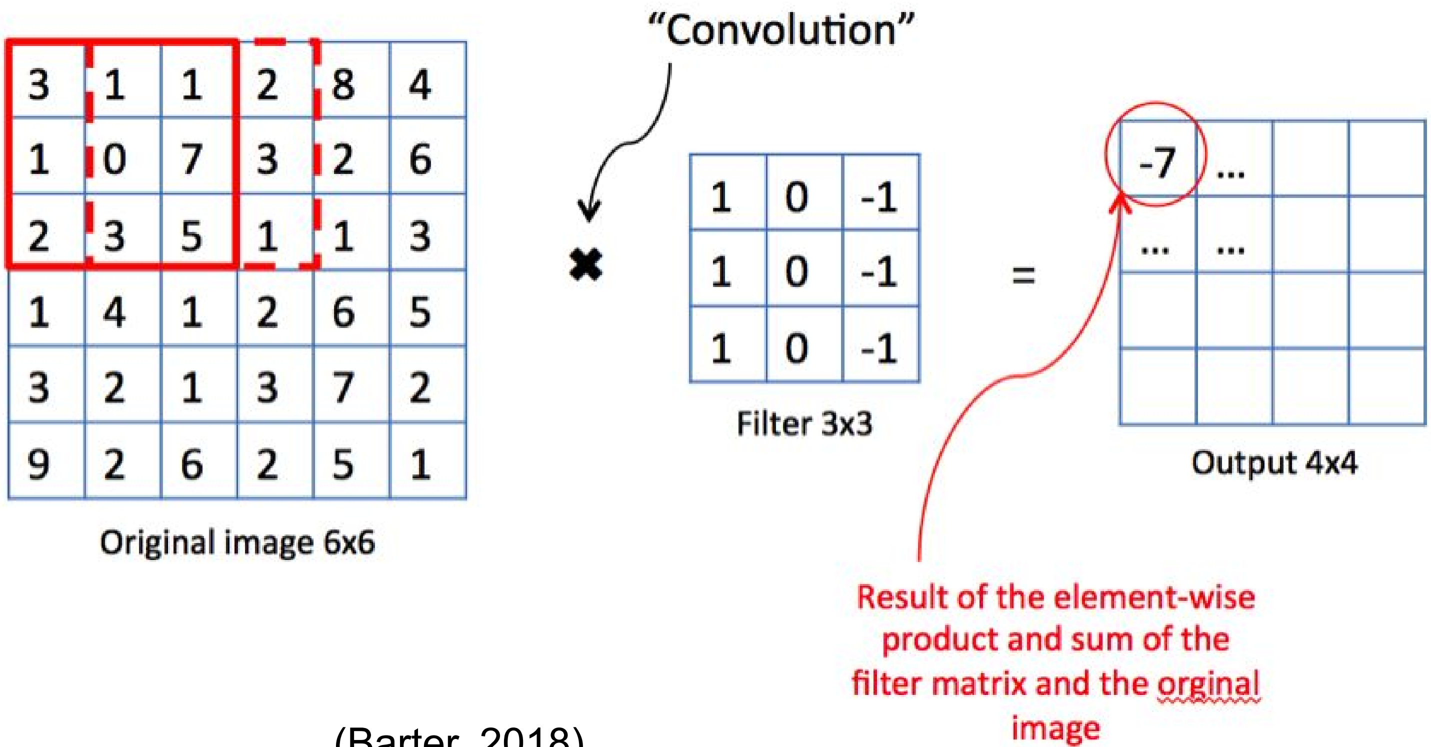
1	0	-1
1	0	-1
1	0	-1

Filter 3x3

(Barter, 2018)

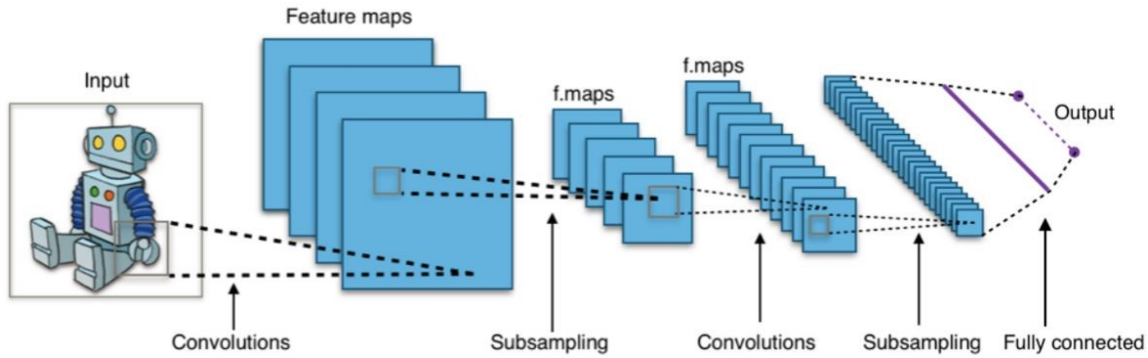
Convolution Layer

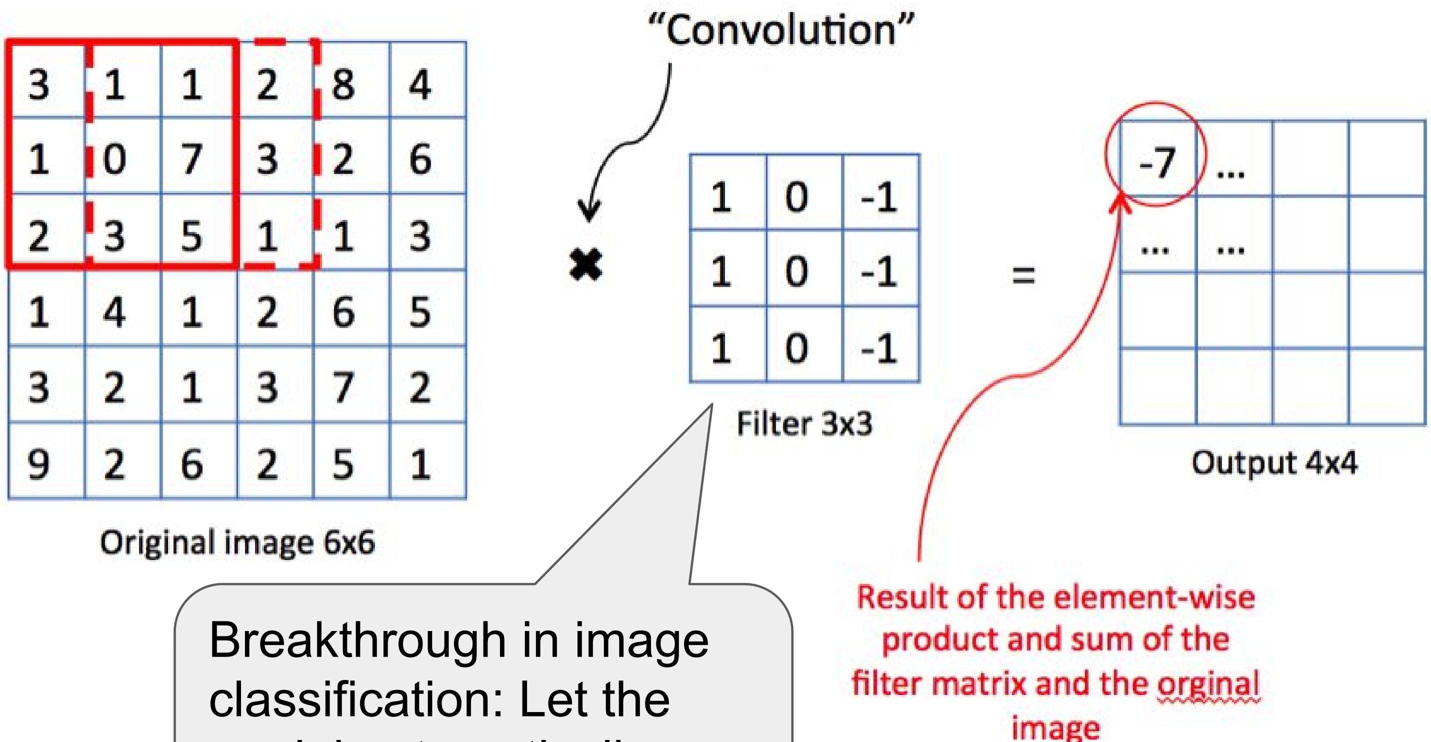




(Barter, 2018)

Convolution Layer

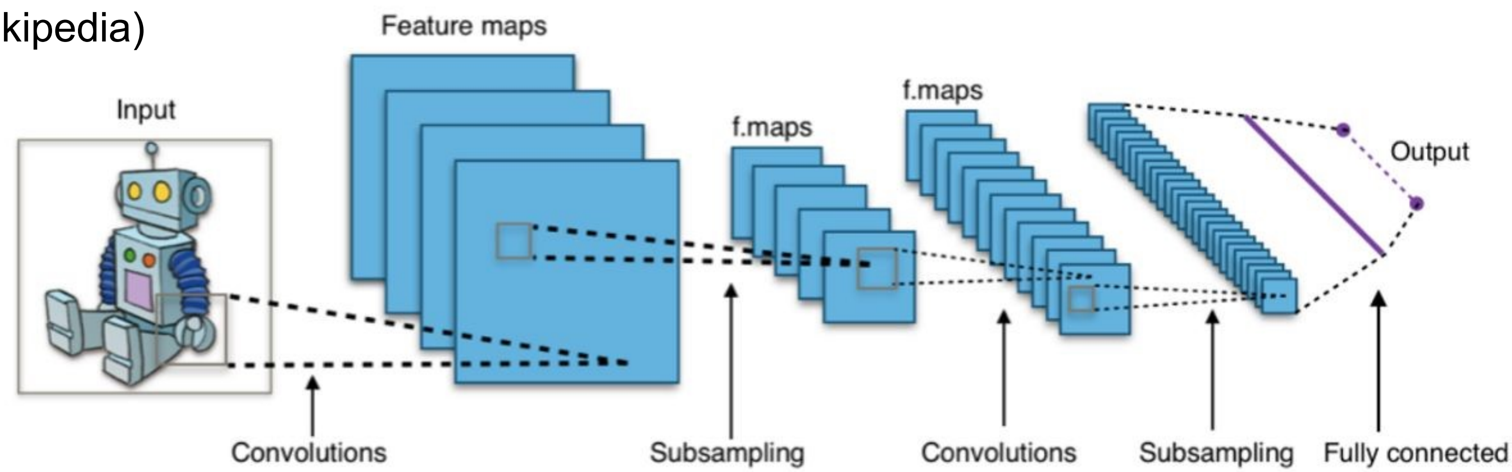




Breakthrough in image classification: Let the model automatically learn the filter weights!

Subsampling (Pooling)

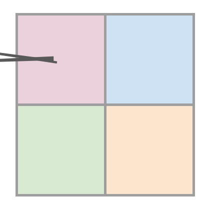
(wikipedia)



Subsampling -- reducing total grid size (i.e. reducing parameters for next layer)

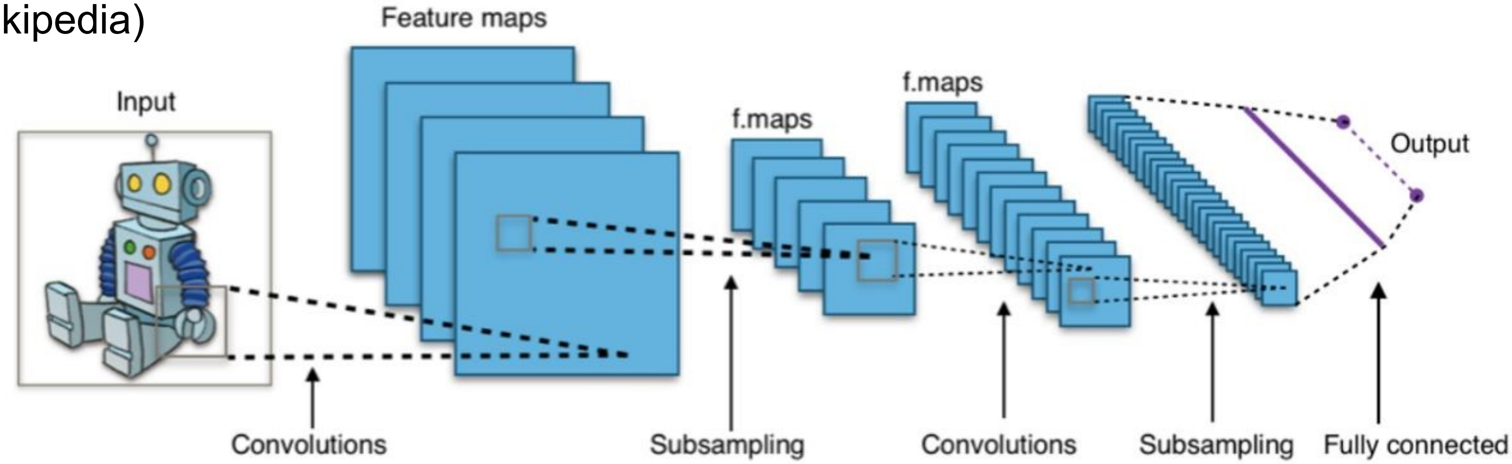
3	4	2	1
1	6	3	7
4	7	9	0
2	1	7	8

2x2 pooling

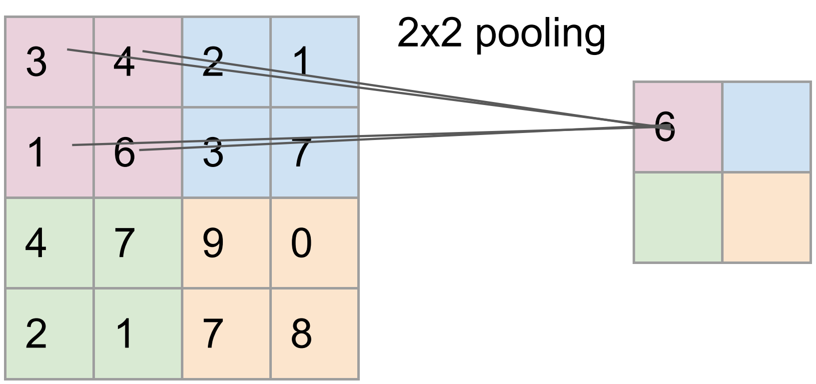


Subsampling (Pooling)

(wikipedia)



Subsampling -- reducing total grid size (i.e. reducing parameters for next layer)

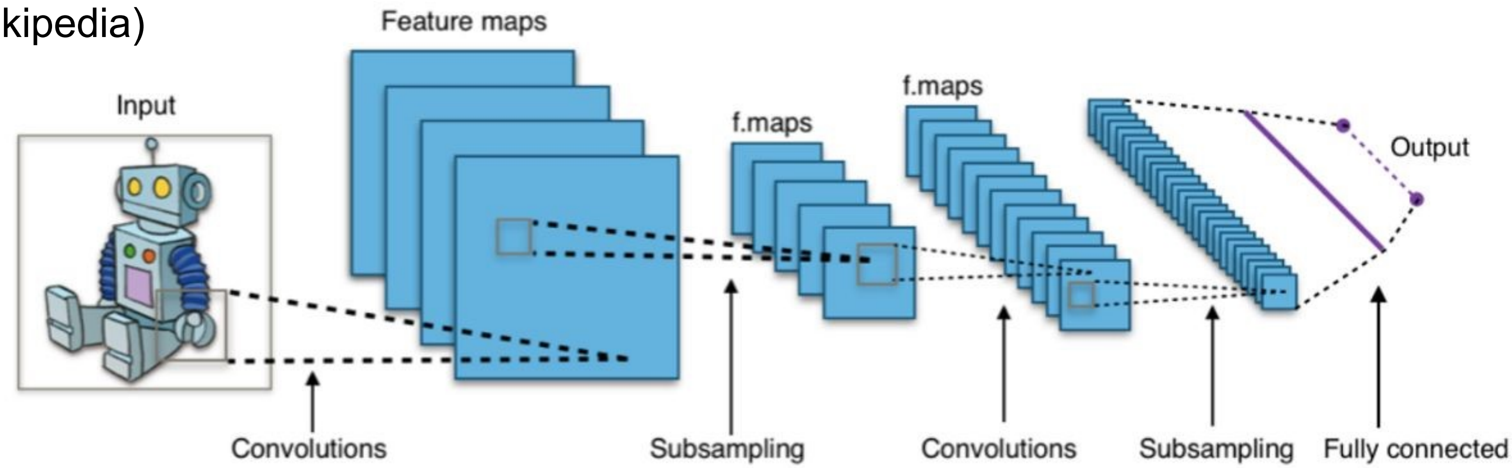


Types of pooling

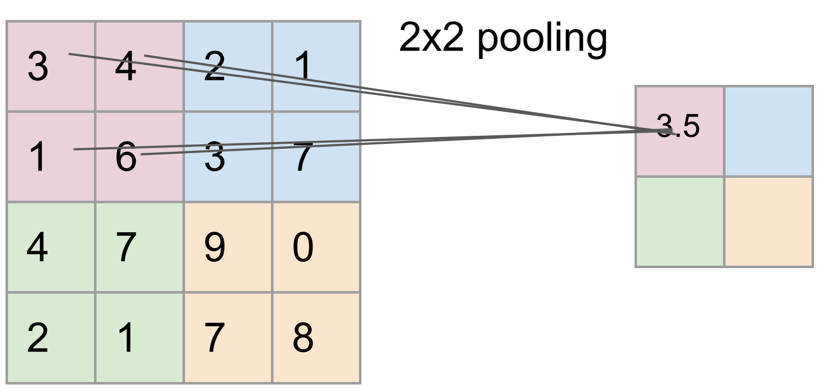
- **max**
- avg

Subsampling (Pooling)

(wikipedia)



Subsampling -- reducing total grid size (i.e. reducing parameters for next layer)



Types of pooling

- max
- **avg**

Standard Training Loss Function

```
RNN_cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
            #where did this come from?
```

Logistic Regression Likelihood: $L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

Final Cost Function: $J^{(t)} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j}^{(t)} log\ \hat{y}_{i,j}^{(t)}$ -- "cross entropy error"

Standard Training Loss Function

```
RNN_cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
            #where did this come from?
```

Logistic Regression Likelihood: $L(\beta_0, \beta_1, ..., \beta_k | X, Y) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

Log Likelihood: $\ell(\beta) = \sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p(x_i))$

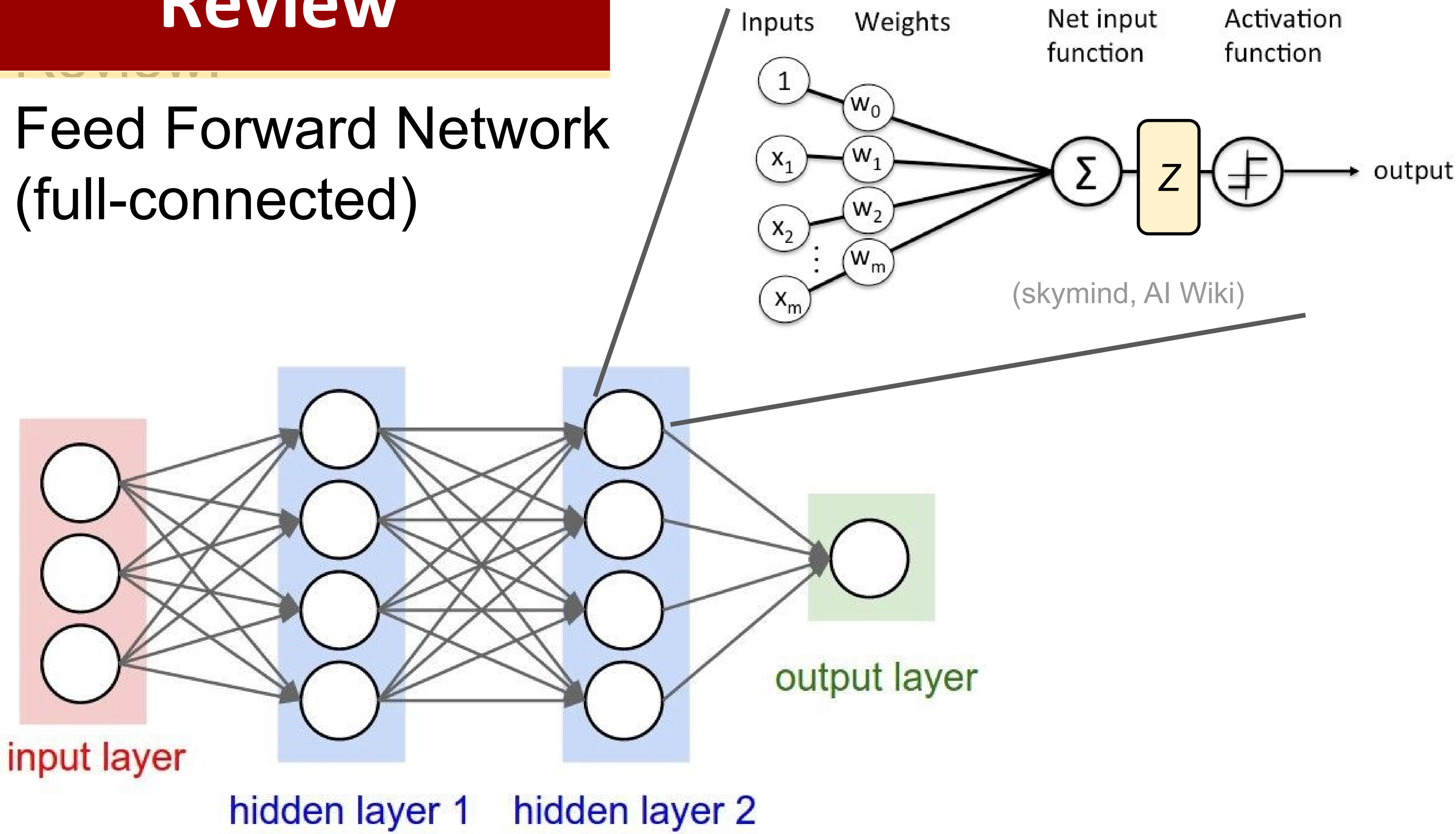
Log Loss: $J(\beta) = -\frac{1}{N}\sum_{i=1}^{N} y_i log\ p(x_i) + (1 - y_i) log\ (1 - p)(x_i))$

Cross-Entropy Cost: $J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_i log\ p(x_{i,j})$ (a “multiclass” log loss)

Final Cost Function: $J^{(t)} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{|V|} y_{i,j}^{(t)} log\ \hat{y}_{i,j}^{(t)}$ -- ”cross entropy error”

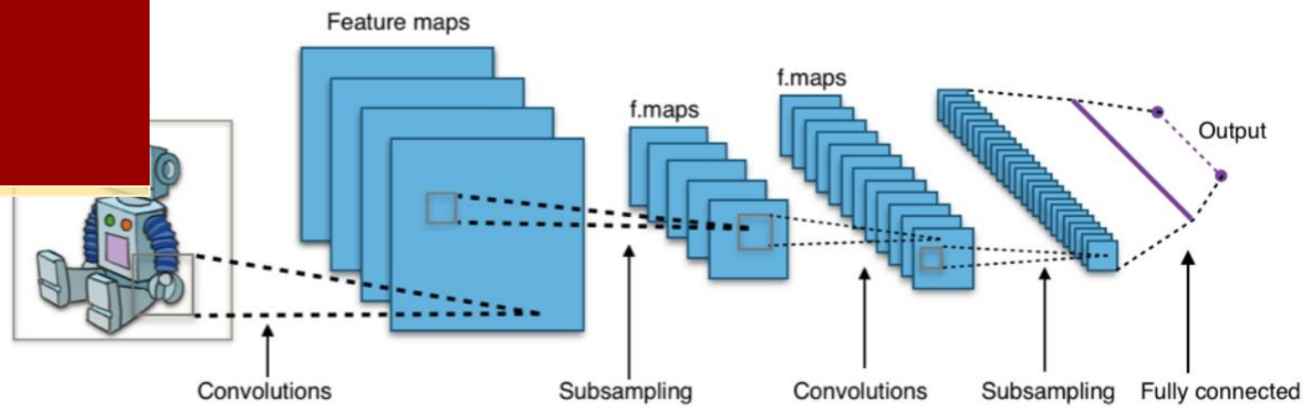
Review

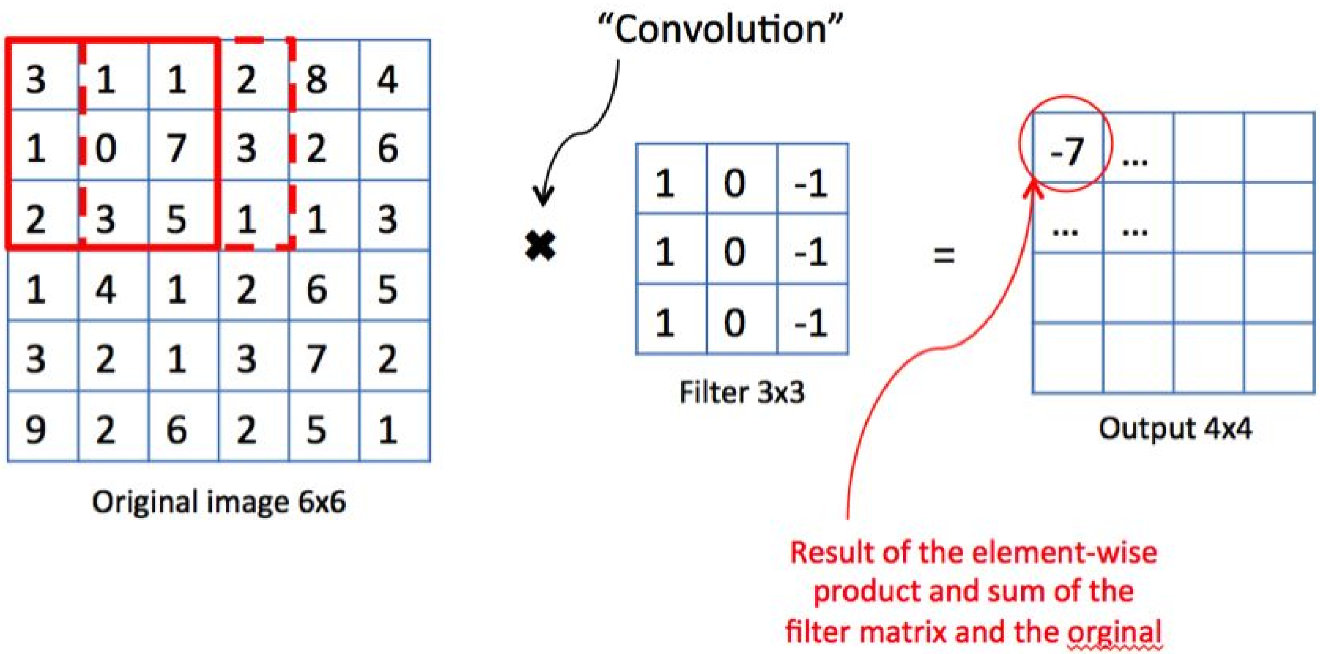
Feed Forward Network (full-connected)



Review

Convolutional NN





(Barter, 2018)

Review

Recurrent Neural Network

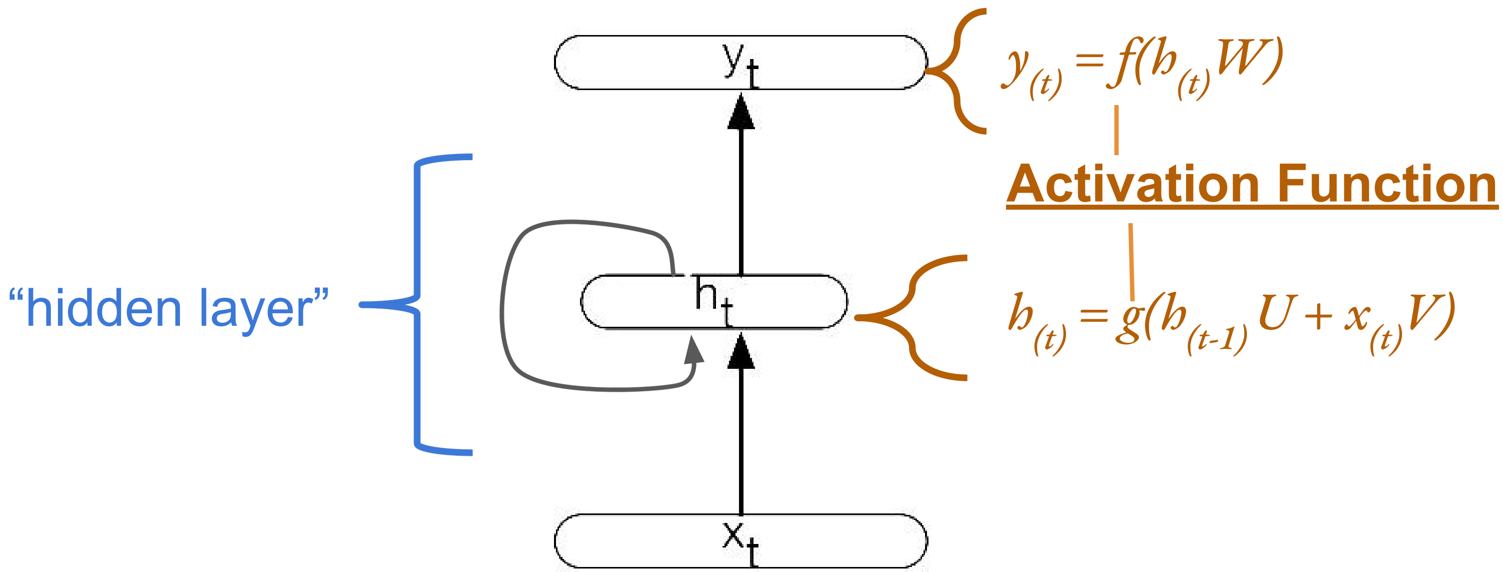
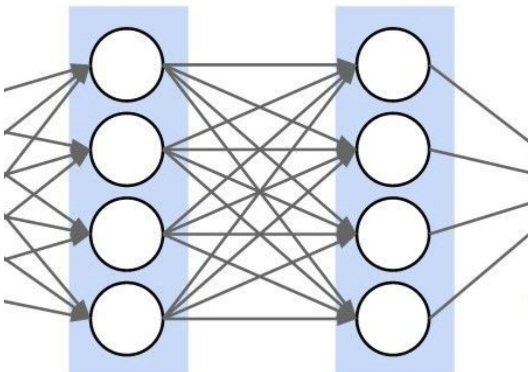


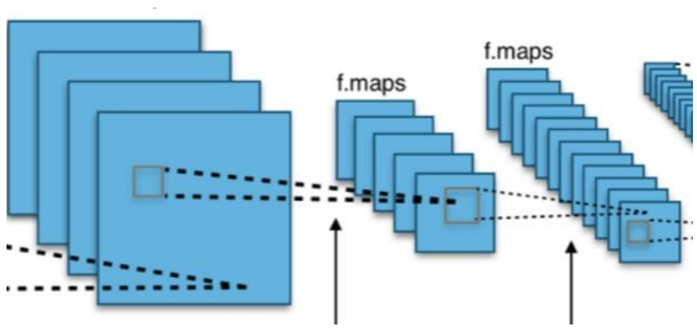
Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

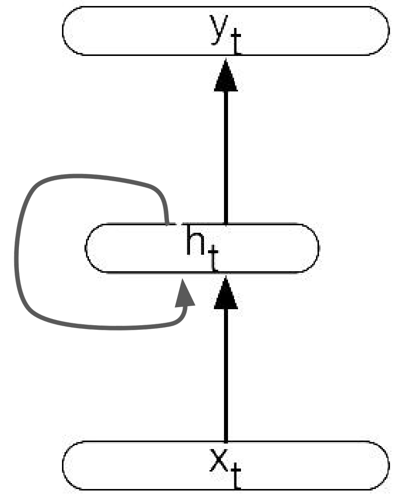
FFN



CNN

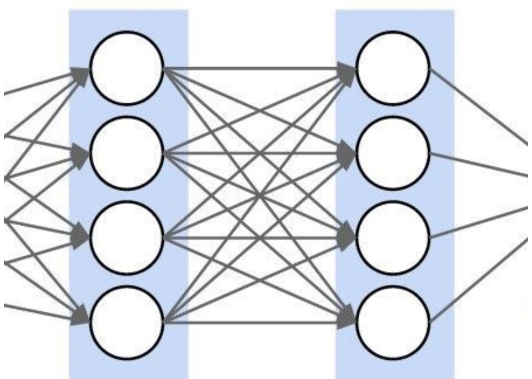


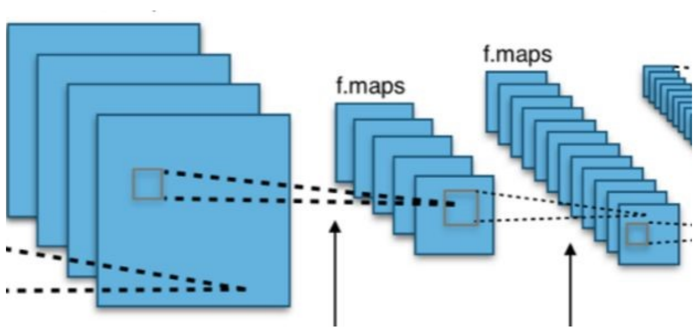
RNN

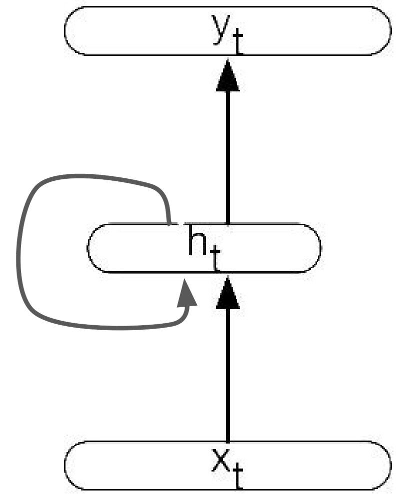


Can model computation (e.g. matrix operations for a single input) be parallelized?

FFN **CNN** **RNN**

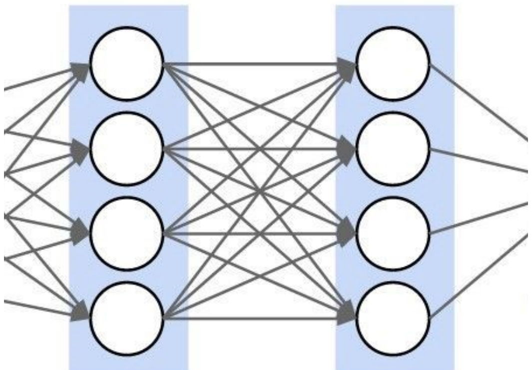




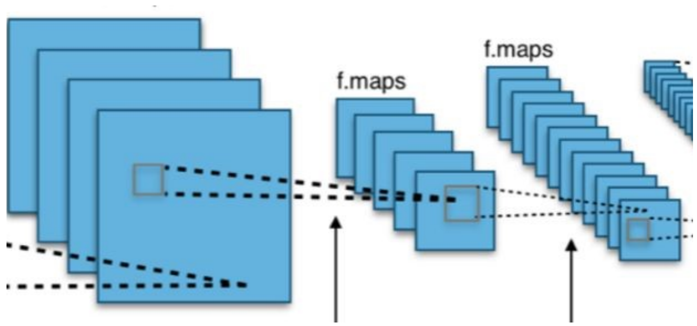


Can model computation (e.g. matrix operations for a single input) be parallelized?

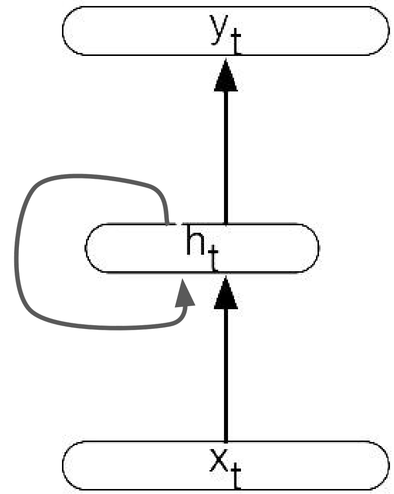
FFN



CNN

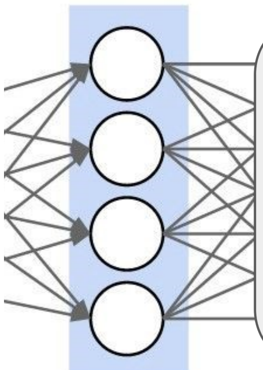


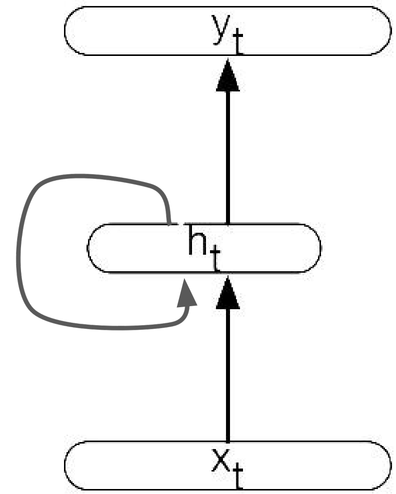
RNN



Can model computation (e.g. matrix operations for a single input) be parallelized?

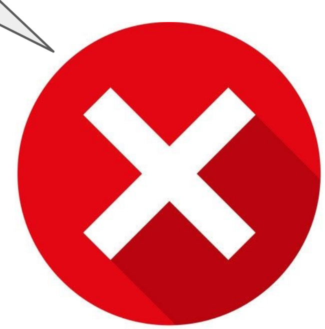
FFN CNN RNN





Ultimately limits how complex the model can be (i.e. it's total number of paramers/weights) as compared to a CNN.

Can model computation (e.g. matrix operations for a single ... t) be parallelized?



The Transformer: Attention-only Models

Can handle sequences and long-distance dependencies, but….

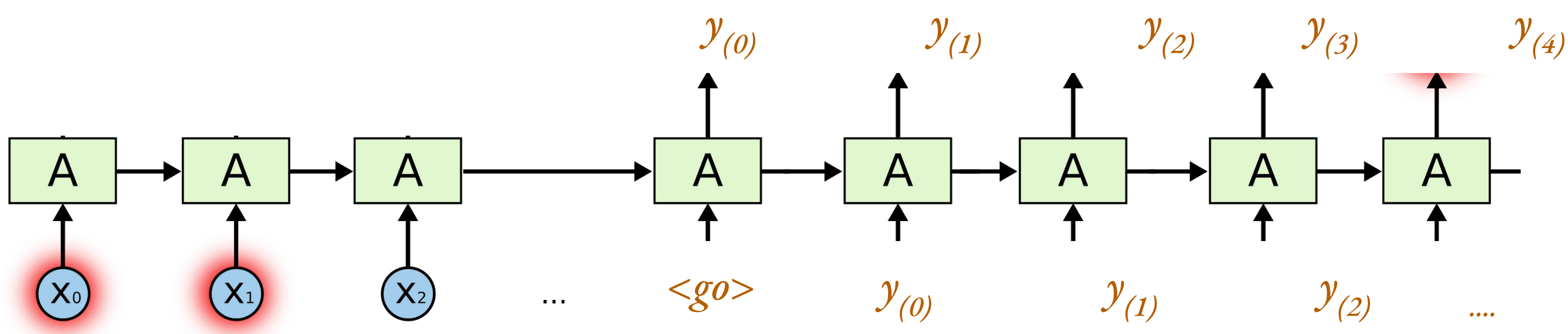
- Don’t want complexity of LSTM/GRU cells
- Constant num edges between input steps
- Enables “interactions” (i.e. adaptations) between words
- **Easy to parallelize -- don’t need sequential processing.**

The Transformer: Attention-only Models

Challenge:

The ball was kicked by kayla.

- Long distance dependency when translating:



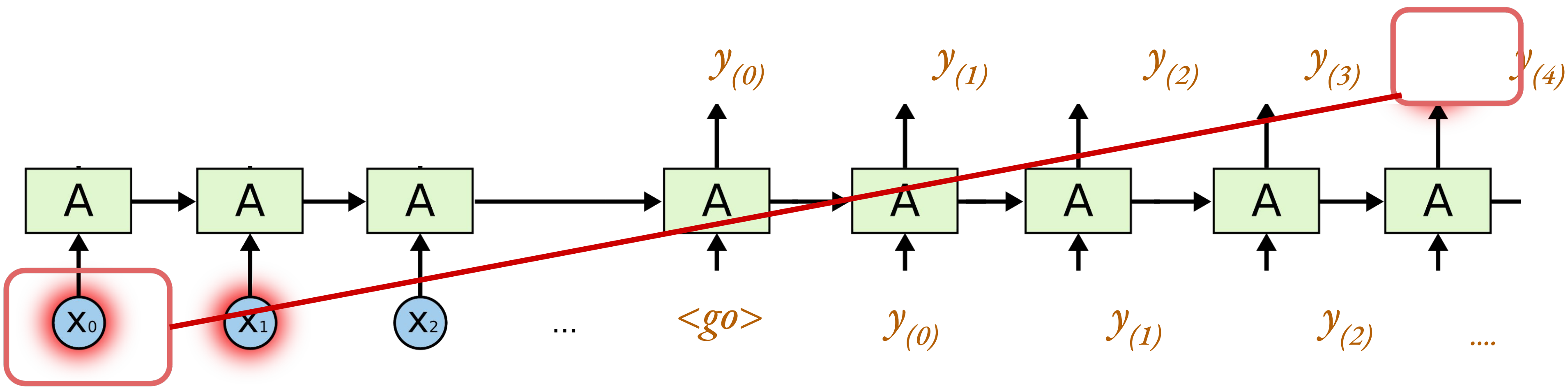
Kayla kicked the ball.

The Transformer: Attention-only Models

Challenge:

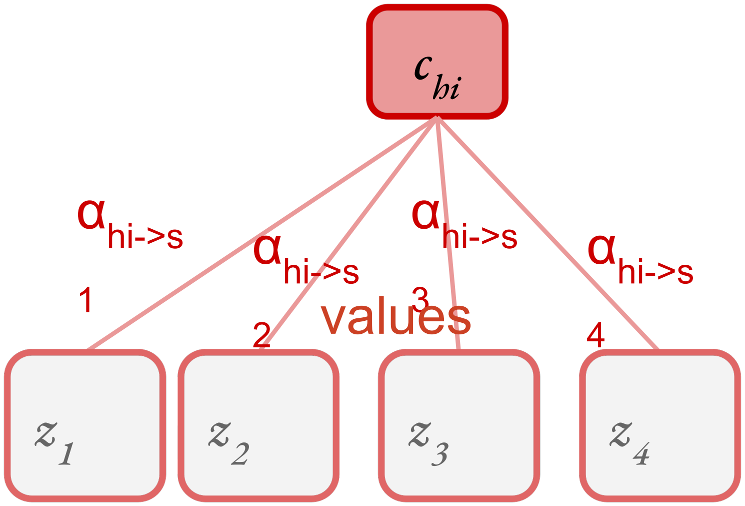
The ball was kicked by kayla.

- Long distance dependency when translating:

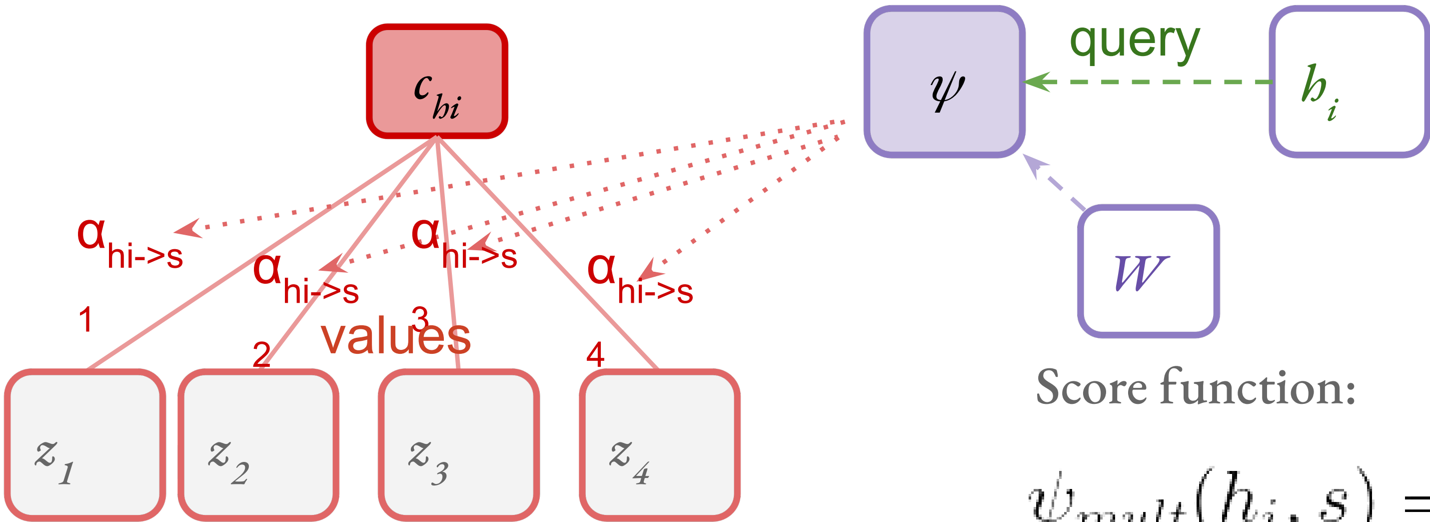


Kayla kicked the ball.

Attention



Attention

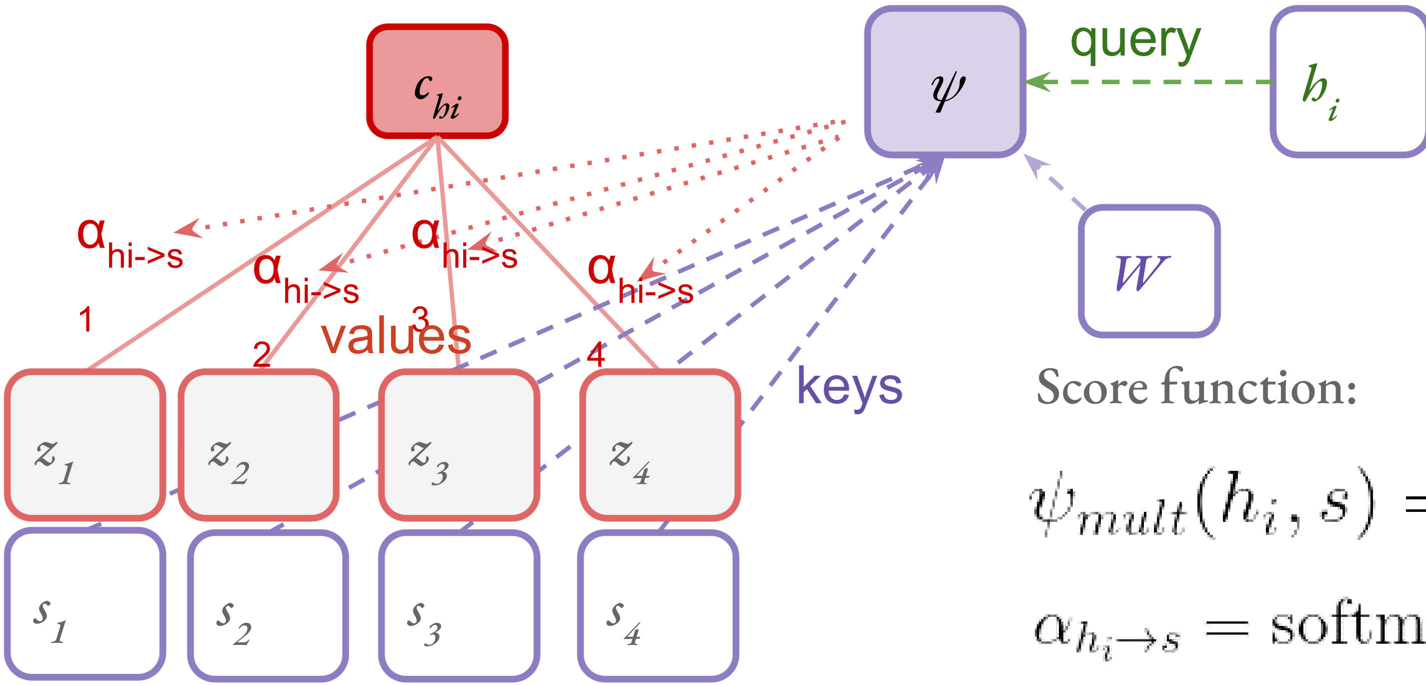


Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

Attention



Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

$$c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \to s_n} z_n$$

The Transformer: Attention-only Models

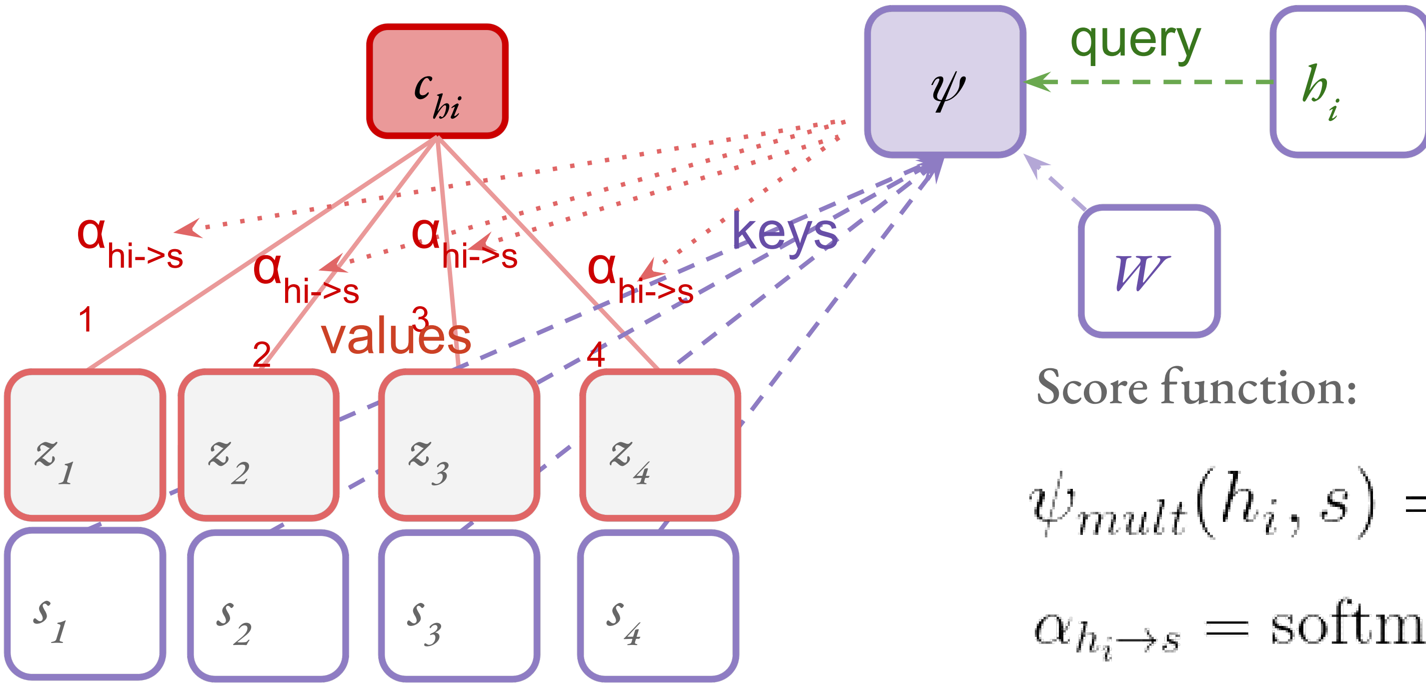
Challenge:

- Long distance dependency when translating:

Attention came about for encoder decoder models.

Then self-attention was introduced:

Attention



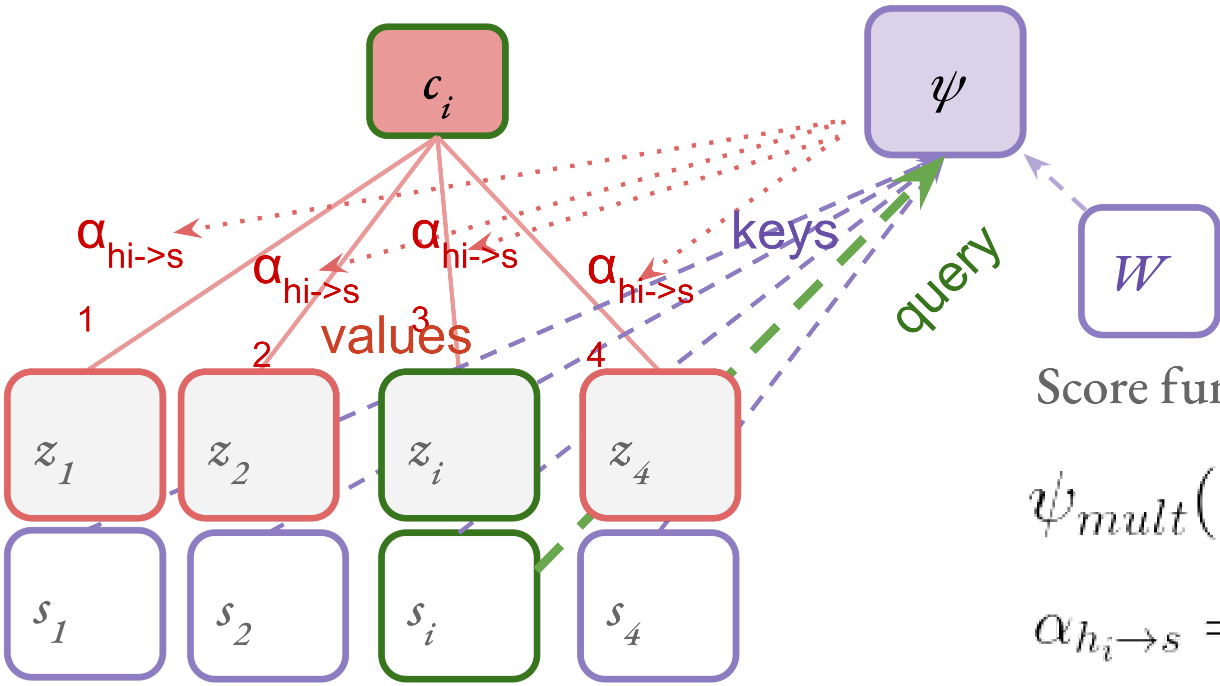
Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

$$c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \to s_n} z_n$$

Attention



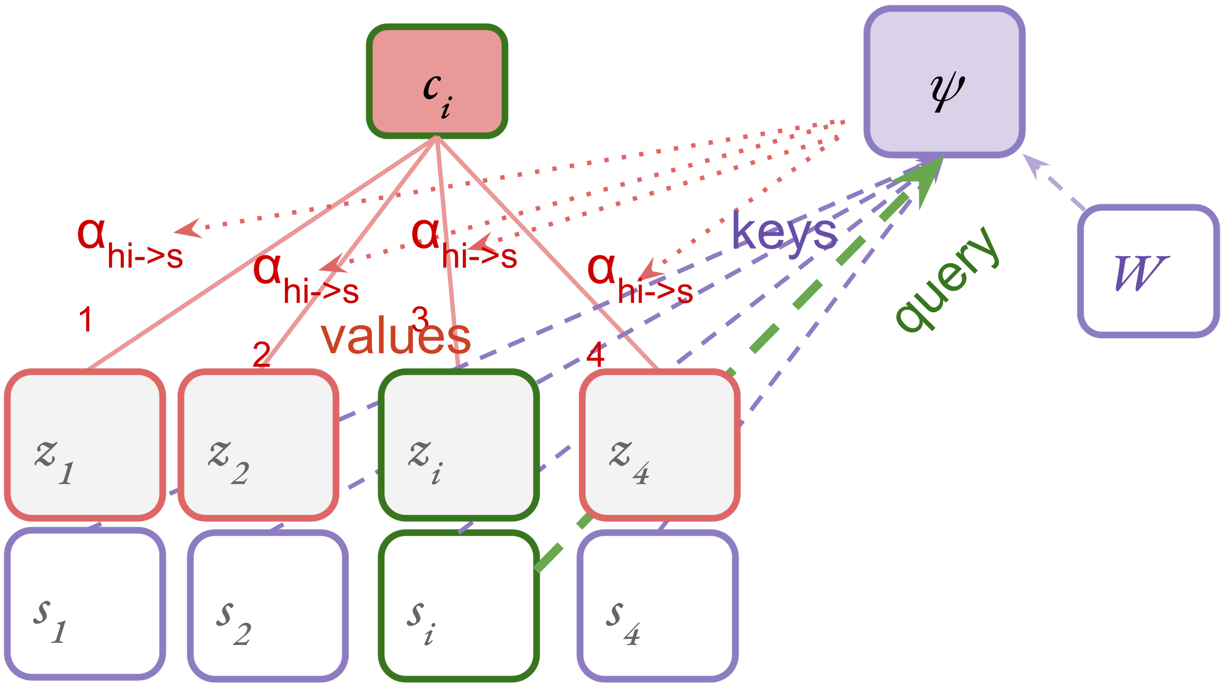
Score function:

$$\psi_{mult}(h_i, s) = s^T W h_i$$

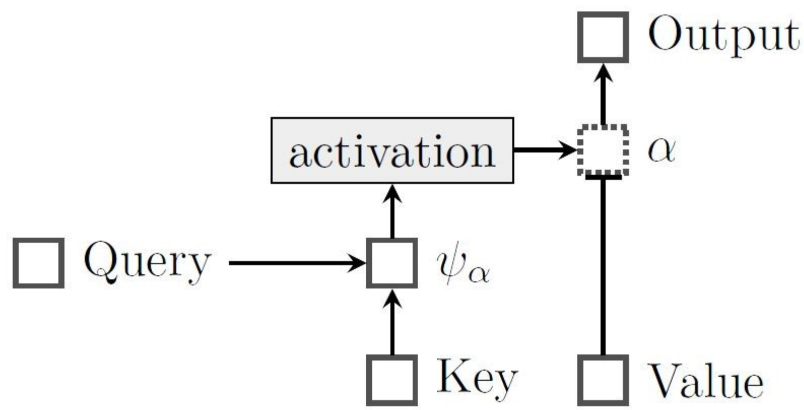
$$\alpha_{h_i \to s} = \text{softmax}(\psi(h_i, s))$$

$$c_{h_i} = \sum_{n=1}^{|s|} \alpha_{h_i \to s_n} z_n$$

Attention



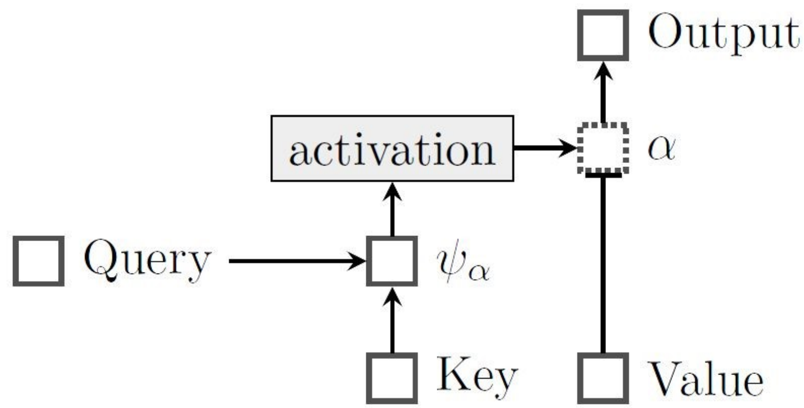
Attention as weighting a value based on a query and key:



(Eisenstein, 2018)

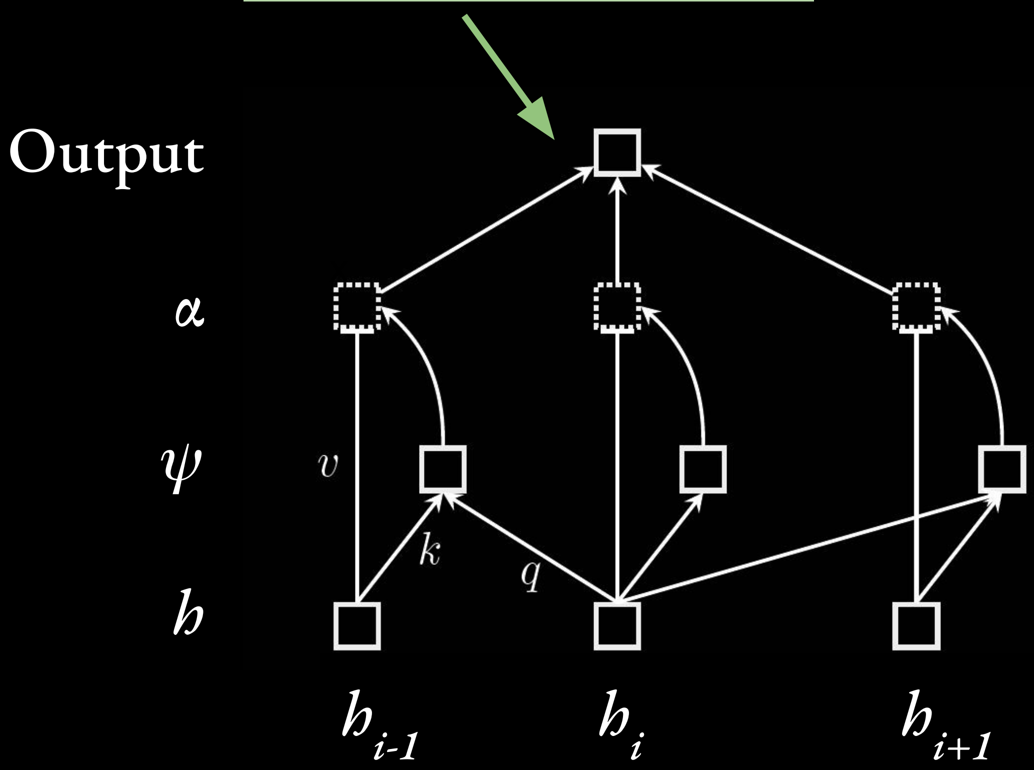
The Transformer: Attention-only Models

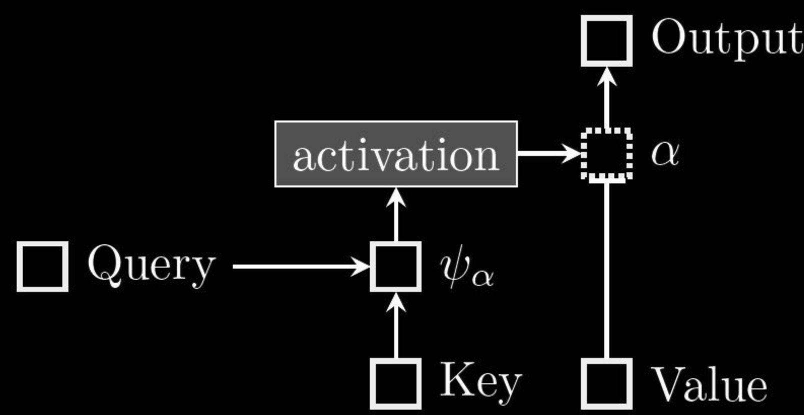
Attention as weighting a value based on a query and key:



(Eisenstein, 2018)

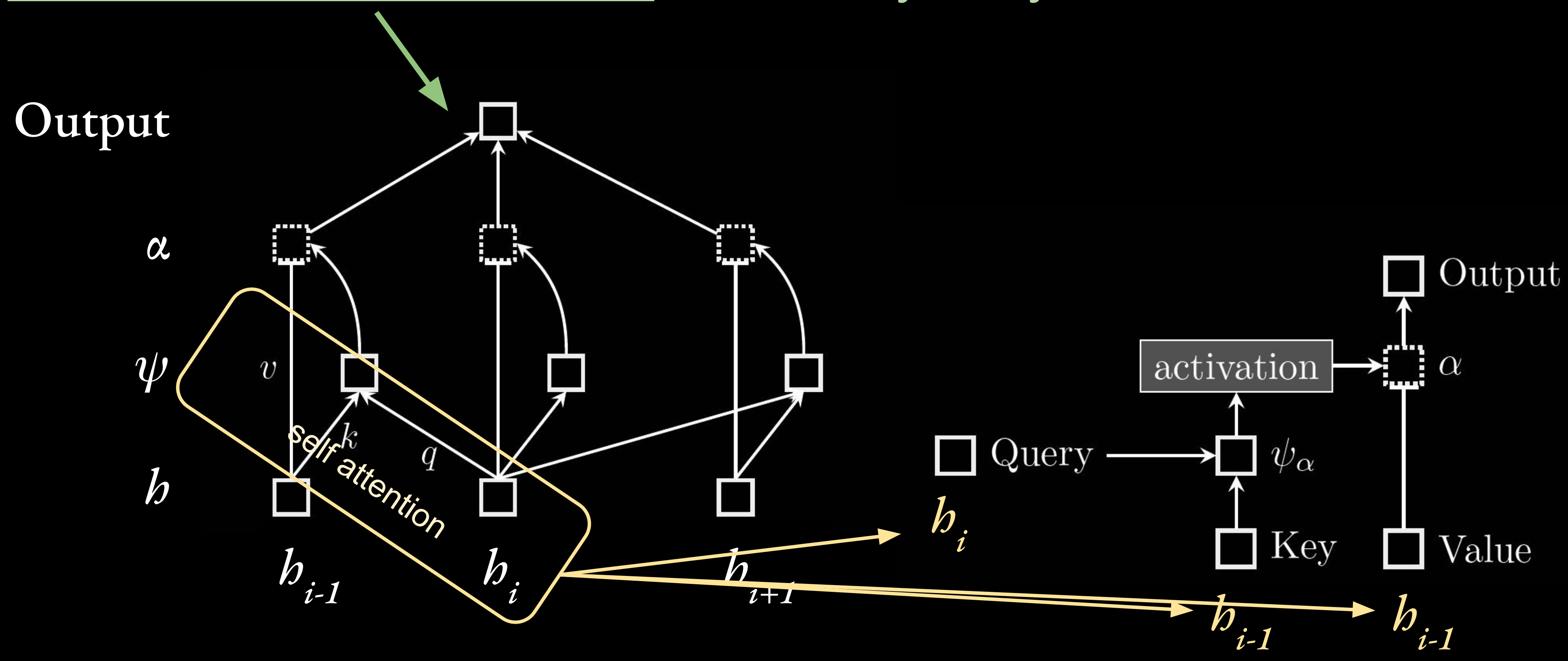
The Transformer: Attention-only Models





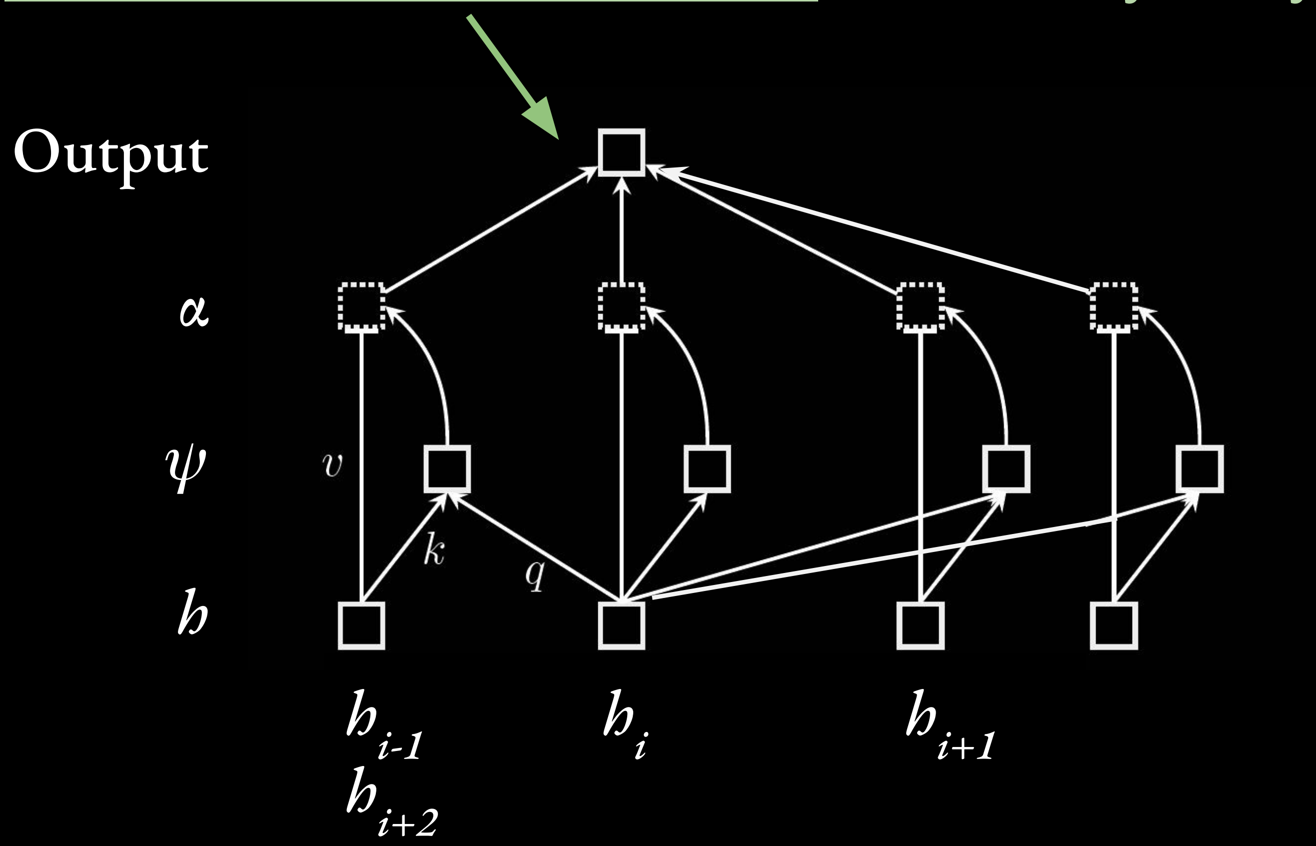
(Eisenstein, 2018)

The Transformer: Attention-only Models

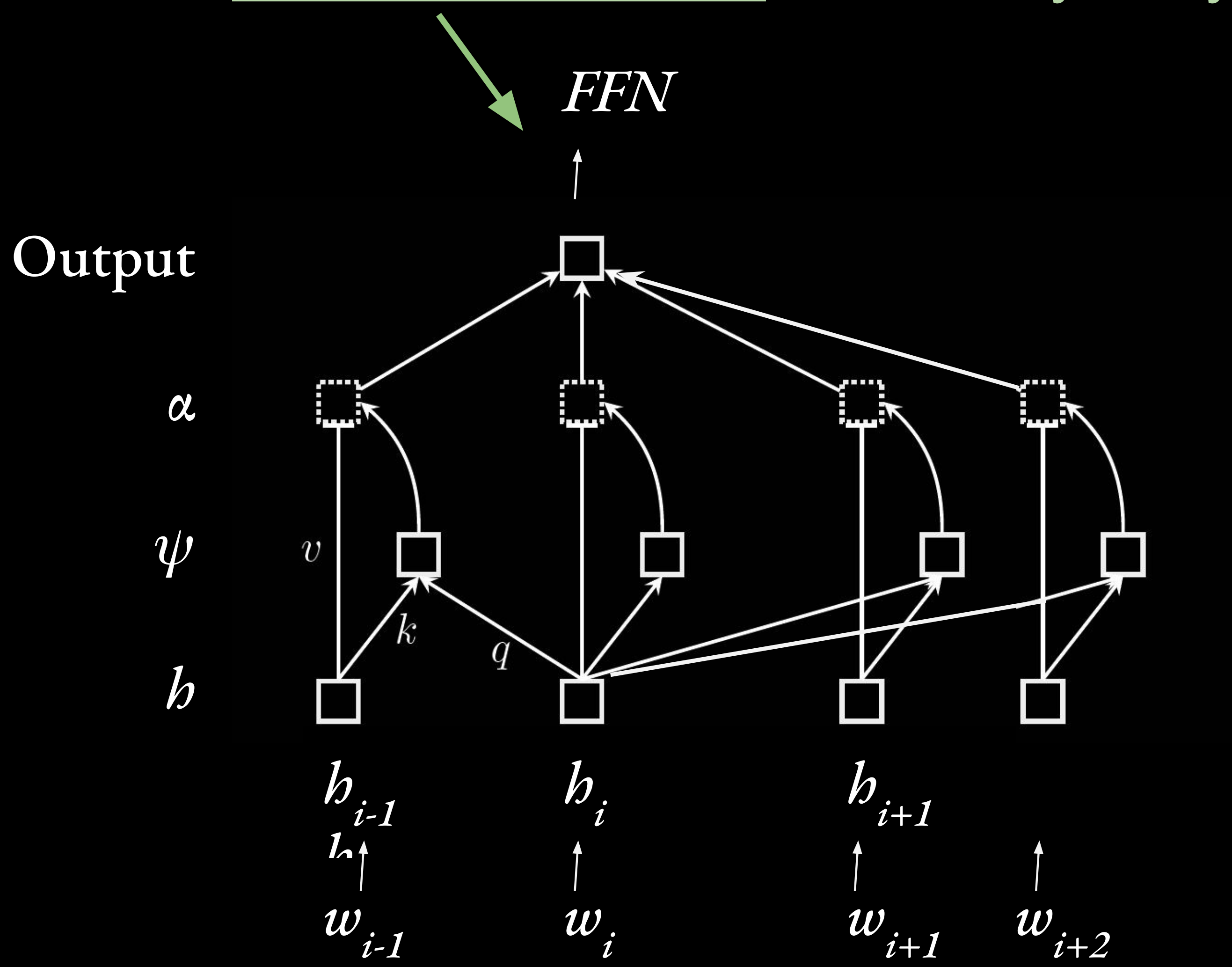


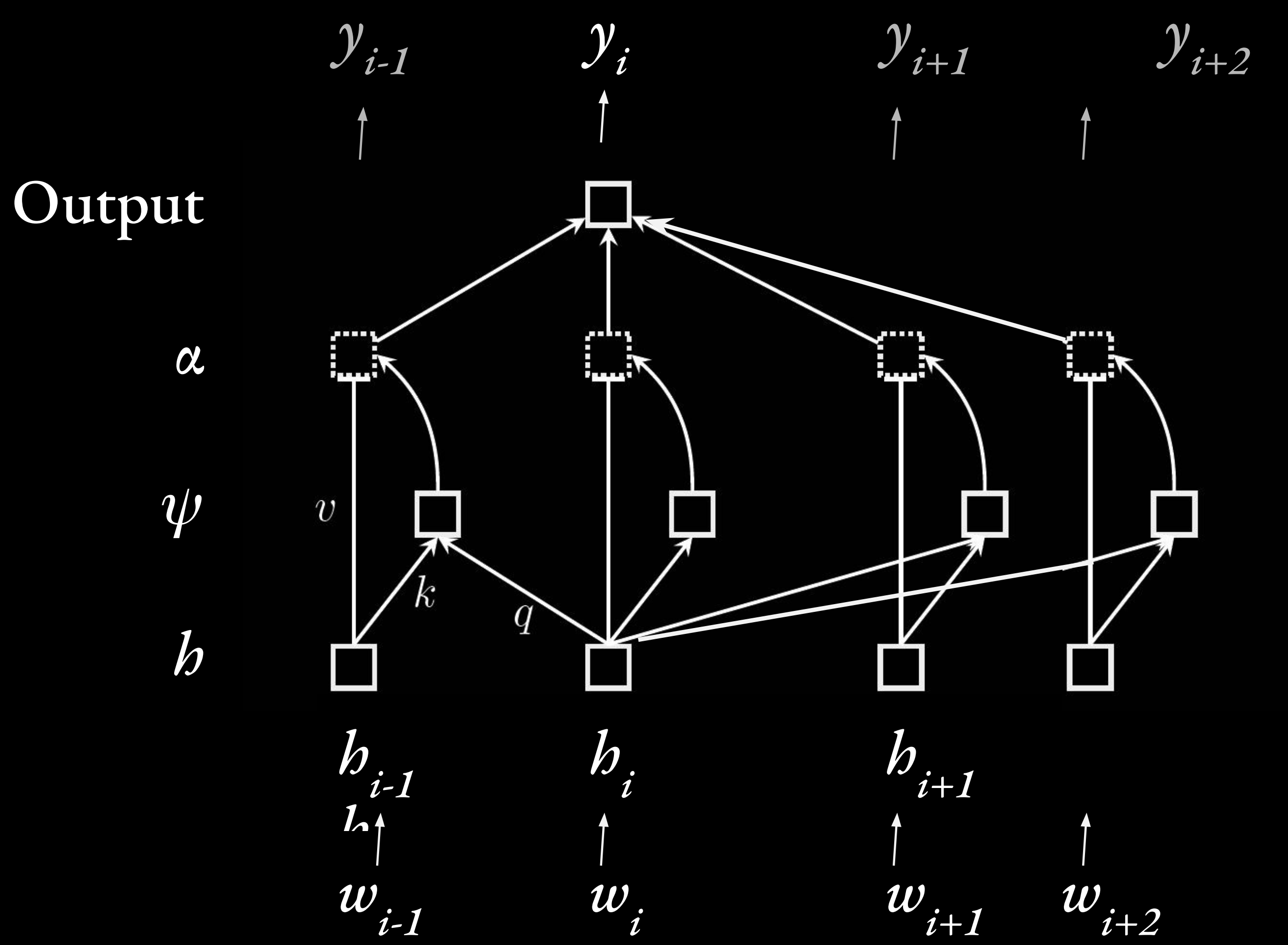
(Eisenstein, 2018)

The Transformer: Attention-only Models



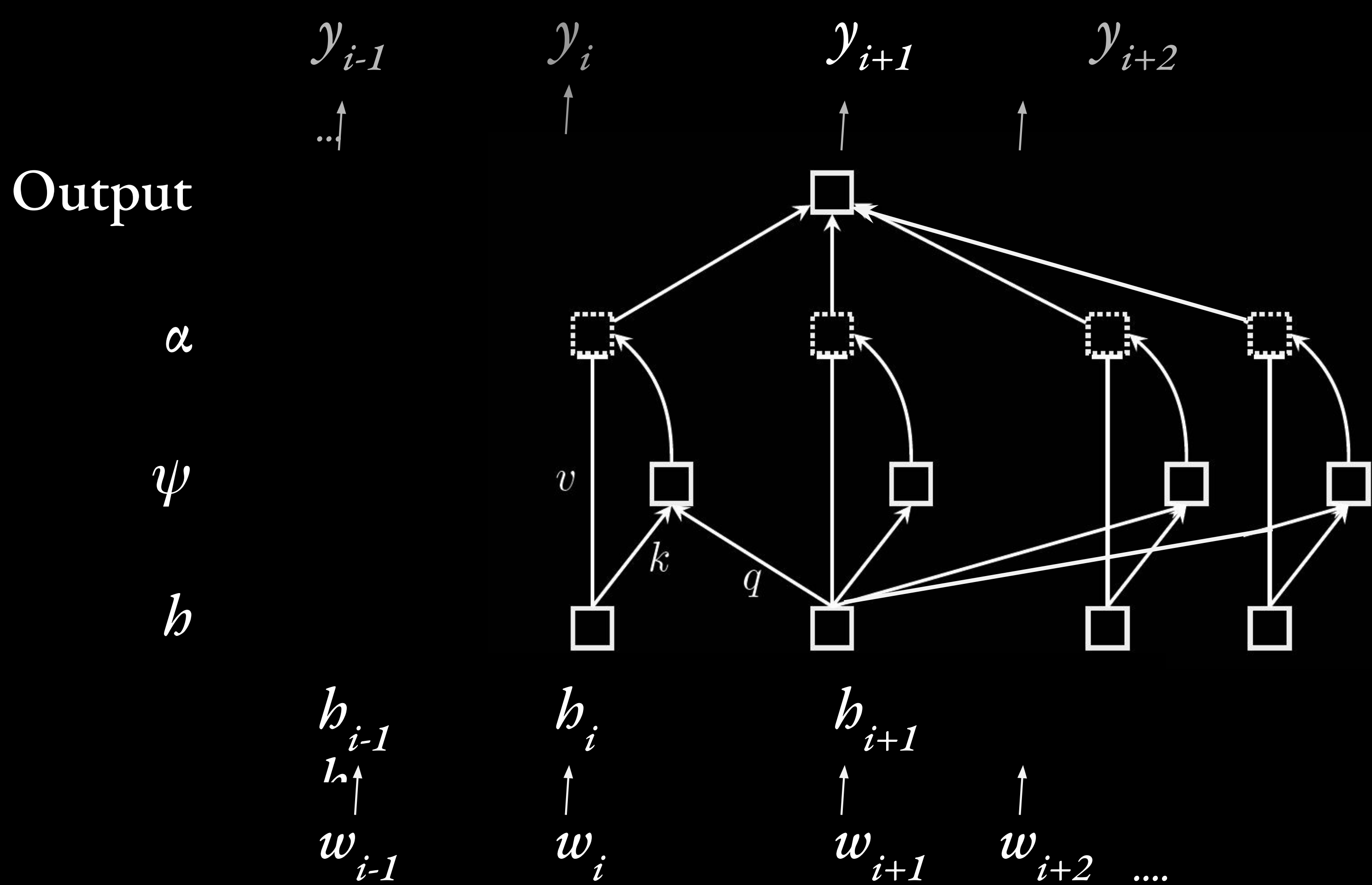
The Transformer: Attention-only Models



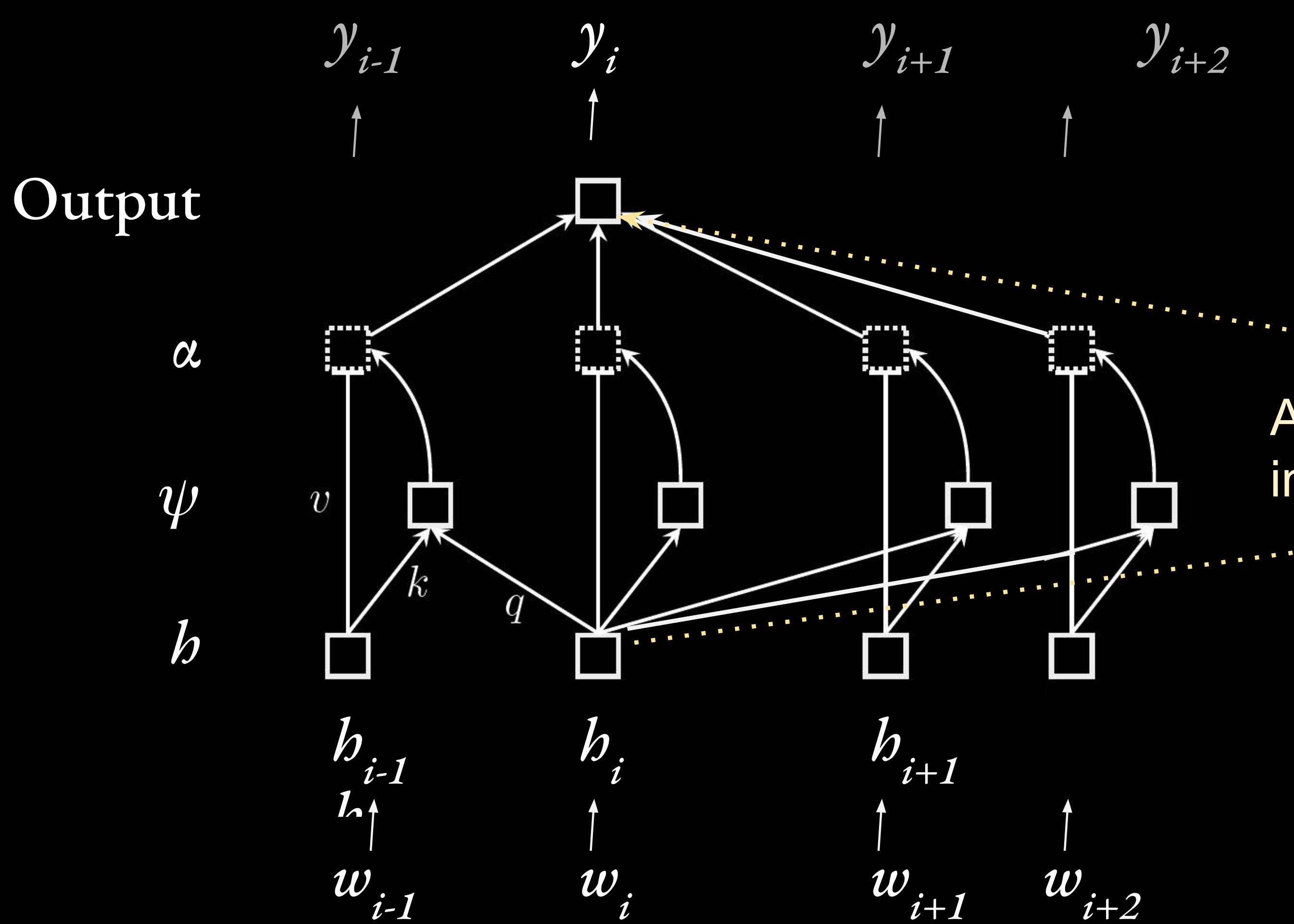


y_{i-1}
y_i
y_{i+1}
y_{i+2}
Output
α
ψ
v
k
q
h
h_{i-1}
h_i
h_{i+1}
w_{i-1}
w_i
w_{i+1}
w_{i+2}

The Transformer: "Attention-only" models

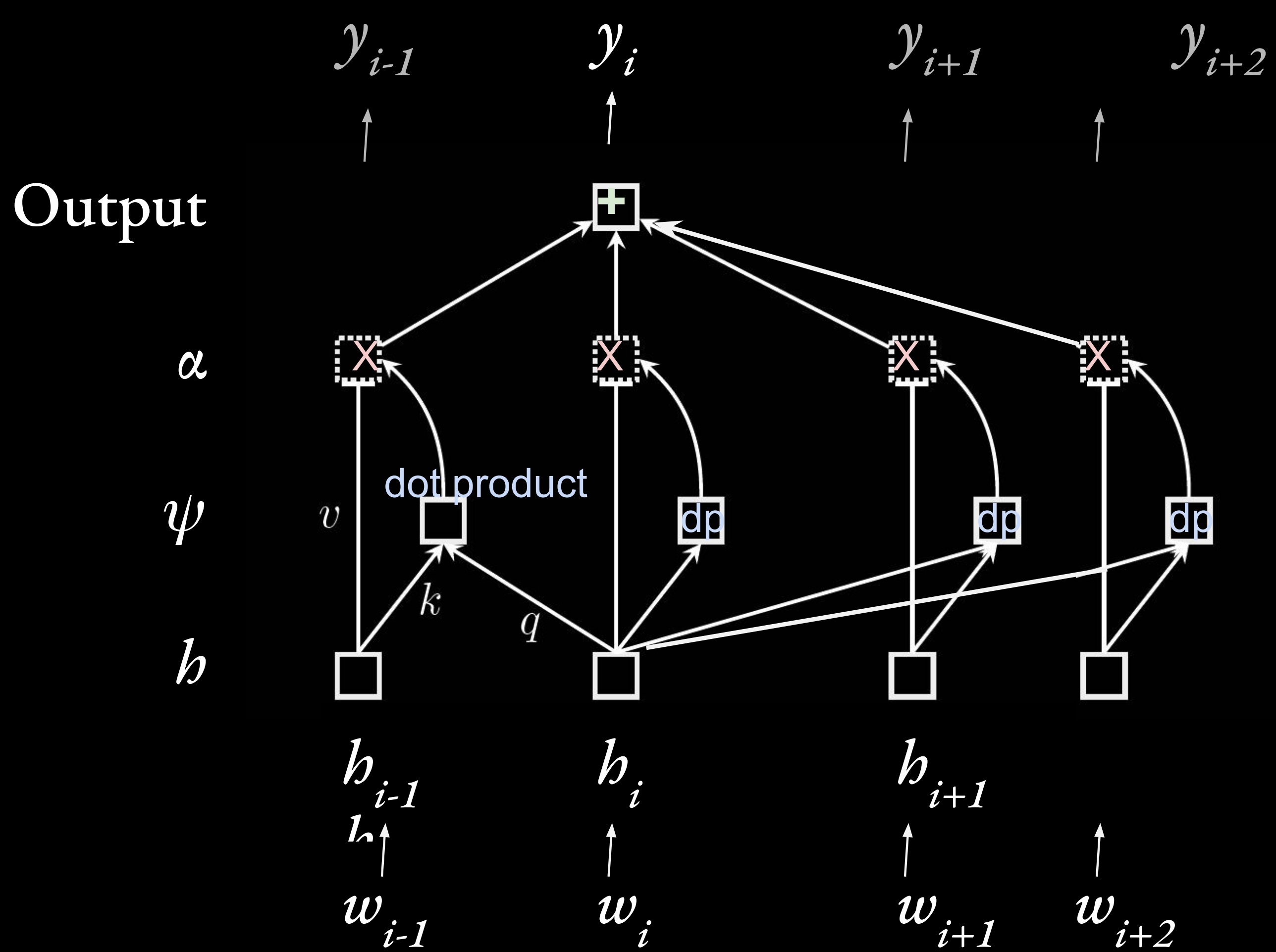


The Transformer: "Attention-only" models



Attend to all hidden states in your "neighborhood".

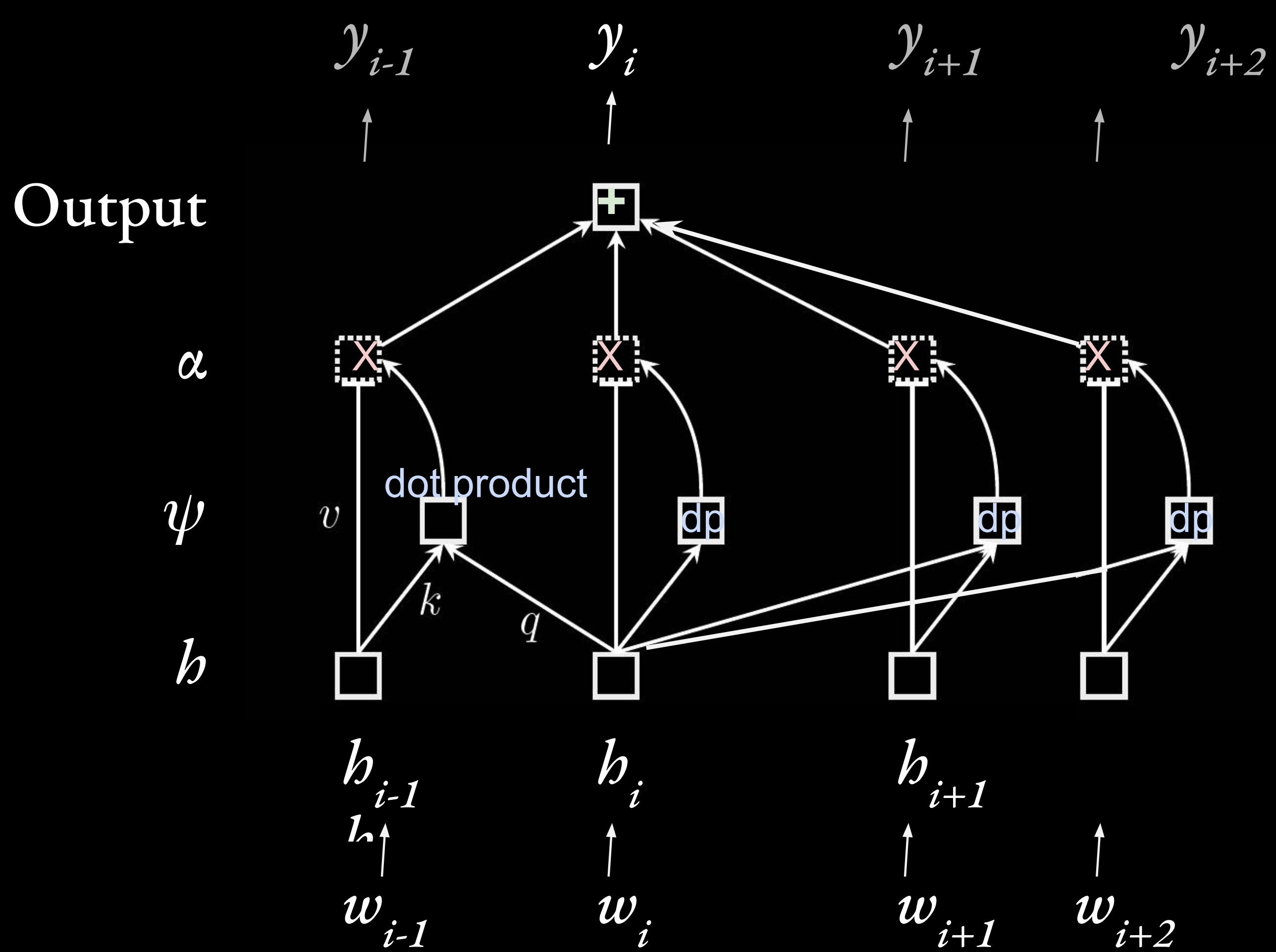
The Transformer: "Attention-only" models



$$\psi_{dp}(h_i, s) = s^T h_i$$

$k^t q$

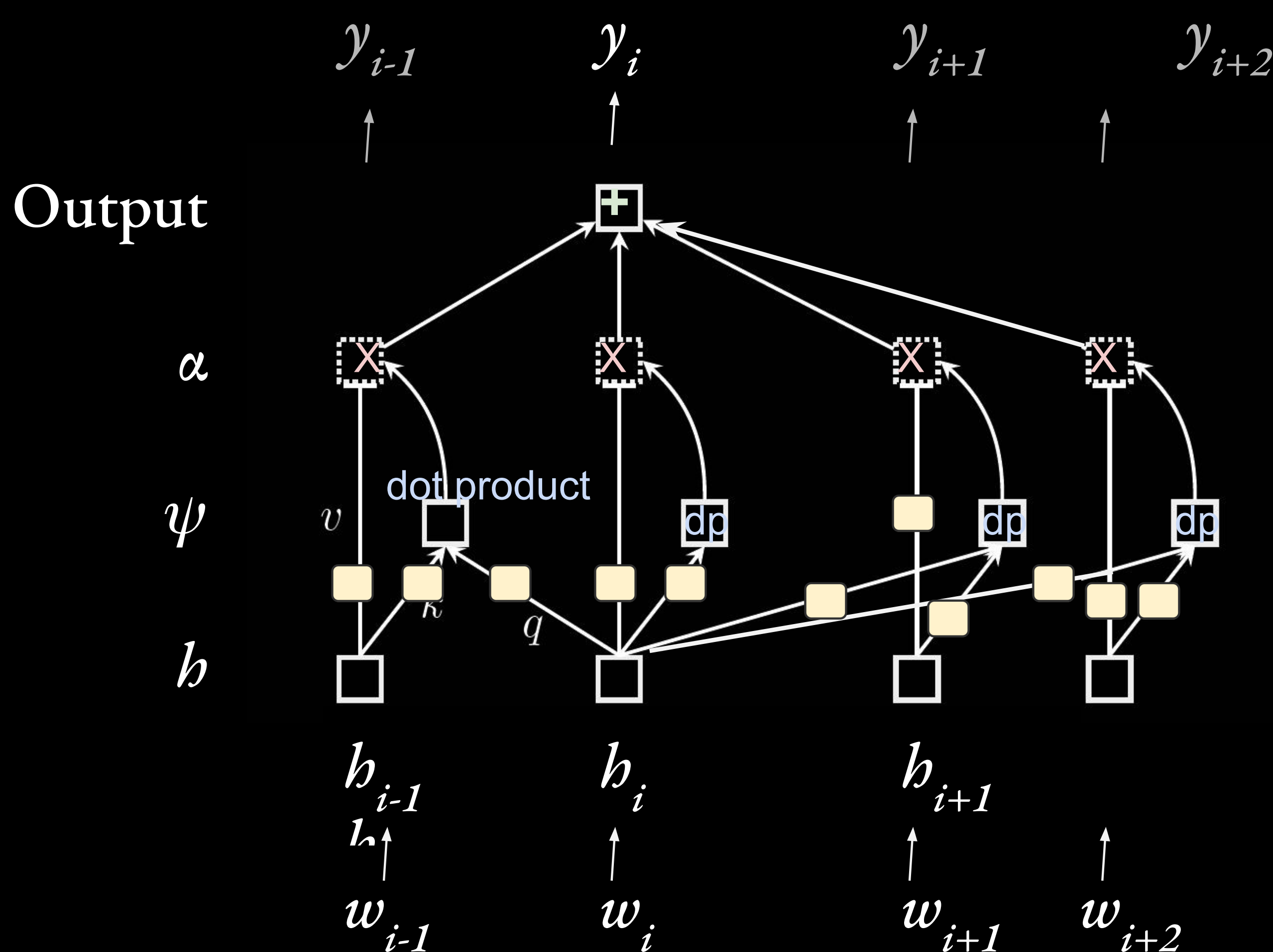
The Transformer: "Attention-only" models



scaling parameter

$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

The Transformer: “Attention-only” models



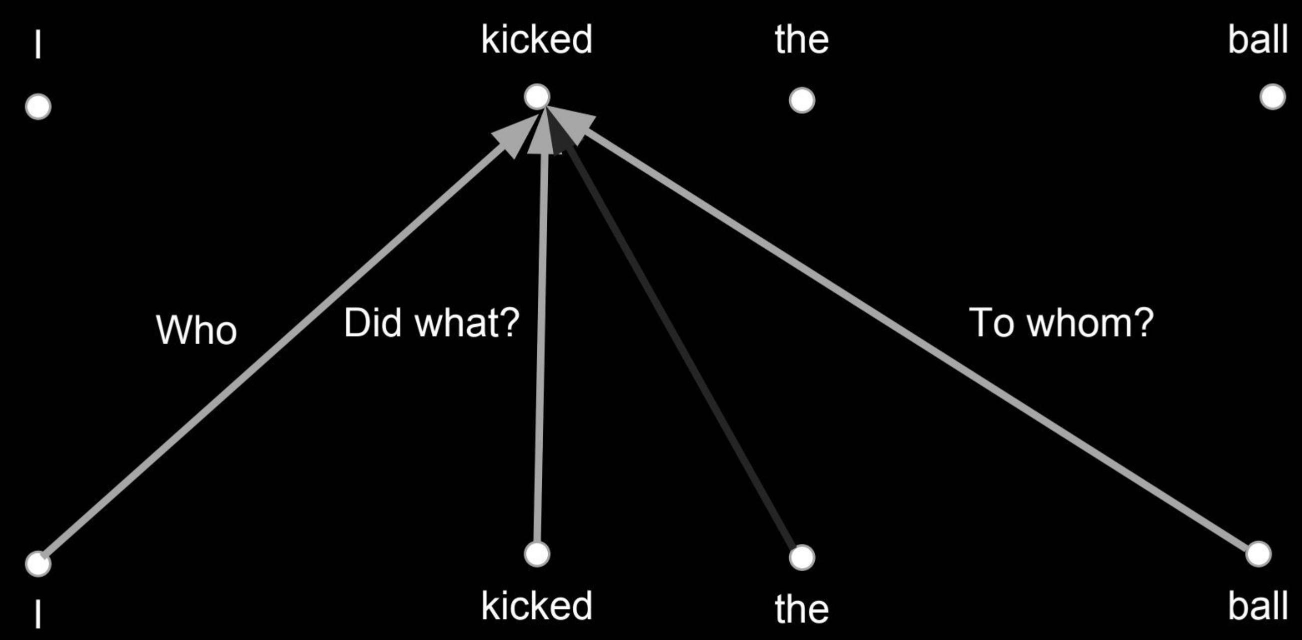
$$\psi_{dp}(k,q) = (k^t q)\,\sigma$$

Linear layer:
W^TX

One set of weights for each of for K, Q, and V

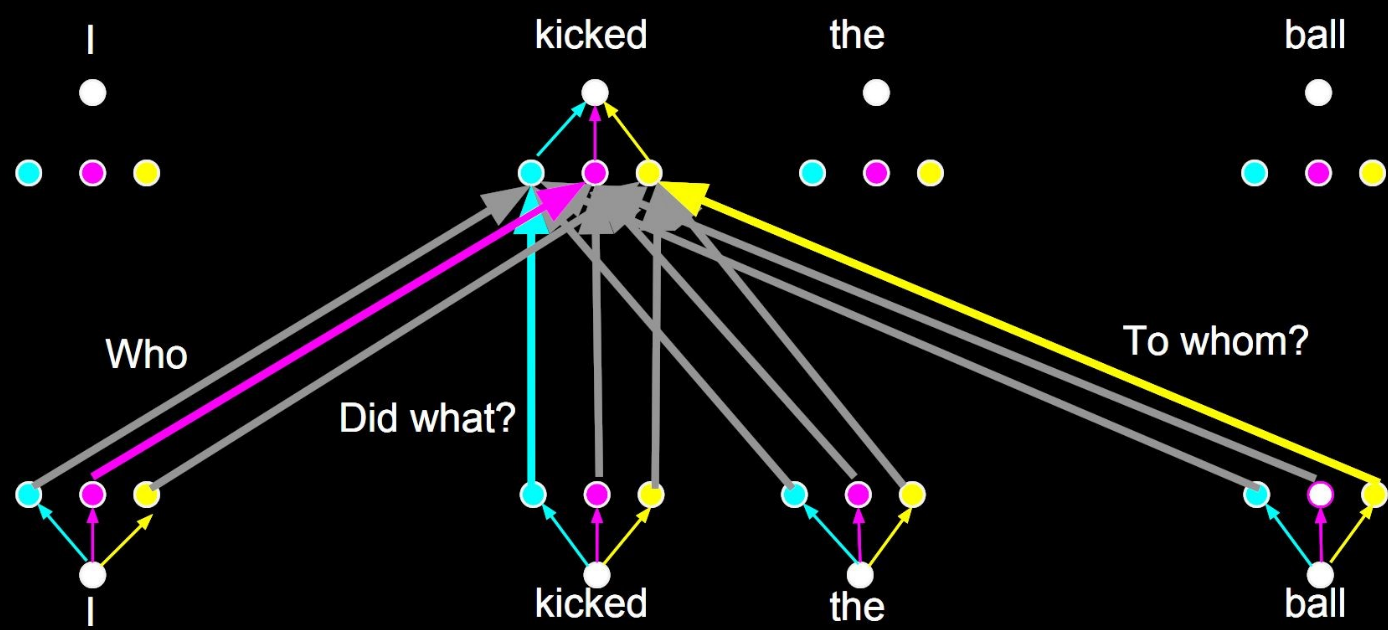
The Transformer

Limitation (thus far): Can't capture multiple types of dependencies between words.

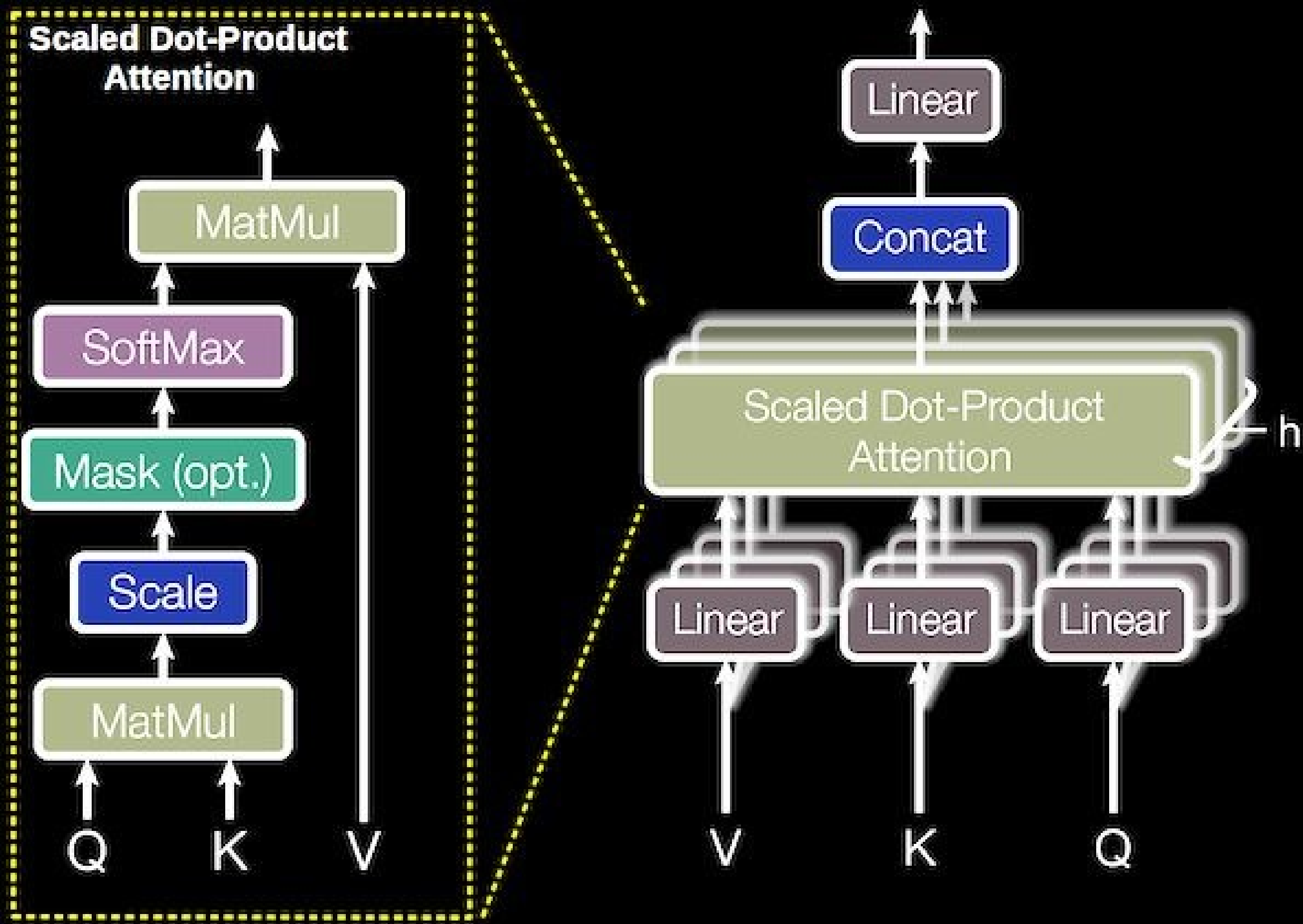


The Transformer

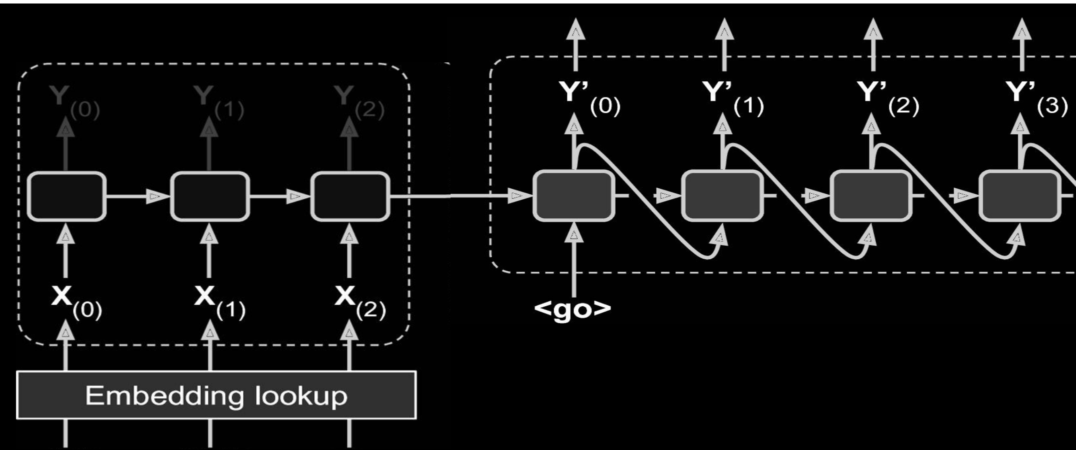
Solution: Multi-head attention

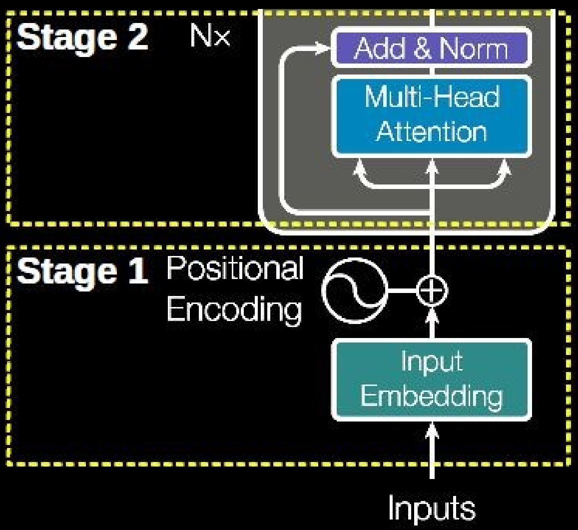


Multi-head Attention

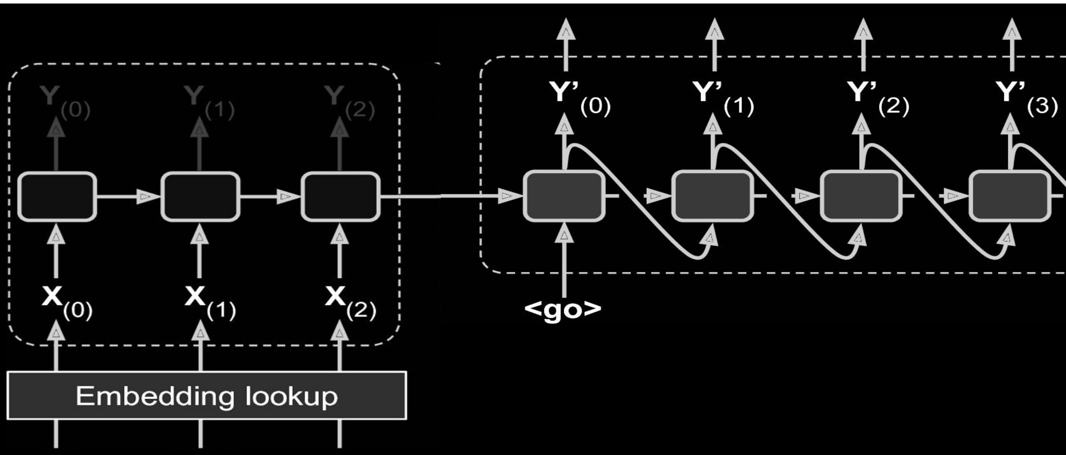


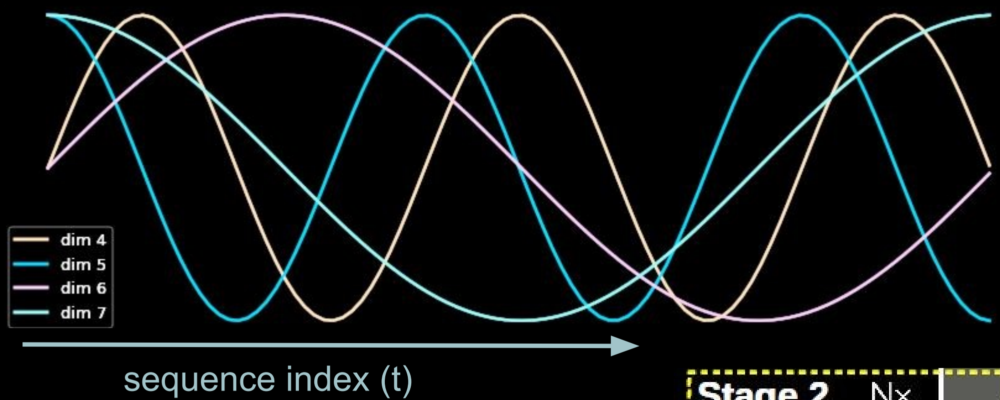
Transformer for Encoder-Decoder

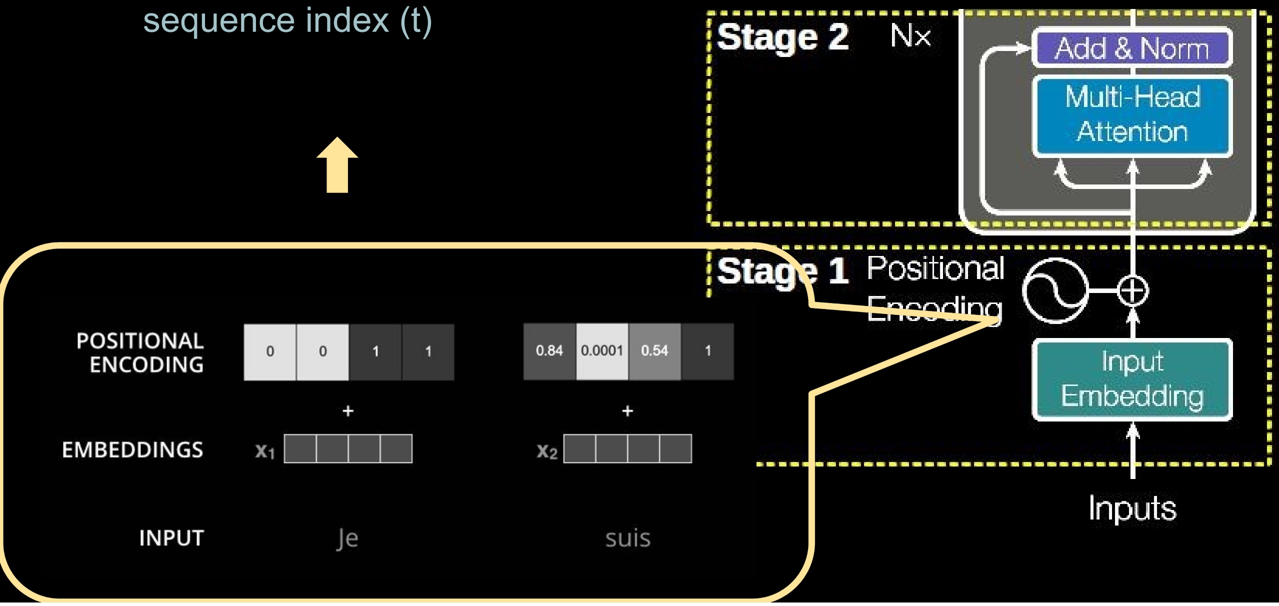




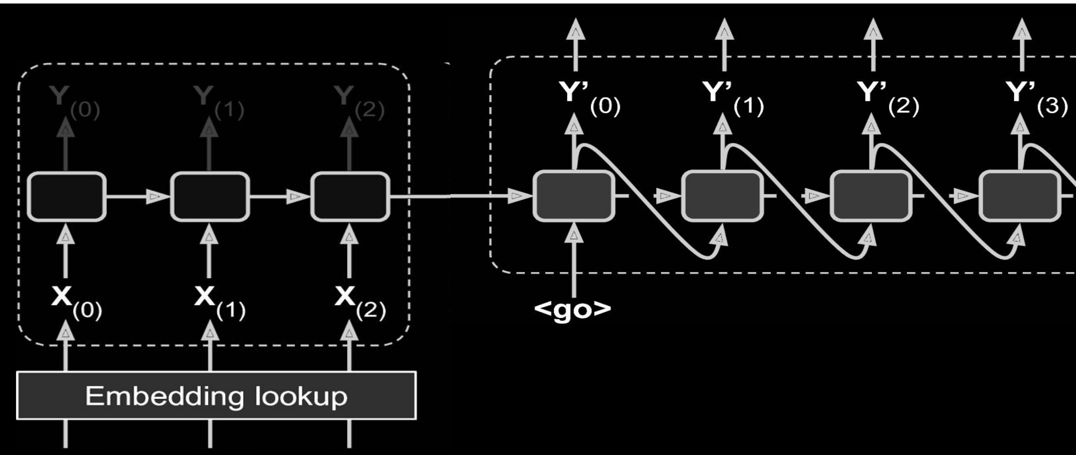
Transformer for Encoder-Decoder

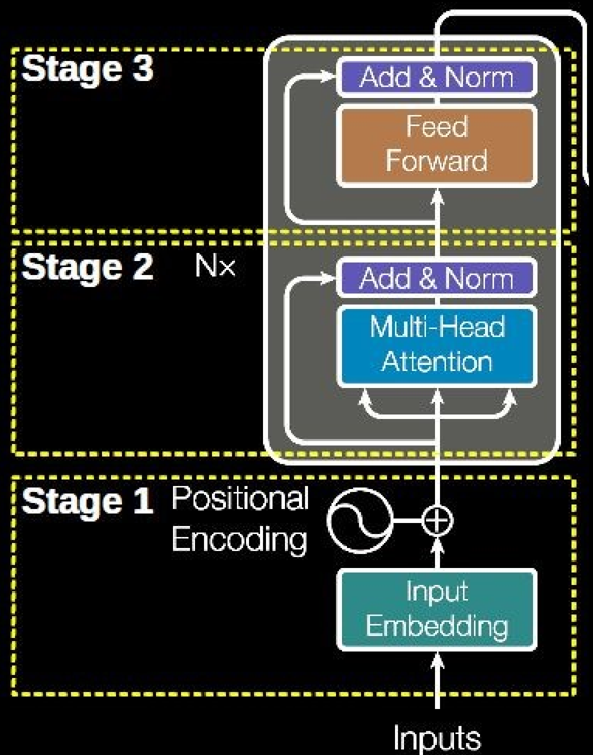




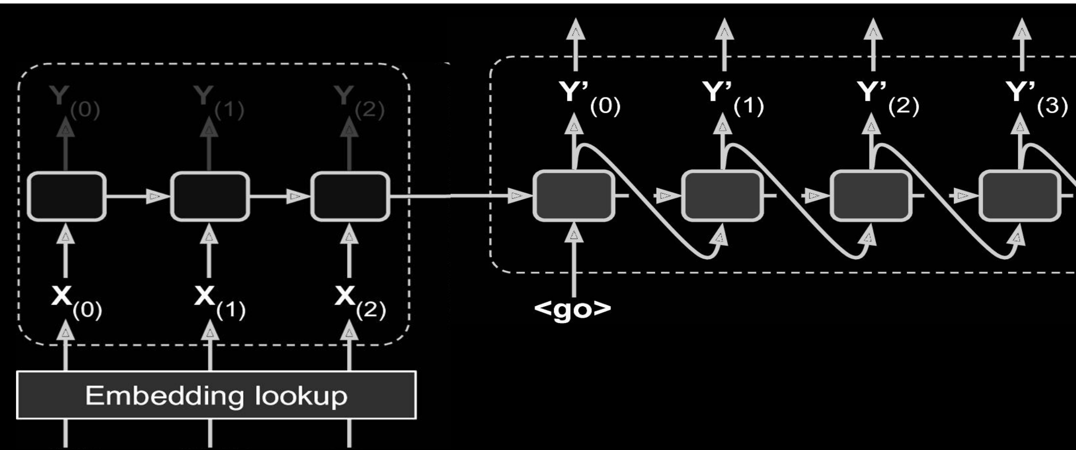


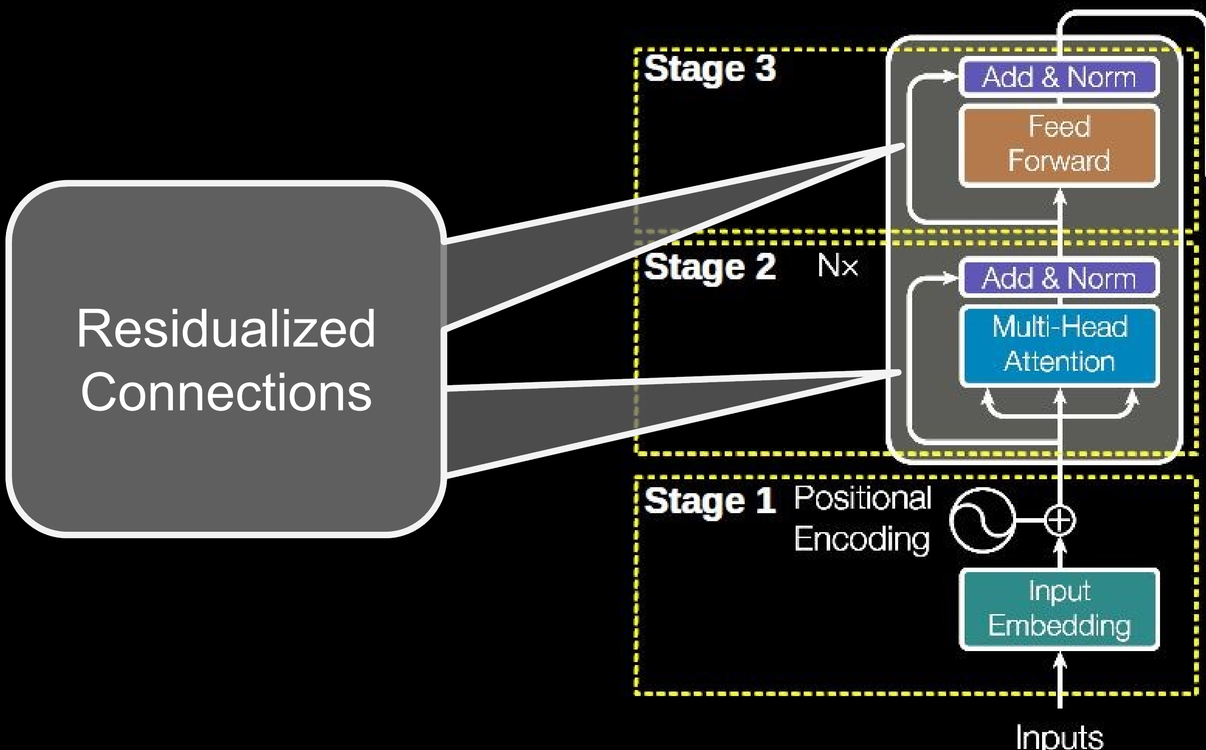
Transformer for Encoder-Decoder



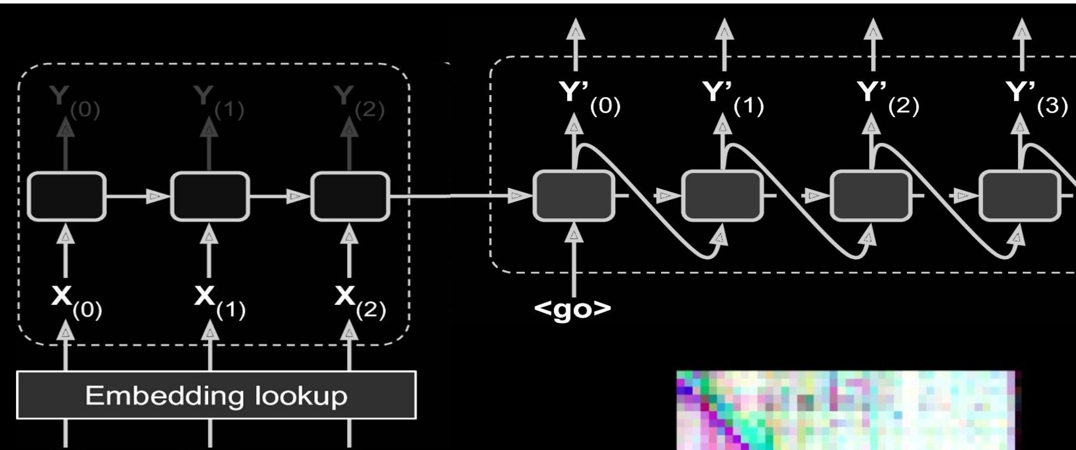


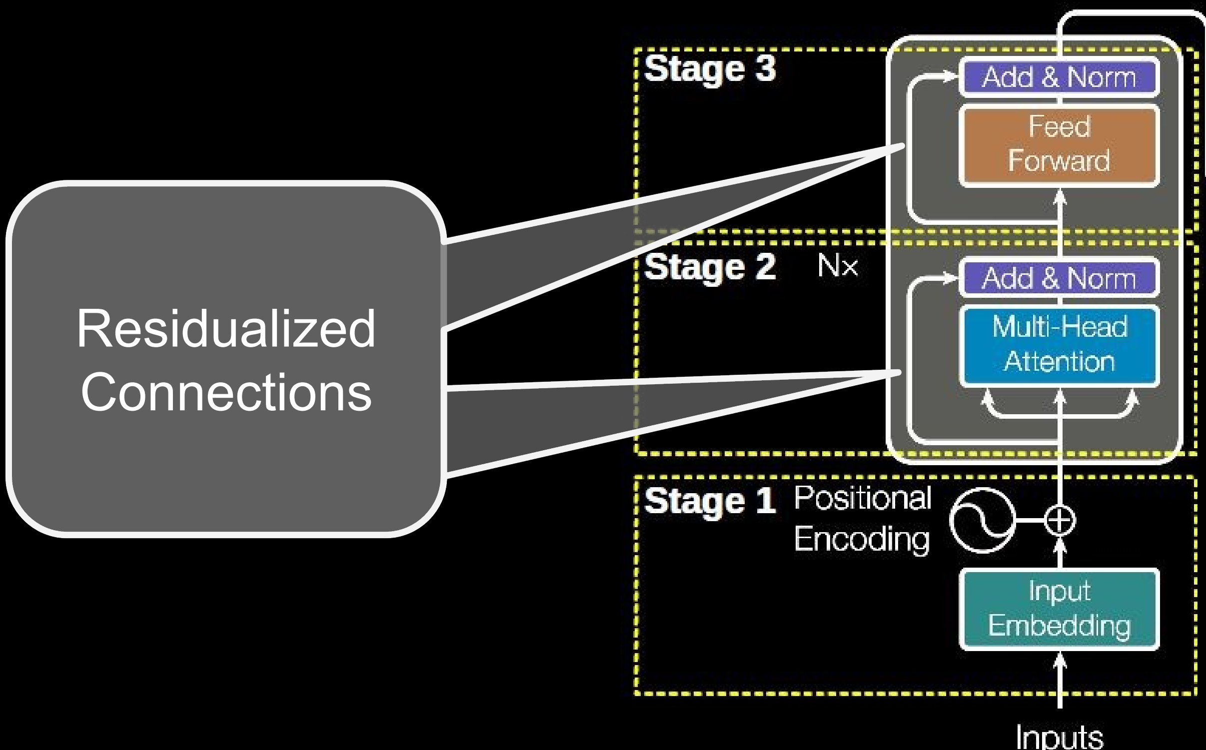
Transformer for Encoder-Decoder



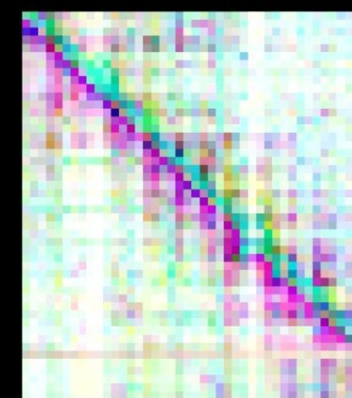


Transformer for Encoder-Decoder

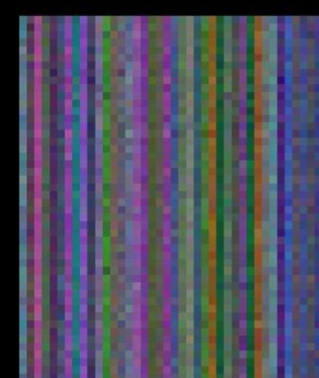




residuals enable positional information to be passed along

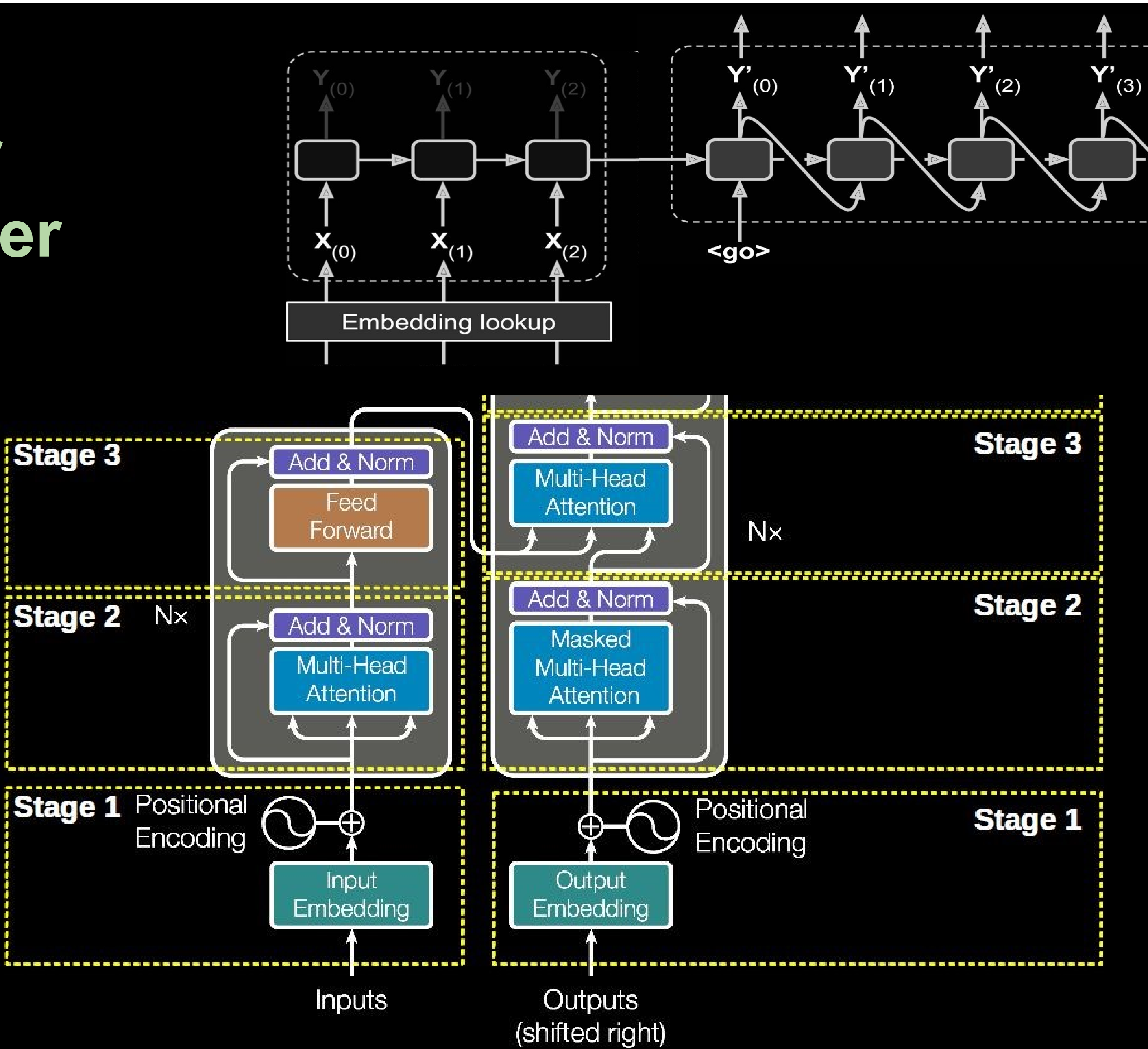


With residuals

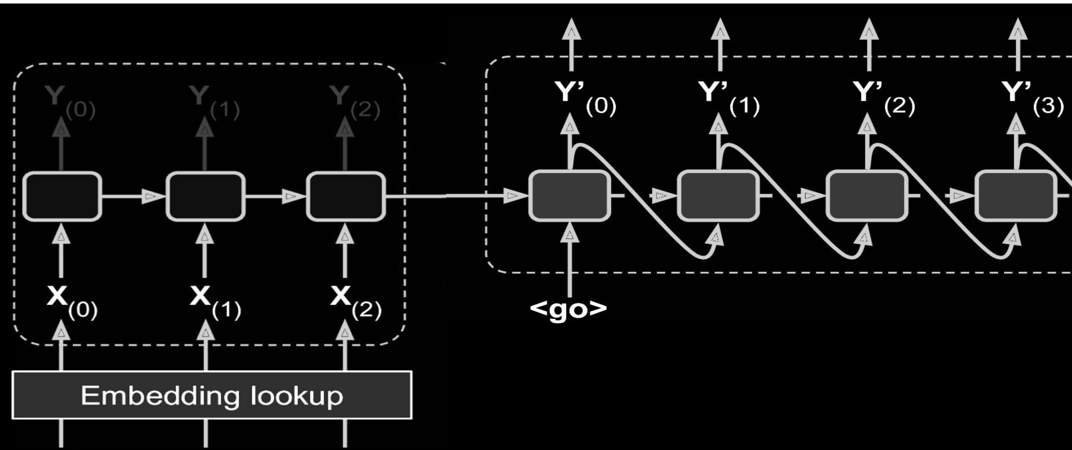


Without residuals

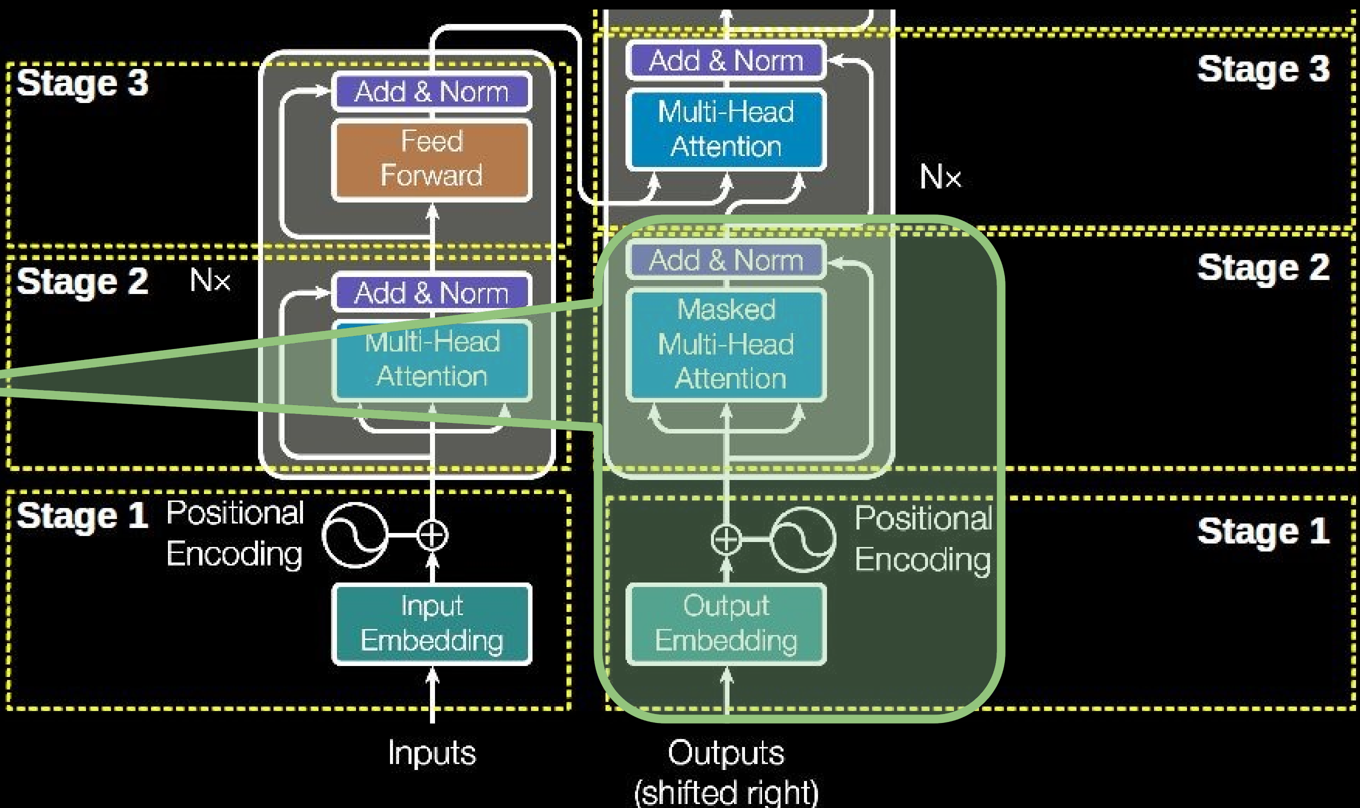
Transformer for Encoder-Decoder



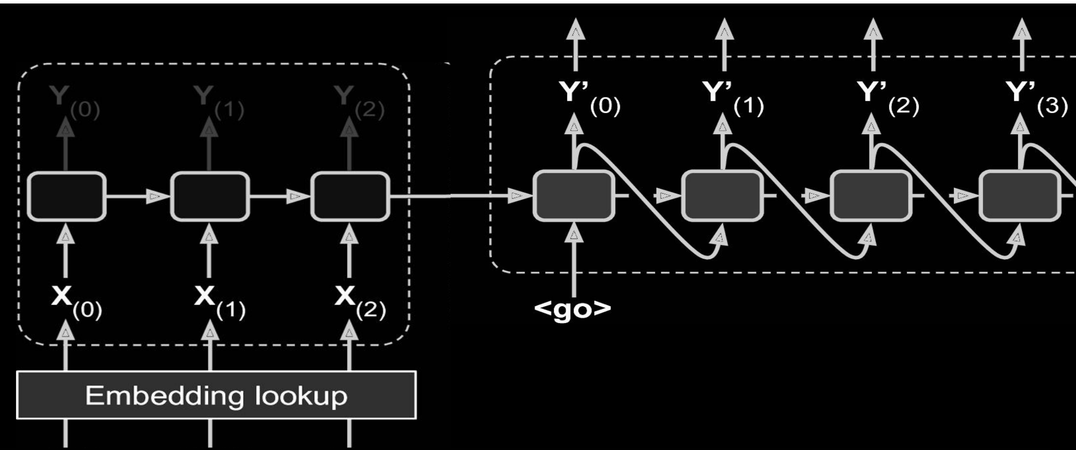
Transformer for Encoder-Decoder



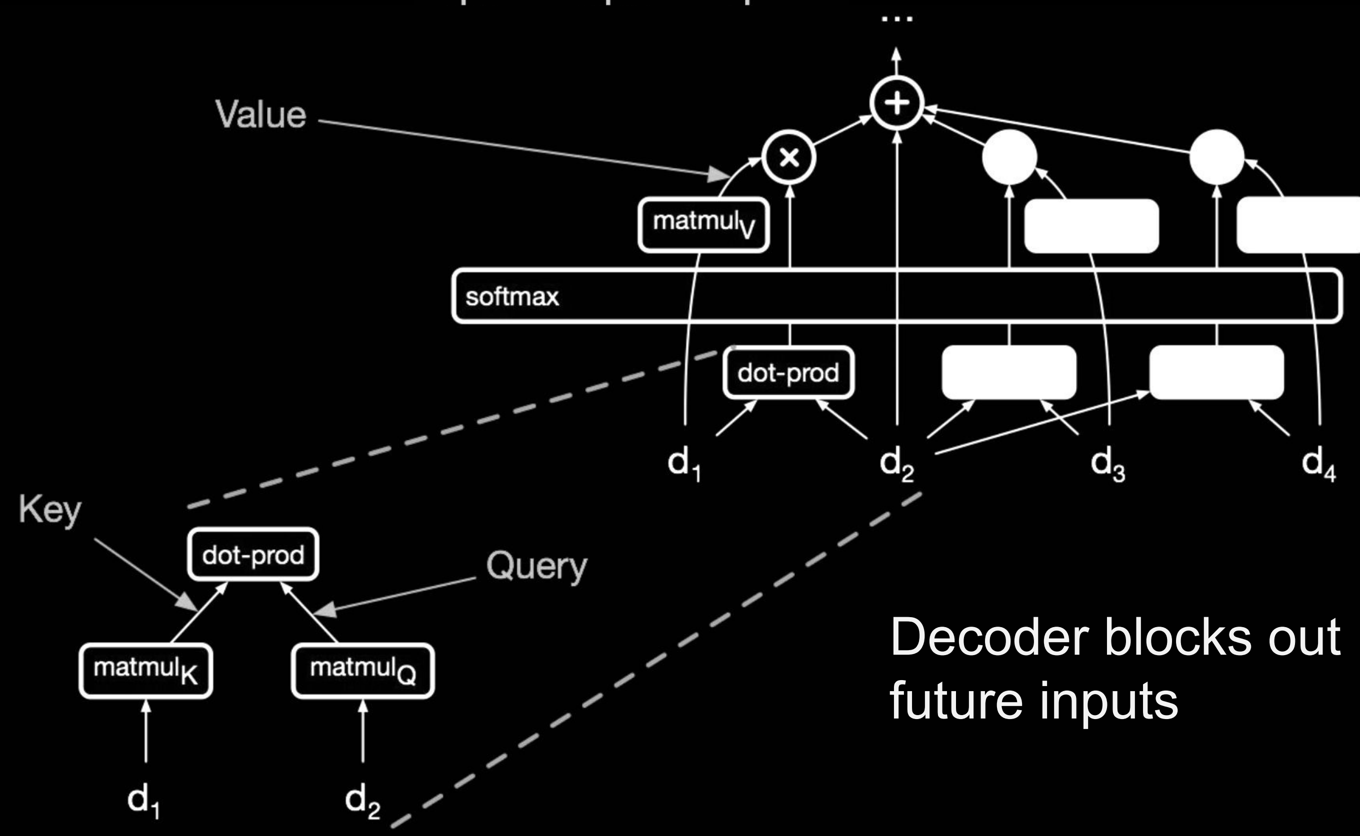
essentially, a language model



Transformer for Encoder-Decoder



essentially, a language model

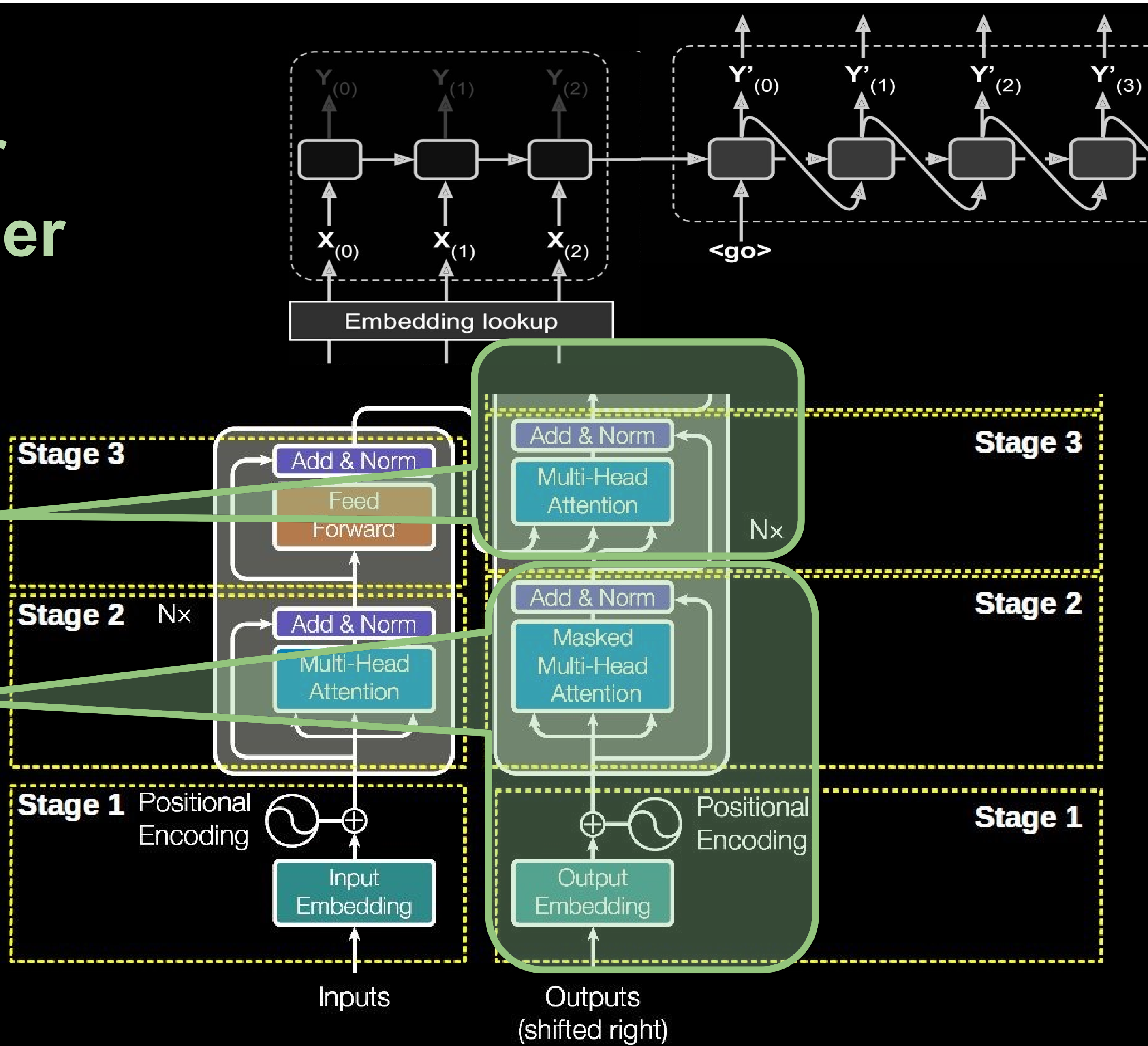


Decoder blocks out future inputs

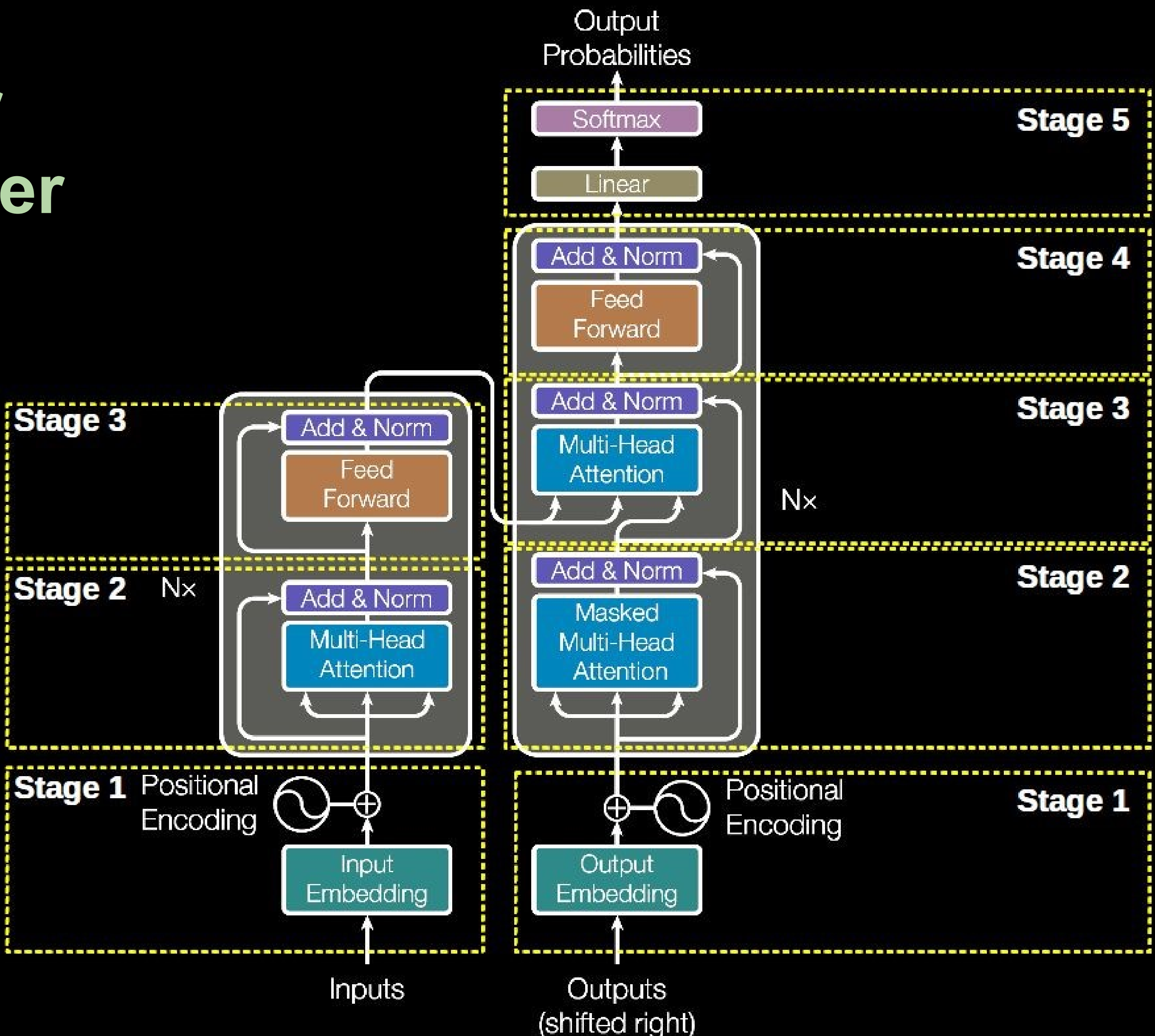
Transformer for Encoder-Decoder

Add conditioning of the LM based on the encoder

essentially, a language model



Transformer for Encoder-Decoder



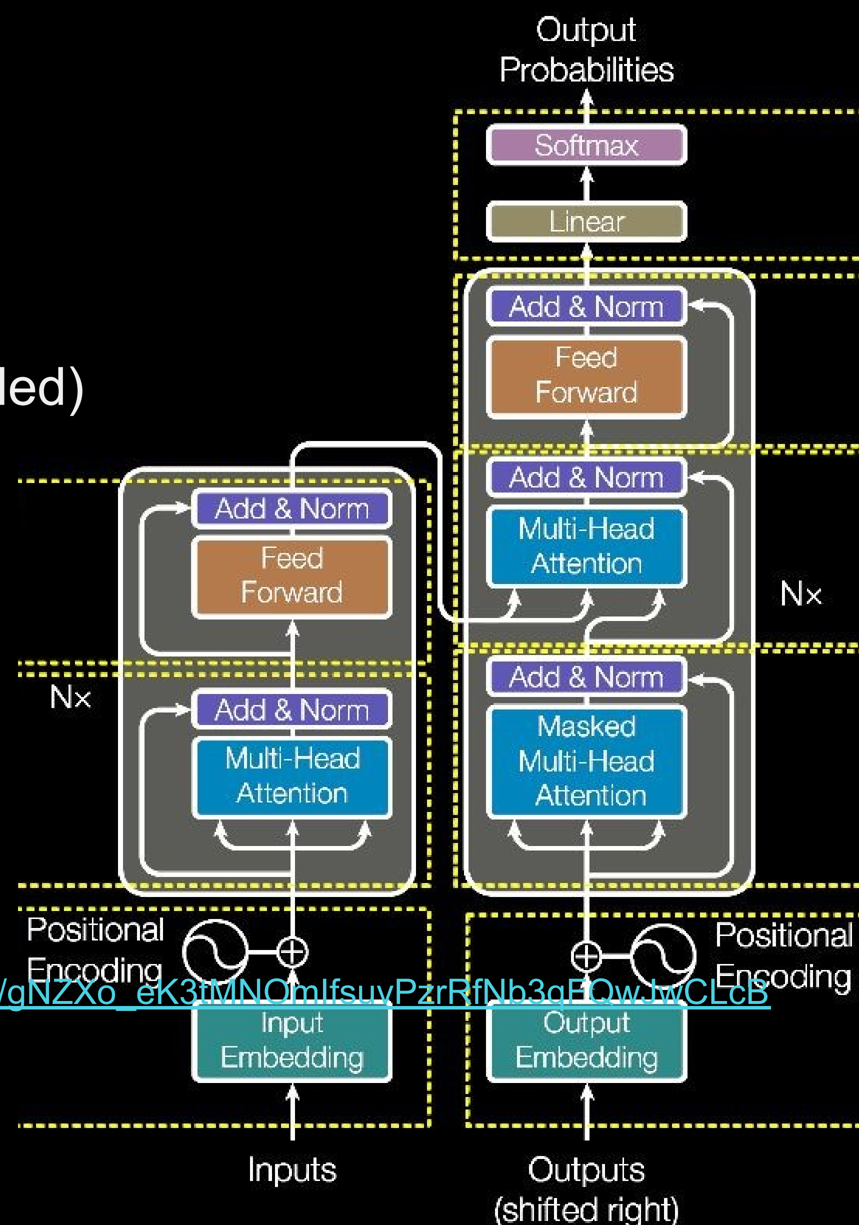
Transformer (as of 2017)

"WMT-2014" Data Set. BLEU scores:

	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

Transformer

- Utilize Self-Attention
- Simple att scoring function (dot product, scaled)
- Added linear layers for Q, K, and V
- Multi-head attention
- Added positional encoding
- Added residual connection
- Simulate decoding by masking



https://4.bp.blogspot.com/-OlrV-PAtEkQ/W3RkOJCBkaI/AAAAAAAADOg/gNZXo_eK3tMNOmIfsuyPzrRfNb3qFQwJwCLcBGAs/s640/image1.gif

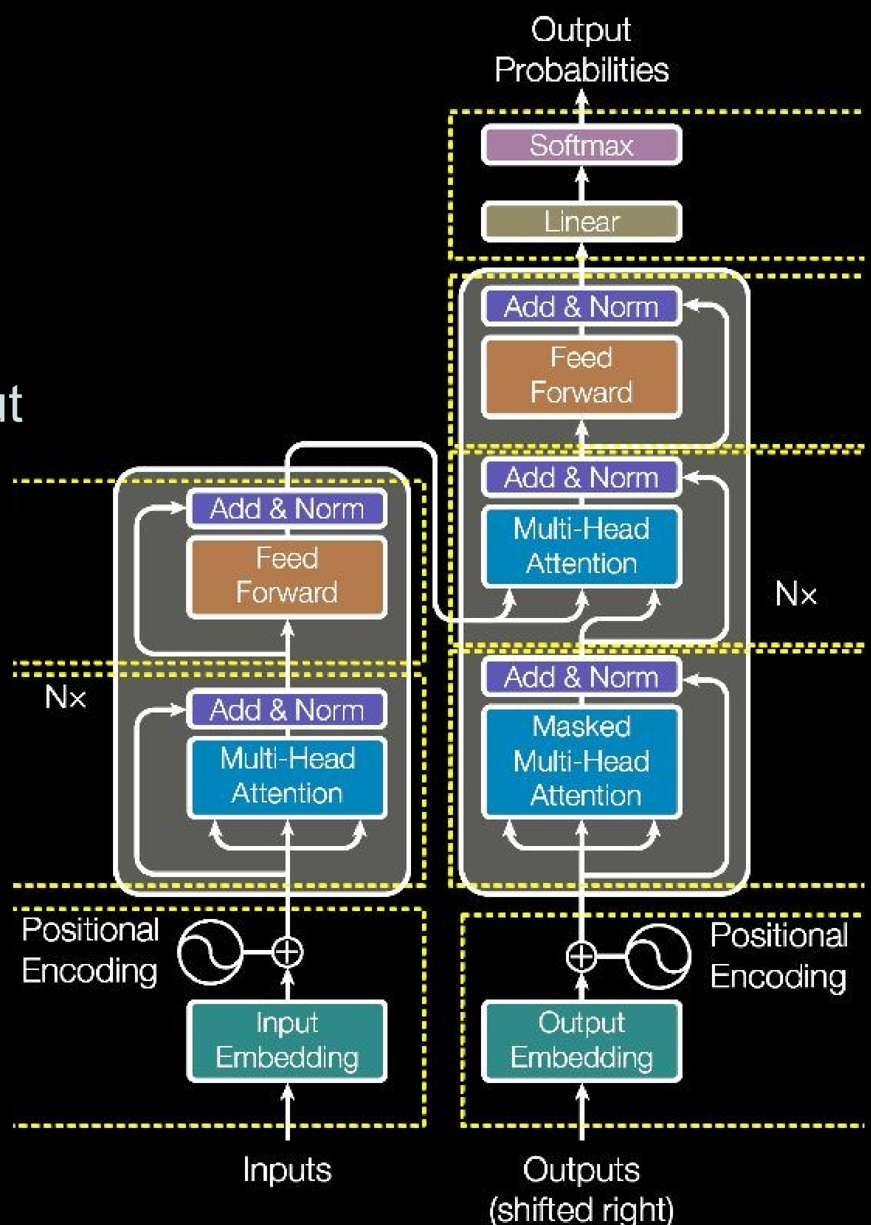
Transformer

Why?

- Don't need complexity of LSTM/GRU cells
- Constant num edges between words (or input steps)
- Enables "interactions" (i.e. adaptations) between words
- Easy to parallelize -- don't need sequential processing.

Drawbacks:

- Only unidirectional by default
- Only a "single-hop" relationship per layer (multiple layers to capture multiple)



BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

Produces contextualized embeddings
(or pre-trained contextualized encoder)

Drawbacks of Vanilla Transformers:

- Only unidirectional by default
- Only a “single-hop” relationship per layer
  (multiple layers to capture multiple)

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

Produces contextualized embeddings
(or pre-trained contextualized encoder)

- Bidirectional context by “masking” in the middle
- A lot of layers, hidden states, attention heads.

Drawbacks of Vanilla Transformers:

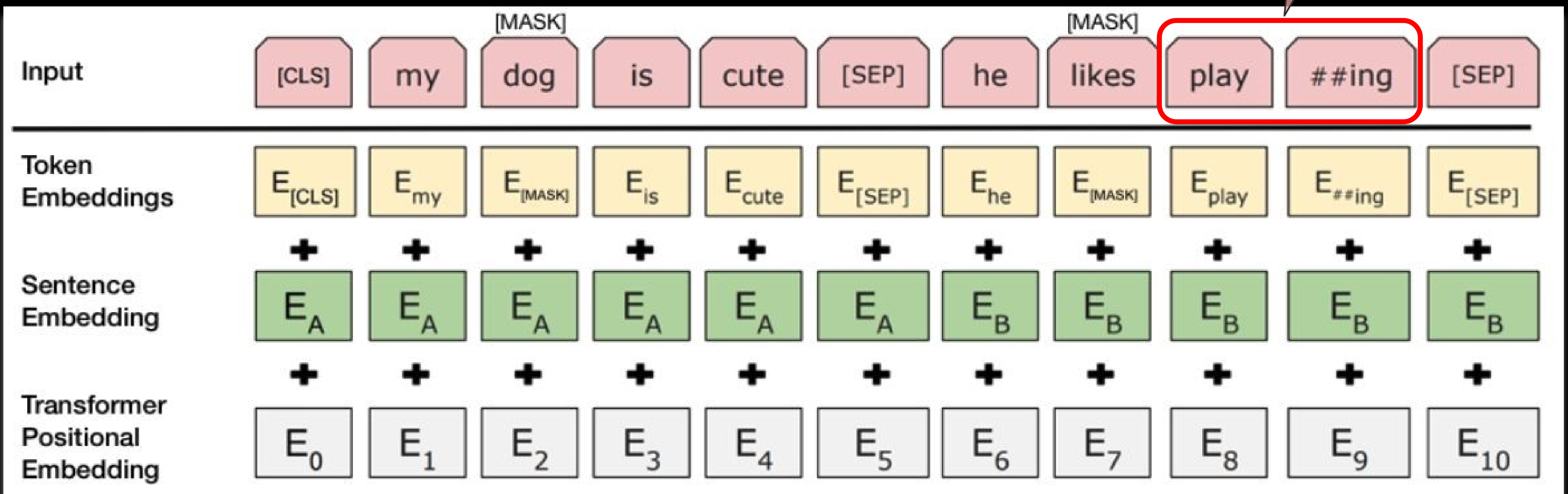
- Only unidirectional by default
- Only a “single-hop” relationship per layer
(multiple layers to capture multiple)

BERT

tokenize into “word pieces”

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

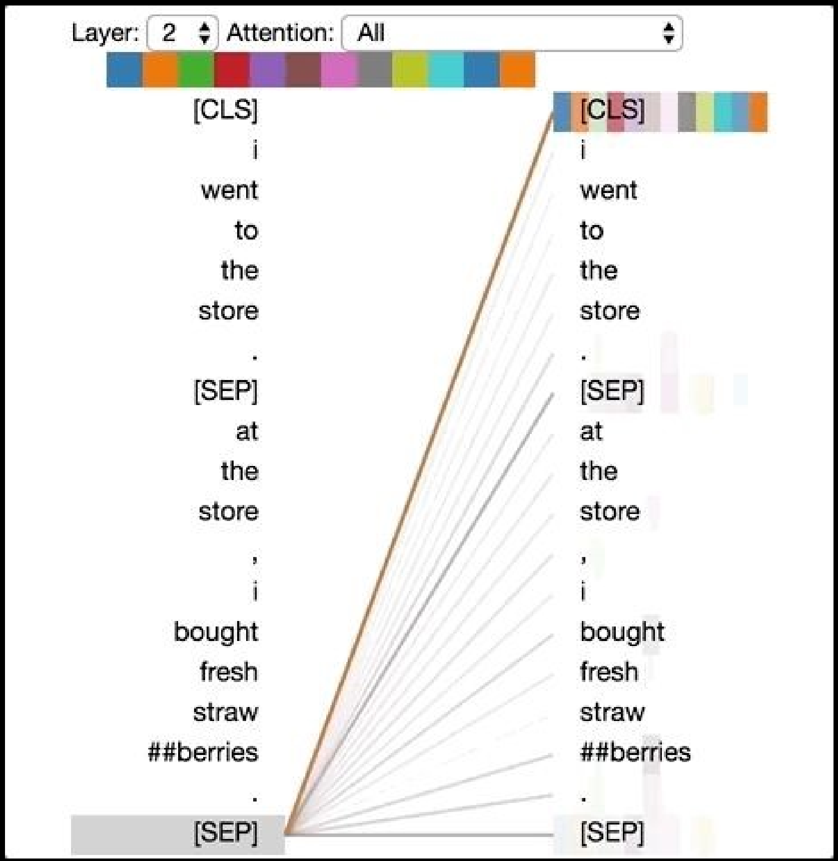
Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence



(Devlin et al., 2019)

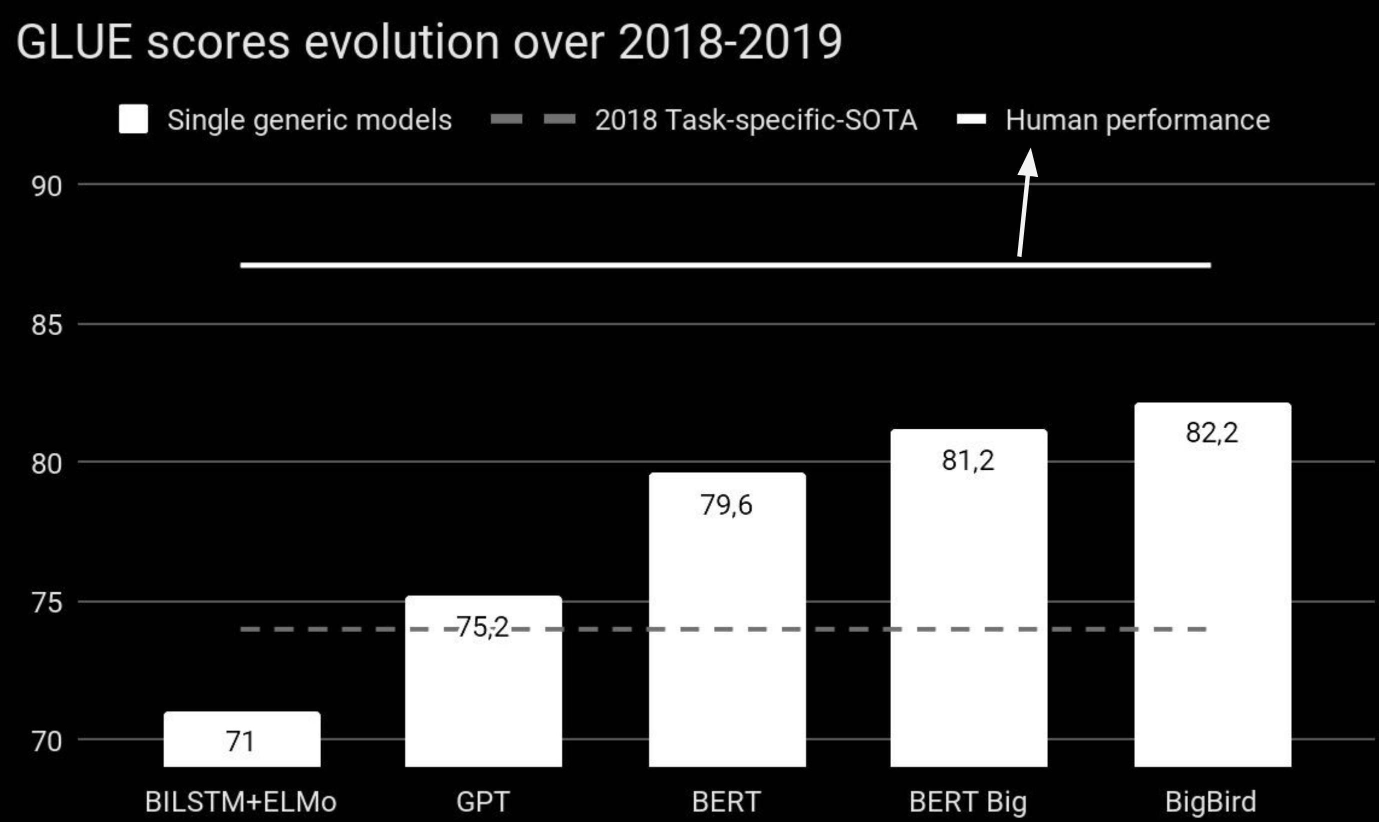
Bert: Attention by Layers

https://colab.research.google.com/drive/1vlOJ1lhdujVjfH857hvYKIdKPTD9Kid8



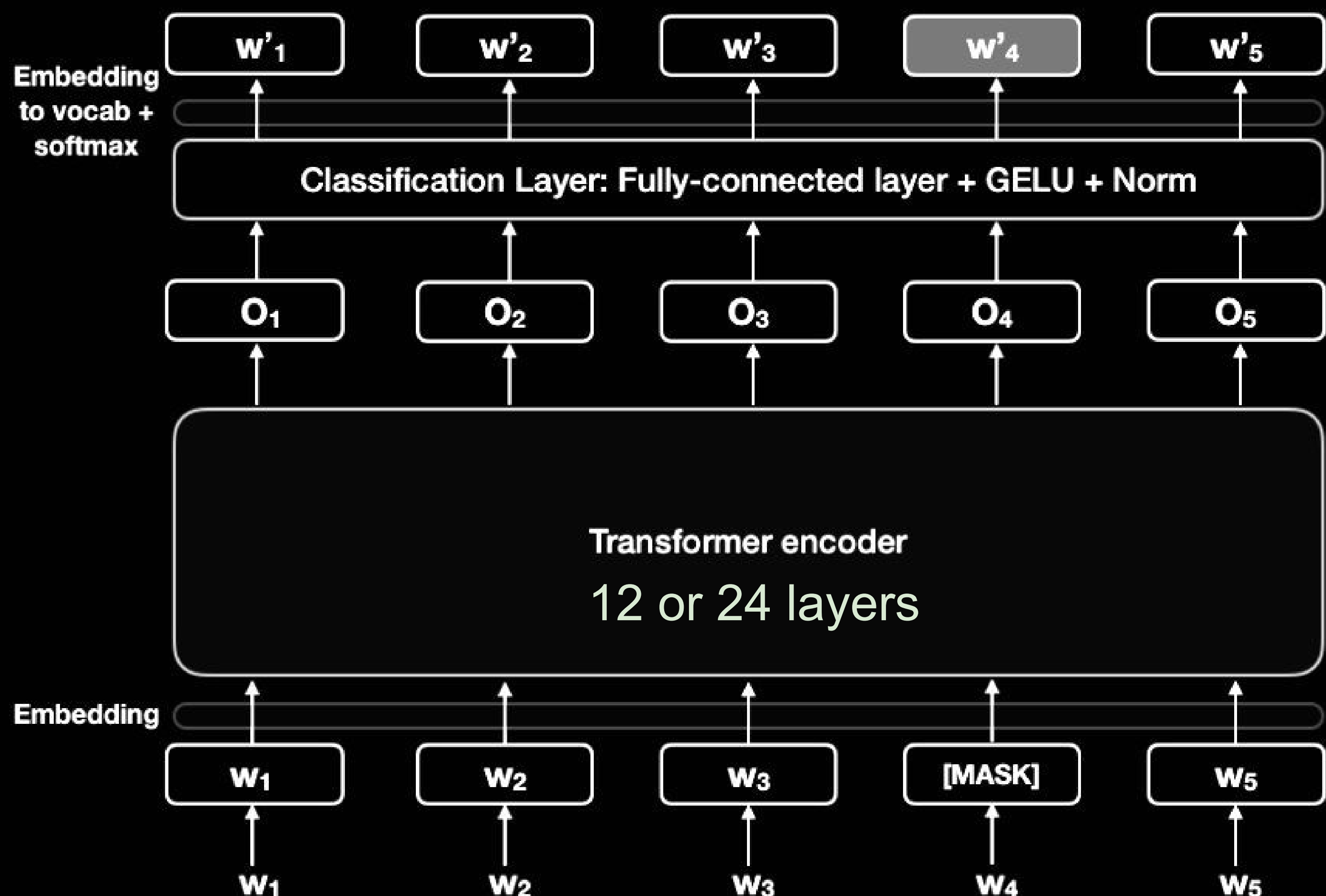
(Vig, 2019)

BERT Performance: e.g. Question Answering

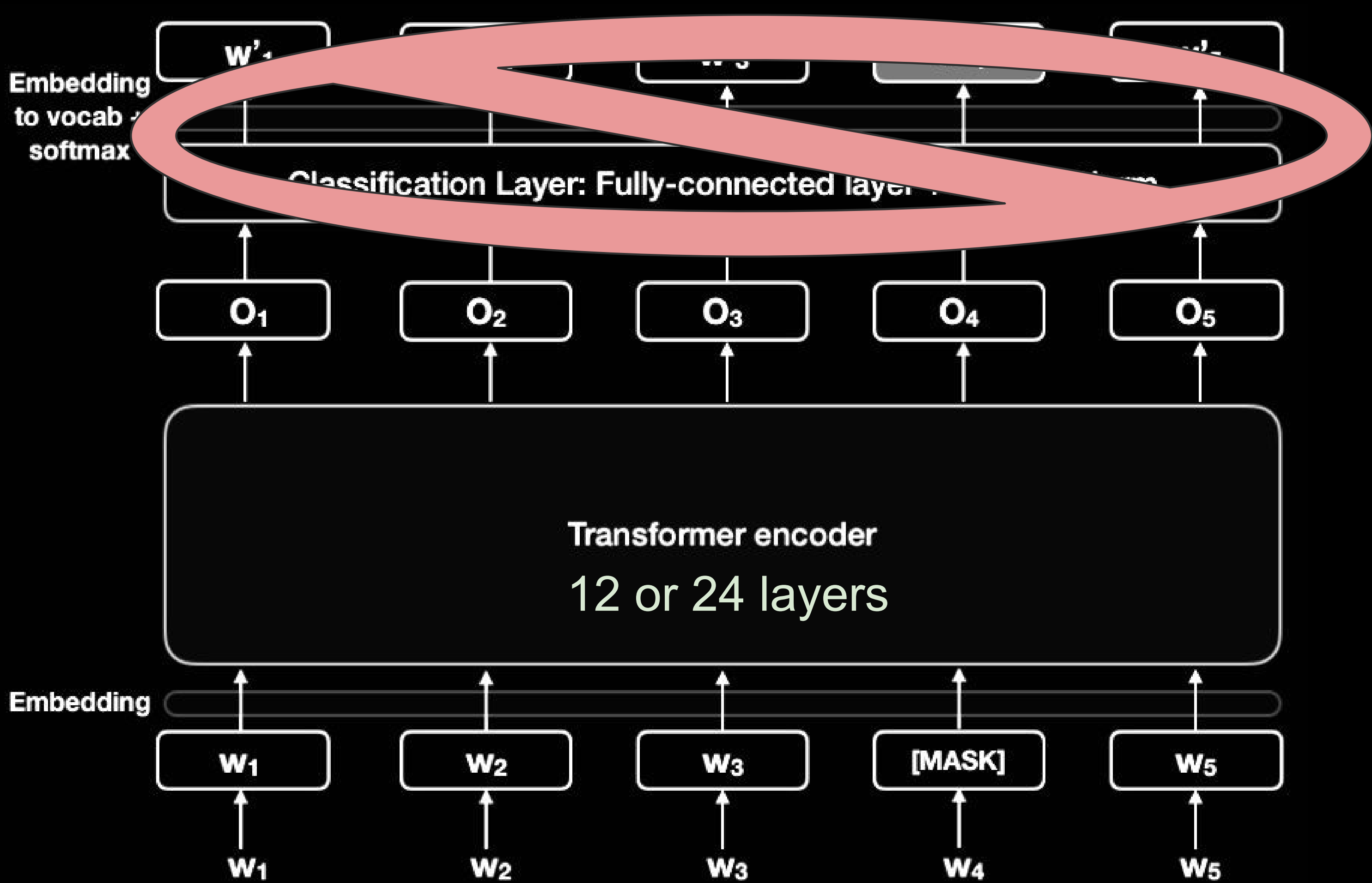


https://rajpurkar.github.io/SQuAD-explorer/

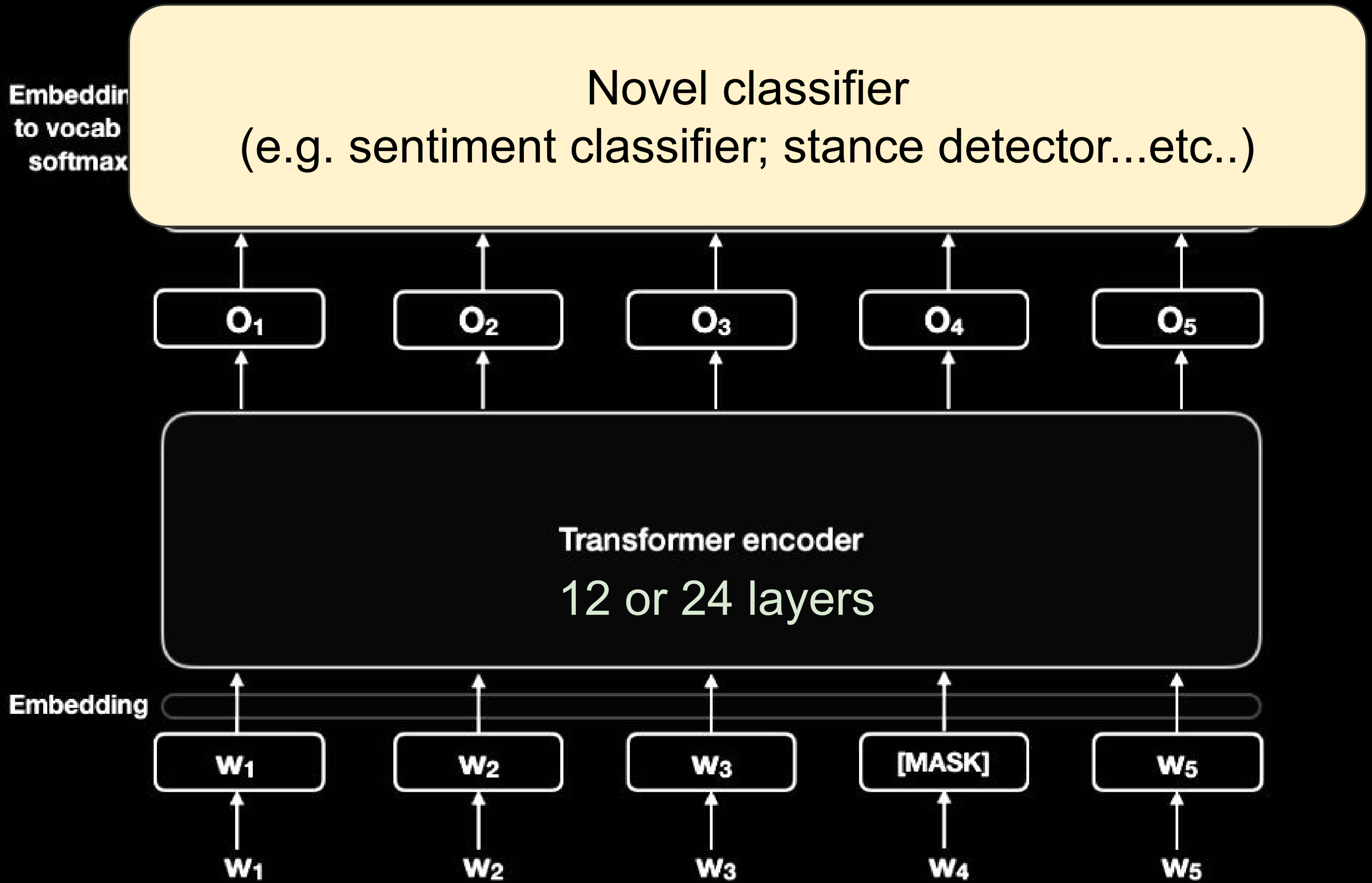
BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



BERT: Pre-training; Fine-tuning



Summary

- Goal is accurate prediction of y (outcome) given features (x)
- Use L1 or L2 penalization (as a regularization) to avoid overfit
- Reason for Train, Dev, Test split
- Components of a neural network
- Batch Normalization
- Distribution options: why is data parallelism preferred?
- Recurrent Neural Network
- Convolution Operation with Filters