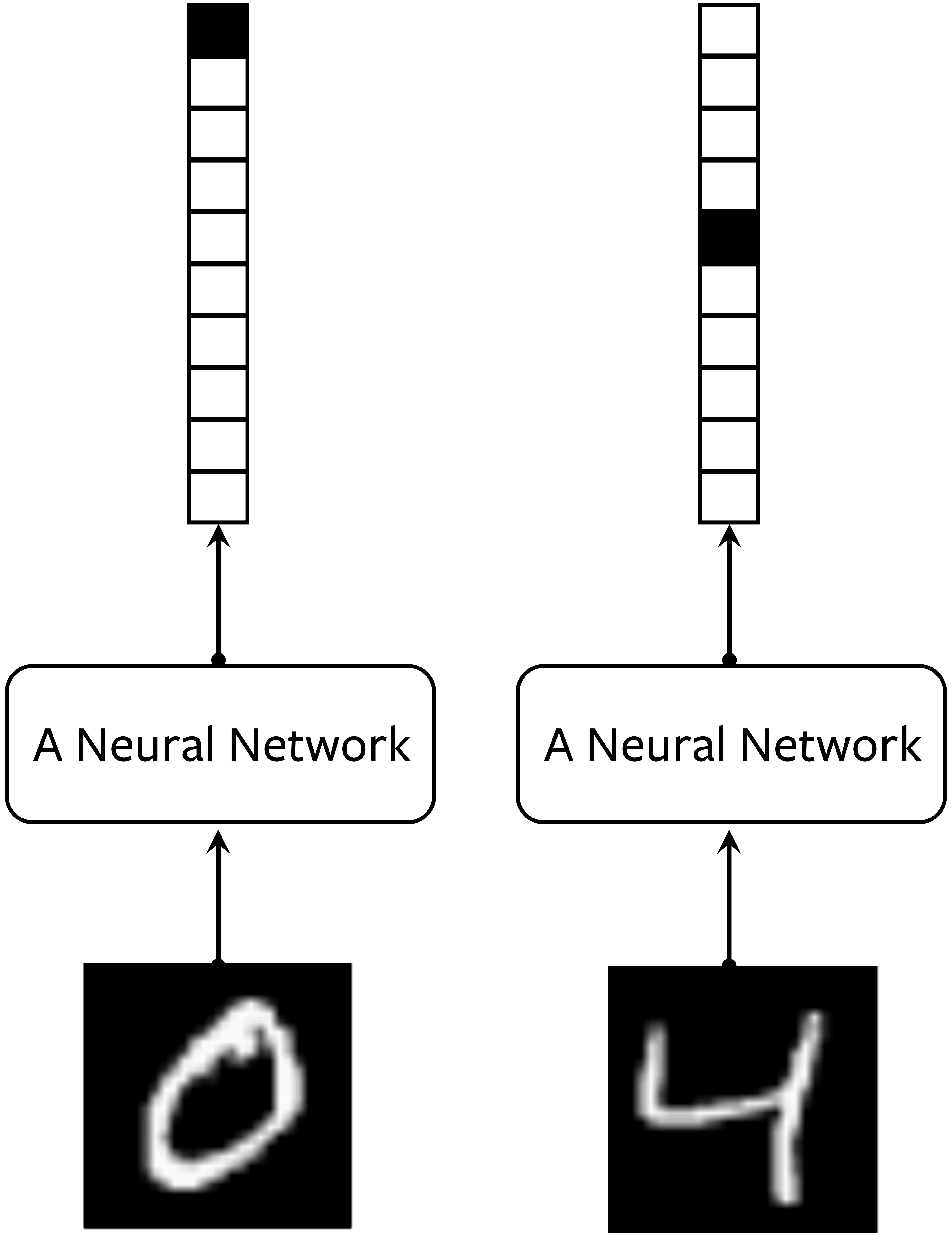
End-to-end approaches for Speech Recognition

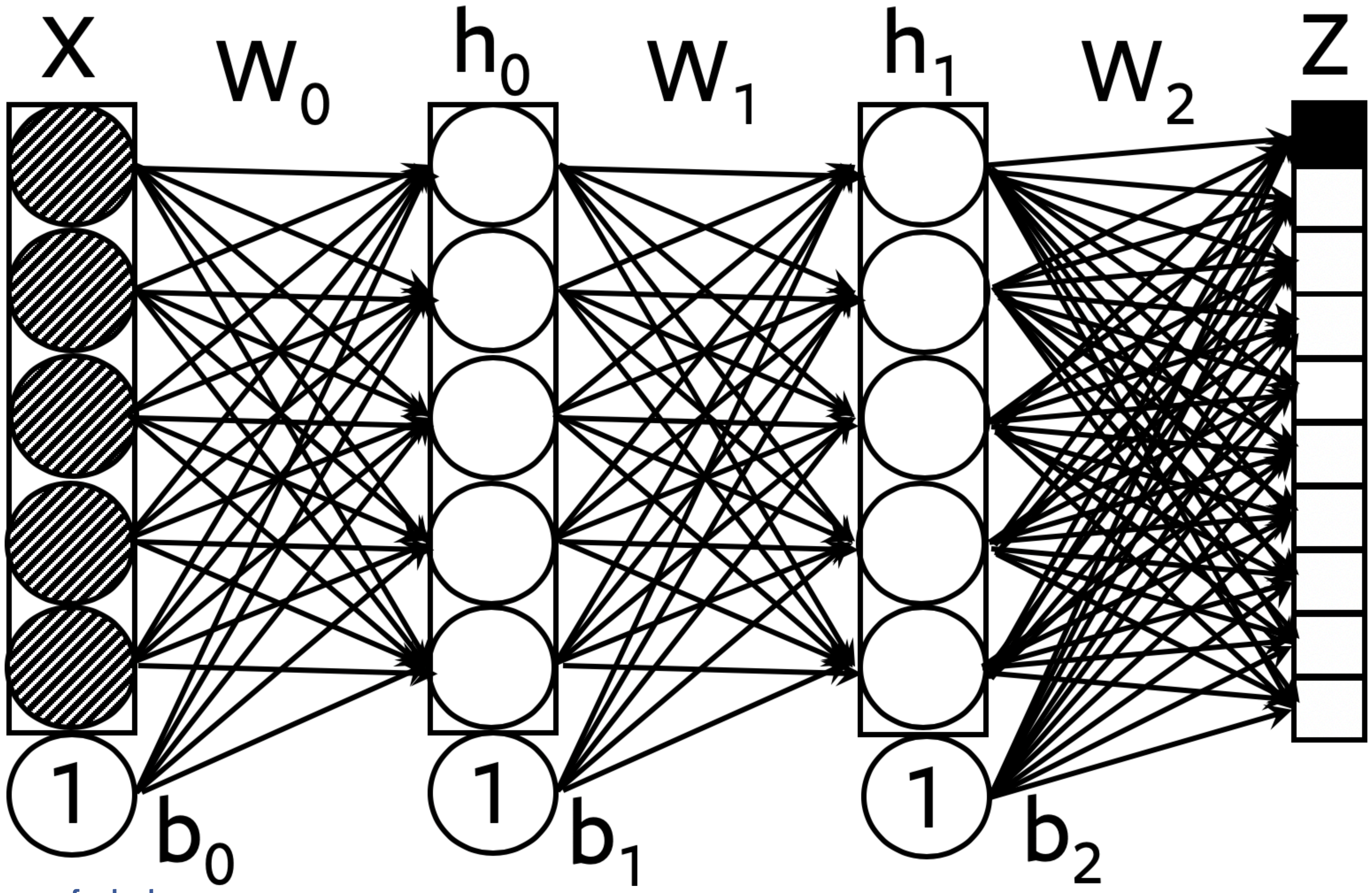
Abdelrahman Mohamed
Facebook AI Research (FAIR)

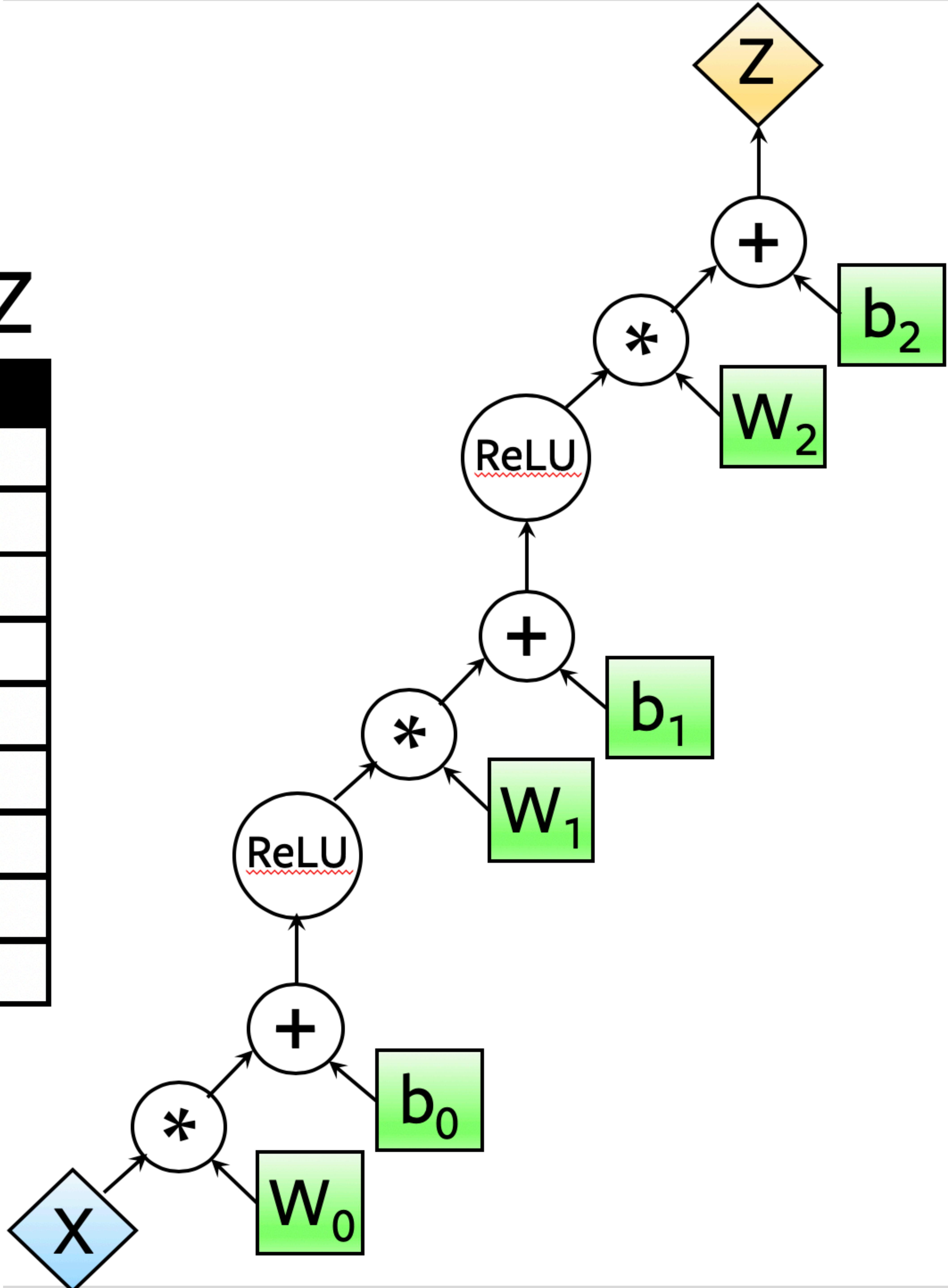
Simple Digit Classification

- MNIST digit classification dataset
- Classify each input into one of 10 classes (1-of-K)
- Complex relationships between input images and output classes.
- For Neural Networks to model such relationships, they require large training data of (input, output) pairs.



Neural Networks in 5 slides (1/5)





facebook
Artificial Intelligence Research

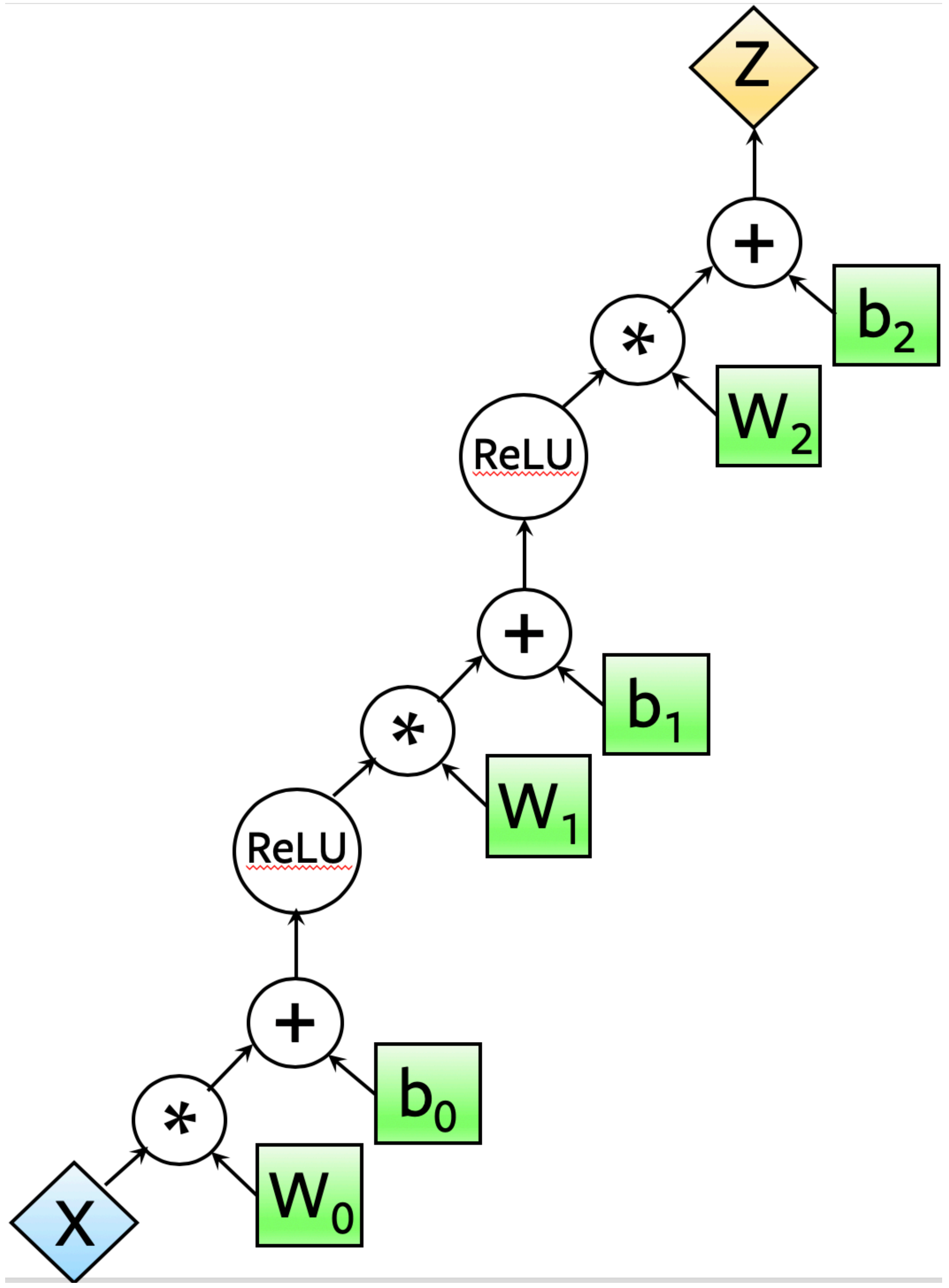
Neural Networks in 5 slides (2/5)

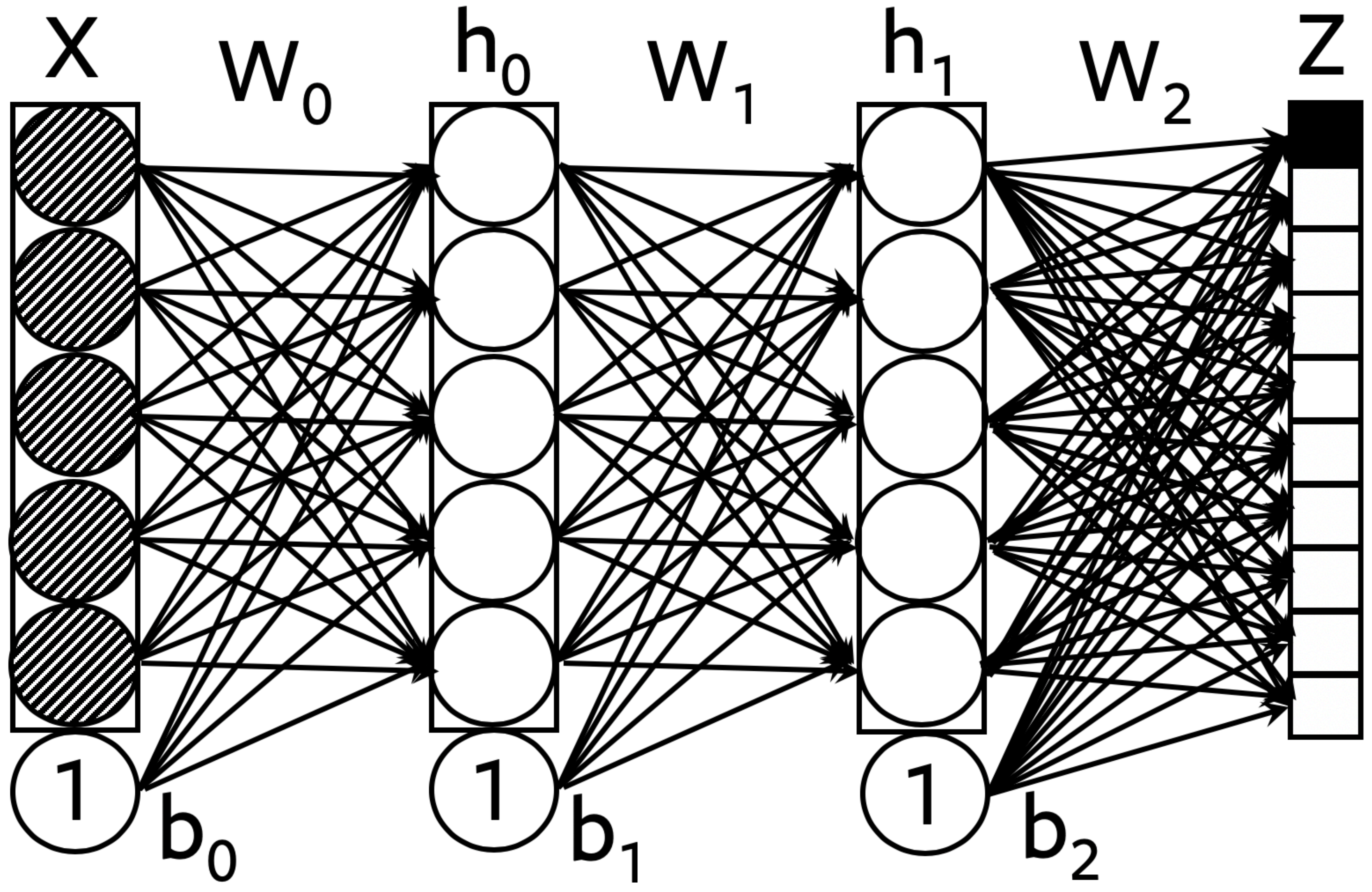
- A Neural Network represents a composition of many differentiable functions.

$$Z(X;\theta) = W_2 * ReLU(W_1 * ReLU(W_0 * X + b_0) + b_1) + b_2$$

$$where\ \theta = \{W_0, b_0, W_1, b_1, W_2, b_2\}$$

- NNs are able to model nonlinear relationships between inputs and outputs.
- Rectified Linear Units (ReLU) cause the final decision boundary to be piecewise linear.



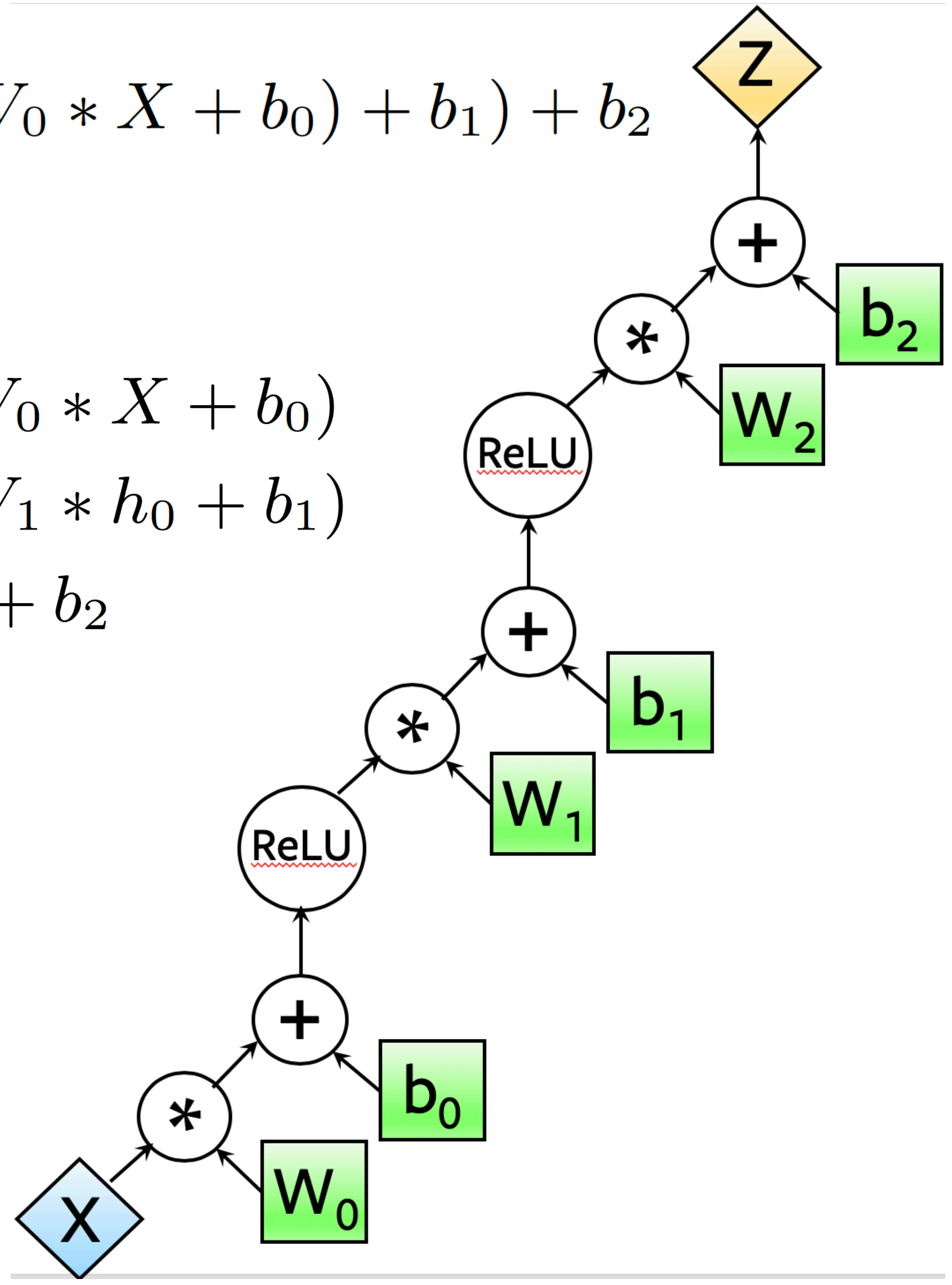


$$h_0 = ReLU(W_0 * X + b_0)$$
$$h_1 = ReLU(W_1 * h_0 + b_1)$$
$$Z = W_2 * h_1 + b_2$$

Neural Networks in 5 slides (3/5)

$$Z(X;\theta) = W_2 * ReLU(W_1 * ReLU(W_0 * X + b_0) + b_1) + b_2$$
$$where\ \theta = \{W_0, b_0, W_1, b_1, W_2, b_2\}$$

$$h_0 = ReLU(W_0 * X + b_0)$$
$$h_1 = ReLU(W_1 * h_0 + b_1)$$
$$Z = W_2 * h_1 + b_2$$



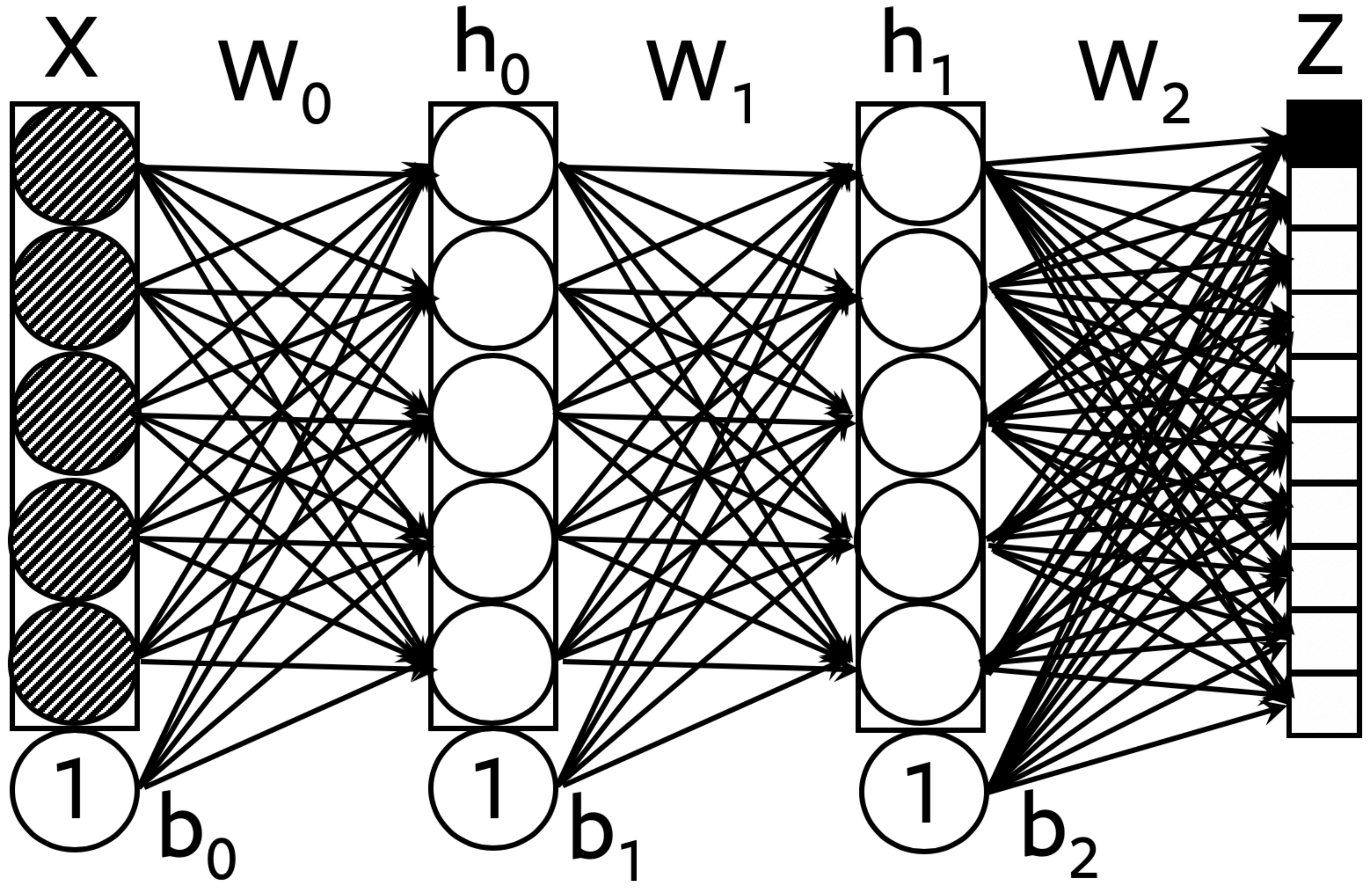
- We need to transform our class scores Z into a probability distribution P using the Softmax function:

$$P = Softmax(Z)$$

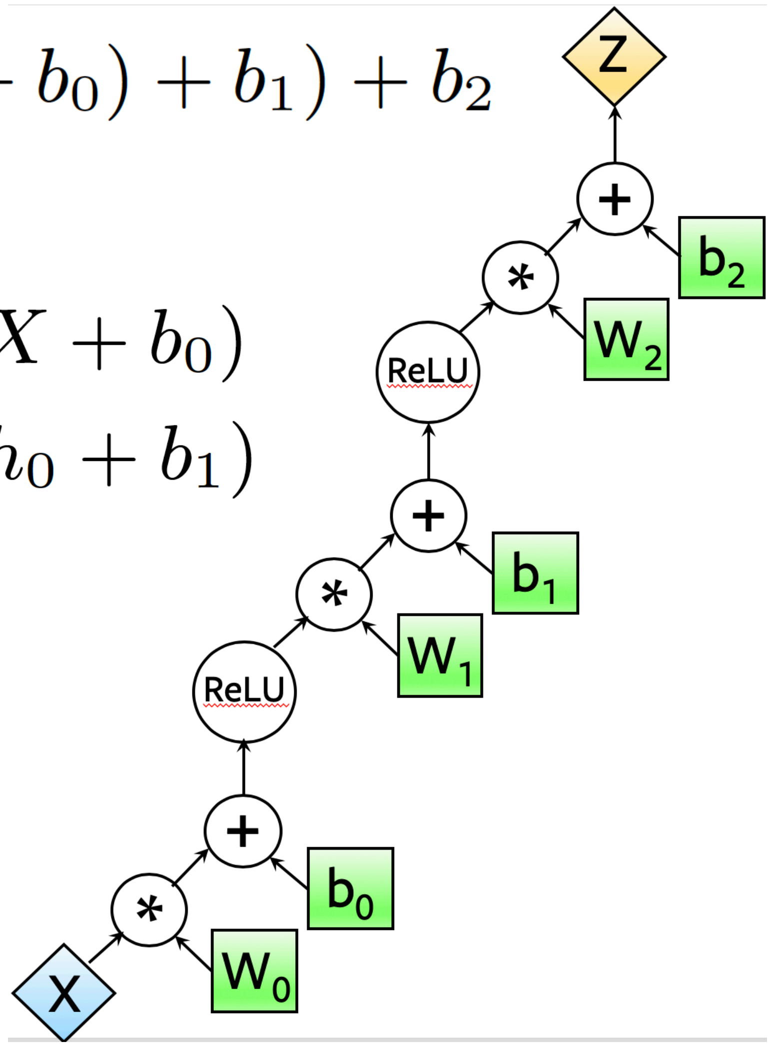
$$P_c = \frac{exp(Z_c)}{\sum_{j=0}^{C} exp(Z_j)}$$

- We measure how good/bad network predictions are using Loss Functions. For classification tasks we use Cross Entropy:

$$\mathcal{L}(\theta) = -\sum_{c=0}^{C} Y_c\, log(P_c)$$
$$= -Y_t\, log(P_t)$$
$$= -log(P_t)$$



$$Z(X;\theta) = W_2 * ReLU(W_1 * ReLU(W_0 * X + b_0) + b_1) + b_2$$
$$where\ \theta = \{W_0, b_0, W_1, b_1, W_2, b_2\}$$



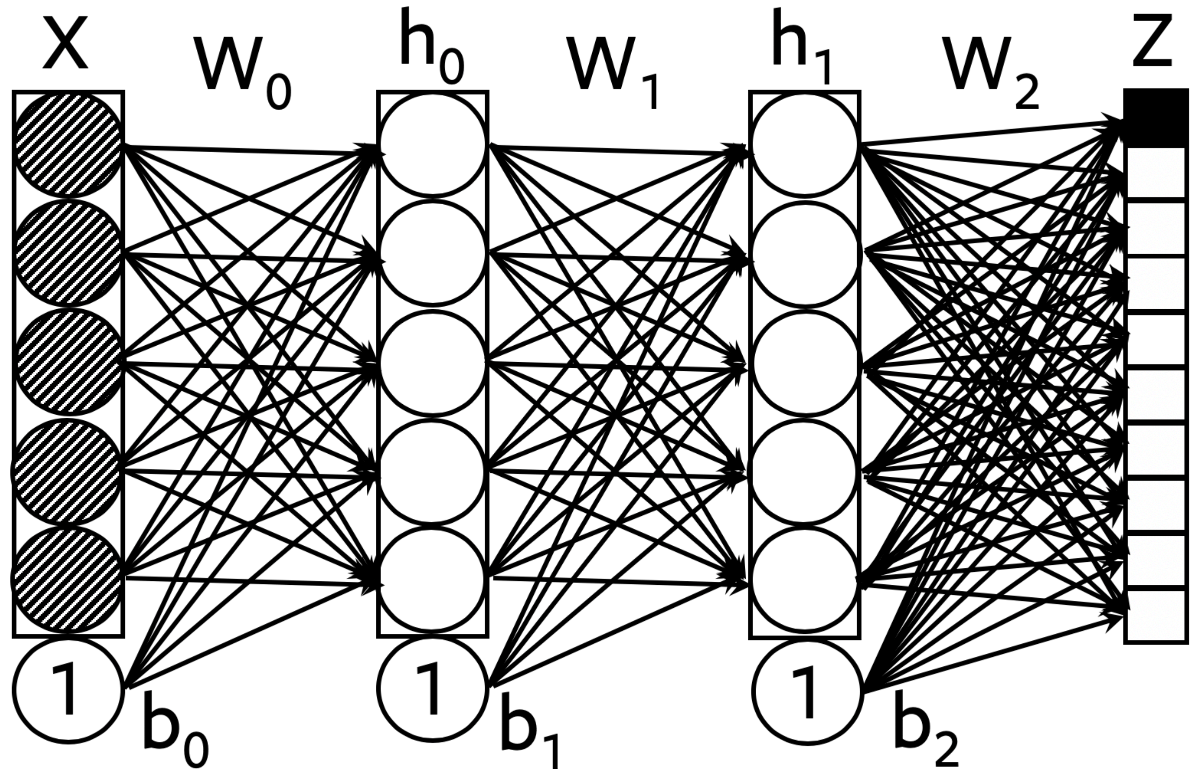
Neural Networks in 5 slides (4/5)

$$h_0 = ReLU(W_0 * X + b_0)$$
$$h_1 = ReLU(W_1 * h_0 + b_1)$$
$$Z = W_2 * h_1 + b_2$$

- To learn the network parameters (aka. Weights) we use Gradient Descent (GD):

$$\theta_{i+1} = \theta_i - \alpha \nabla \mathcal{L}(\theta_i)$$

- GD uses the derivatives of the loss function w.r.t different parameters to find a direction that reduces the overall loss.
- This is done using the "Chain Rule" (aka. The Backpropagation algorithm).
- The loss function and the gradients are computed over a few (input, output) pairs per update, aka. Minibatch GD.
- There are many extensions of GD: ADAM, NAG, AdaDelta, ...



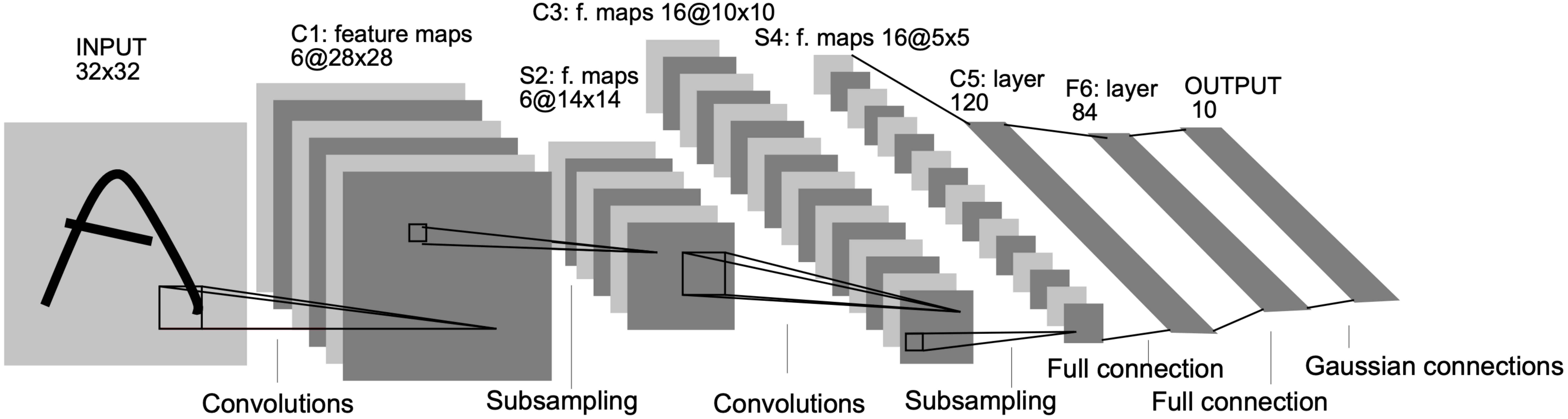
$$P = Softmax(Z)$$

$$P_c = \frac{exp(Z_c)}{\sum_{j=0}^{C} exp(Z_j)}$$

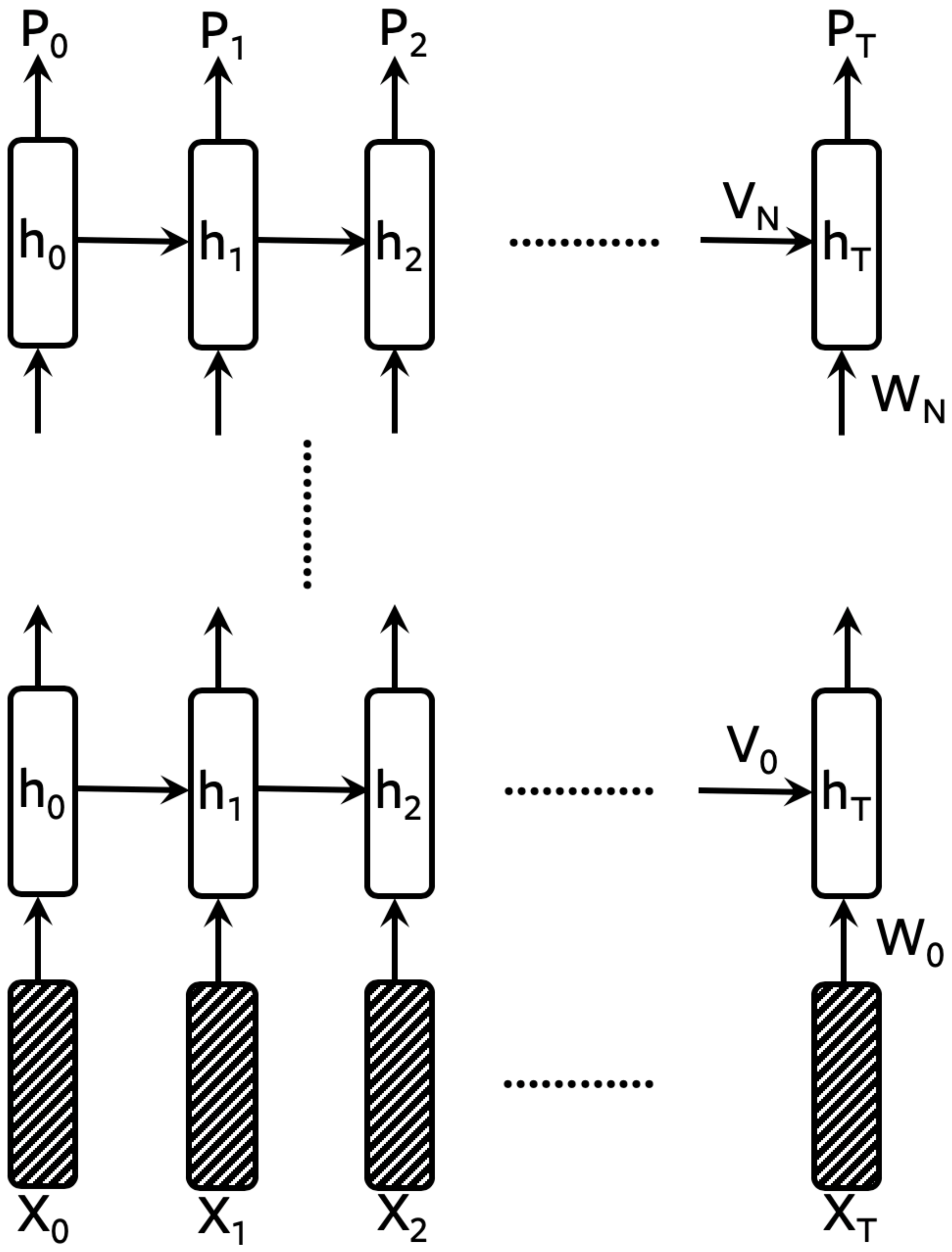
$$\mathcal{L}(\theta) = -\sum_{c=0}^{C} Y_c\, log(P_c)$$
$$= -Y_t\, log(P_t)$$
$$= -log(P_t)$$

Neural Networks in 5 slides (5/5)

- Different types of inputs requires different network architectures.
- Convolutional Neural Networks (CNNs) captures spatial relationships

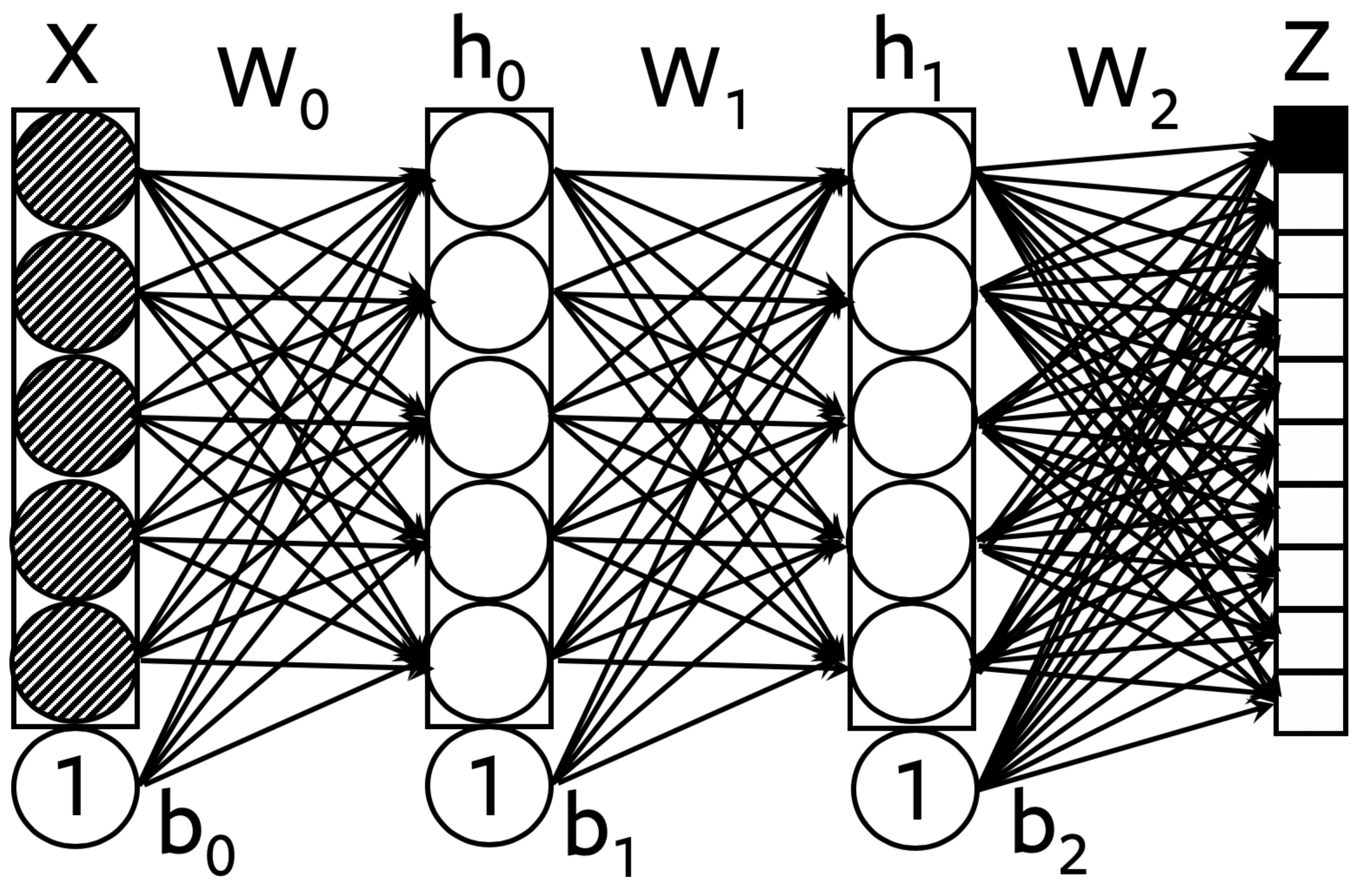


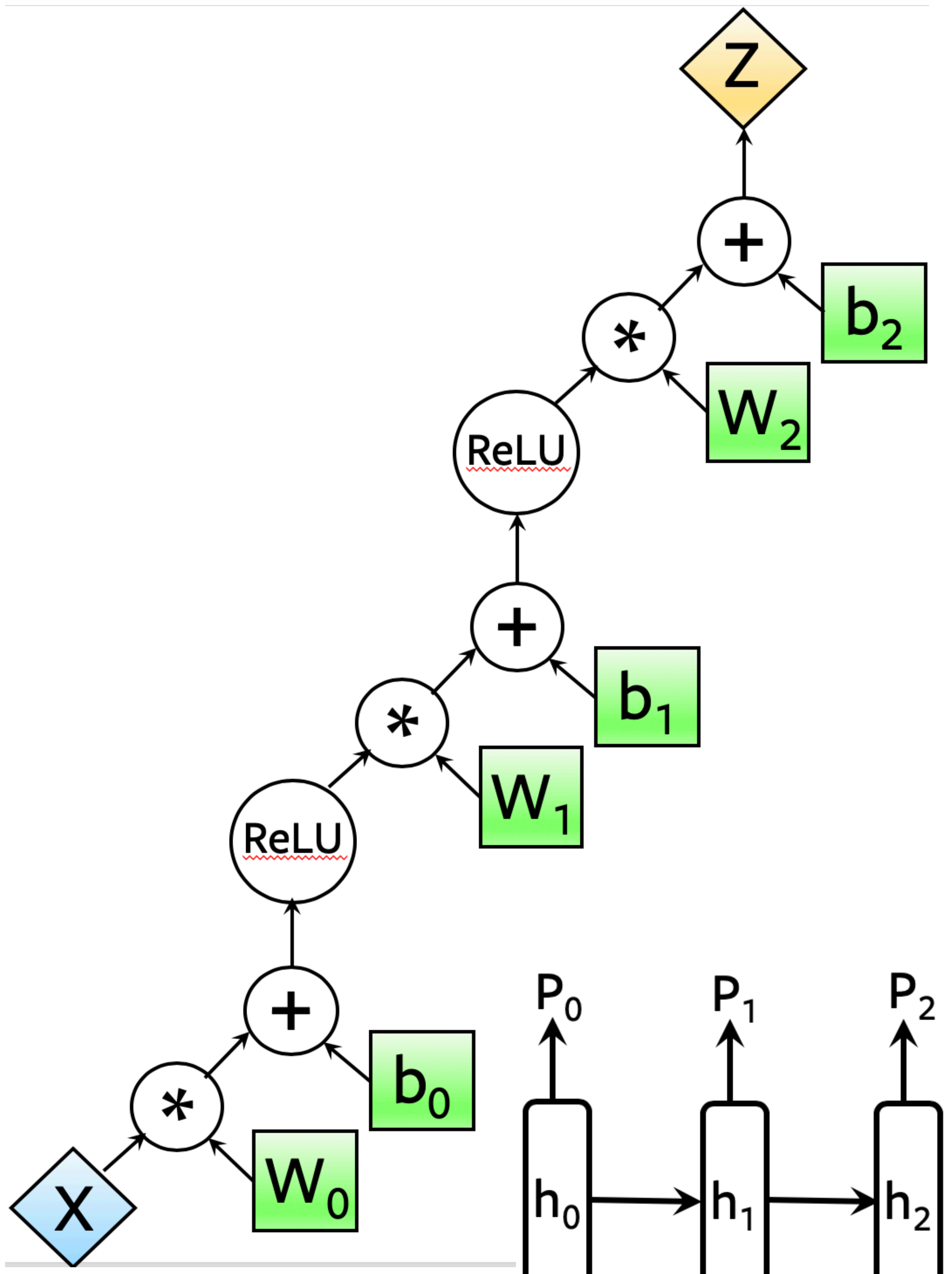
- For sequential inputs, we use Recurrent Neural Networks (RNNs) to model the temporal relationships.
- And many more ….



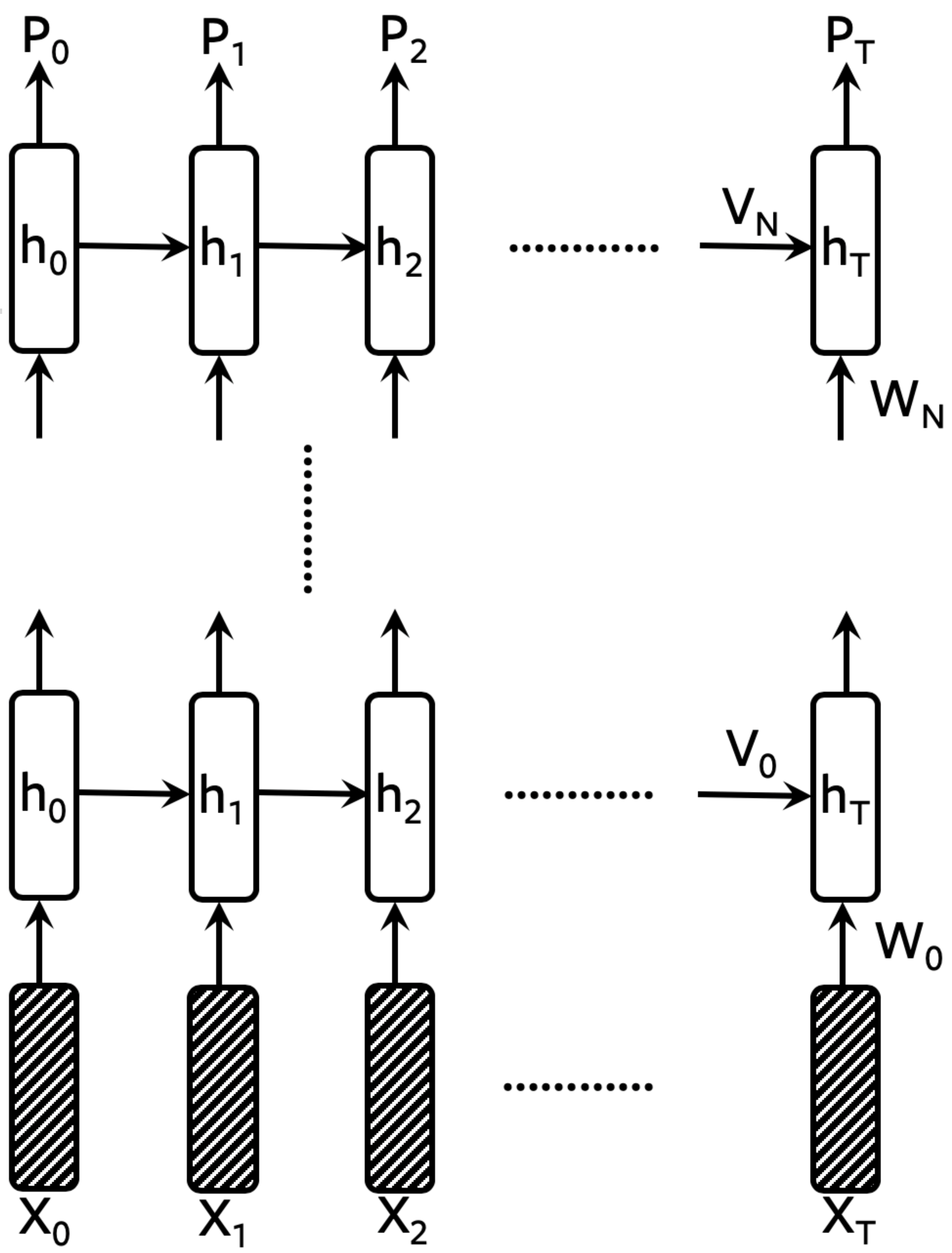
Neural Networks: one slide summary

$$h_0 = ReLU(W_0 * X + b_0)$$
$$h_1 = ReLU(W_1 * h_0 + b_1)$$
$$Z = W_2 * h_1 + b_2$$
$$Z(X;\theta) = W_2 * ReLU(W_1 * ReLU(W_0 * X + b_0) + b_1) + b_2$$
$$where\ \theta = \{W_0, b_0, W_1, b_1, W_2, b_2\}$$

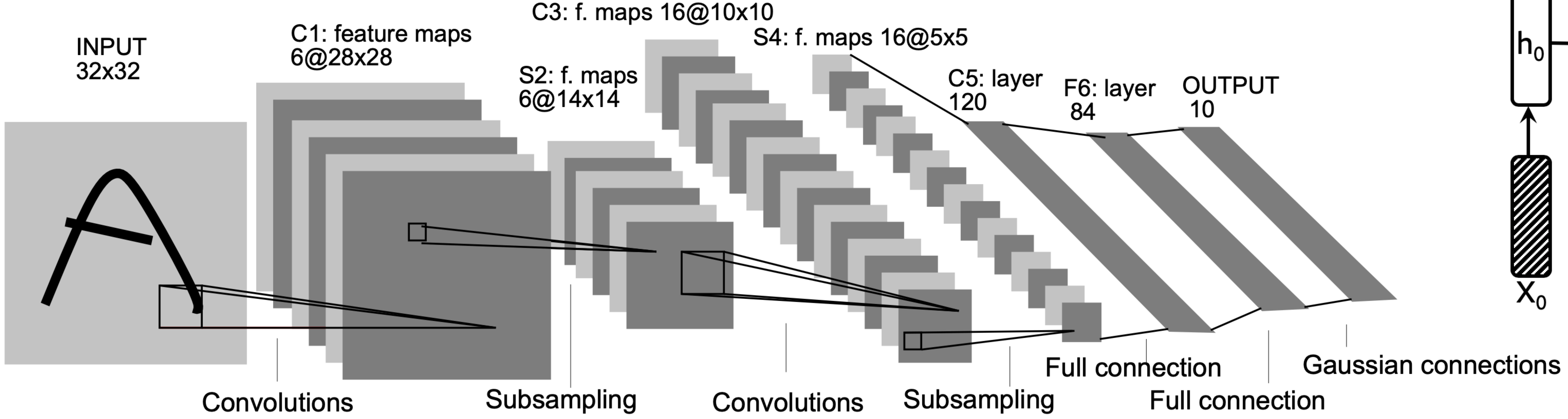




$$P = Softmax(Z)$$
$$P_c = \frac{exp(Z_c)}{\sum_{j=0}^{C} exp(Z_j)}$$

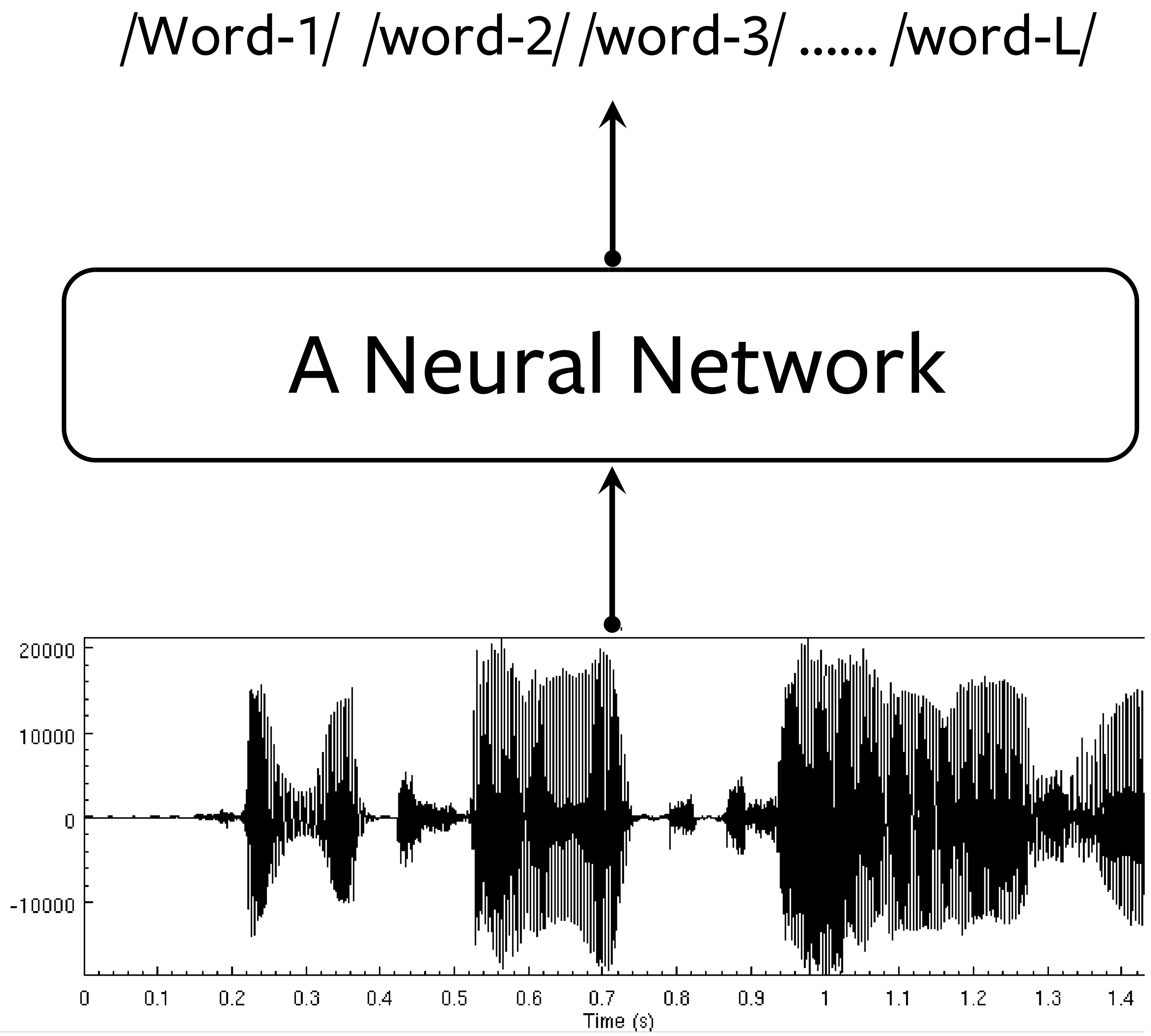


$$\mathcal{L}(\theta) = -\sum_{c=0}^{C} Y_c\, log(P_c)$$
$$= -Y_t\, log(P_t)$$
$$= -log(P_t)$$
$$\theta_{i+1} = \theta_i - \alpha \nabla \mathcal{L}(\theta_i)$$

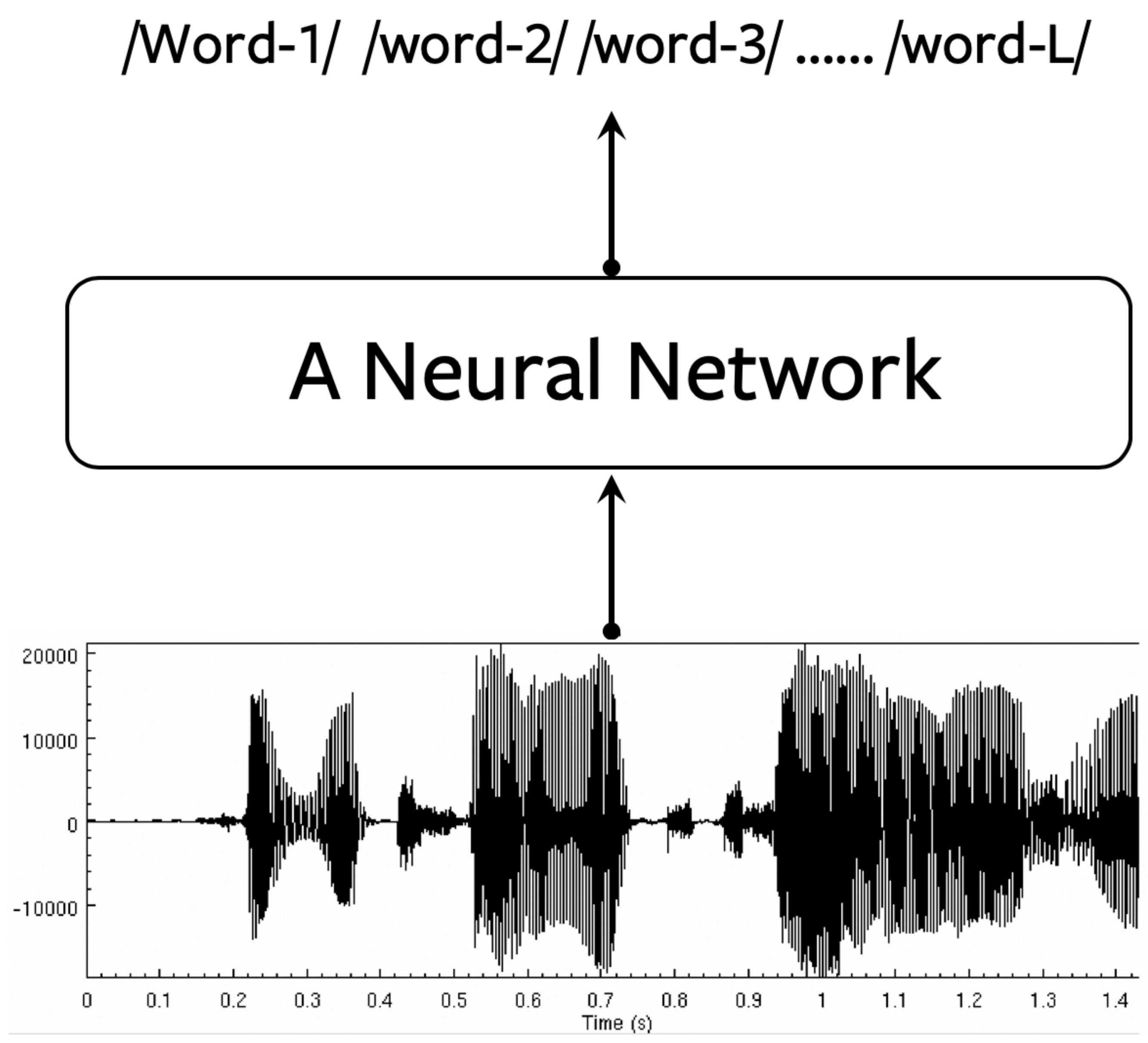


What about Speech Recognition?

- We need to classify the input speech into a sequence of output words.
- Different input and output lengths.
- Words correspond to variable-length sub-sequences in the input stream.
- To solve this problem, traditional ASR systems used Expectation Maximization (EM) and Viterbi algorithm to generate input-output alignment.



ASR HMM/NN hybrid: 3 sub-problems given



- Language Modeling:

$$P(W_0, W_1, W_2, ..., W_m) = \prod_{i=1}^{m} P(W_i|W_0, W_1, ..., W_{i-1})$$
$$\approx \prod^{m} P(W_i|W_{i-2}, W_{i-1})$$

- Pronunciation dictionary:
  - Decompose words into small units of sound, known as Phonemes.
  - Mostly done by human experts.
- Acoustic Model:
  - Maps the audio signal segments to phonemes.
  - Using alignments, it boils down to a standard classification task.
- Decoder: Searches over all hypotheses weighing their probabilities.

End-to-End approaches for ASR

- Alignment-free:

  - No pre-alignment using a previously trained model

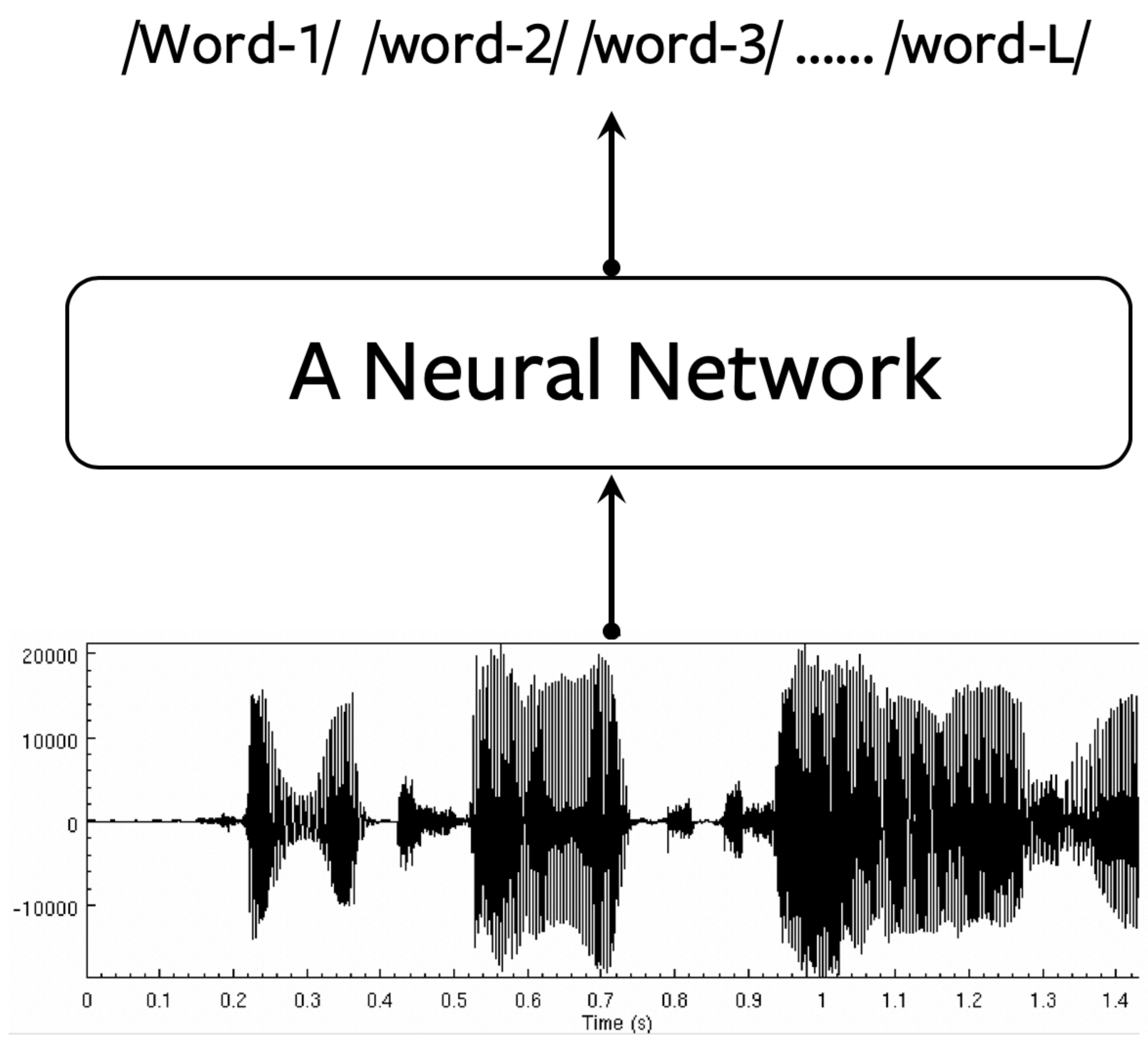
- Direct optimization of the output sequence:

  - Using sequence level loss, e.g. CTC

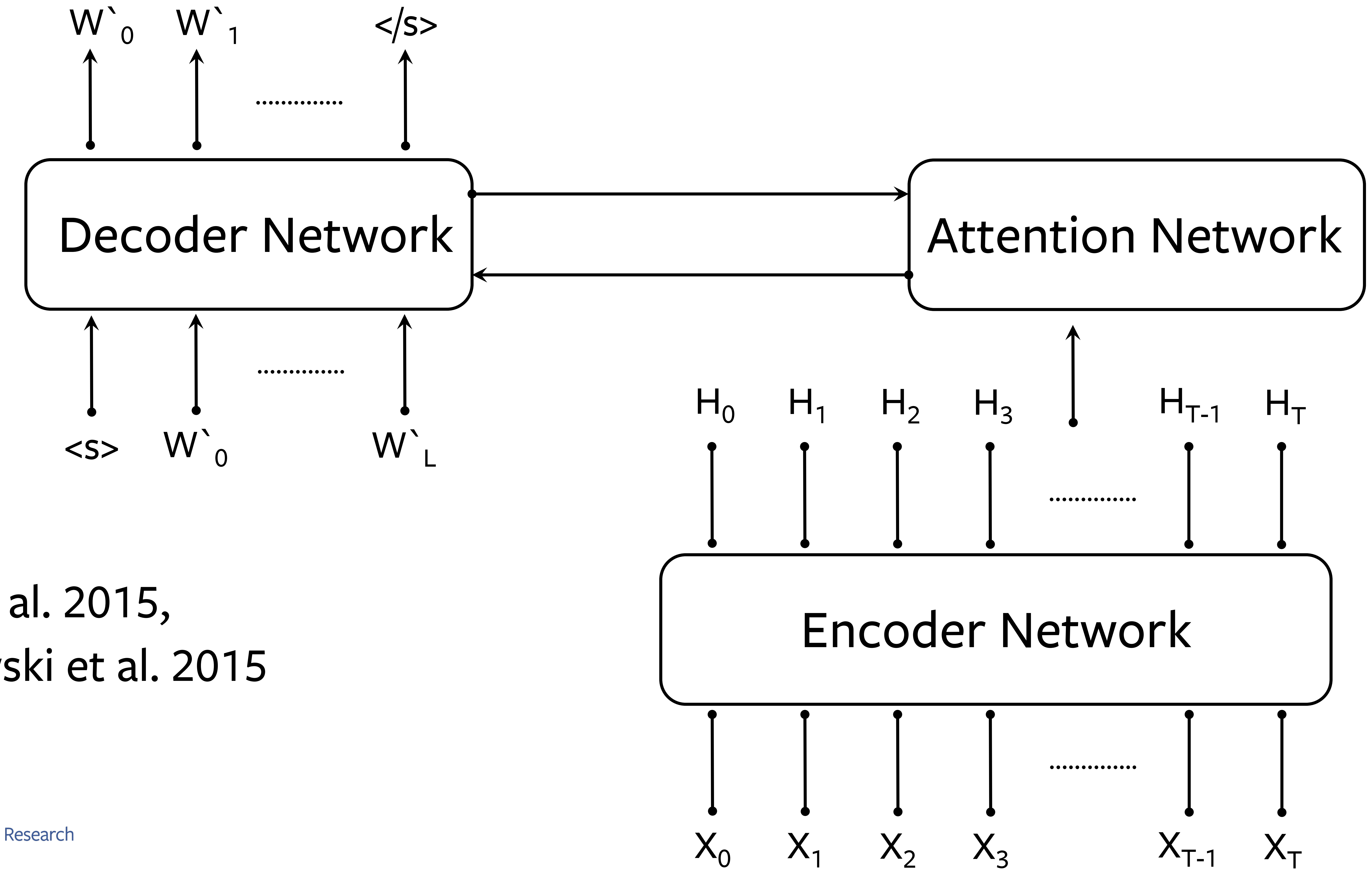
  - Cross-Entropy at each output unit conditioned on previously predicted units, e.g. Enc/Dec with attention.

- Joint training of the acoustic and the language models.
- No pronunciation dictionary!

  output units = Characters, Word Pieces, Words

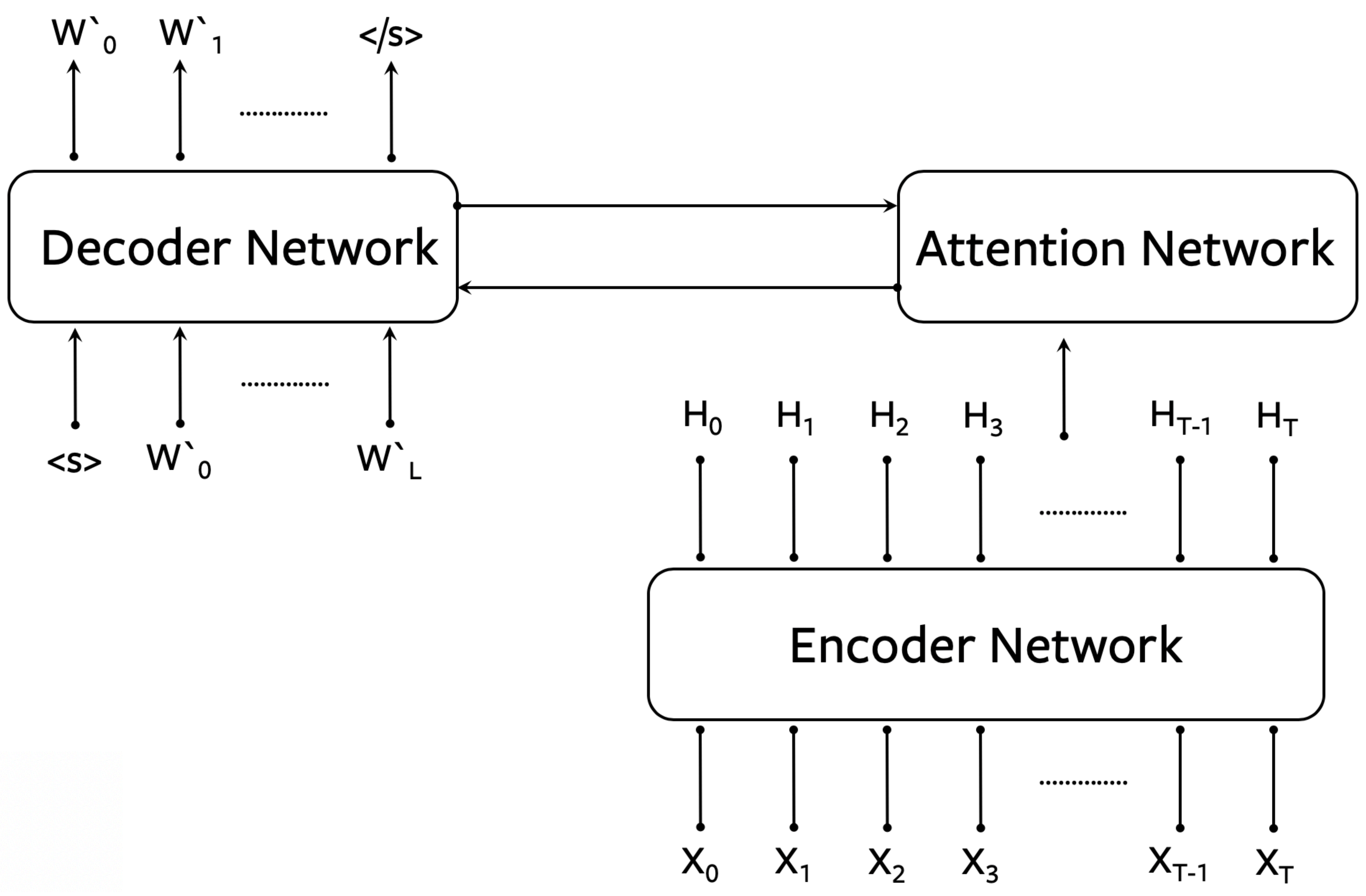


Encoder-Decoder with attention for ASR



Chan et al. 2015,
Chorowski et al. 2015

Encoder-Decoder with attention for ASR



- Mapping to standard ASR components:

  Encoder Network = Acoustic Model

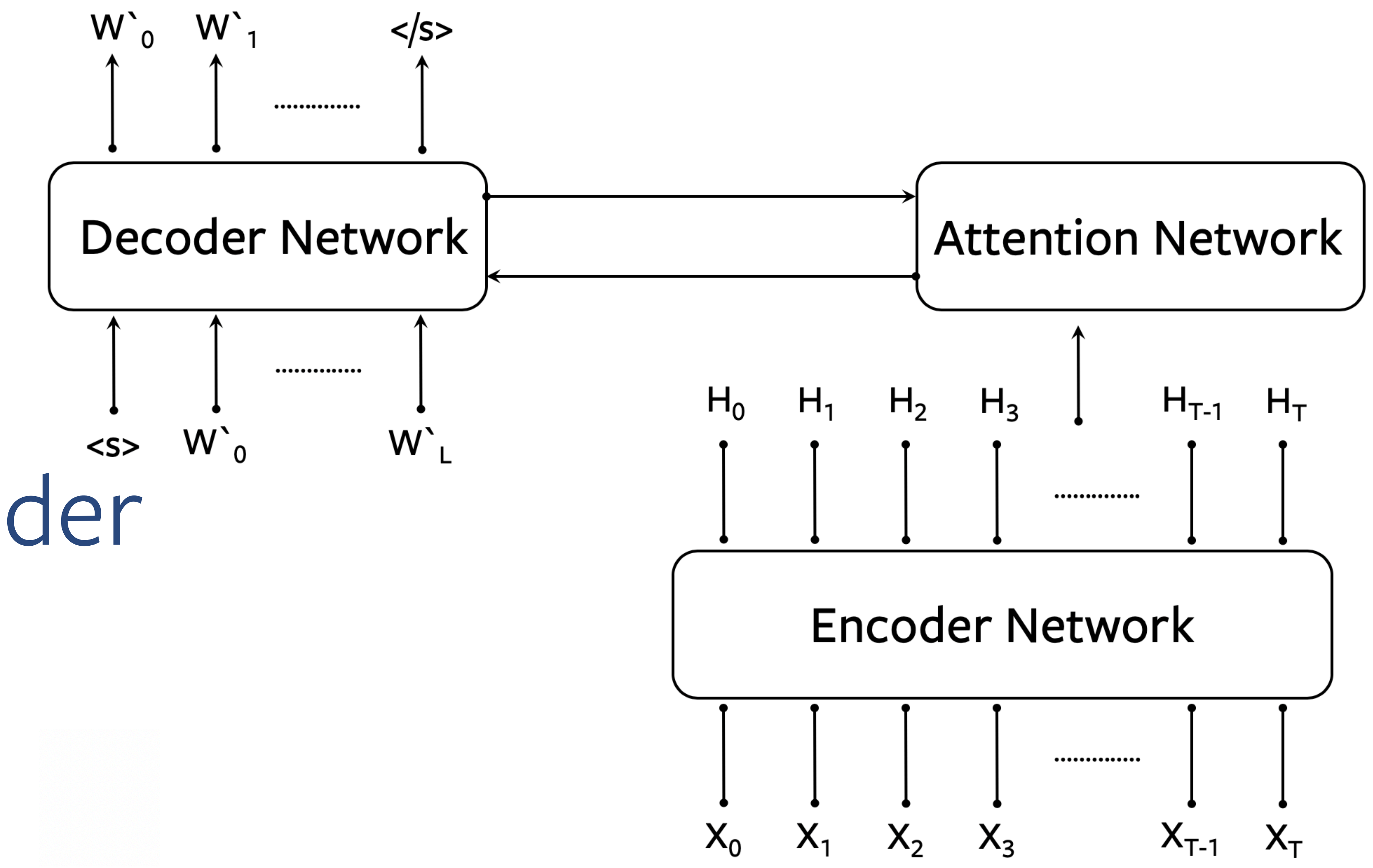
  Decoder Network = Language Model

  Attention Network = input-output alignment

- All components are trained jointly using Gradient Descent.
- Beam search during inference to generate the n-best hypotheses.
- Interesting synergies between the acoustic and language models.

The attention operation

- (Q)uery= S_{t-1}, (K)eys=H, (V)alues=H
- It performs soft alignment via weighted sum of encoder states H (Values).
- The similarity between S_{t-1} (Query) and H (Keys) determines summation weights.
- The context/summary vector C_t is used by the decoder to compute the next decoder state S_t
- Q, K, V can have projection weights to new dim.

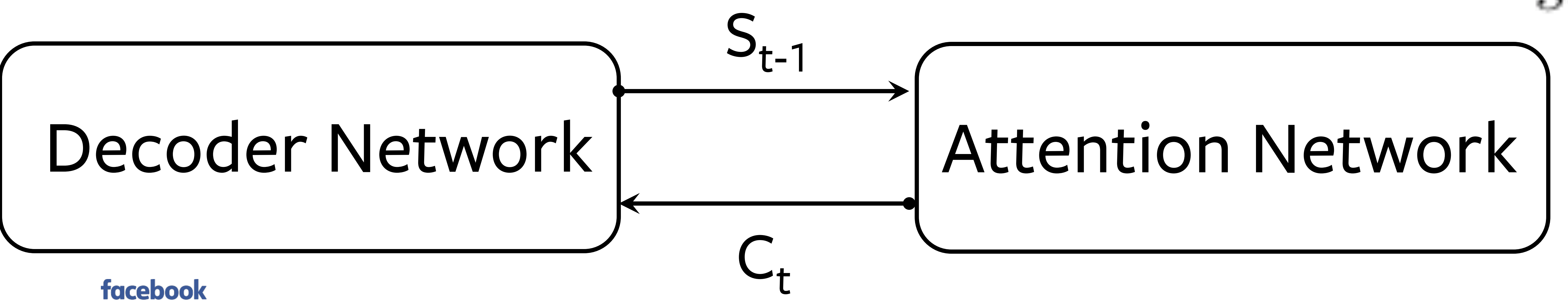


$$H = [H_0, H_1, ..., H_T]$$

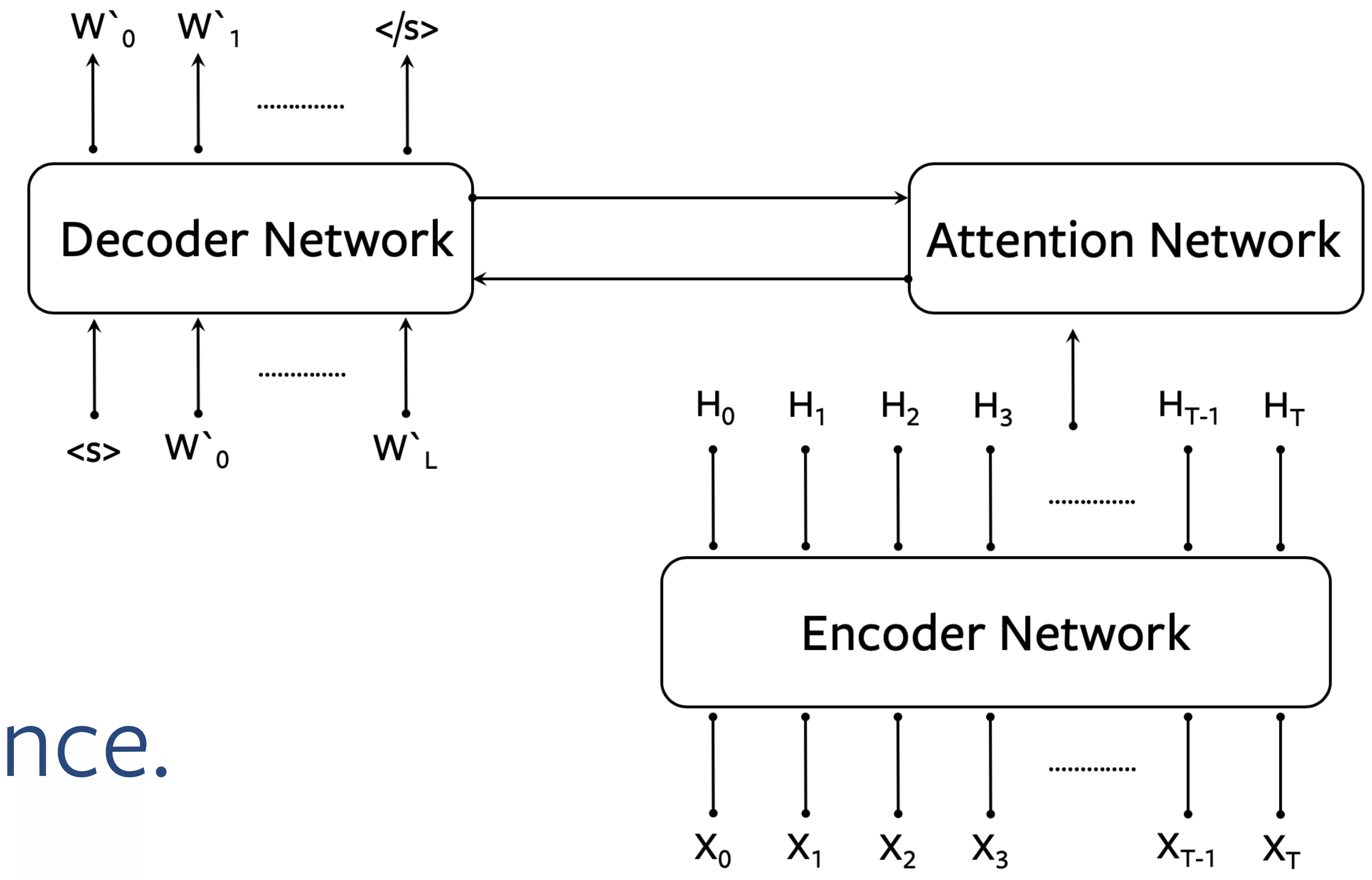
$$s_{t-1} = \text{Decoder Network state at t-1}$$

$$\alpha_t = Softmax(s_{t-1}^T H)$$

$$c_t = \sum_j \alpha_t^j H_j$$



Encoder-Decoder with attention for ASR



- Challenges:
  - Can't leverage text-only data from the web.
  - Attention random start can lead to slow convergence.
  - Doesn't enforce alignment monotonicity.
  - Multinomial attention doesn't fit the segmental nature of speech input.
  - Joint training …
    - -> can lead to easy overfitting of training data.
    - -> made the whole system more sensitive to training hyperparameters.
    - -> is computationally more expensive compared to hybrid HMM/NN system.

From LSTMs to Transformers (Vaswani et al. 2017)

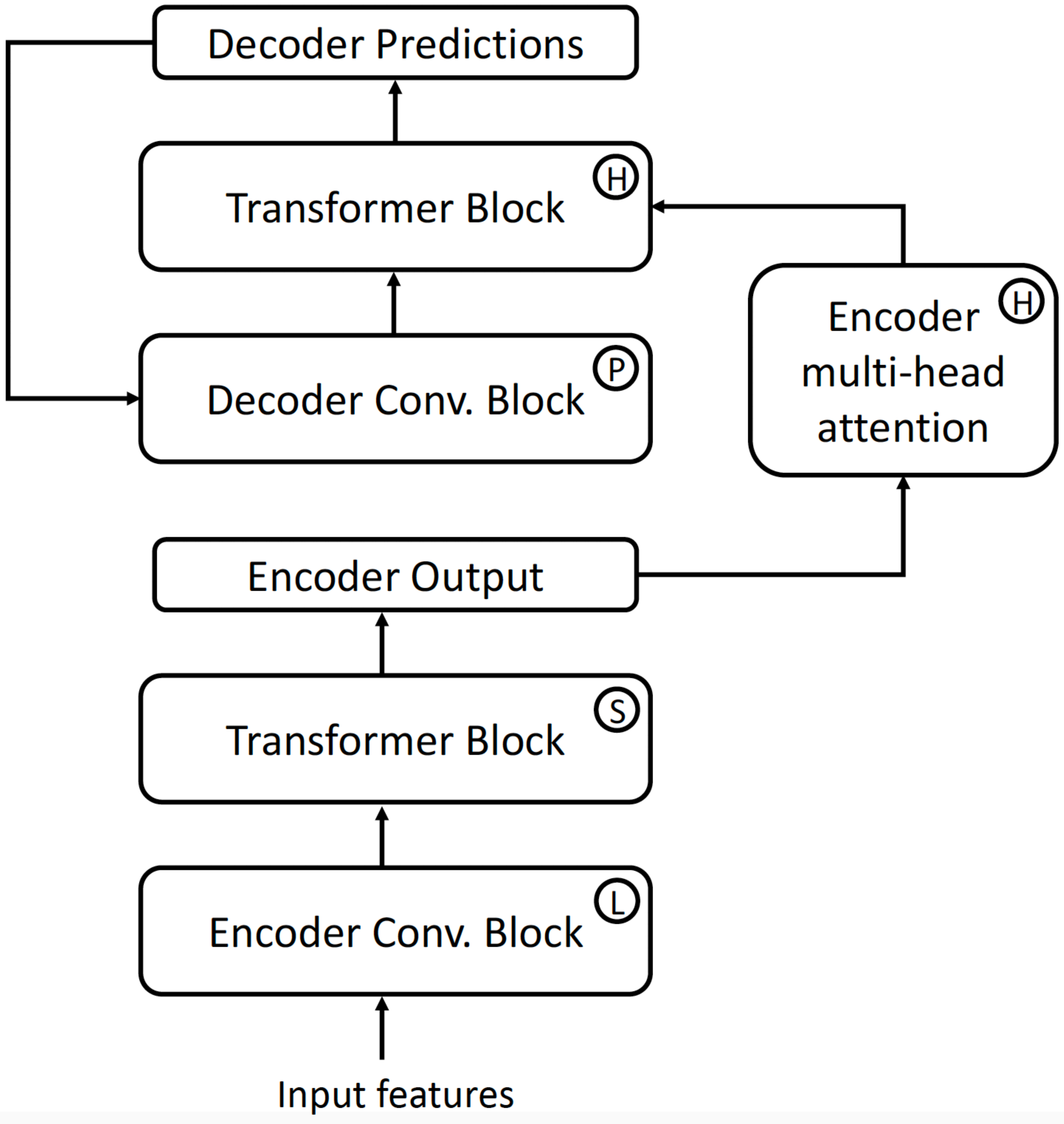
- LSTMs problems:
  - The sequential dependency between time steps limits parallel computation
  - The relationship between two different timesteps fades with distance.

- Transformer networks were proposed for Neural Machine Translation achieving a new state-of-the-art results.

- We present the first successful application of transformers to ASR with a convolutional preprocessing layers for both encoder and decoder.

Transformers with convolutional context for ASR

- We use L 2-D convolutional blocks to process input features before Transformer blocks.
- Same for the decoder, 1-D convolutions precede transformer blocks.
- Each decoder transformer block has a separate attention network.



Transformer block: Self-attention

- Self-attention is the main component of a transformer block:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- timesteps in a sequence are used as (Q)ueries, (K)eys, and (V)alues.

- Similarities between keys and queries determine combination weights of values for each time step.

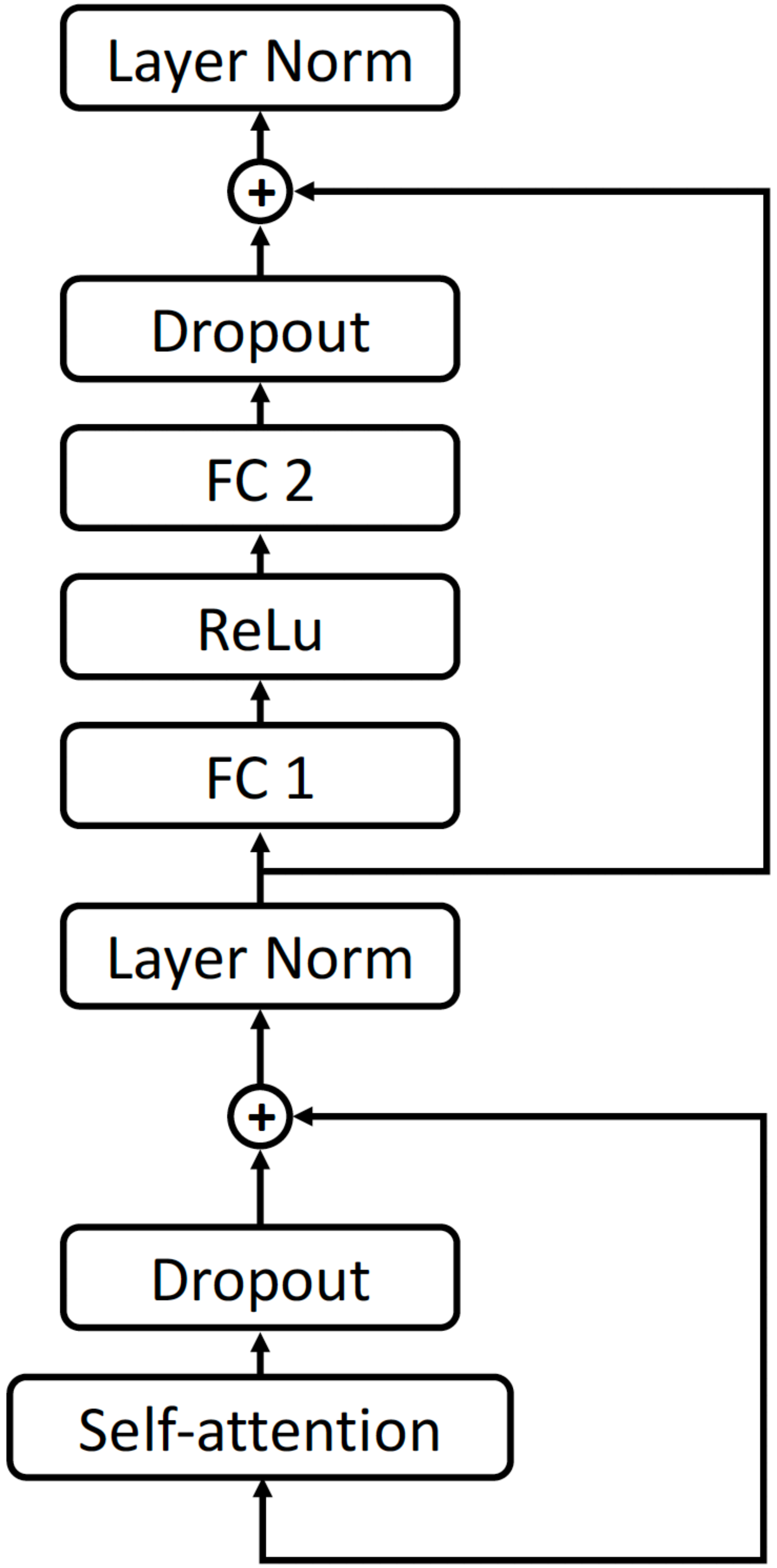
- Each timestep now is a weighted bag-of-features of all input positions

Transformer block: Multi-head Self-attention

- This self-attention operation is done h times in parallel, one per attention head, hence the name “multi-head”:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

- (Q)ueries, (K)eys, (V)alues have projection matrices that are different for each head.
- Outputs of all heads are concatenated then projected to the output.
- On top of self-attention, there are many operations per timestep.
- Multi-head self-attention is used for enc/dec cross attention as well.



Convolutional layers preserve positional information

- The weighted sum in the self-attention operation loses the position information of the input.
- Positional information to be preserved for constructing the correct output order.
- The original proposal of Transformer networks used sinusoidal positional embeddings

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Where d_{model} is the transformer block output dimension.

- We are NOT using positional embedding!
- We think that using convolution before transformers is all what is needed to recover the correct output sequence order.

Experiments on 1000h Librispeech dataset

- Our canonical model has:

  (1) two 2-D convolutional blocks, each with two conv. Layers of 64, 128 features

  (2) 10 encoder transformer blocks with dim=1024, 16 heads, ReLU layer size=2048

  (3) decoder input word embedding dim=512

  (4) three 1-D conv. layers each with kernel size=3

  (5) 10 decoder transformer blocks each with encoder-side multihead attention

  (6) 5000 subword target units

- This canonical model (223M params) takes about 24 hours to perform 80 epochs on 2 machines each with 8GPUs.

Experiments on 1000h Librispeech dataset

Model	dev clean	dev other	test clean	test other
Conv. context	5.2	13.7	5.3	14.0
Sin. pos. embd	5.8	14.2	5.4	14.8
(1) + (2)	5.2	13.8	5.3	14.0
One layer of enc. att.	6.4	15.2	6.3	15.9
32 heads in enc/dec	5.3	14.1	5.4	14.6
4k transformer ReLU	5.3	13.2	5.3	13.4

Experiments: Effect of convolutional context size

Conv. depth	Cxt size (num. kernels)	dev clean	dev other	test clean	test other
1	3 (3)	5.3	13.8	5.4	14.1
	5 (5)	5.4	14.1	5.5	14.0
	7 (7)	5.4	13.9	5.4	14.5
	9 (9)	5.4	13.9	5.5	14.0
	11 (11)	5.3	13.6	5.4	13.8
2	5 (3-3)	5.3	14.0	5.2	14.5
	7 (3-5)	5.5	14.7	5.9	14.8
	9 (5-5)	5.2	13.9	5.6	14.2
	11 (5-7)	5.2	14.1	5.4	14.6
3	7 (3-3-3)	5.2	13.7	5.3	14.0
	9 (3-3-5)	5.3	13.8	5.4	14.1
	11 (3-5-5)	5.6	14.3	5.4	14.2
4	9 (3-3-3-3)	5.0	13.5	5.4	13.9
	11 (3-3-3-5)	5.0	13.6	5.2	13.7

Experiments: Putting best configs together

Model	LM on extra text	dev clean	dev other	test clean	test other
CAPIO spk adpt[22]	RNNLM	3.12	8.28	3.51	8.58
LSTM[23]	4gramLM	4.79	14.31	4.82	15.30
Gated Cnv[24]	4gramLM	4.6	13.8	4.8	14.5
Tsf w/sin pos embd[13]	4gramLM	-	-	4.8	13.1
TDS Cnv[25]	4gramLM	3.75	10.70	4.21	11.87
LSTM[23]	LSTMLM	3.54	11.52	3.82	12.76
Fully Cnv[26]	ConvLM	3.16	10.05	3.44	11.24
TDS Cnv[25]	ConvLM	3.01	8.86	3.28	9.84
TDS Cnv[25]	None	5.04	14.45	5.36	15.64
LSTM[23]	None	4.87	14.37	4.87	15.39
Cnv Cxt Tsf **(ours)**	None	**4.8**	**12.7**	**4.7**	**12.9**

Experiments: Final remarks

- AdaDelta algorithm with fixed learning rate=1.0 and gradient clipping at 10.0 were used.

- The reported result was achieved with a fixed recipe of training for 80 epochs followed by averaging the last 30 checkpoints

- No scheduled sampling or label smoothing was used

- The reported 12% and 16% gains on the “dev-other” and “test-other” sets of Librispeech shows that the transformer network has better ability to deal with challenging acoustic conditions.

Thank You

facebook
Artificial Intelligence Research