Machine Translation and Advanced Deep Learning Models for Sequence Processing

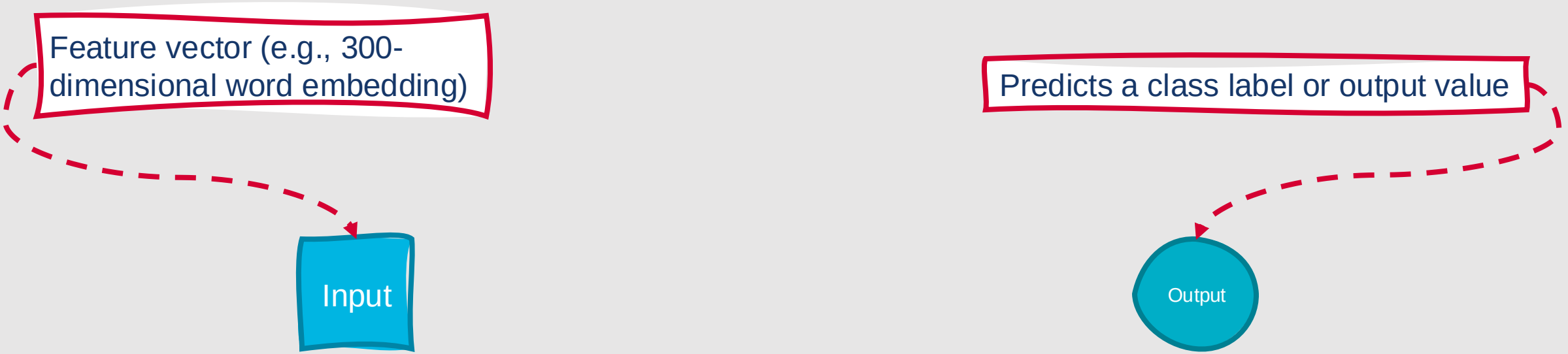
Natalie Parde

UIC CS 521

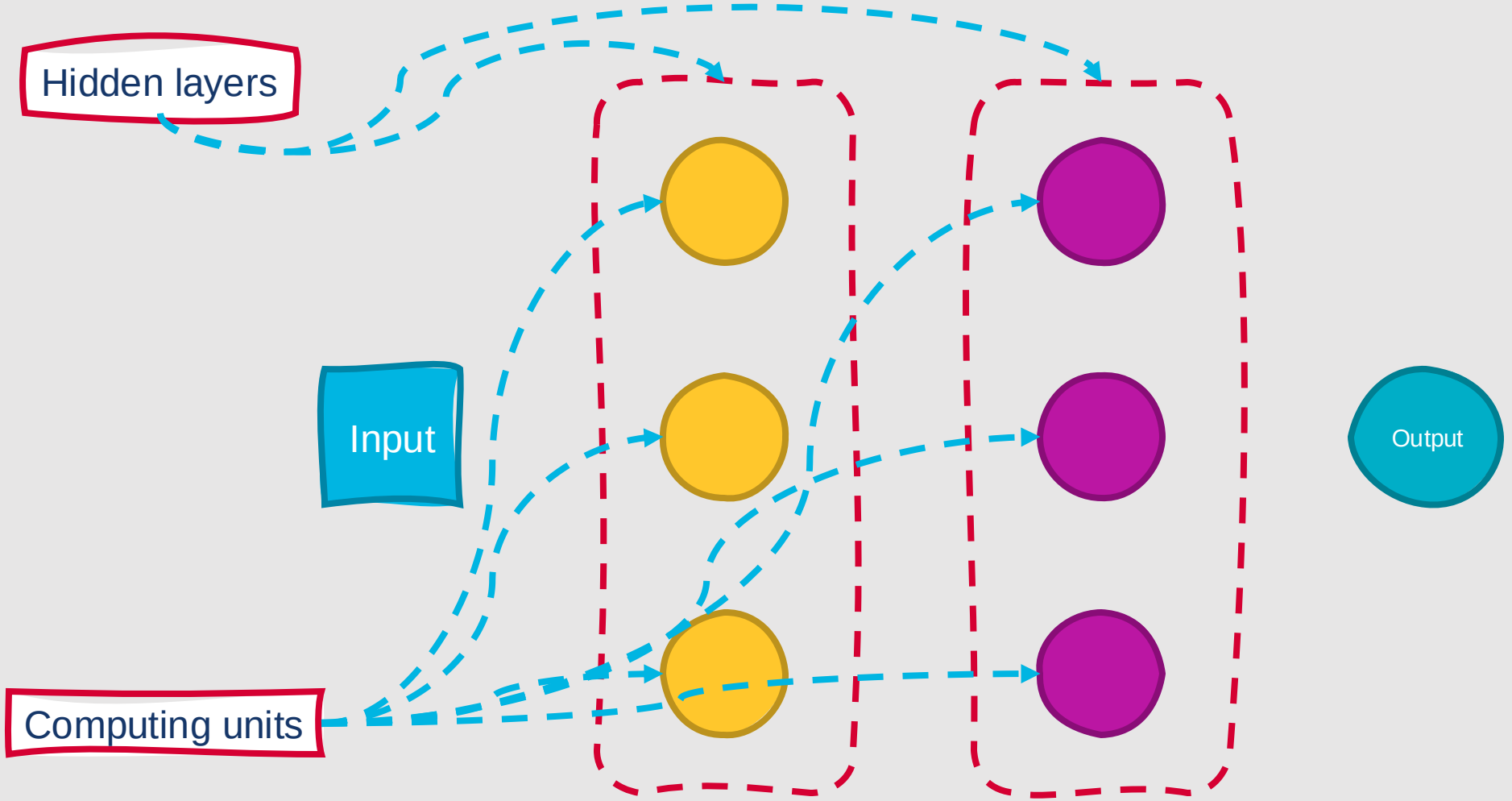
Review: Essentials of Deep Learning

- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Stacked models
- Bidirectional models

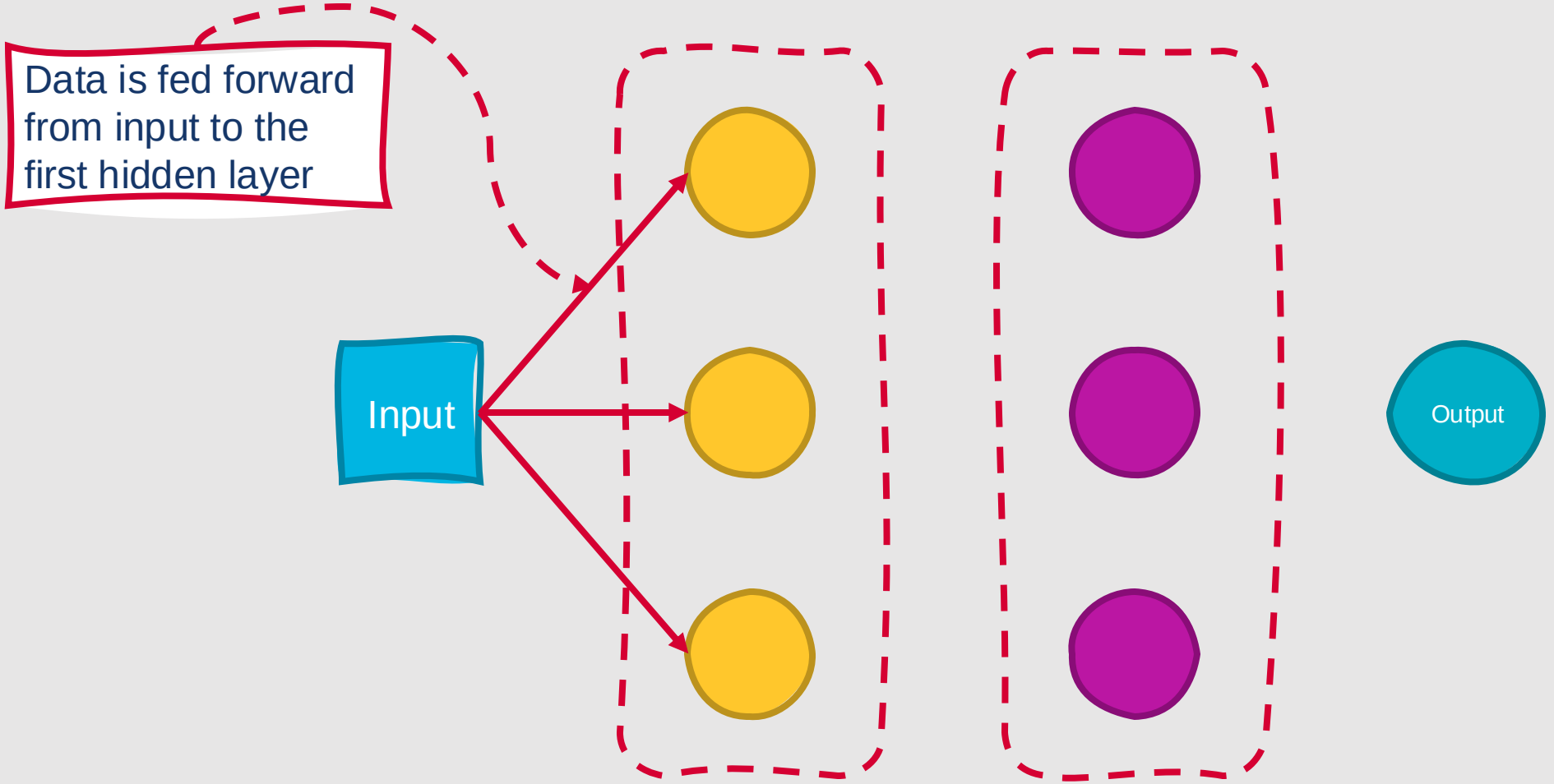
Feedforward Neural Networks



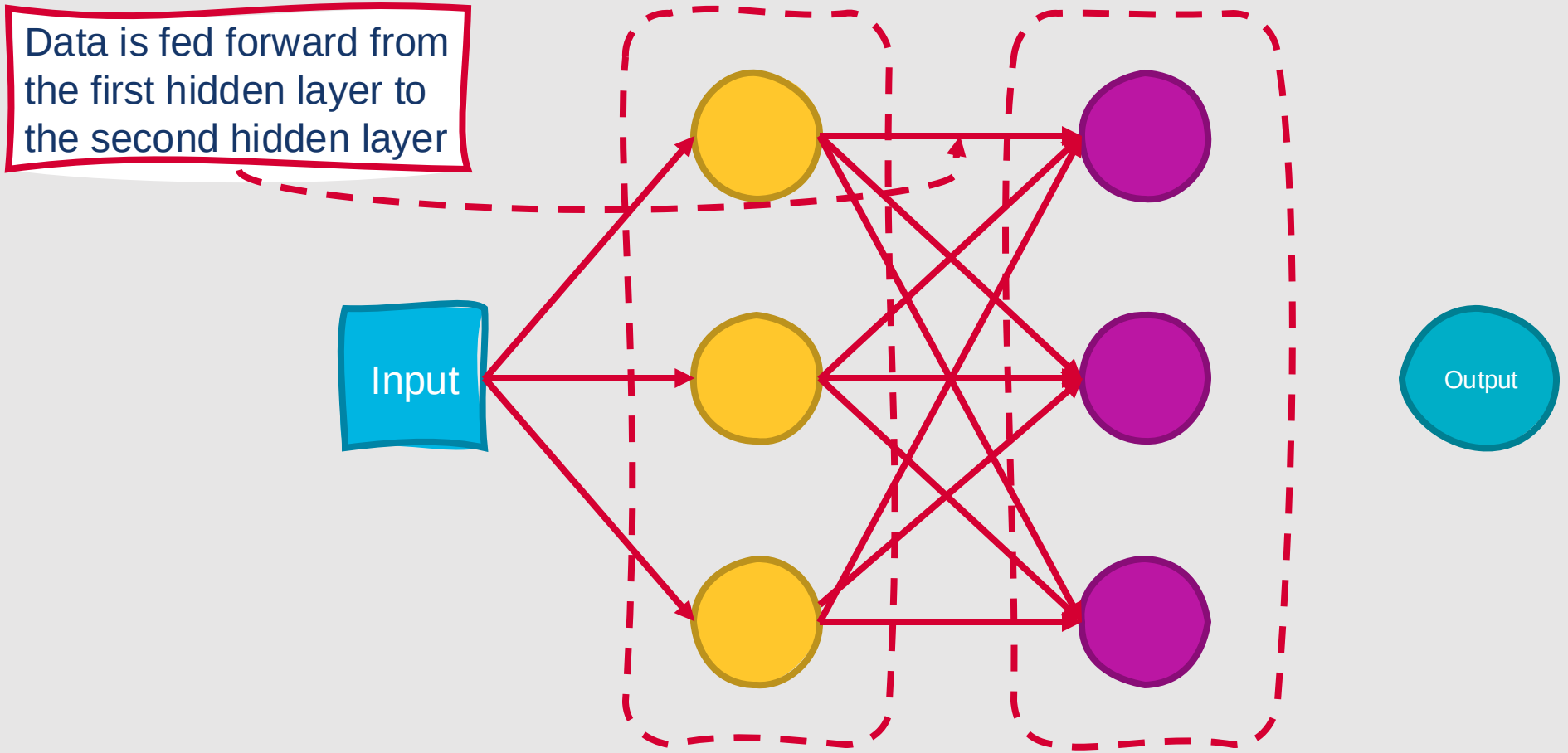
Feedforward Neural Networks



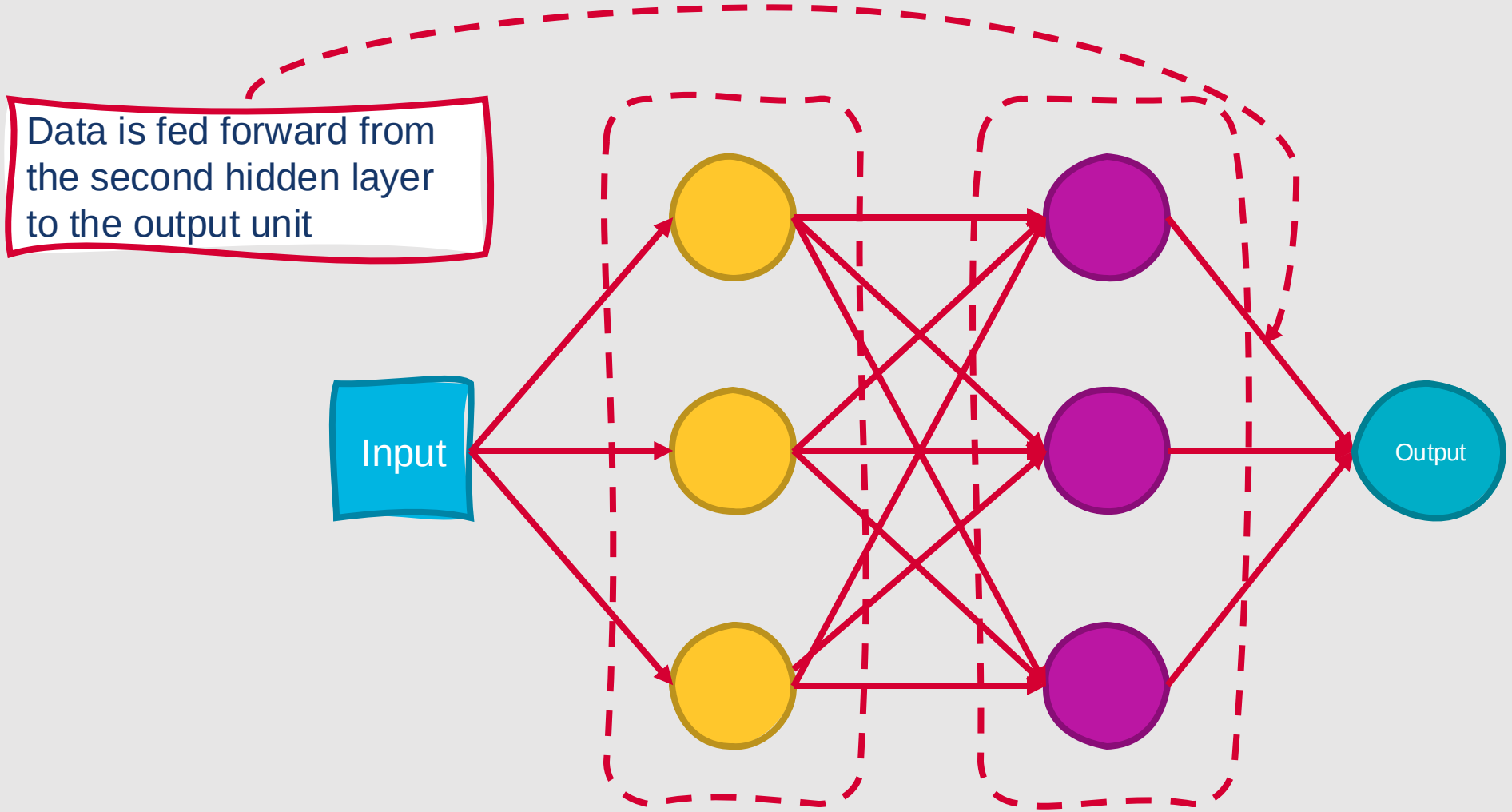
Feedforward Neural Networks



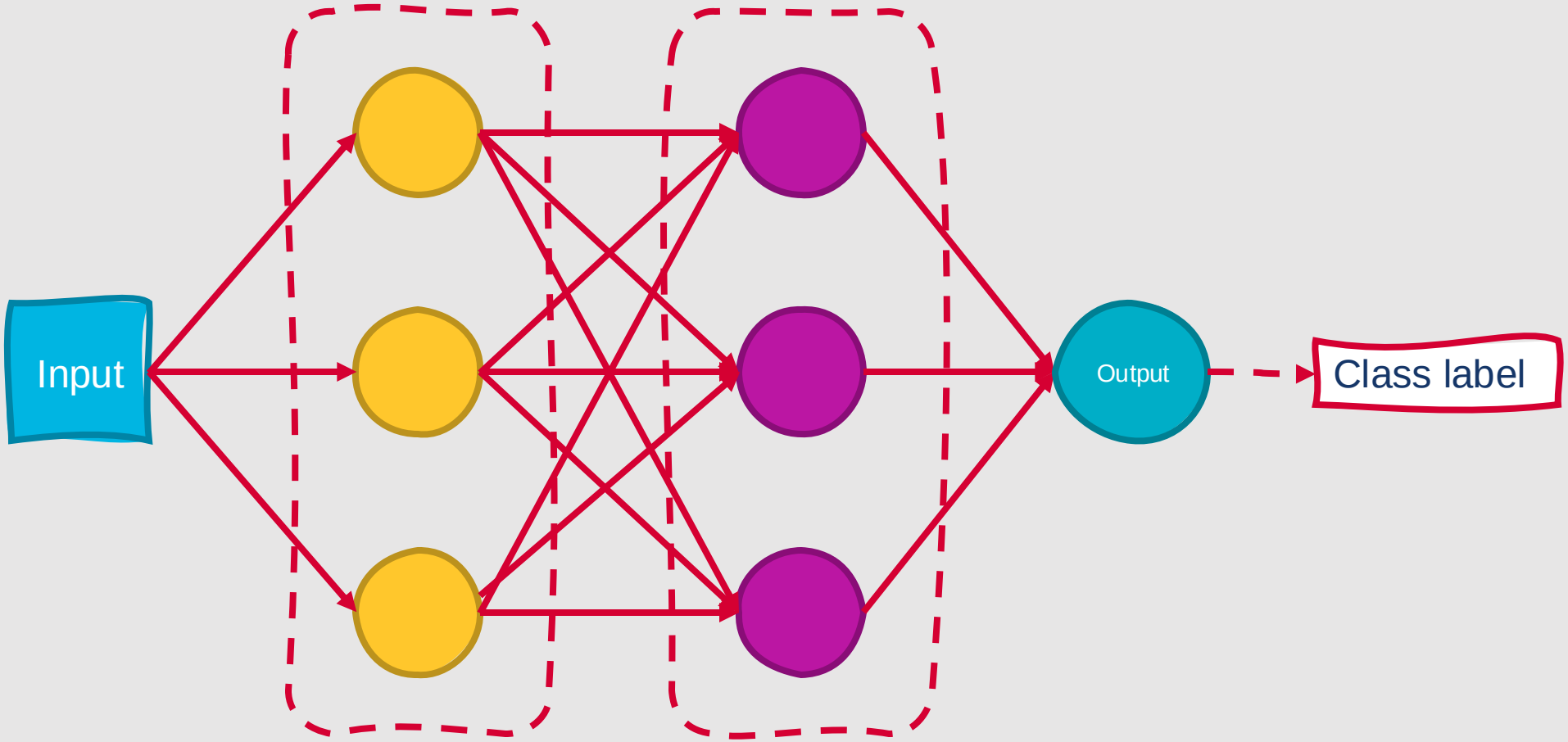
Feedforward Neural Networks



Feedforward Neural Networks



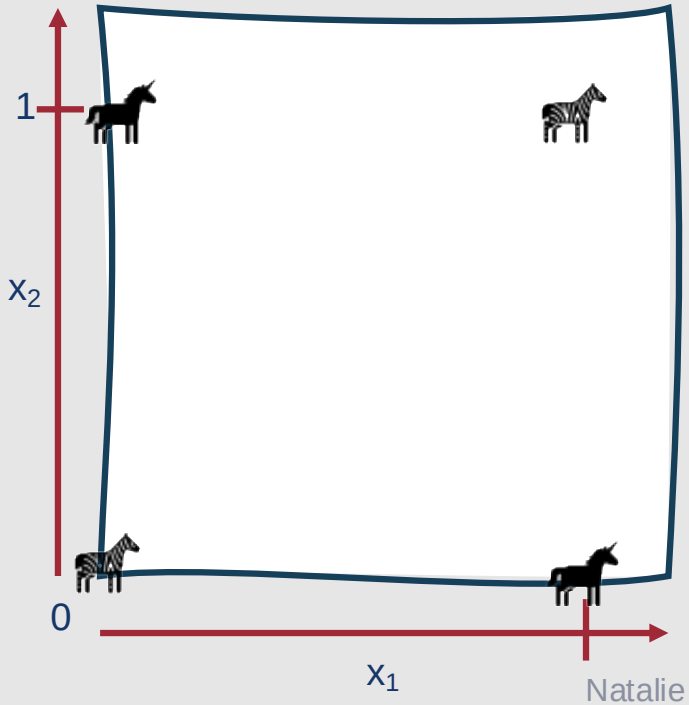
Feedforward Neural Networks



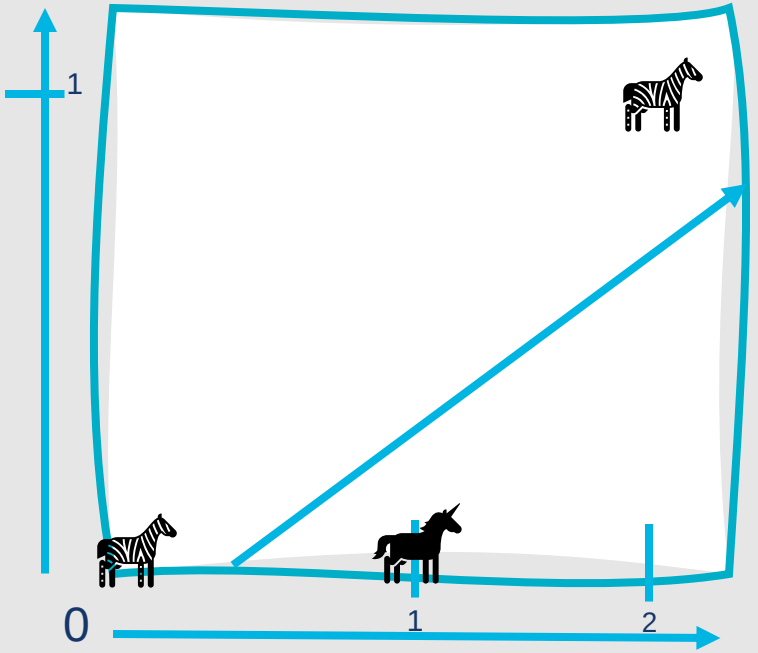
Why does this work?

- When computational units are combined, the outputs from each successive layer provide **new representations** for the input that can better separate the data into the target classes

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



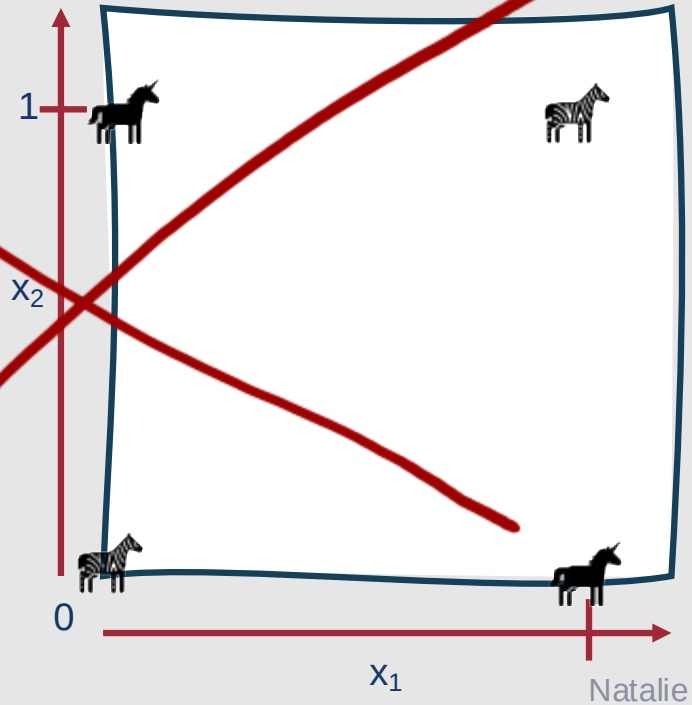
XOR		
h0	h1	y
0	0	0
1	0	1
2	1	0



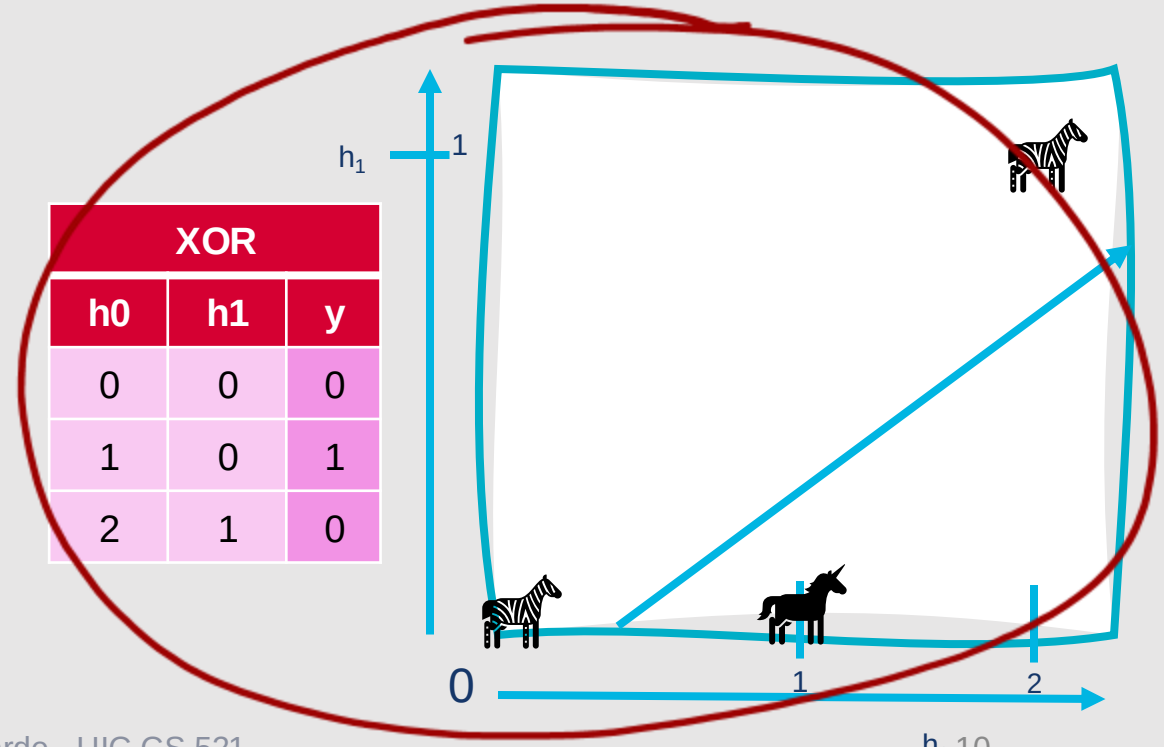
Why does this work?

- When computational units are combined, the outputs from each successive layer provide **new representations** for the input that can better separate the data into the target classes

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



XOR		
h0	h1	y
0	0	0
1	0	1
2	1	0



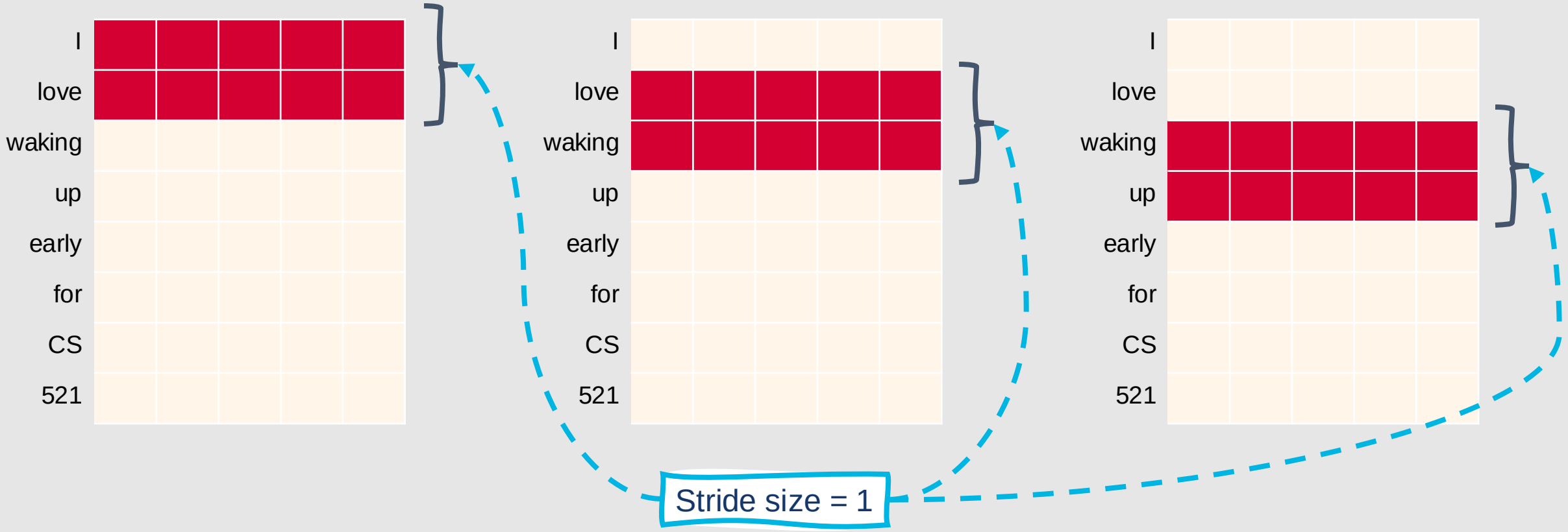
Review: Essentials of Deep Learning

- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Stacked models
- Bidirectional models

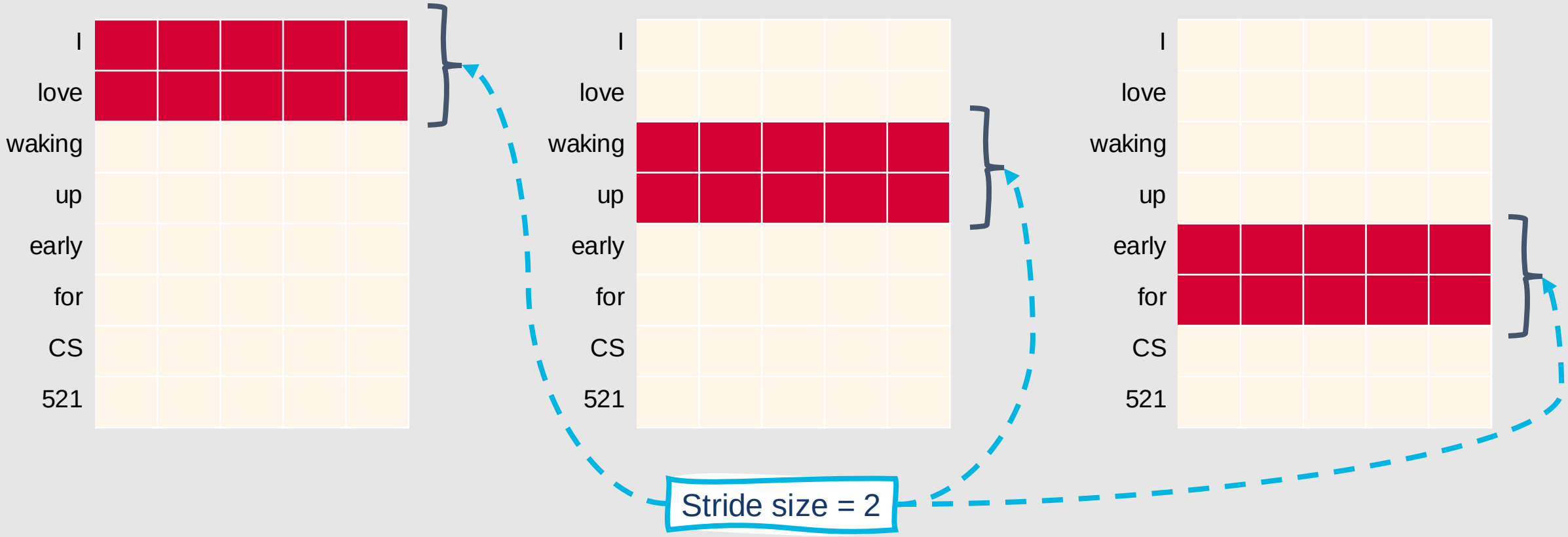
Convolutional Neural Networks

- Neural networks that incorporate one or more **convolutional layers**
- Designed to reflect the inner workings of the visual cortex system
- Require that fewer parameters are learned relative to feedforward networks for equivalent input data

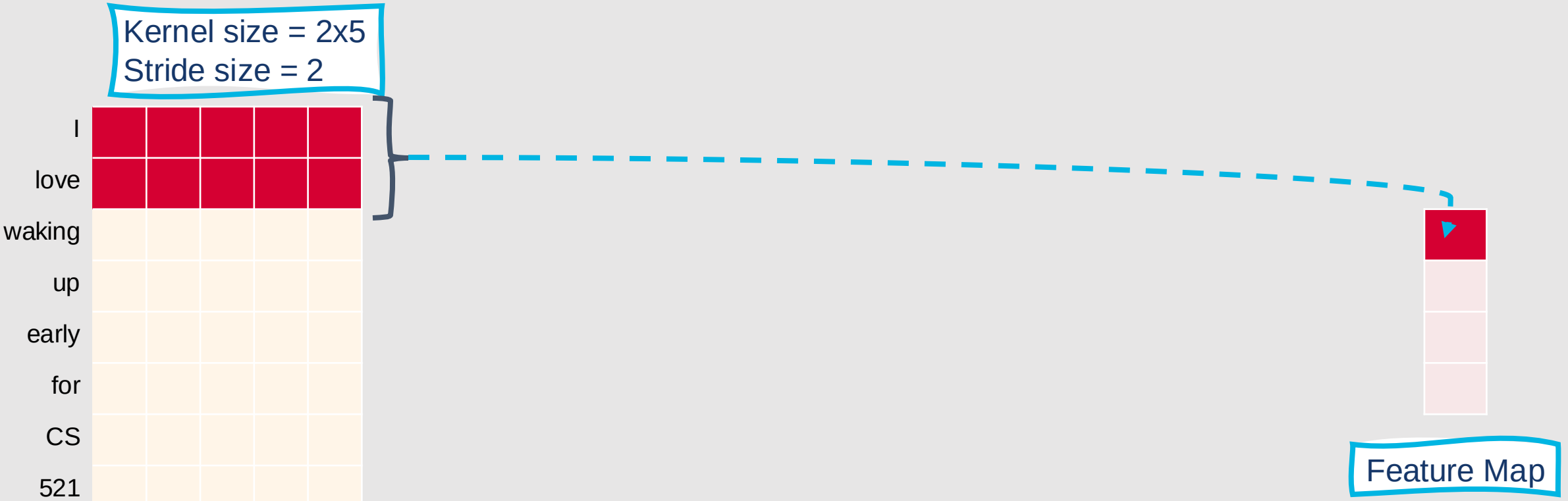
In NLP, convolutions are typically performed on entire rows of an input matrix, where each row corresponds to a word.



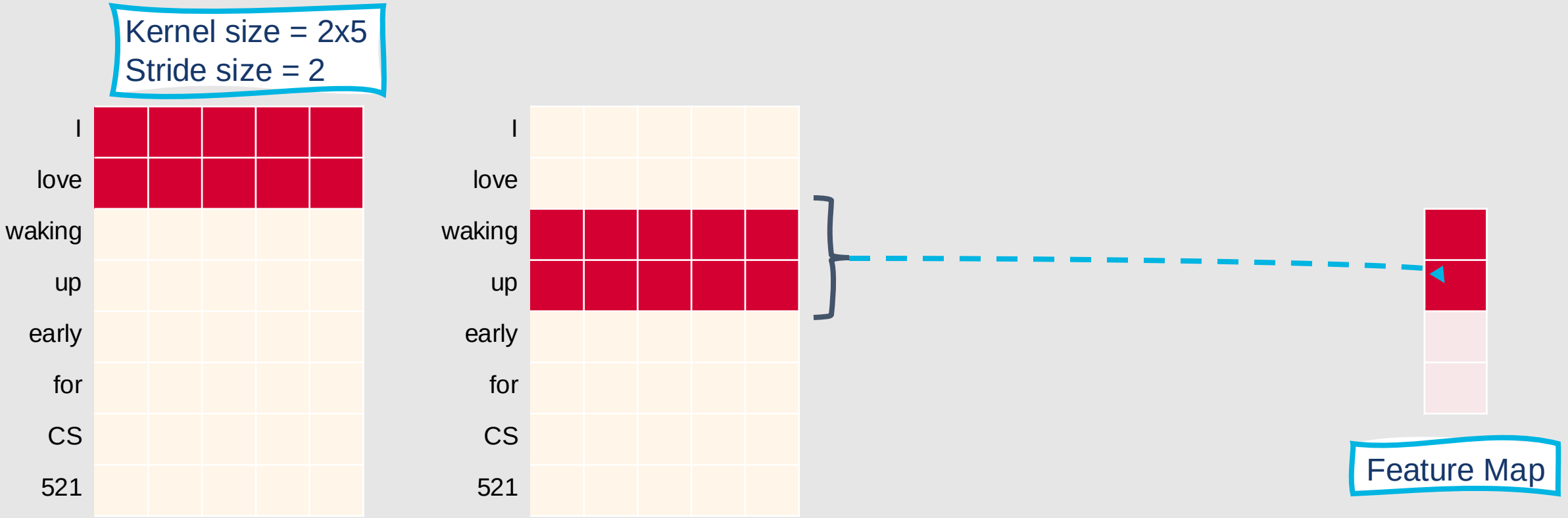
In NLP, convolutions are typically performed on entire rows of an input matrix, where each row corresponds to a word.



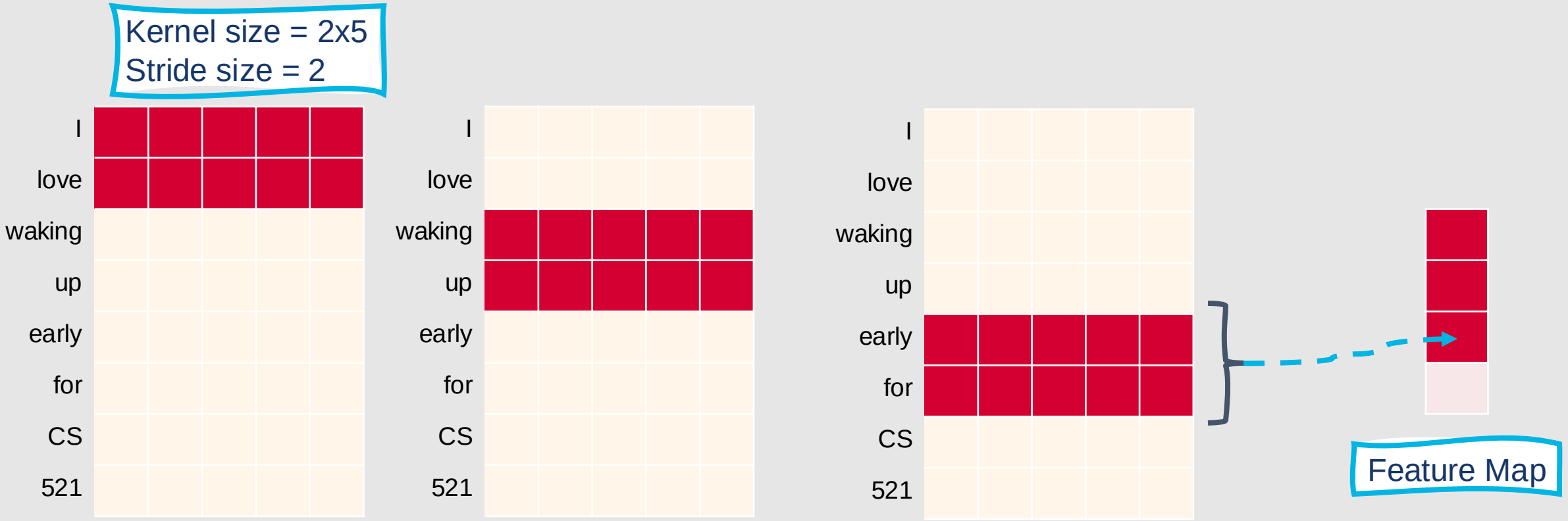
After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.



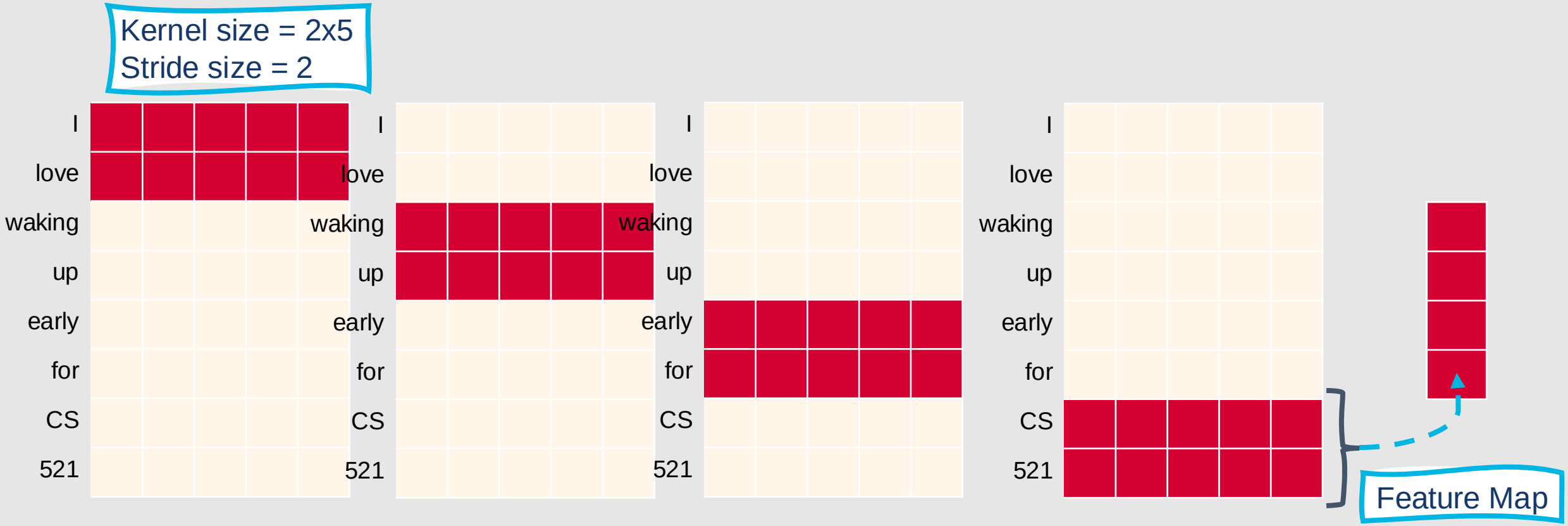
After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.

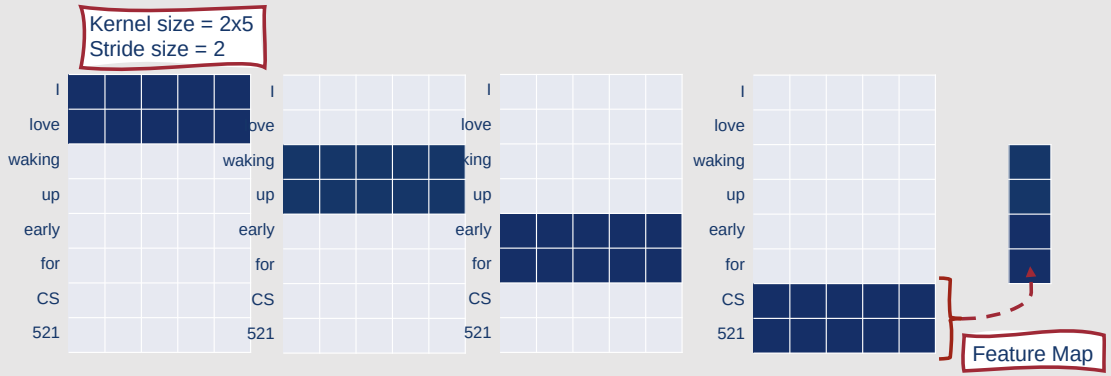


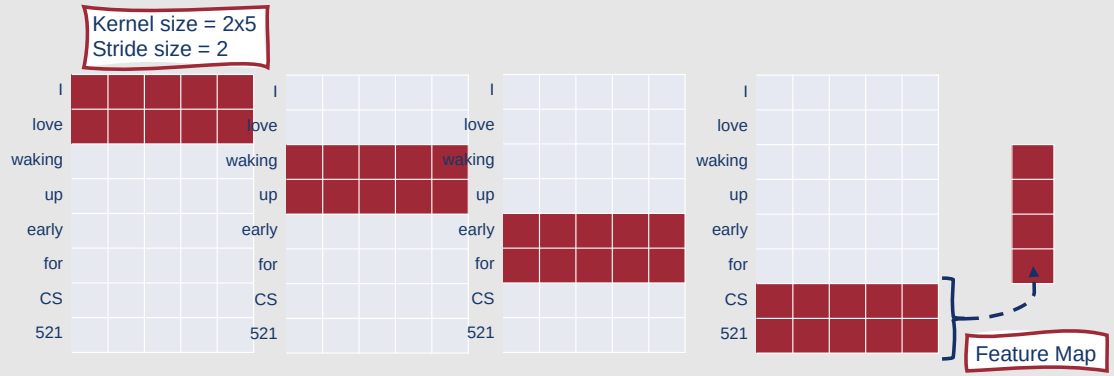
After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.



After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.

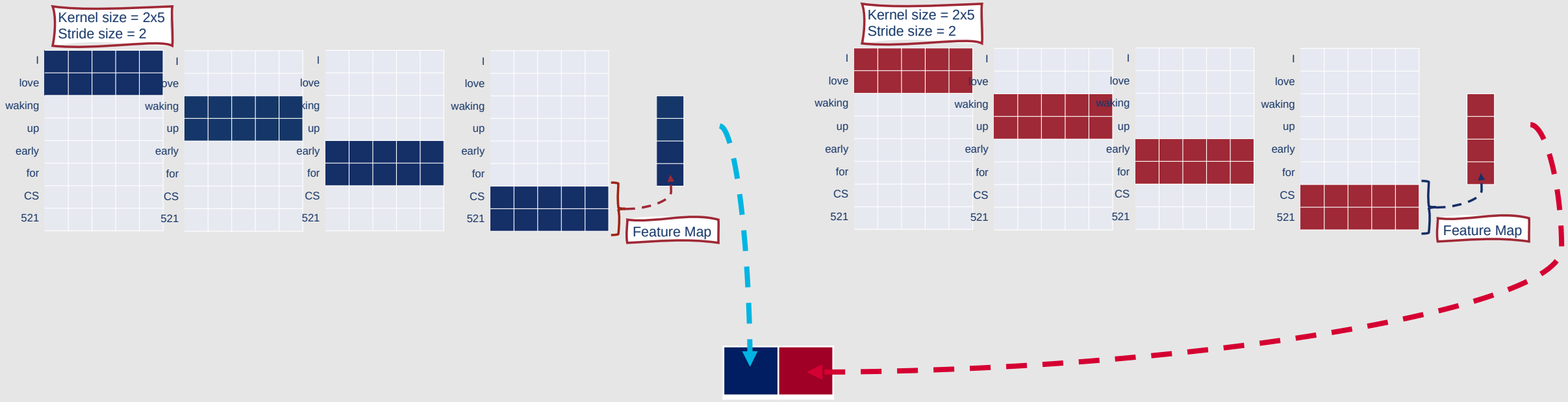




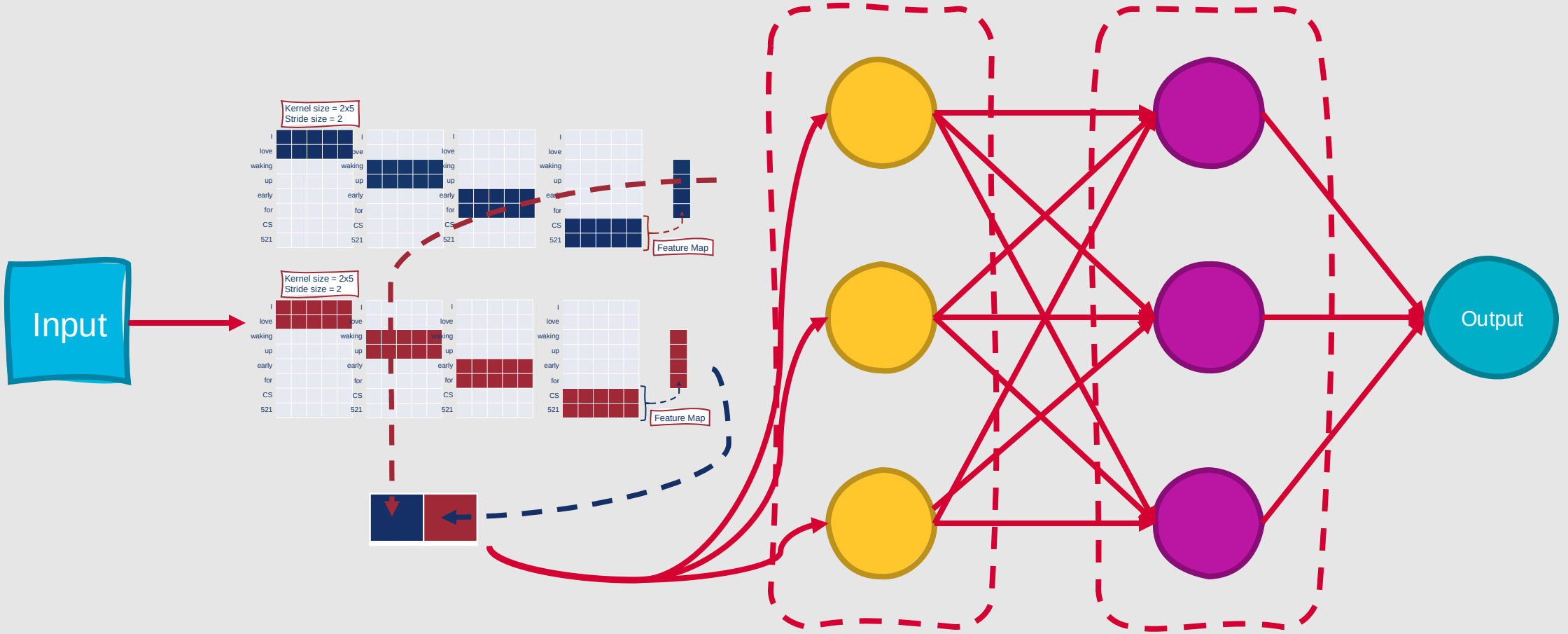


It’s common to extract multiple different feature maps from the same input.

Pooling Layers



The output from pooling layers is passed along as input to the rest of the network.



Review: Essentials of Deep Learning

- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Stacked models
- Bidirectional models

Recurrent Neural Networks (RNNs)

Built-in capacity to handle temporal information

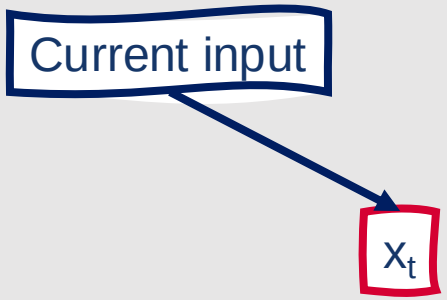
- Contain cycles within their connections, where the value of a unit is dependent upon outputs from previous timesteps

Can accept variable length inputs without the use of fixed-size windows

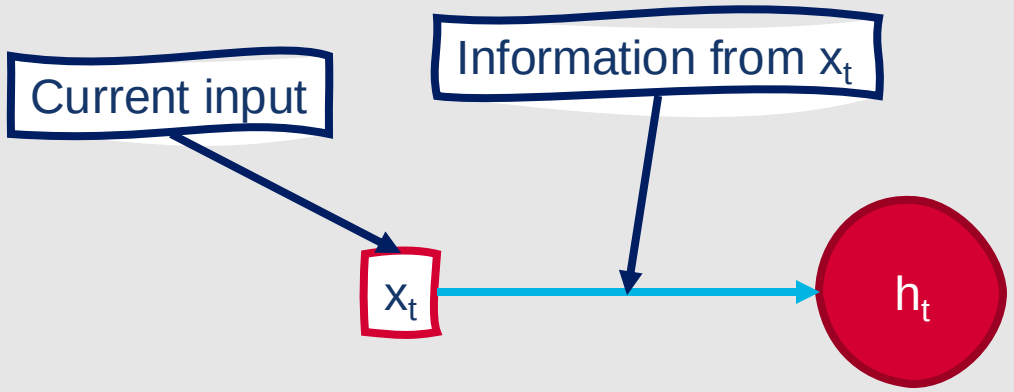
Many varieties exist

- “Vanilla” RNNs
- Long short-term memory networks (LSTMs)
- Gated recurrent units (GRUs)

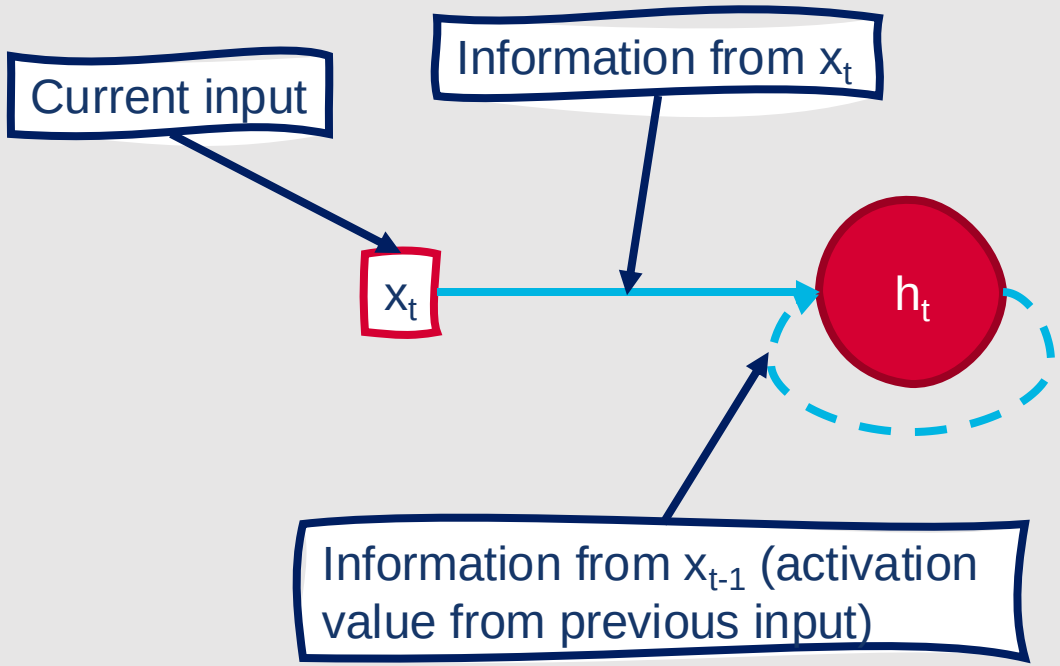
Vanilla RNN Unit



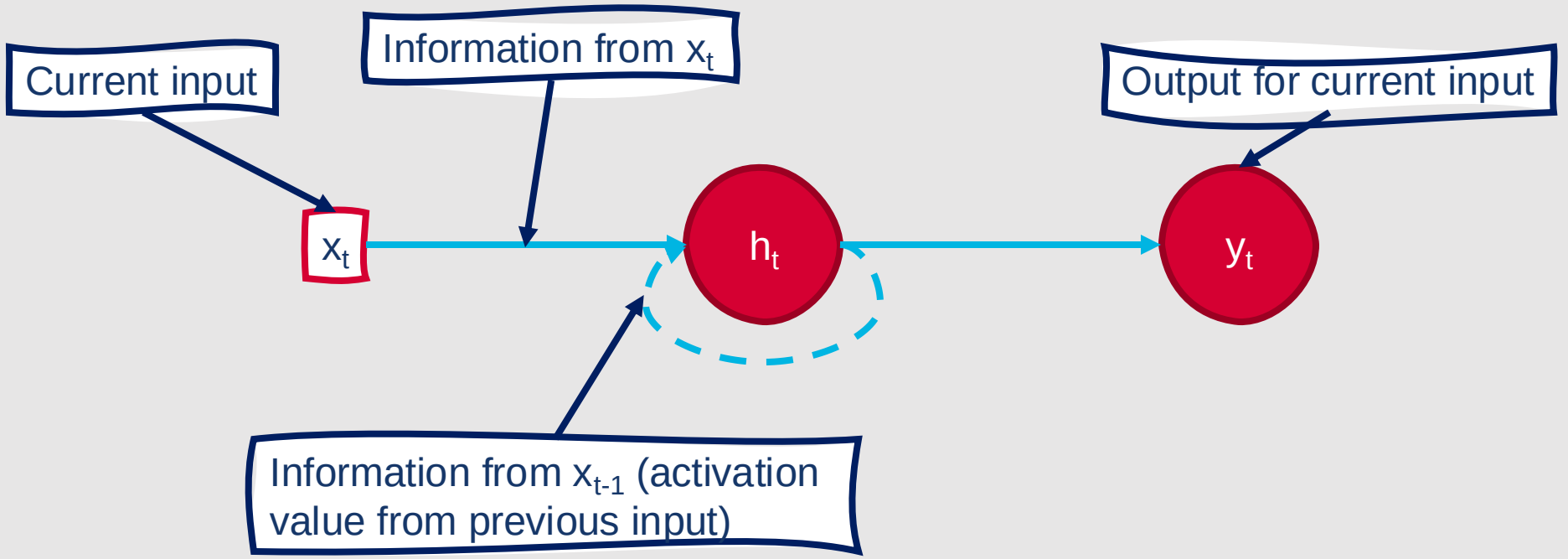
Vanilla RNN Unit



Vanilla RNN Unit



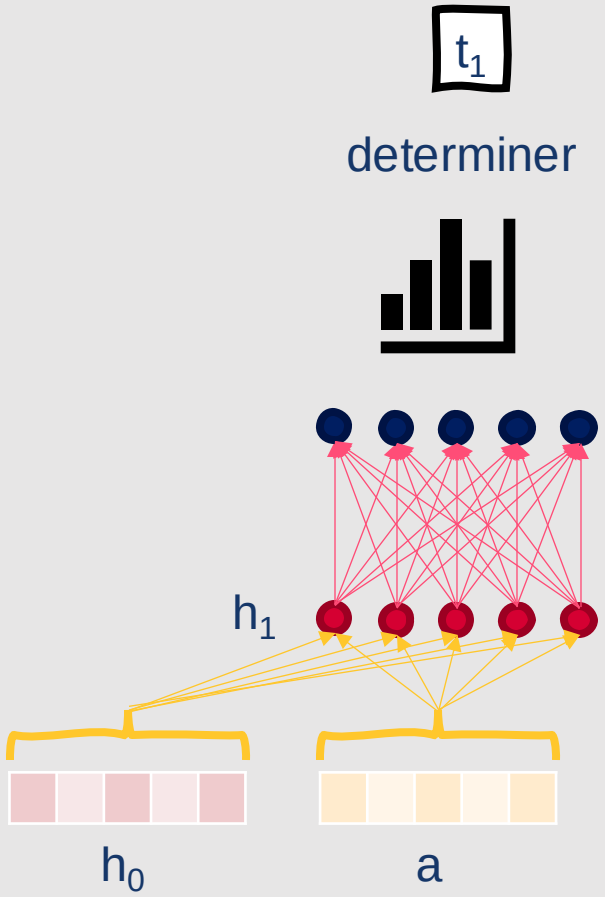
Vanilla RNN Unit



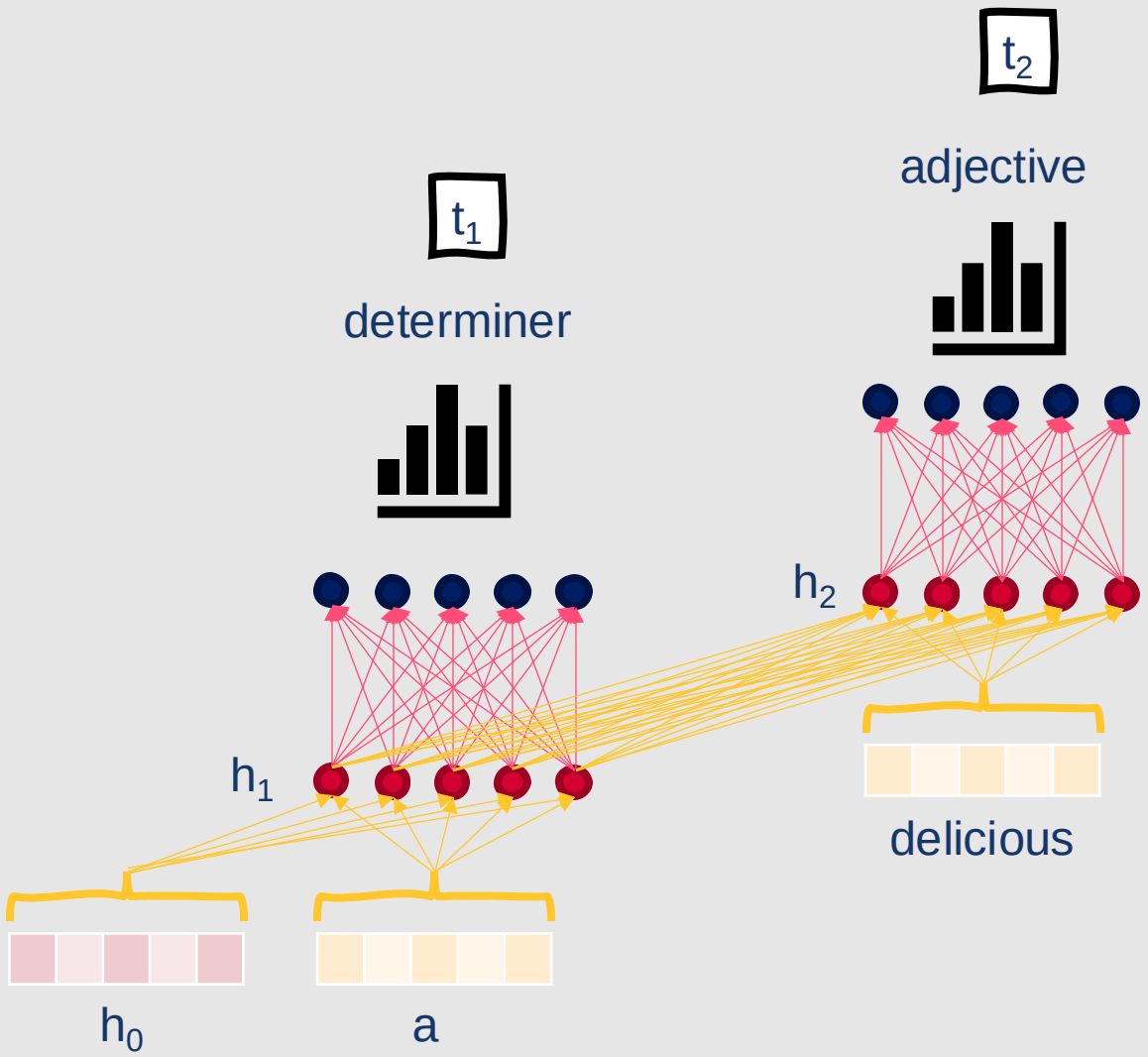
Sequence processing models like RNNs are also useful for many classification problems.

- **Sequence Labeling Tasks:** Given a fixed set of labels, assign a label to each element of a sequence
  - Example: Part-of-speech tagging
- Inputs → word embeddings
- Outputs → label probabilities generated by the softmax (or other activation) function over the set of all labels
- **Sequence Classification Tasks:** Given an input sequence, assign the entire sequence to a class (rather than the individual tokens within it)

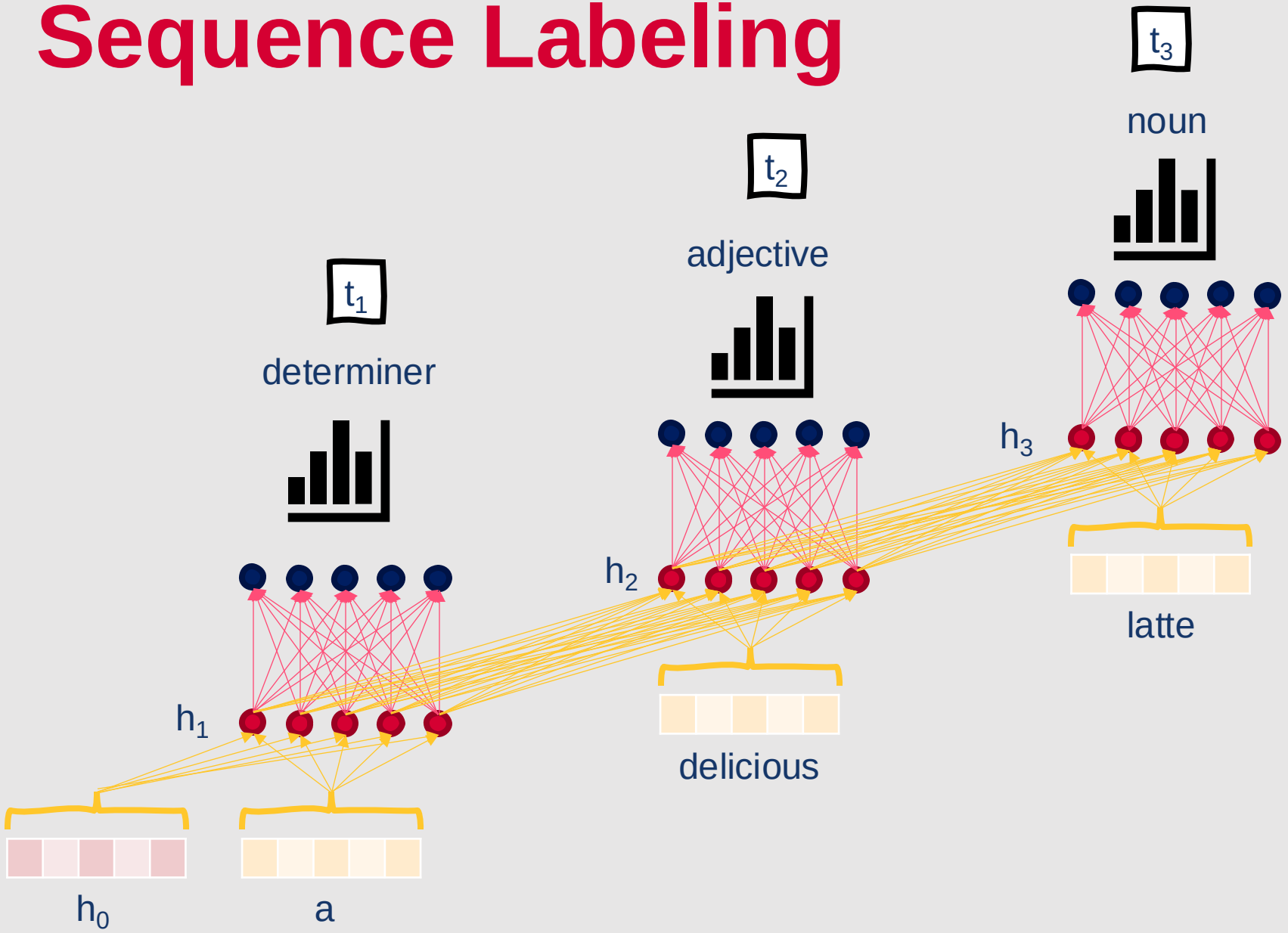
Sequence Labeling



Sequence Labeling



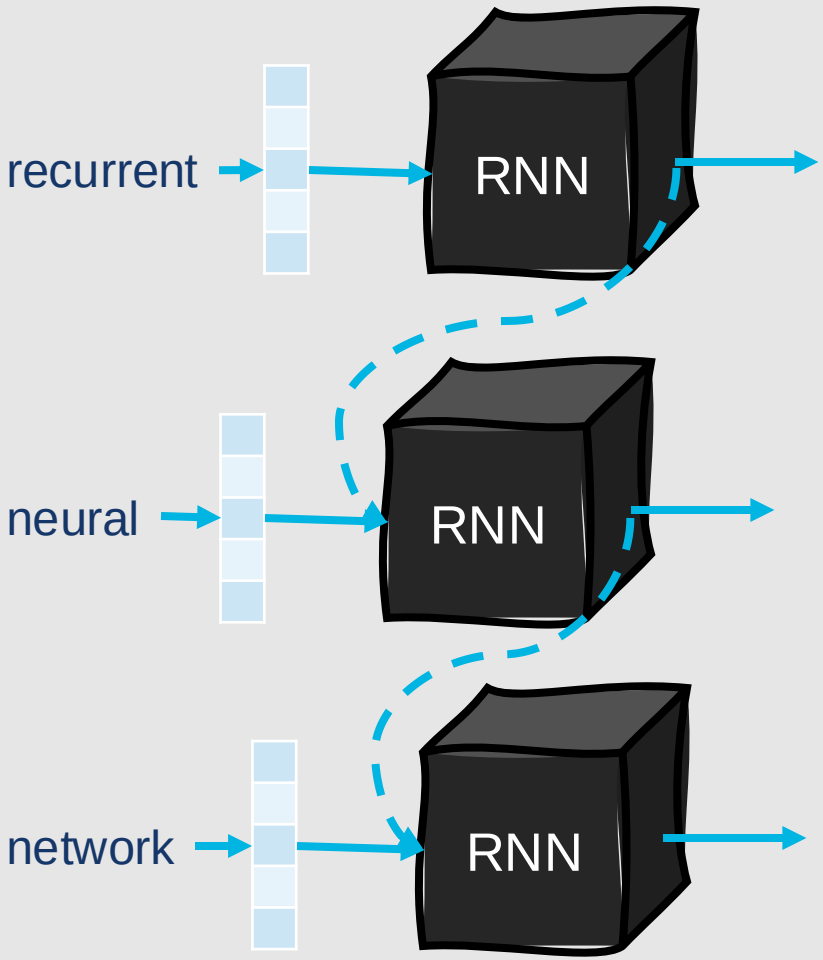
Sequence Labeling



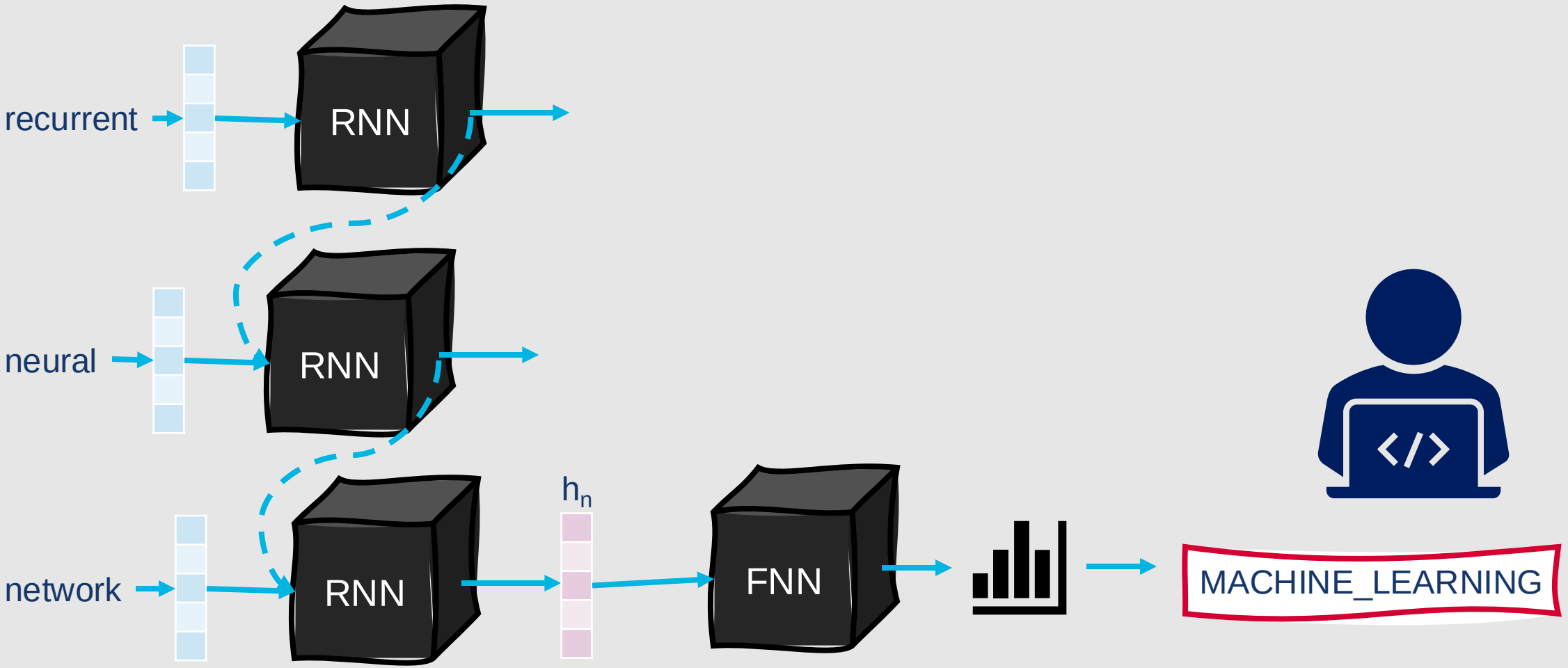
How to use RNNs for sequence classification?

1. Pass the sequence through an RNN one word at a time, as usual
2. Assume that the hidden layer for the final word, $\mathbf{h}_n$, acts as a compressed representation of the entire sequence
3. Use $\mathbf{h}_n$ as input to a subsequent feedforward neural network
4. Choose a class via softmax over all the possible classes

Sequence Classification



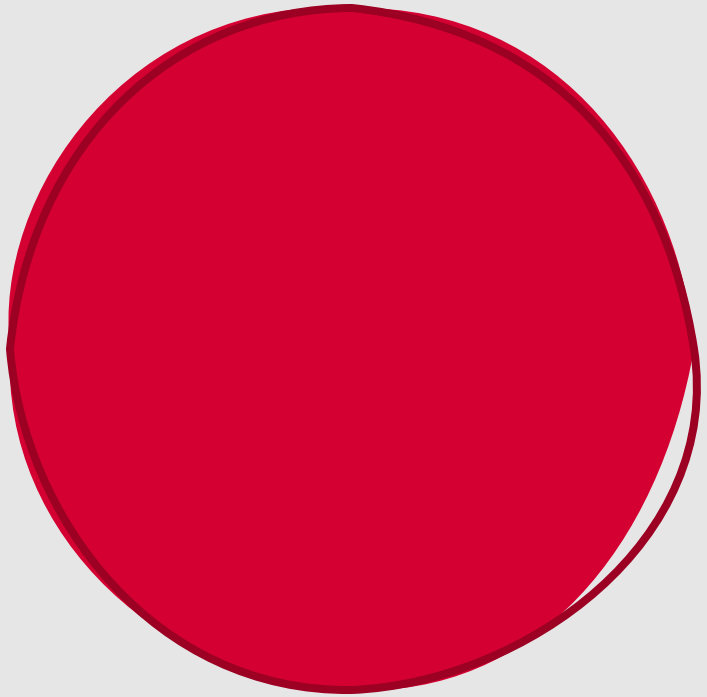
Sequence Classification



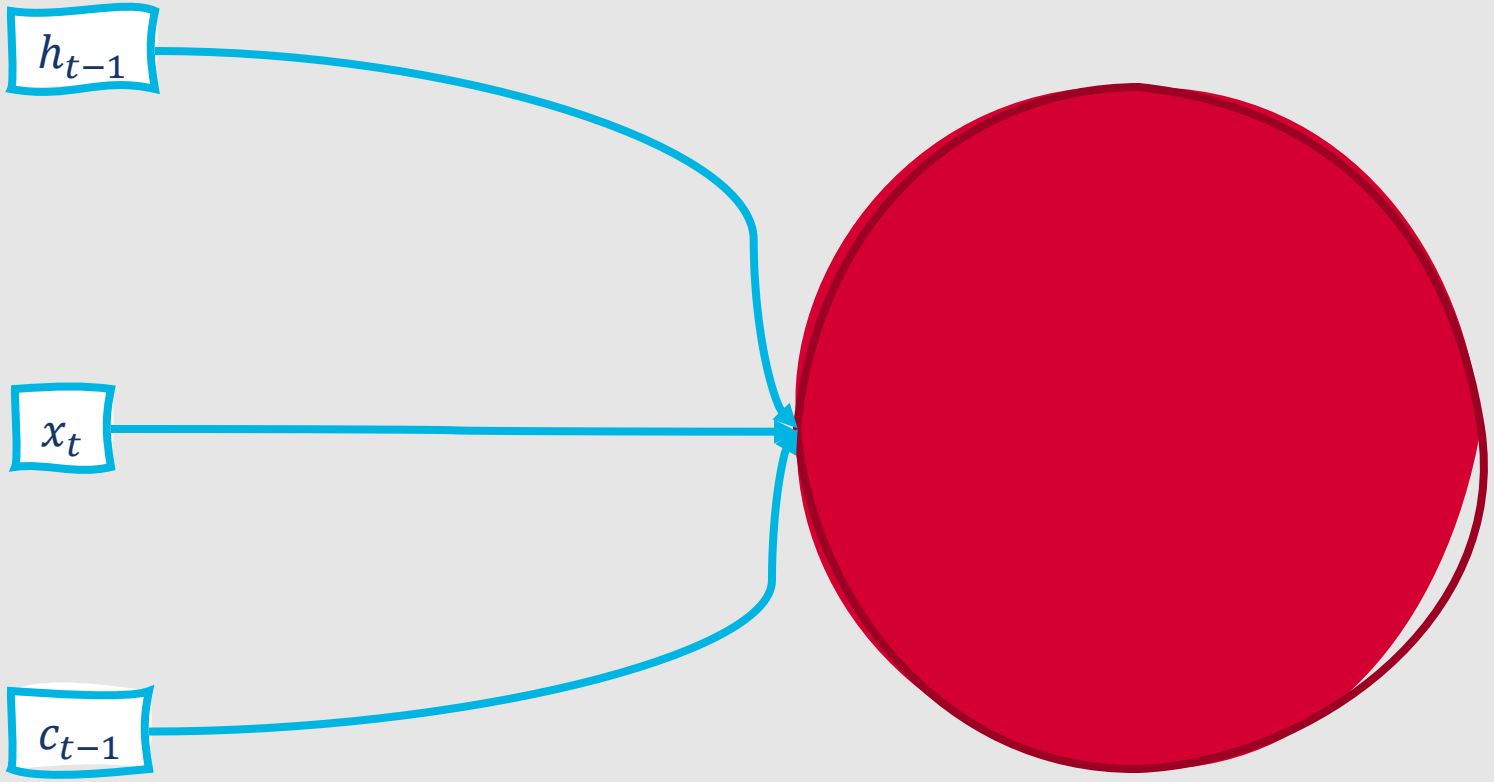
Long Short-Term Memory Networks (LSTMs)

- **Remove information** no longer needed from the context, and **add information** likely to be needed later
- Do this by:
  - Adding an **explicit context layer** to the architecture
  - This layer controls the flow of information into and out of network layers using specialized neural units called **gates**

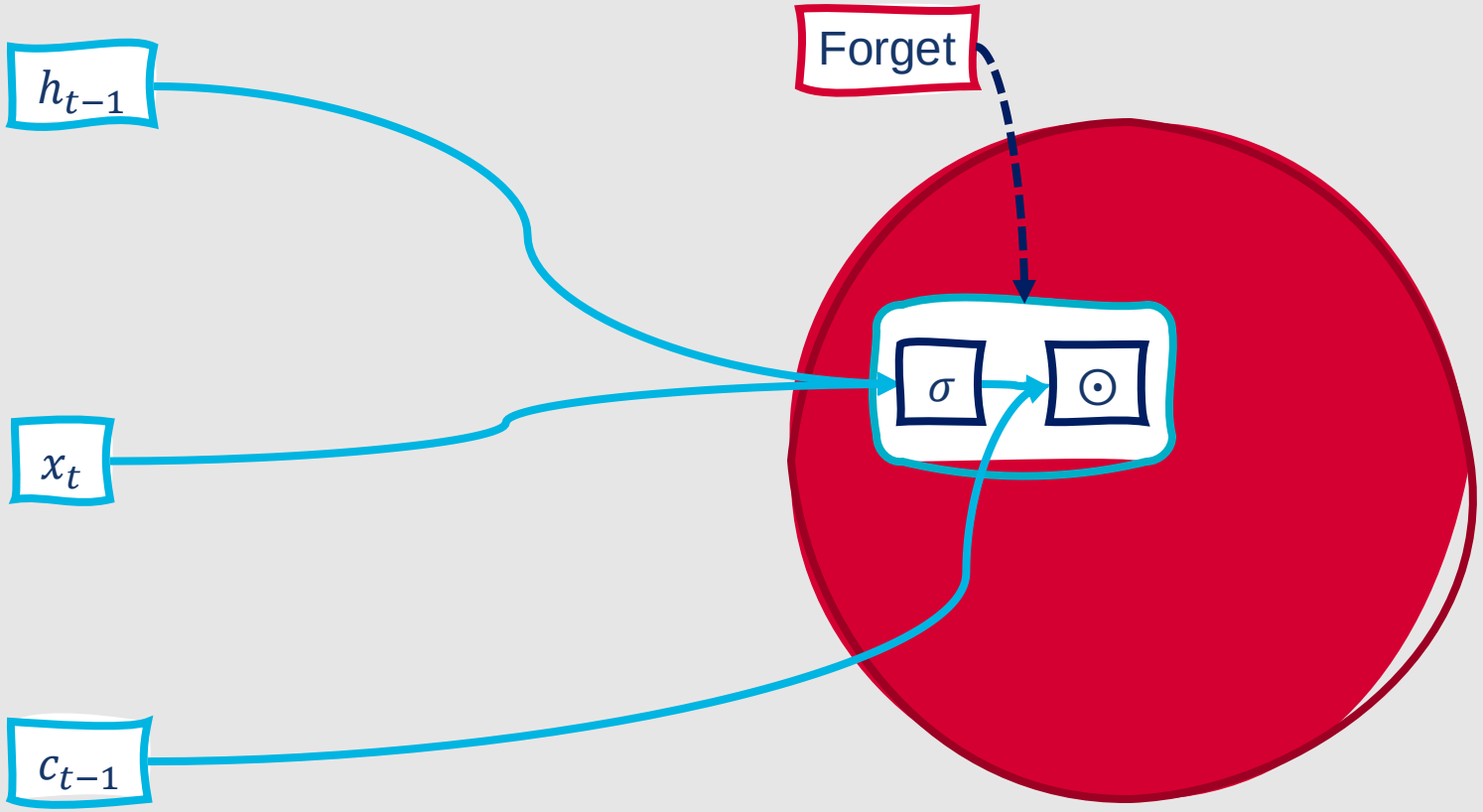
What does this process look like in a single LSTM unit?



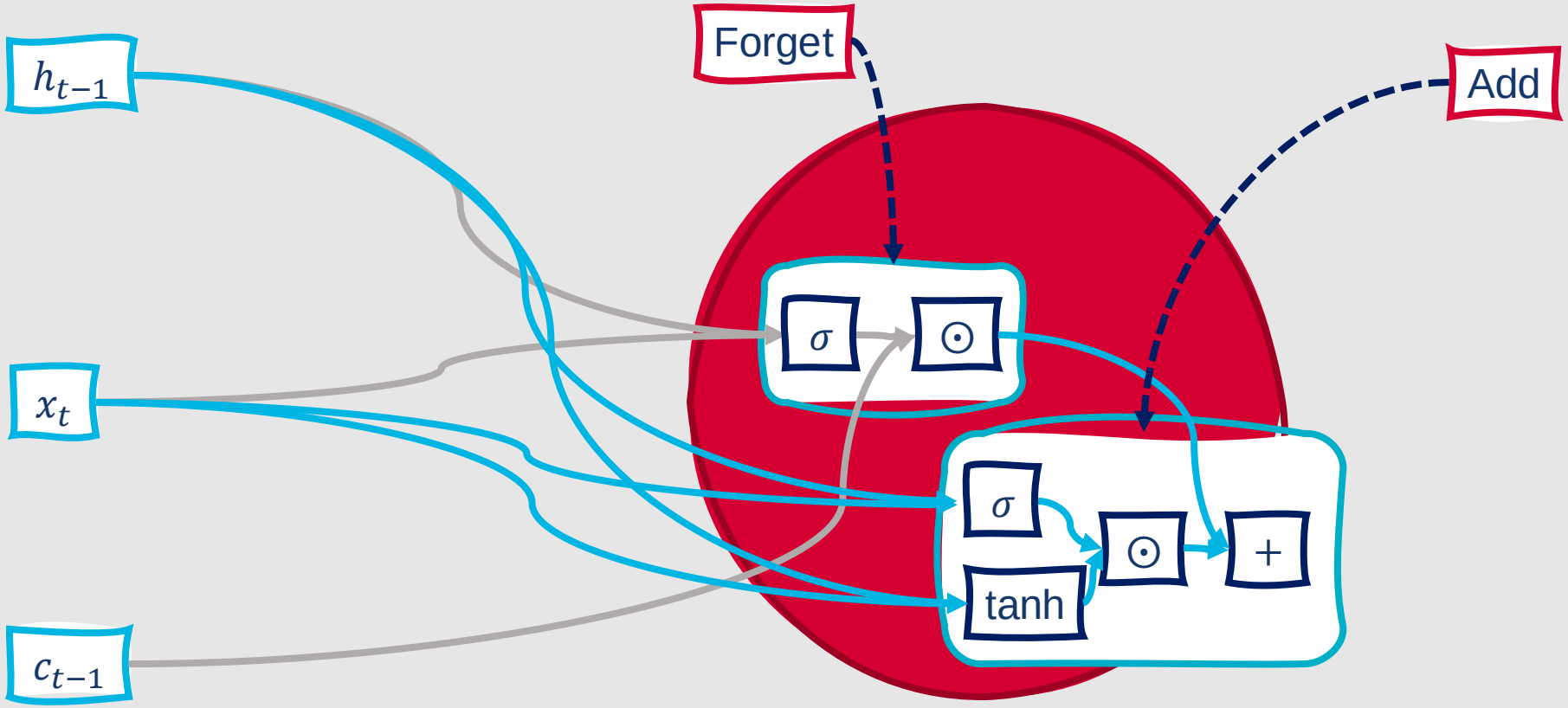
What does this process look like in a single LSTM unit?



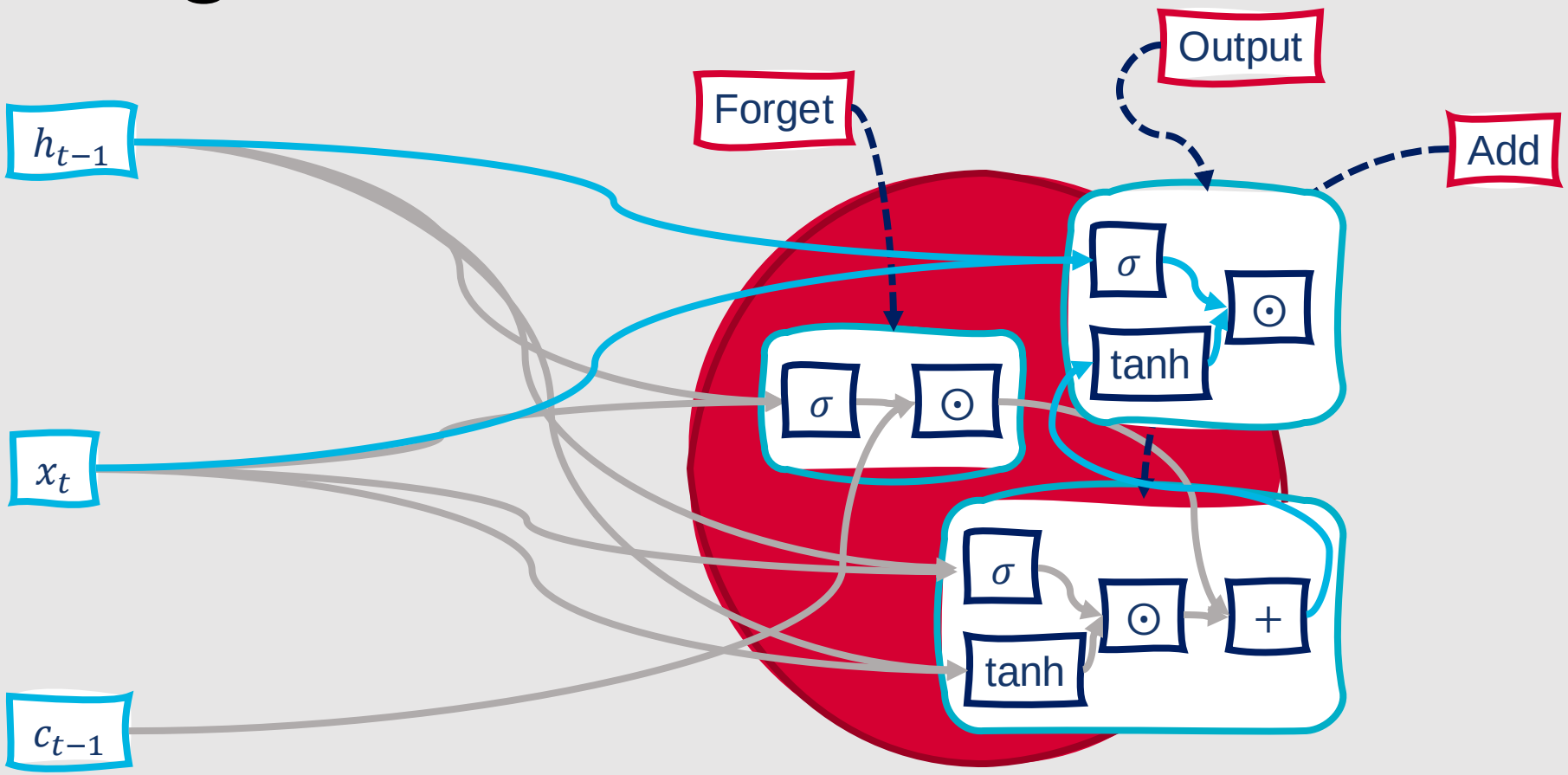
What does this process look like in a single LSTM unit?



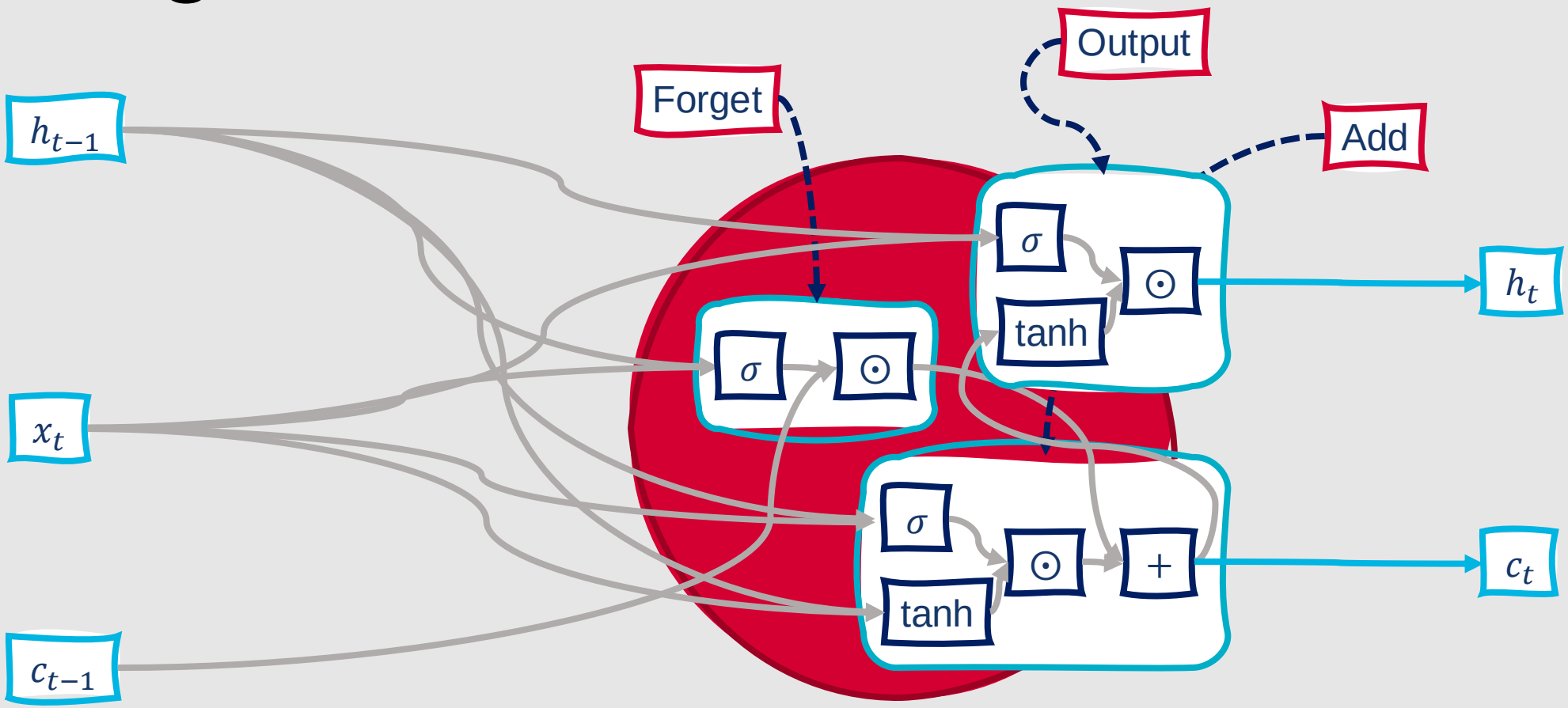
What does this process look like in a single LSTM unit?



What does this process look like in a single LSTM unit?



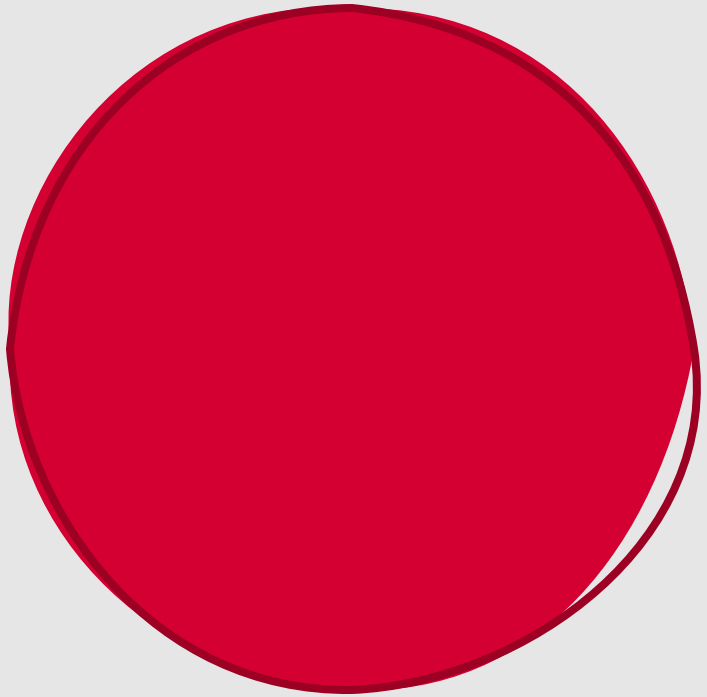
What does this process look like in a single LSTM unit?



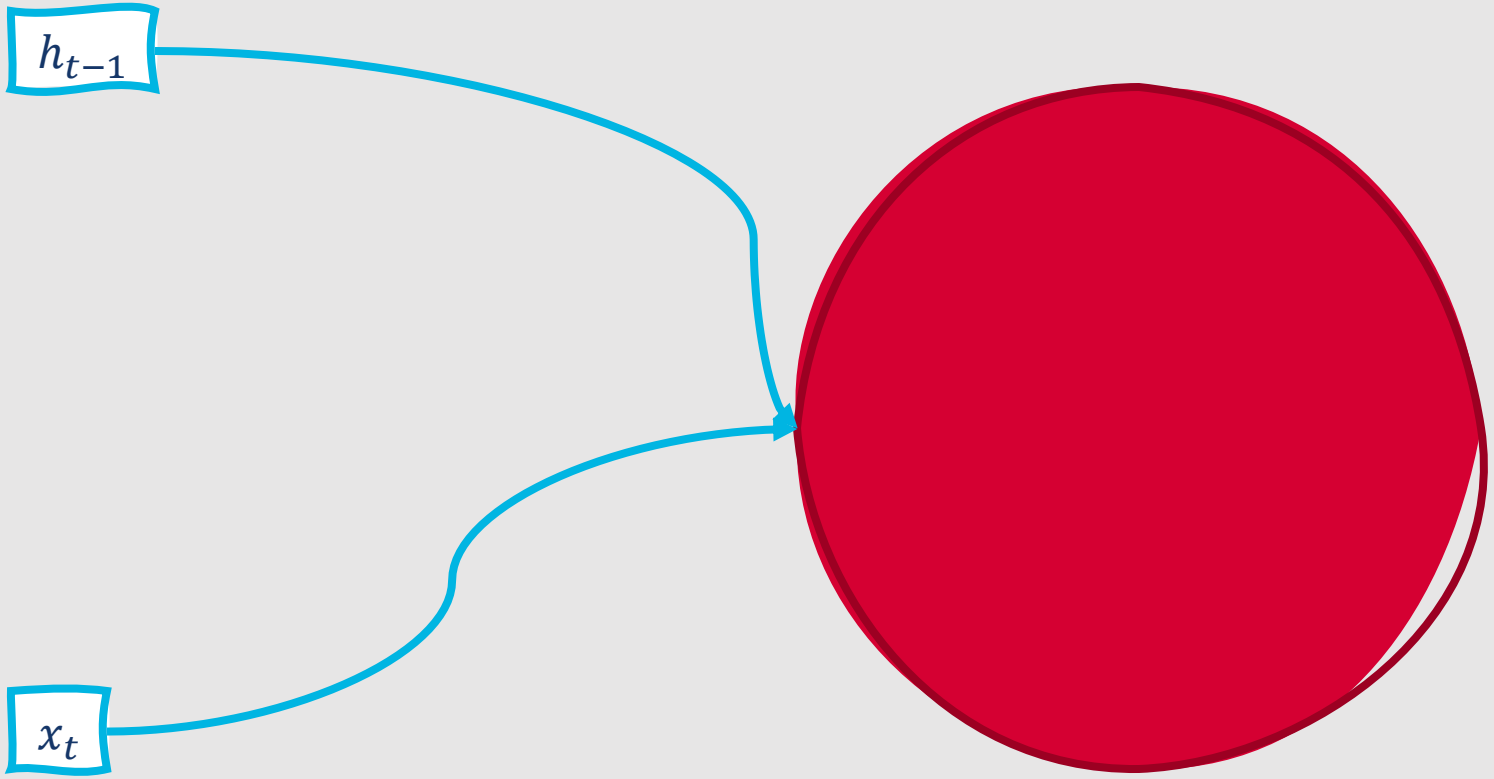
Gated Recurrent Units (GRUs)

- Also manage the context that is passed through to the next timestep, but do so using a simpler architecture than LSTMs
  - No separate context vector
  - Only two gates
    - **Reset** gate
    - **Update** gate
- Gates still use a similar design to that seen in LSTMs
  - **Feedforward layer** + **sigmoid activation** + **pointwise multiplication** with the layer being gated, resulting in a **binary-like mask**

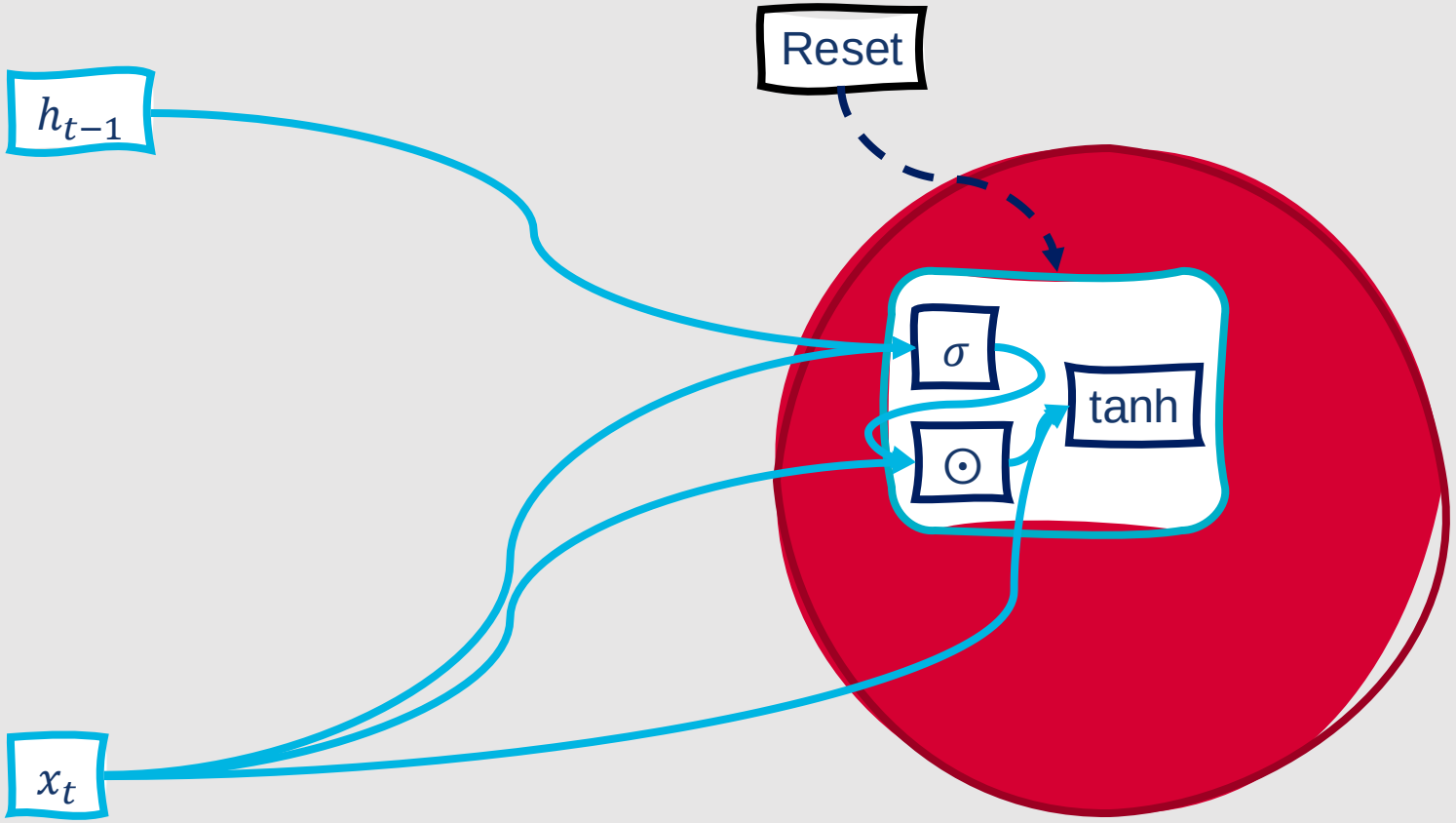
What does this process look like in a single GRU unit?



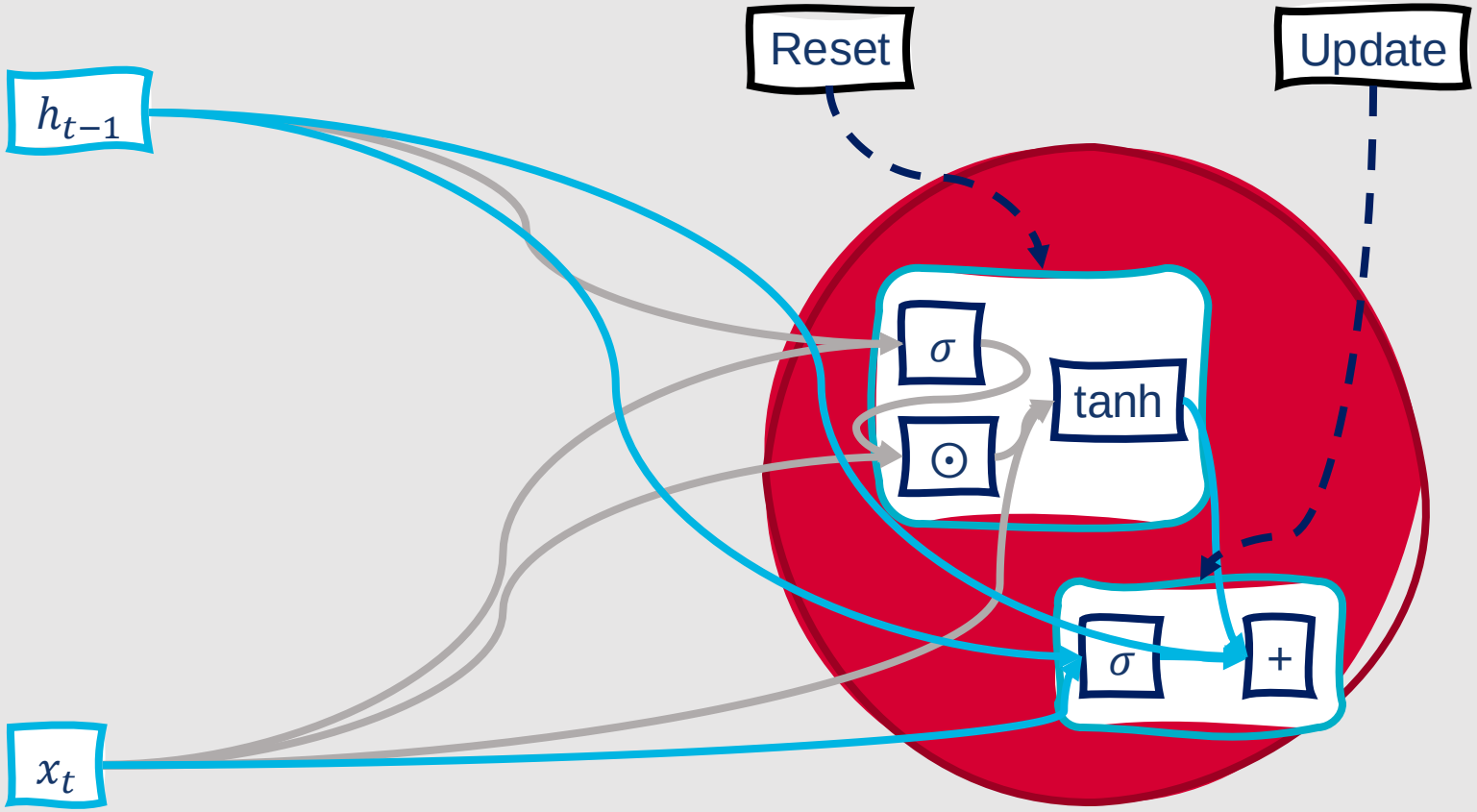
What does this process look like in a single GRU unit?



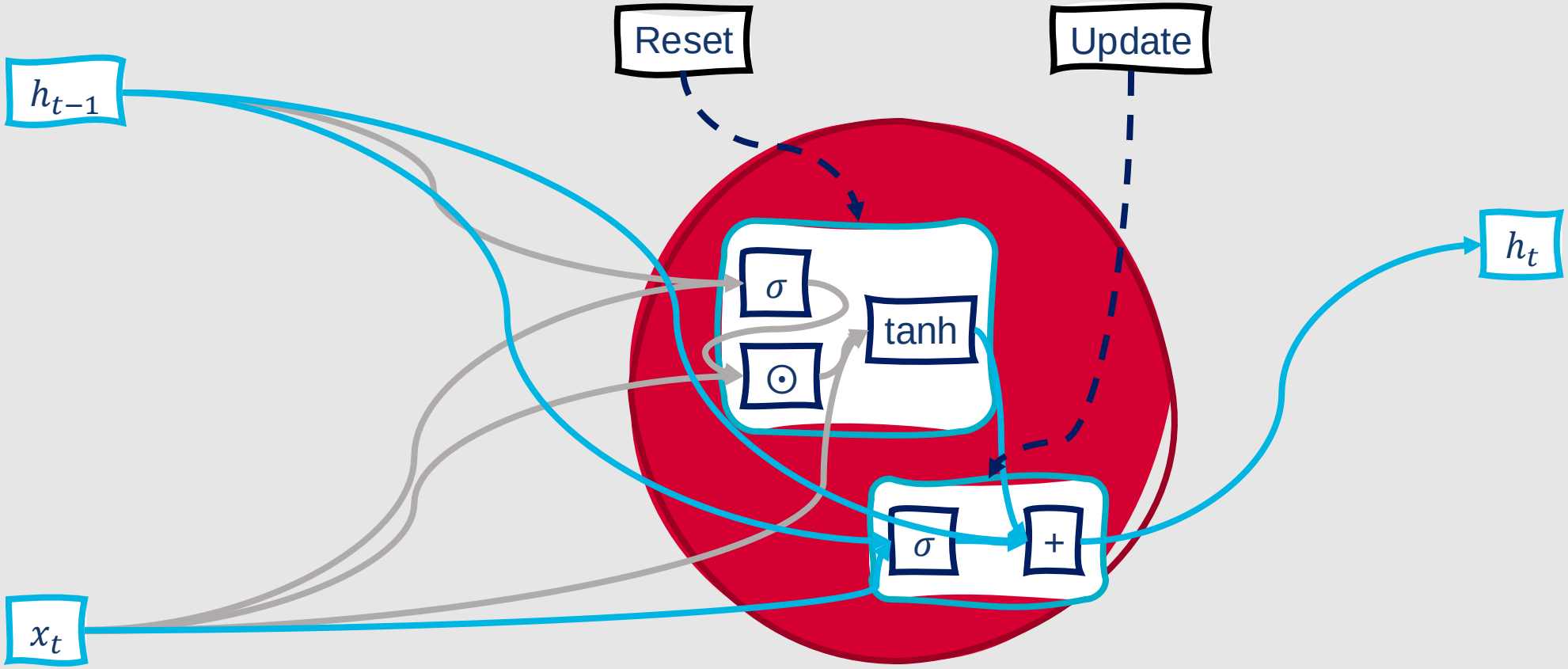
What does this process look like in a single GRU unit?



What does this process look like in a single GRU unit?



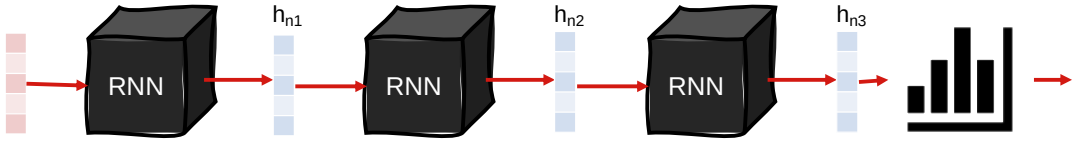
What does this process look like in a single GRU unit?



Review: Essentials of Deep Learning

- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Stacked models
- Bidirectional models

Stacked RNNs



- Use the entire sequence of outputs from one RNN as the input sequence to another
- Capable of outperforming single-layer networks
- Why?
  - Having more layers allows the network to learn representations at differing levels of **abstraction** across layers
    - Early layers → more fundamental properties
    - Later layers → more meaningful groups of fundamental properties
- Optimal number of RNNs to stack together?
  - Depends on application and training set
- More RNNs in the stack → increased training costs

Review: Essentials of Deep Learning

- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Stacked models
- Bidirectional models

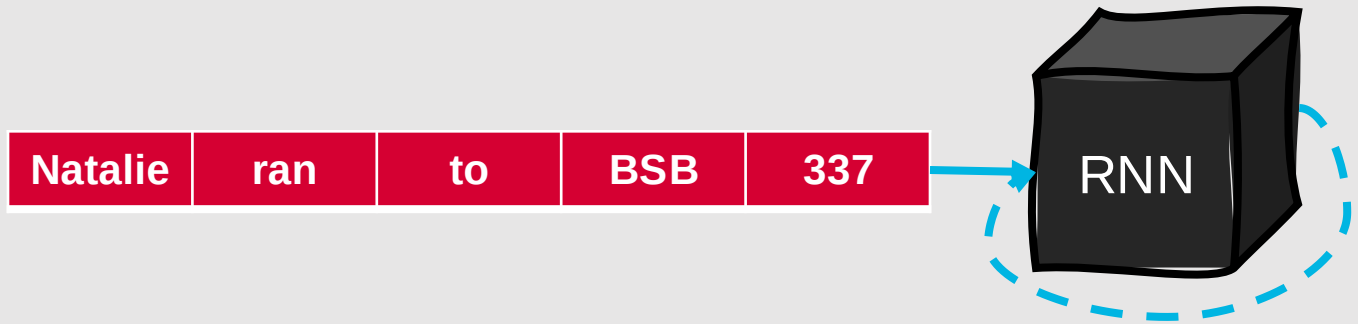
Bidirectional RNNs

- How can we make use of information both before and after the current timestep?
  - Train an RNN on an input sequence in **reverse**
    - $h_t^b = RNN_{backward}(x_t^n)$
      - h_t^b corresponds to information from the current timestep to the end of the sequence
  - **Combine** the forward and backward networks

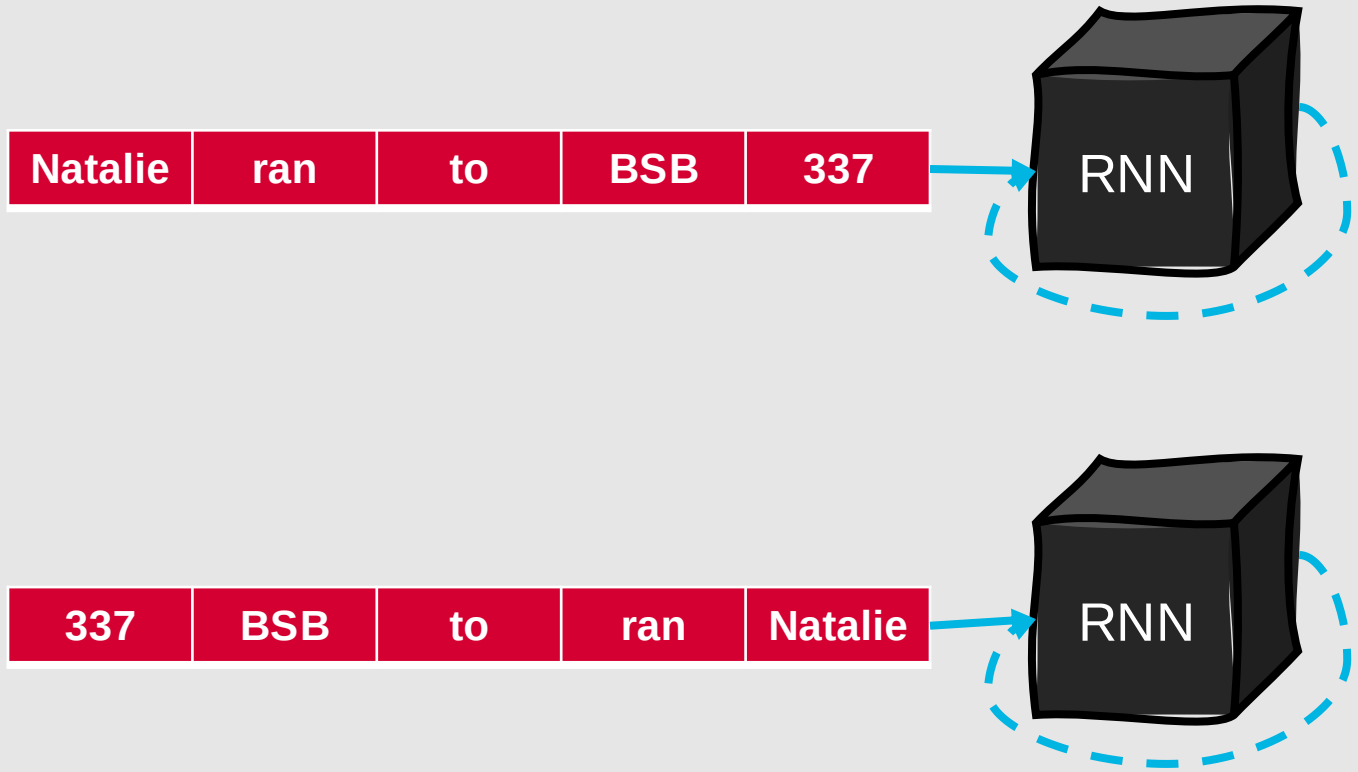
Bidirectional RNNs

- Two independent RNNs
  - One where the input is processed from start to end
  - One where the input is processed from end to start
- Outputs combined into a single representation that captures both the prior and future contexts of an input at each timestep
  - $h_t = h_t^f \oplus h_t^b$
- How to combine the contexts?
  - Concatenation
  - Element-wise addition, multiplication, etc.

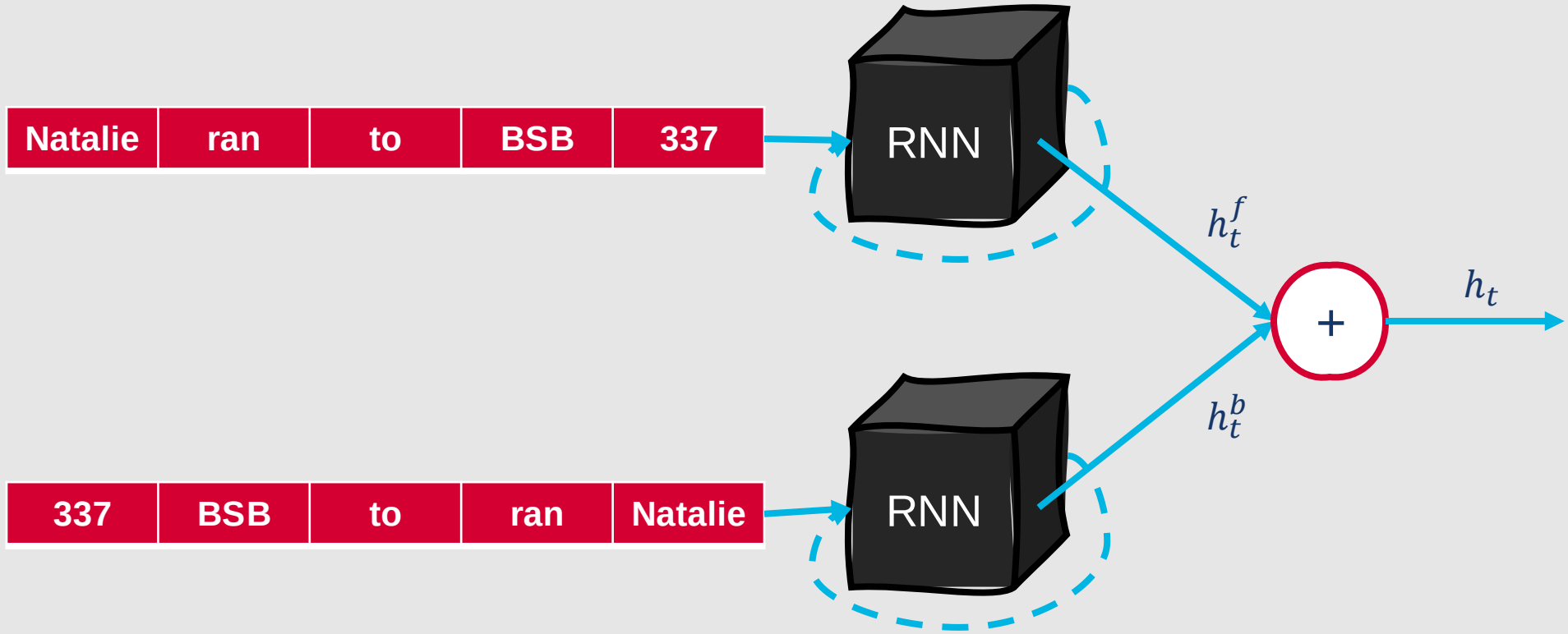
Bidirectional RNNs



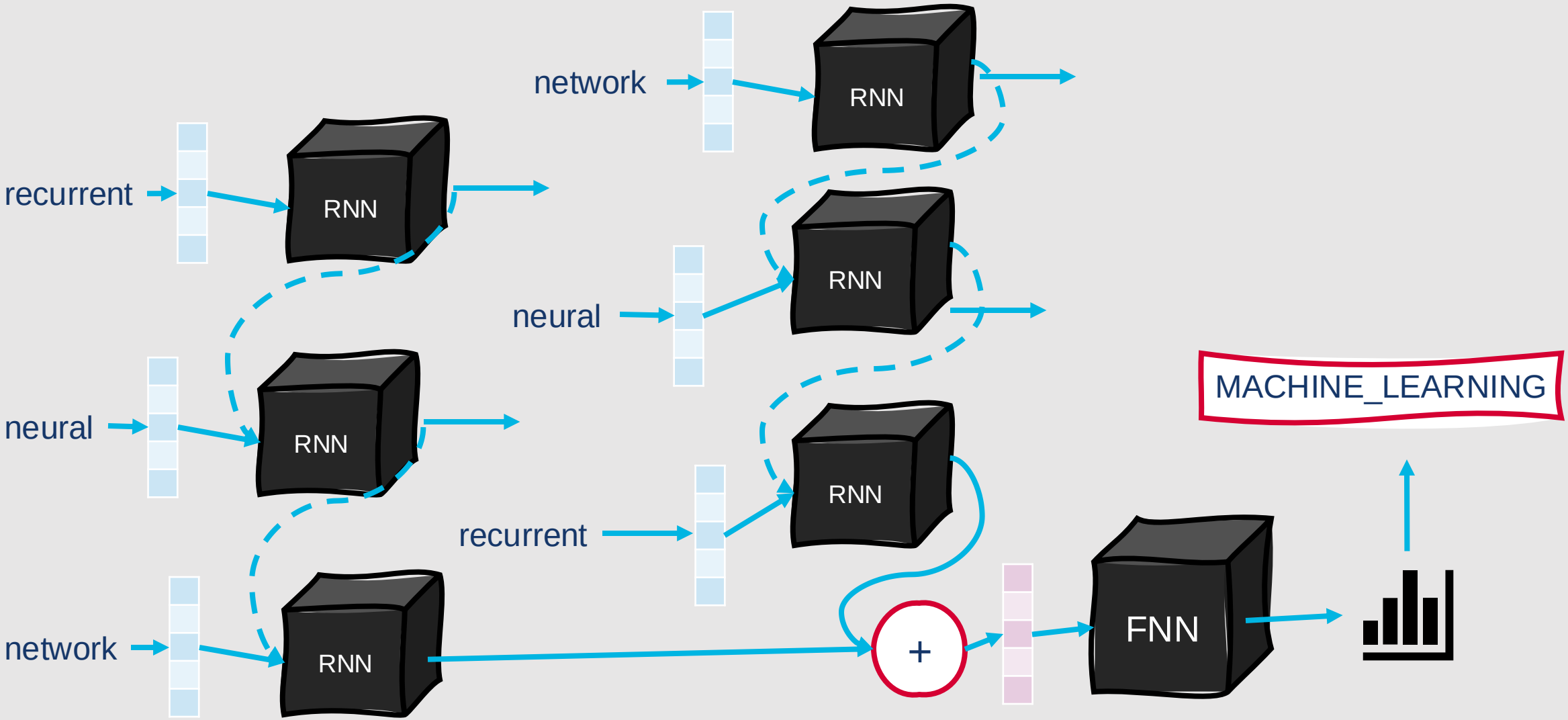
Bidirectional RNNs



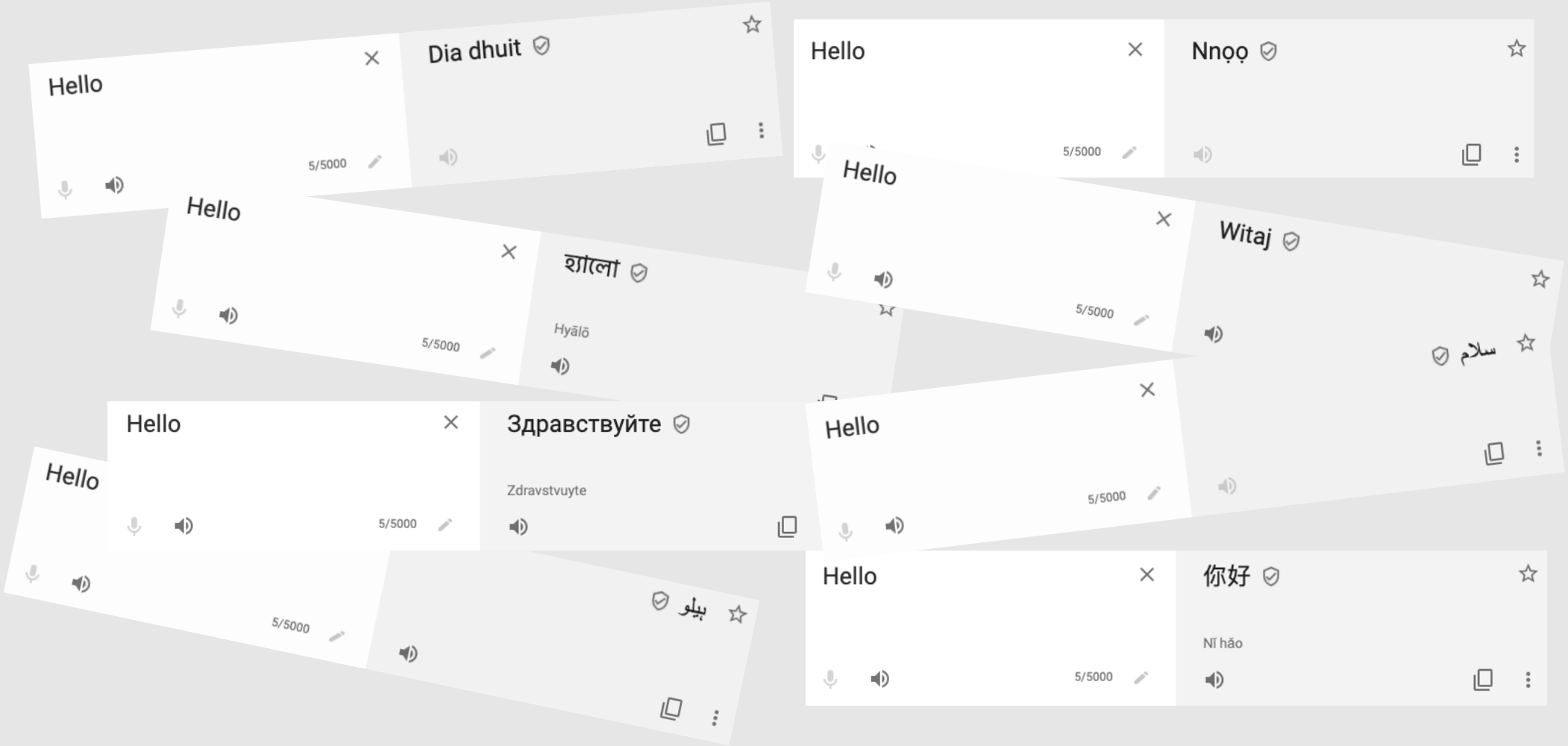
Bidirectional RNNs

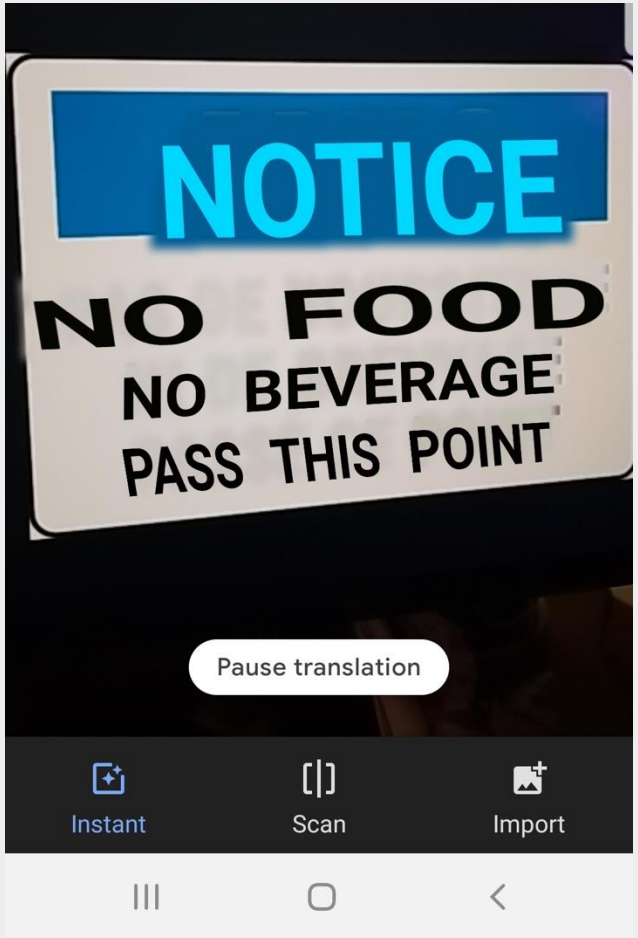


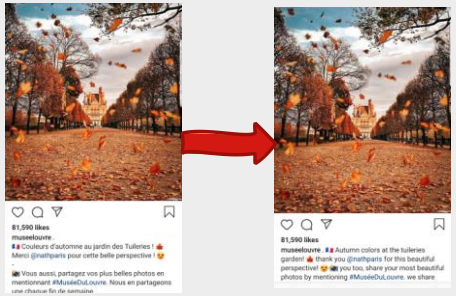
Sequence Classification with a Bidirectional RNN



Machine Translation: The process of automatically converting a text from one language to another.







Machine translation is increasingly ubiquitous, but also challenging for many reasons.

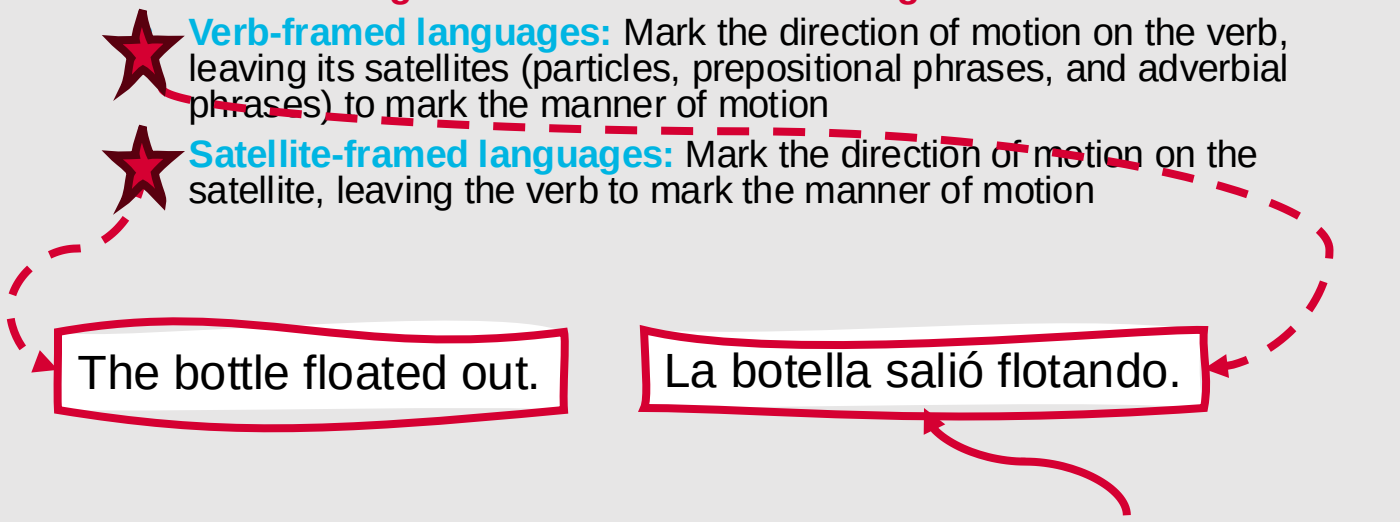
- Structural and lexical differences between languages
- Differences in word order
- Morphological differences
- Stylistic and cultural differences

Cross-Linguistic Similarities and Differences

- **Typological Differences:**
  - Systematic structural differences between languages
- **Morphological Differences:**
  - Number of morphemes per word
    - **Isolating languages:** Each word generally has one morpheme
    - **Polysynthetic languages:** Each word may have many morphemes
  - Degree to which morphemes can be segmented
    - **Agglutinative languages:** Morphemes have well-defined boundaries
    - **Fusion languages:** Morphemes may be conflated with one another

Cross-Linguistic Similarities and Differences

- **Syntactic Differences:**
  - Primary difference between languages: Word order
    - **SVO languages:** Verb tends to come between the subject and object
    - **SOV languages:** Verb tends to come at the end of basic clauses
    - **VSO languages:** Verb tends to come at the beginning of basic clauses
  - Languages with similar basic word order also tend to share other similarities
    - SVO languages generally have prepositions
    - SOV languages generally have postpositions
- **Differences in Argument Structure and Linking**
  - **Verb-framed languages:** Mark the direction of motion on the verb, leaving its satellites (particles, prepositional phrases, and adverbial phrases) to mark the manner of motion
  - **Satellite-framed languages:** Mark the direction of motion on the satellite, leaving the verb to mark the manner of motion



Cross-Linguistic Similarities and Differences

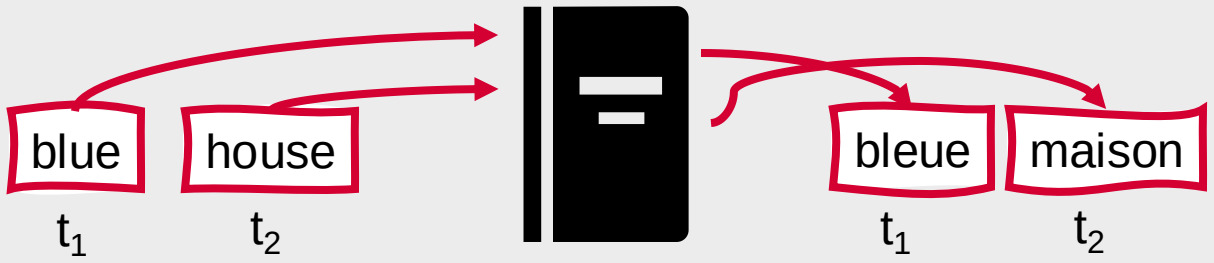
- **Differences in Permissible Omissions:**
  - **Pro-Drop languages:** Can omit pronouns when talking about certain referents
  - Some pro-drop languages permit more pronoun omission than others
    - **Referentially dense** and **sparse** languages
  - Converting text from pro-drop languages (e.g., Japanese) to non-pro-drop languages (e.g., English) requires that all missing pronoun locations are identified and their appropriate **anaphors** recovered
- **Differences in noun-adjective order**
  - Blue house → Maison bleue
- **Differences in homonymy and polysemy**
- **Differences in grammatical constraints**
  - Some languages require gender for nouns
  - Some languages require gender for pronouns
- **Lexical gaps**
  - No word or phrase in the target language can express the meaning of a word in the source language

Machine Translation

- Classical Machine Translation
  - Direct translation
  - Transfer approaches
  - Interlingua approaches
  - Statistical methods
- Modern Machine Translation
  - Encoder-decoder models

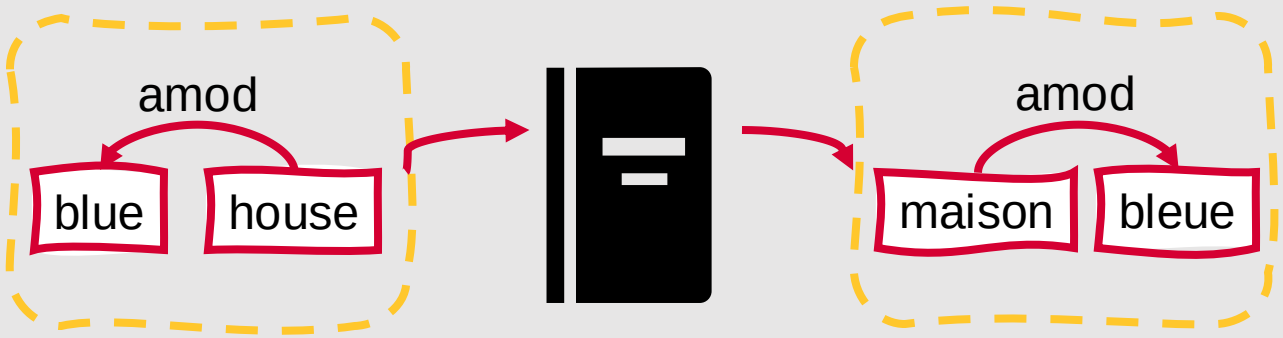
Classical Machine Translation

- **Direct translation**
  1. Take a large bilingual dictionary
  2. Proceed through the source text word by word
  3. Translate each word according to the dictionary
- No intermediate structures
- Simple reordering rules may be applied
  - For example, moving adjectives so that they are after nouns when translating from English to French
- Dictionary entries may be relatively complex
  - Rule-based programs for translating a word to the target language

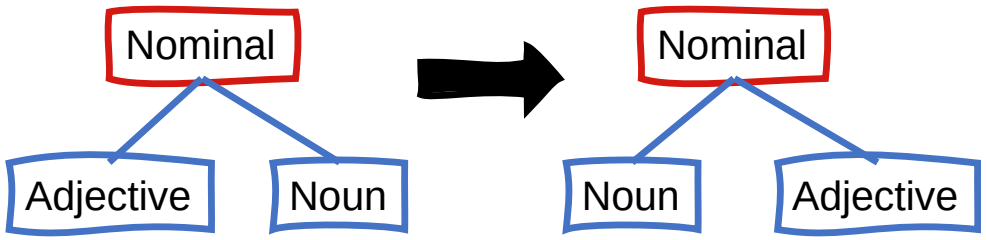


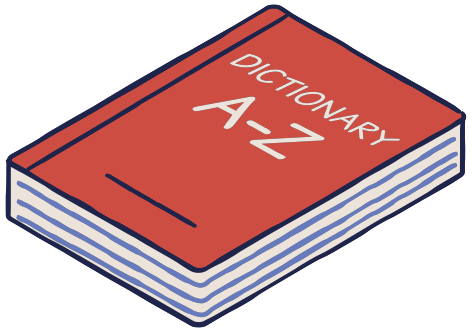
Classical Machine Translation

- **Transfer approaches**
  - Parse the input text
  - Apply rules to transform the source language parse structure into a target language parse structure
- Two subcategories of transformations:
  - **Syntactic** transfer
  - **Lexical** transfer



Transfer Approaches





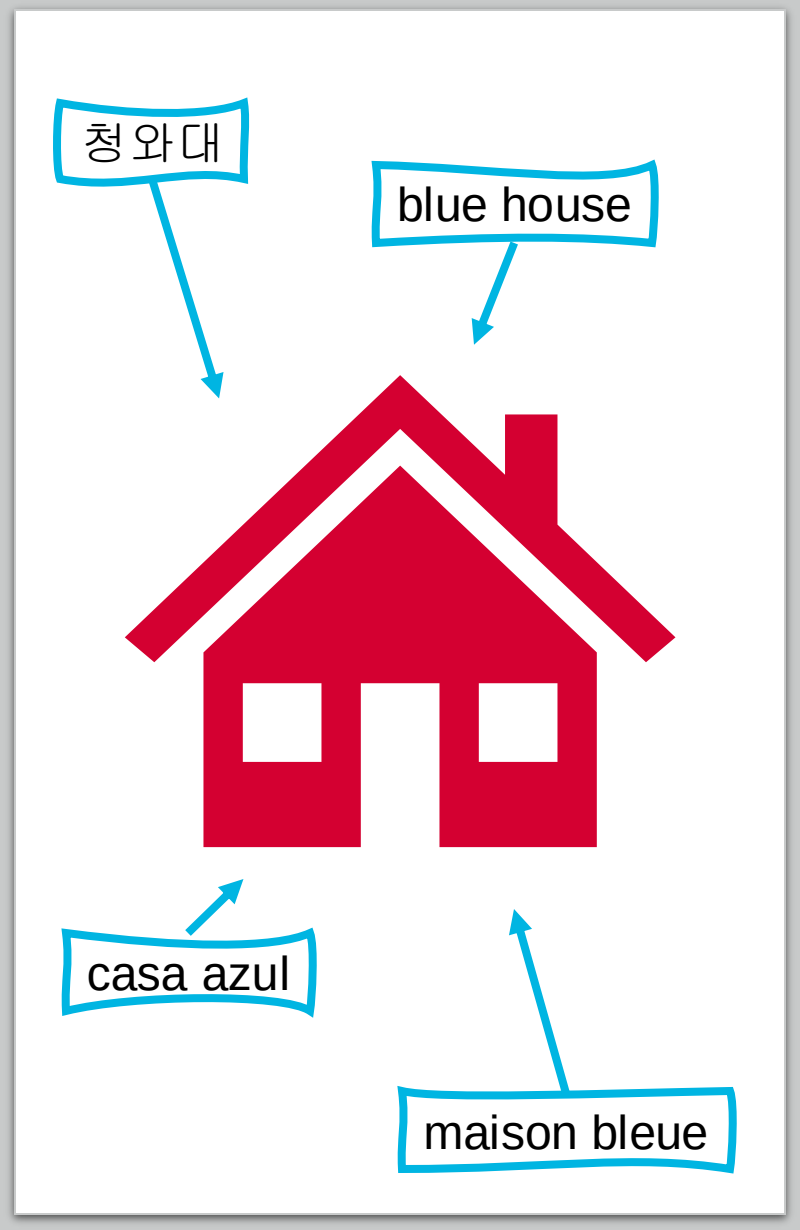
- **Syntactic Transfer:** Modifies the source parse tree to resemble the target parse tree
  - For some languages, may also include **thematic structures**
    - Directional or locative prepositional phrases vs. recipient prepositional phrases
- **Lexical Transfer:** Generally based on a bilingual dictionary
  - As with direct translation, dictionary entries can be complex to accommodate many possible translations

Classical Machine Translation

- **Interlingua approaches**
  - Convert the source language text into an abstract meaning representation
  - Generate the target language text based on the abstract meaning representation
- Require more analysis work than transfer approaches
  - Semantic analysis
  - Sentiment analysis
- No need for syntactic or lexical transformations

Interlingua Approaches

- Goal: Represent all sentences that mean the same thing in the same way, regardless of language
- What kind of representation scheme should be used?
  - Classical approaches:
    - First-order logic
    - Semantic primitives
    - Event-based representation
  - More recently, neural machine translation models follow a similar intuition



When to use each classical approach?

Direct Translation

- Pros:
  - Simple
  - Easy to implement
- Cons:
  - Cannot reliably handle long-distance reorderings
  - Cannot handle reorderings involving phrases or larger structures
  - Too focused on individual words

Transfer Approaches

- Pros:
  - Can handle more complex language phenomena than direct translation
- Cons:
  - Still not sufficient for many cases!

Interlingua Approaches

- Pros:
  - Direct mapping between meaning representation and lexical realization
  - No need for transformation rules
- Cons:
  - Extra (often unnecessary) work
    - Classical approaches require an exhaustive analysis and formalization of the semantics of the domain

Statistical Machine Translation

- Models automatically learn to map from the source language to the target language
  - No need for intermediate transformation rules
  - No need for an explicitly defined internal meaning representation
- Goal: Produce an output that maximizes some function representing translation **faithfulness** and **fluency**
- One possible approach: **Bayesian noisy channel model**
  - Assume a possible target language translation t_i and a source language sentence s
  - Select the translation t' from the set of all possible translations $t_i \in T$ that maximizes the probability $P(t_i|s)$, using Bayes' rule
    - $t' = \underset{t_i \in T}{\operatorname{argmax}} P(s|t_i)P(t_i)$
  - Often a **phrase-based translation model** is used

The Phrase-Based Translation Model

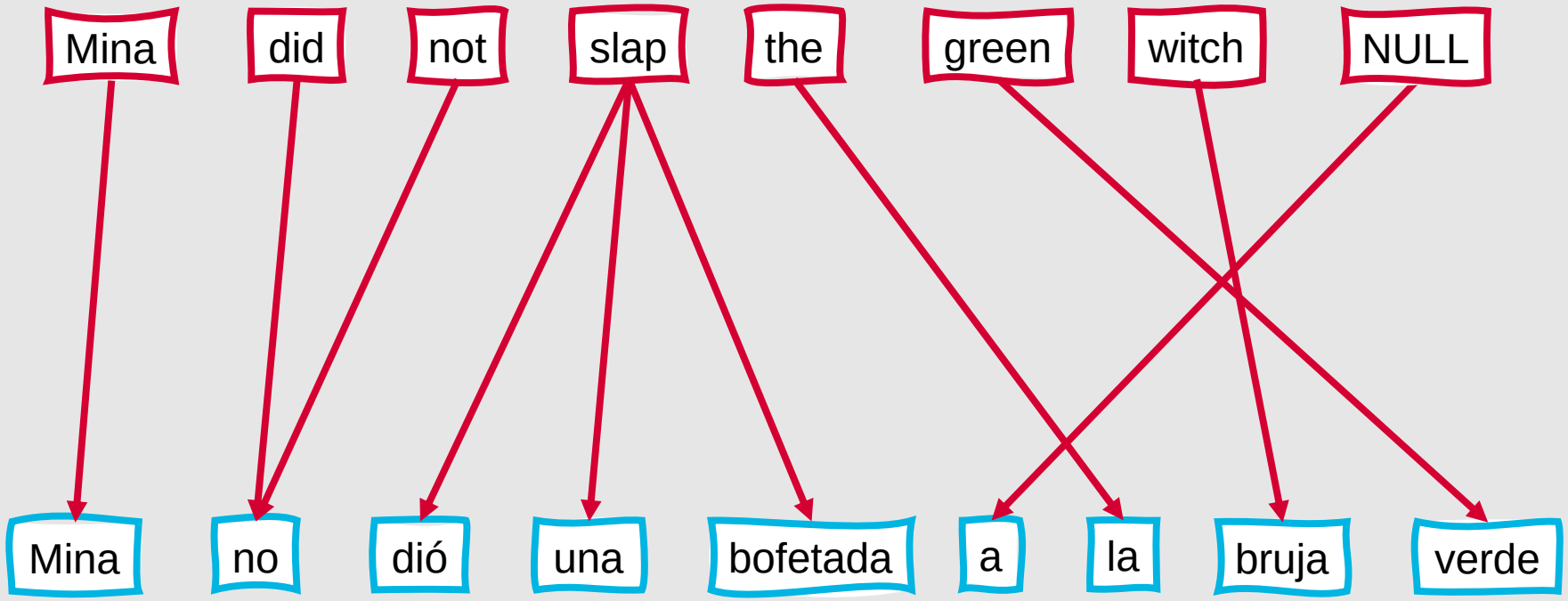
- Computes the probability that a given translation t_i generates the original sentence s based on its **constituent phrases**
- Stages of phrase-based translation:
  1. Group the words from the source sentence into phrases
  2. Translate each source phrase into a target language phrase
  3. (Optionally) reorder the target language phrases

Probability in Phrase-Based Translation Models

- Relies on two probabilities:
  - **Translation probability**
    - Probability of generating a source language phrase from a target language phrase, $\phi(\overline{t_i}, \overline{s_i})$
  - **Distortion probability**
    - Probability of two consecutive target language phrases being separated in the source language by a word span of a particular length, $d(a_i - b_{i-1})$
- To learn these probabilities, we need to train two sets of parameters:
  - $\phi(\overline{t_i}, \overline{s_i})$
  - $d(a_i - b_{i-1})$
- We learn these using phrase-aligned bilingual training sets

Decoding for Phrase-Based Machine Translation

- Aligned phrases can be stored in a **phrase-translation** table
- **Decoding algorithms** can then search through this table to find the overall translation that maximizes the phrase translation probabilities



Summary: Review of Deep Learning Architectures and Classical Machine Translation

- Popular deep learning architectures in natural language processing include **feedforward neural networks**, **convolutional neural networks**, and **recurrent neural networks**
- These architectures can be **stacked** or combined to form **bidirectional** architectures
- Machine translation is challenging due to many **typological**, **morphological**, and other differences between languages
- Classical machine translation used **dictionary-based**, **direct transfer**, and **interlingua** approaches
- A popular statistical MT model is the **Bayesian noisy channel approach**, which relies on **phrase-based translation**

Machine Translation

- Classical Machine Translation
  - Direct translation
  - Transfer approaches
  - Interlingua approaches
  - Statistical methods
- Modern Machine Translation
  - Encoder-decoder models

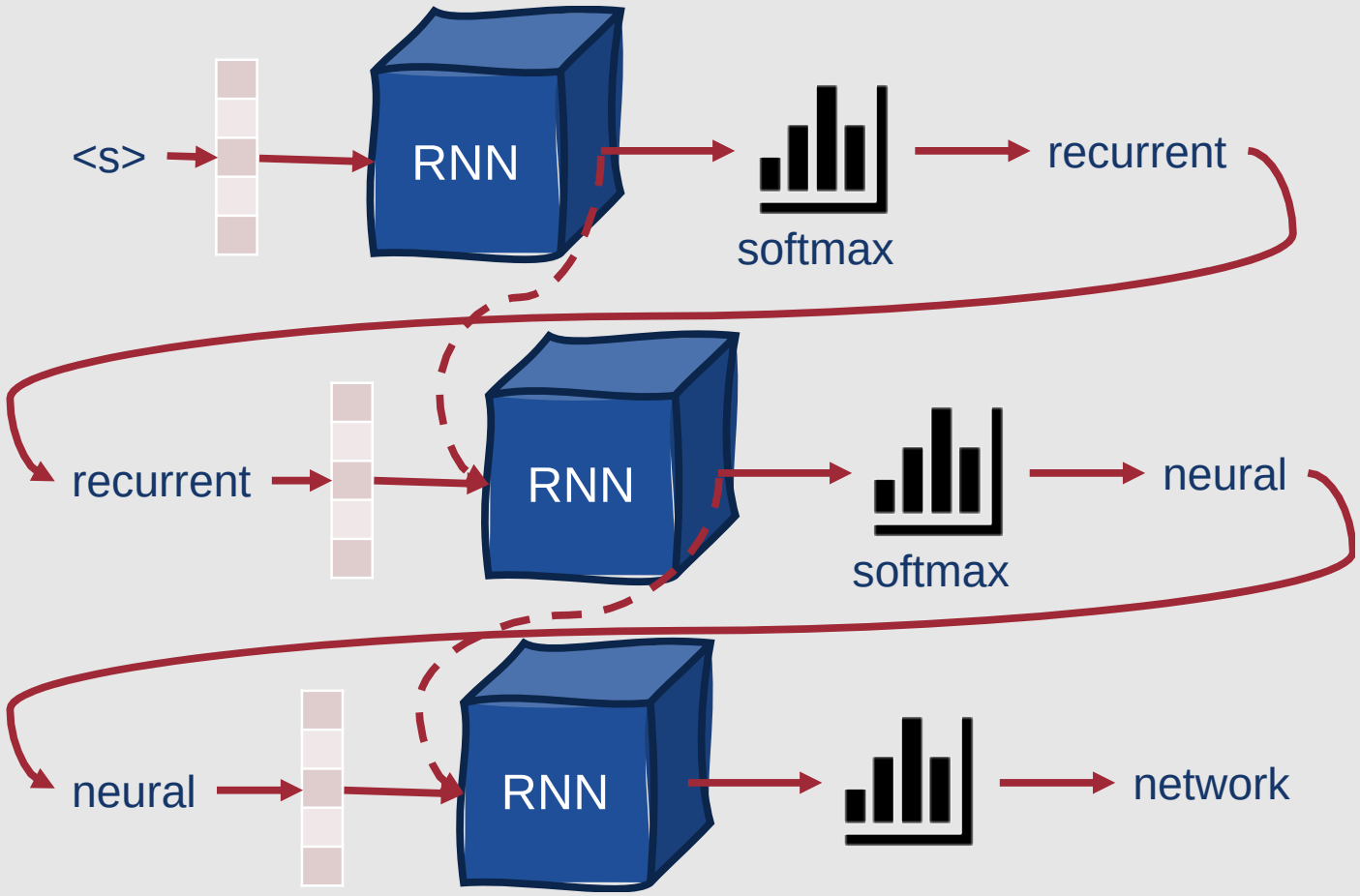
Encoder-Decoder Models

- Generate **contextually-appropriate**, **arbitrary-length** output sequences
- Basic premise:
  - Use a neural network to **encode an input to an internal representation**
  - Pass that internal representation as input to a second neural network
  - Use that neural network to **decode the internal representation to a task-specific output sequence**
- This method allows networks to be trained in an **end-to-end** fashion

Where did this idea come from?

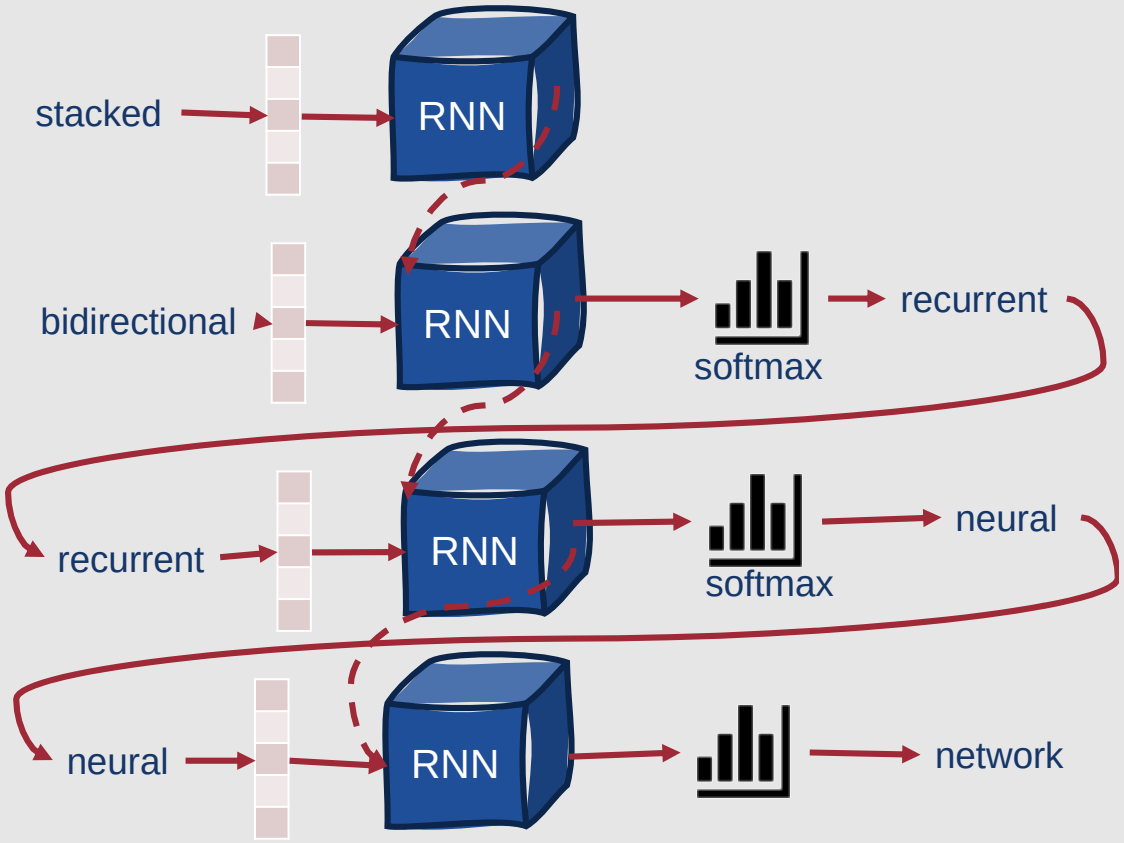
Sequence processing models generate language using a process of autoregressive generation:

- Start with a seed token (e.g., <s>)
- Predict the most likely next word in the sequence
- Use that word as input at the next timestep
- Repeat until an end token (or max length) is reached



This setup can be extended to generate text given a specific prefix….

- Pass the specified prefix through the language model, in sequence
- End with the hidden state corresponding to the last word of the prefix
- Start the autoregressive process at that point
- **Goal: Output sequence should be a reasonable completion of the prefix**

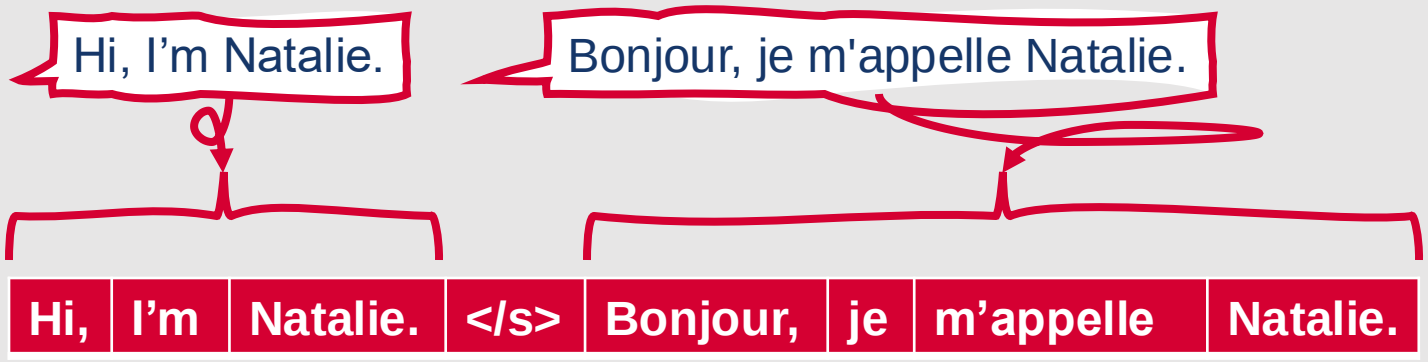


We can build upon this idea to transform one type of sequence to another.

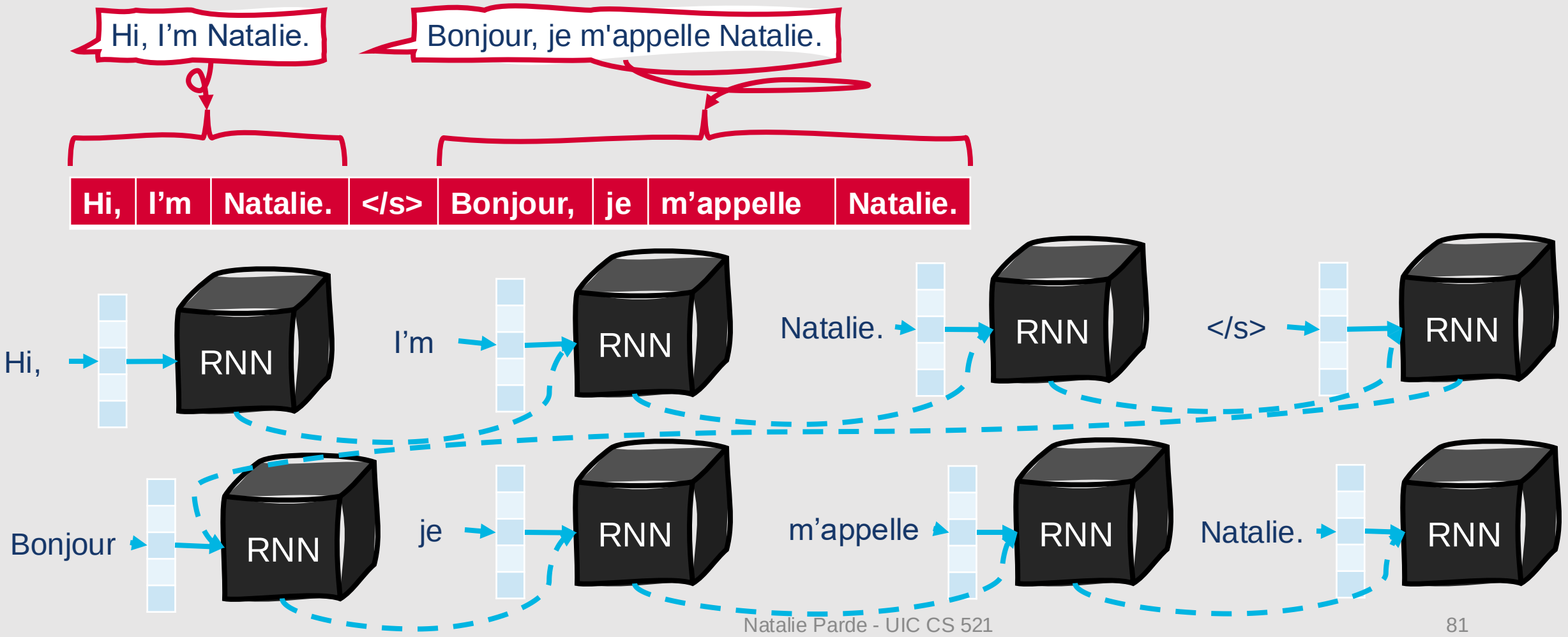
- Machine translation example:
  1. Take a **sequence of text from Language #1**
  2. Take the **translation of that text from Language #2**
  3. **Concatenate the two sequences**, separated by a marker
  4. Use these concatenated sequences to **train the autoregressive model**
  5. Test the model by **passing in only the first part of a concatenated sequence** (text from Language #1) and checking to see what the generated completion (text from Language #2) looks like

Intuition: Machine Translation

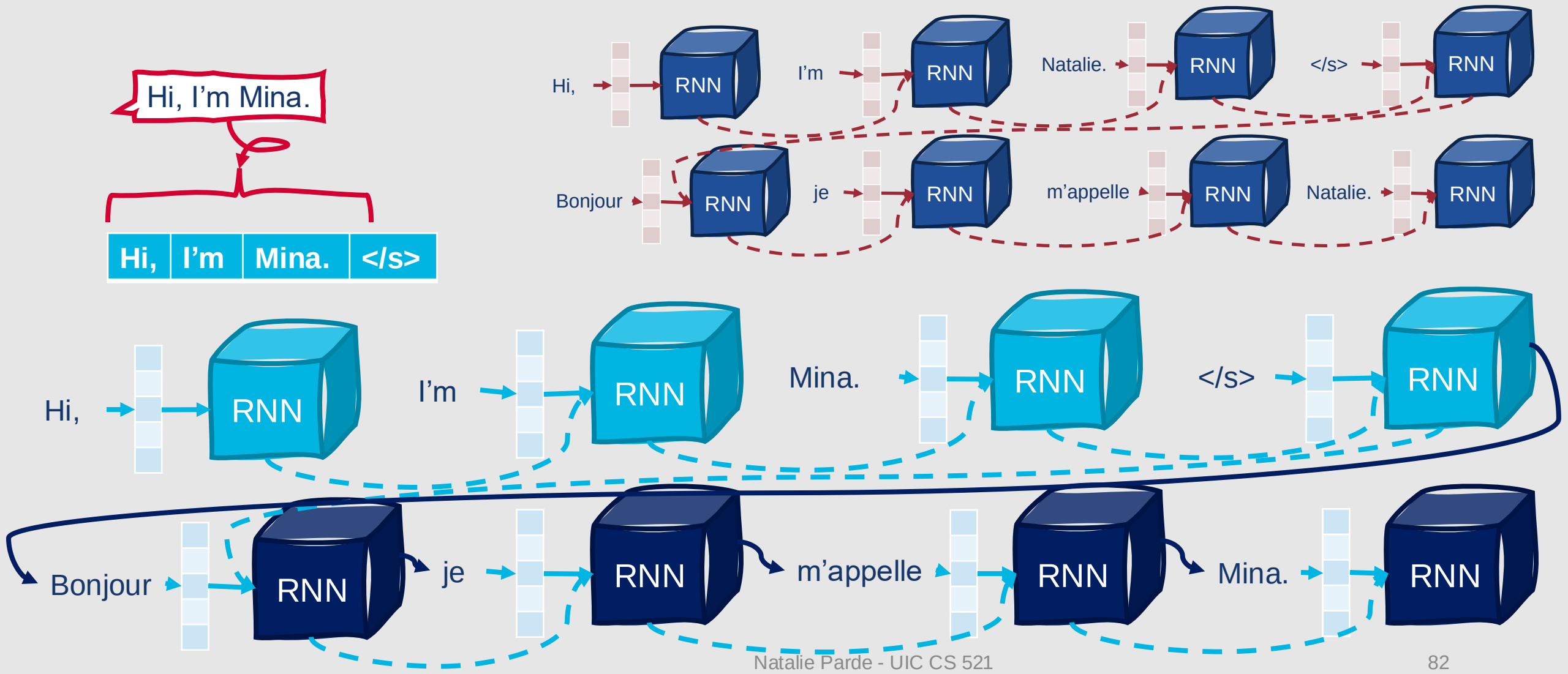
Intuition: Machine Translation



Intuition: Machine Translation



Intuition: Machine Translation



Key Elements of an Encoder-Decoder Network

- **Encoder**
  - Accepts an input sequence, x_1^n
  - Generates a sequence of contextualized representations, h_1^n
- **Context vector**
  - A function, c, of h_1^n that conveys the basic meaning of x_1^n to the decoder
  - (Might just be equivalent to h_1^n)
- **Decoder**
  - Accepts c as input
  - Generates an arbitrary-length sequence of hidden states, h_1^m, from which a corresponding sequence of output states y_1^m can be obtained

Encoders

- Can be any type of neural network, but is generally assumed to be a sequence processing model:
  - Feedforward network
  - CNN
  - RNN
  - LSTM/BiLSTM
  - GRU/BiGRU
  - Transformer

- These networks can be stacked on top of one another

Decoders

- Need to perform autoregressive generation to produce the output sequence
- Can be any type of sequence processing network
  - RNN
  - LSTM
  - GRU
  - Transformer
- Formally….
  - $c = h_n^e$
  - $h_0^d = c$
  - $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d)$
  - $z_t = f(h_t^d)$
  - $y_t = \text{softmax}(z_t)$

Final hidden state of the encoder

First hidden state of the decoder

Decoders

- Need to perform autoregressive generation to produce the output sequence
- Can be any type of sequence processing network
  - RNN
  - LSTM
  - GRU
  - Transformer
- Formally….
  - $c = h_n^e$
  - $h_0^d = c$
  - $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d)$
  - $z_t = f(h_t^d)$
  - $y_t = \text{softmax}(z_t)$

Embedding for the output sampled from the previous step

Some type of sequence processing model

Decoders

- Need to perform autoregressive generation to produce the output sequence
- Can be any type of sequence processing network
  - RNN
  - LSTM
  - GRU
  - Transformer
- Formally….
  - $c = h_n^e$
  - $h_0^d = c$

Regular ending steps (activation function applied to hidden state outputs, and softmax applied to activation outputs)

- $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d)$
- $z_t = f(h_t^d)$
- $y_t = \text{softmax}(z_t)$

A couple useful extensions….

- Formally….
  - $c = h_n^e$
  - $h_0^d = c$
  - $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d) \rightarrow h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d, c)$
  - $z_t = f(h_t^d)$
  - $y_t = \text{softmax}(z_t)$

Make the context vector available at each timestep when decoding, so that its influence doesn't diminish over time

A couple useful extensions….

- Formally….
  - $c = h_n^e$
  - $h_0^d = c$

  - $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d) \rightarrow h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d, c)$
  - $z_t = f(h_t^d)$
  - $y_t = \text{softmax}(z_t) \rightarrow y_t = \text{softmax}(\widehat{y_{t-1}}, z_t, c)$

Condition output on not only the hidden state, but the previous output and encoder context (easier to keep track of what's been generated already)

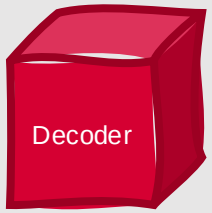
What other ways can we improve the decoder's output quality?

- **Beam search**
- Improved **context vector**
  - Final hidden state tends to be more focused on the end of the input sequence
  - Can be addressed by using bidirectional RNNs, summing the encoder hidden states, or averaging the encoder hidden states

Beam Search

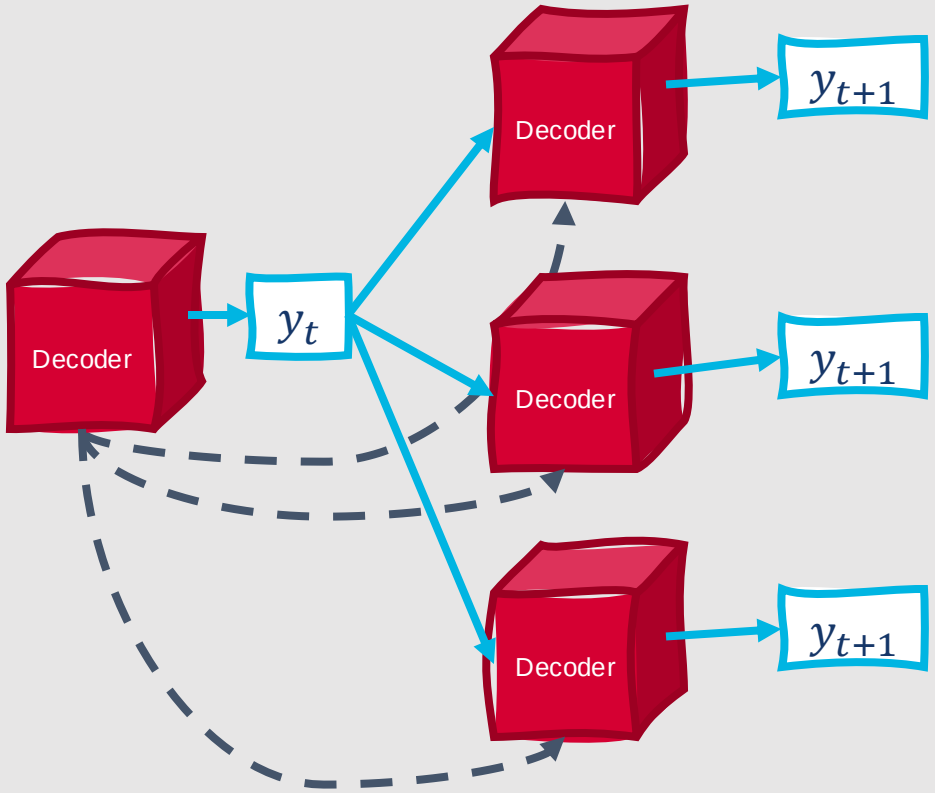
- Selects from multiple possible outputs by framing the task as a **state space search**
- Combines **breadth-first search** with a **heuristic filter**
  - Continually prunes search space to stay a fixed size (**beam width**)
- Results in a set of *b* **hypotheses**, where *b* is the beam width

How does beam search work?



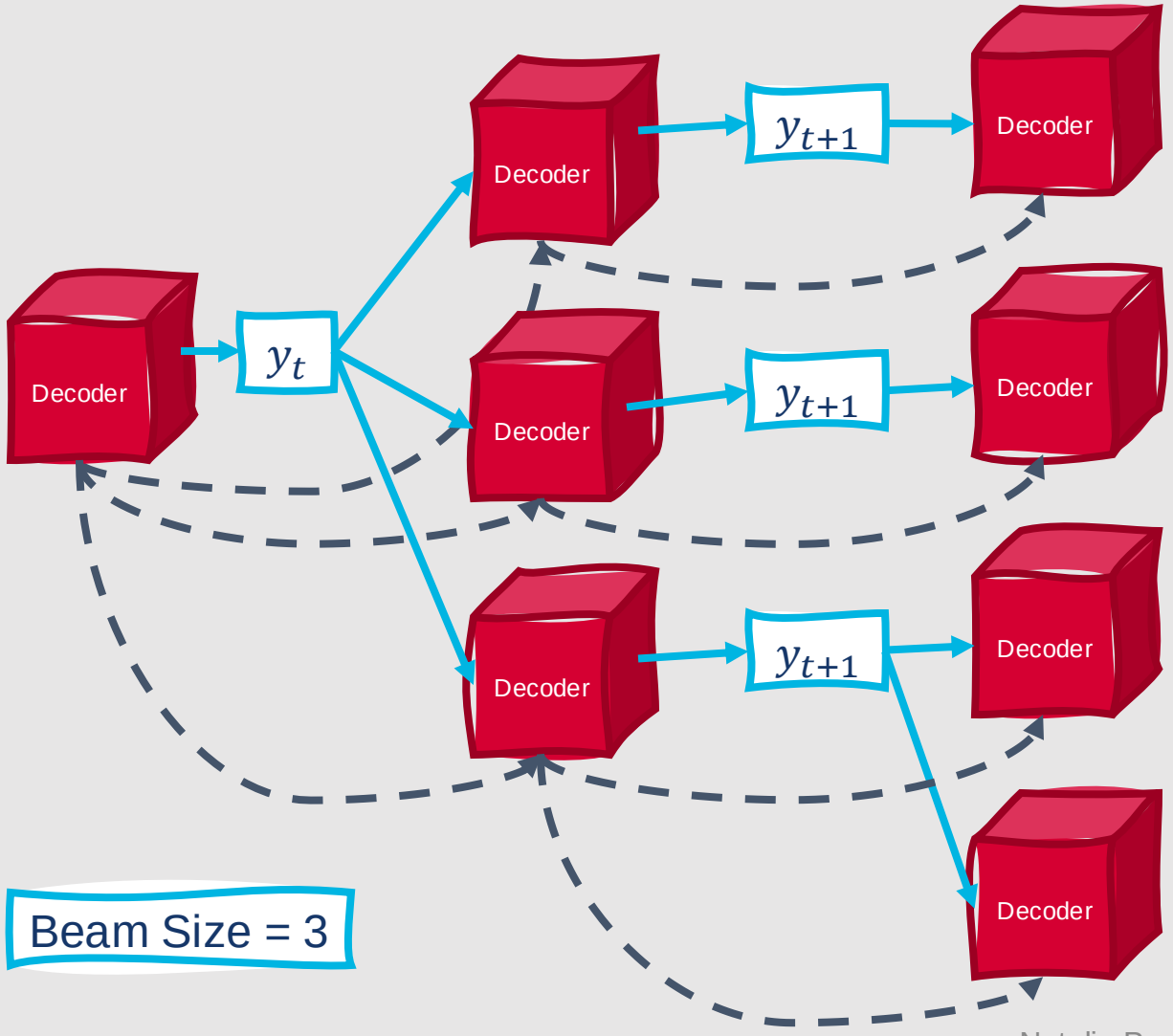
Beam Size = 3

How does beam search work?

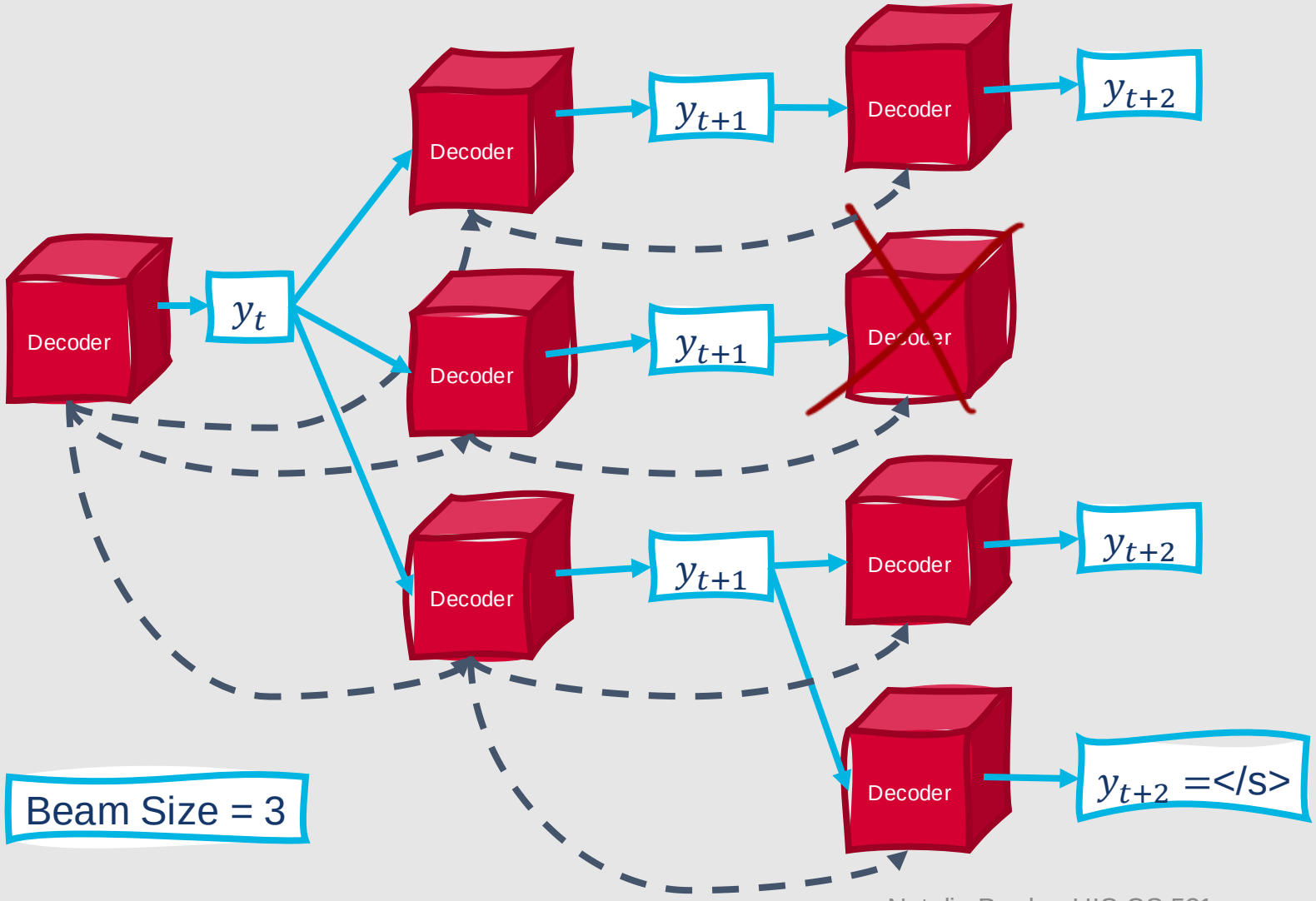


Beam Size = 3

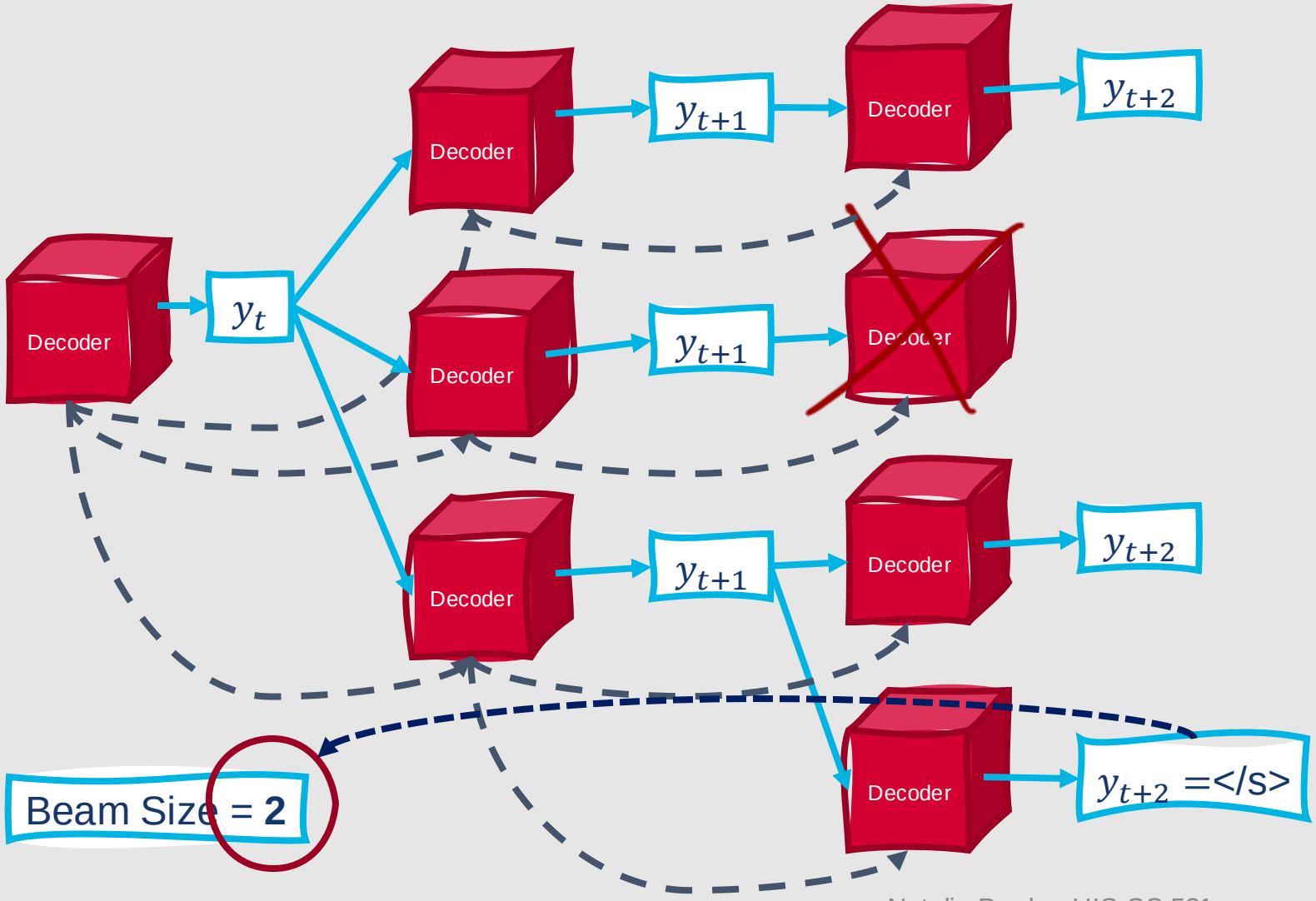
How does beam search work?



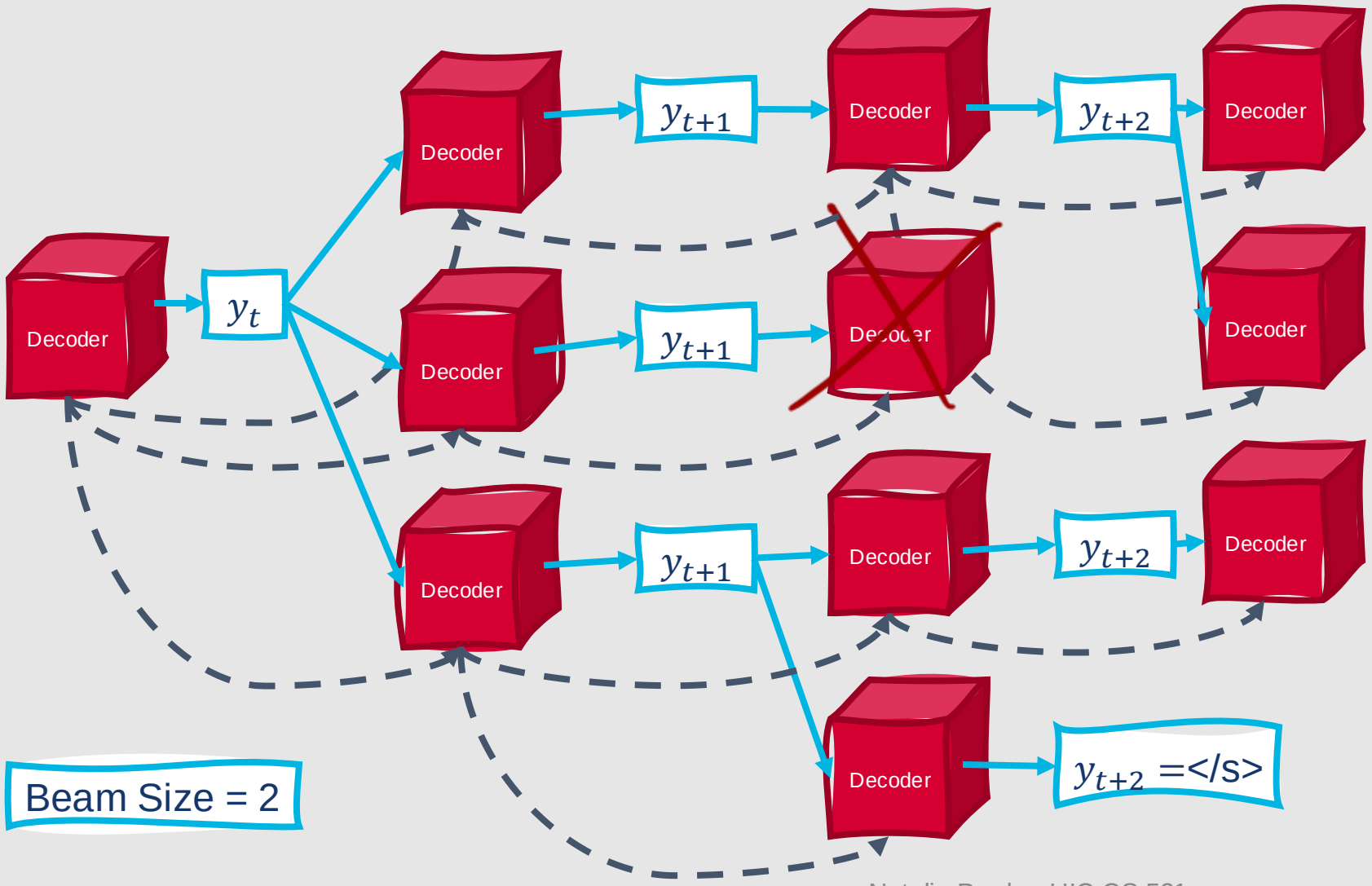
How does beam search work?



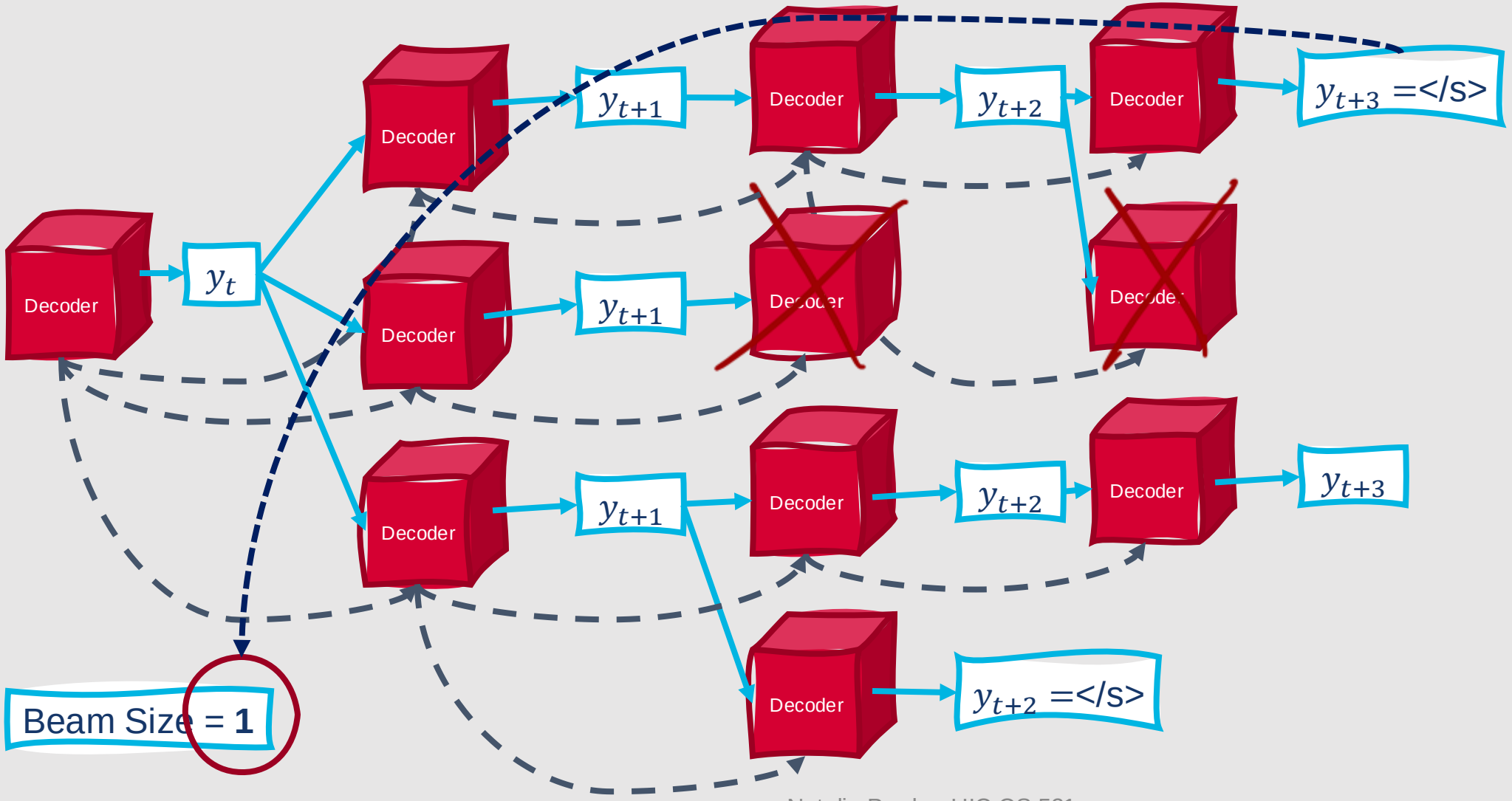
How does beam search work?



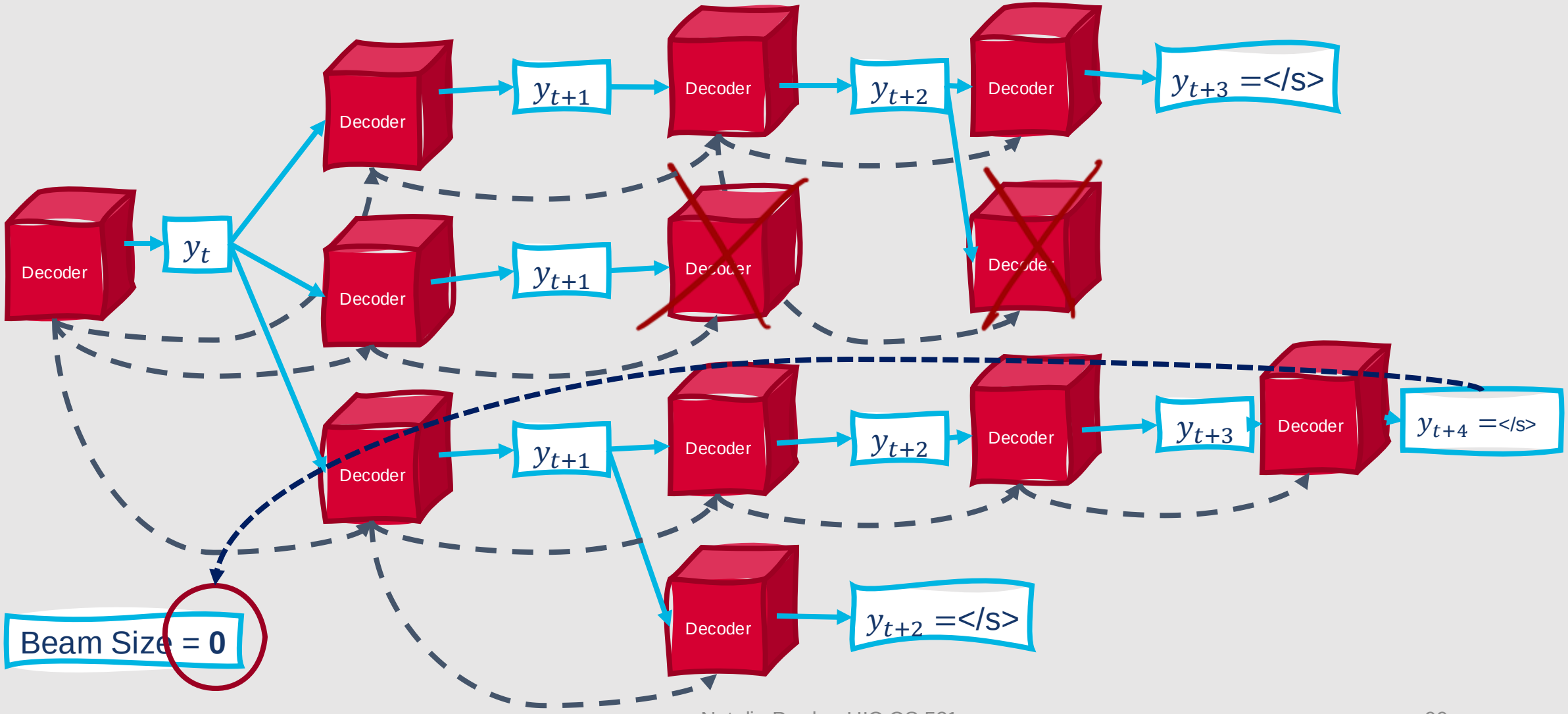
How does beam search work?



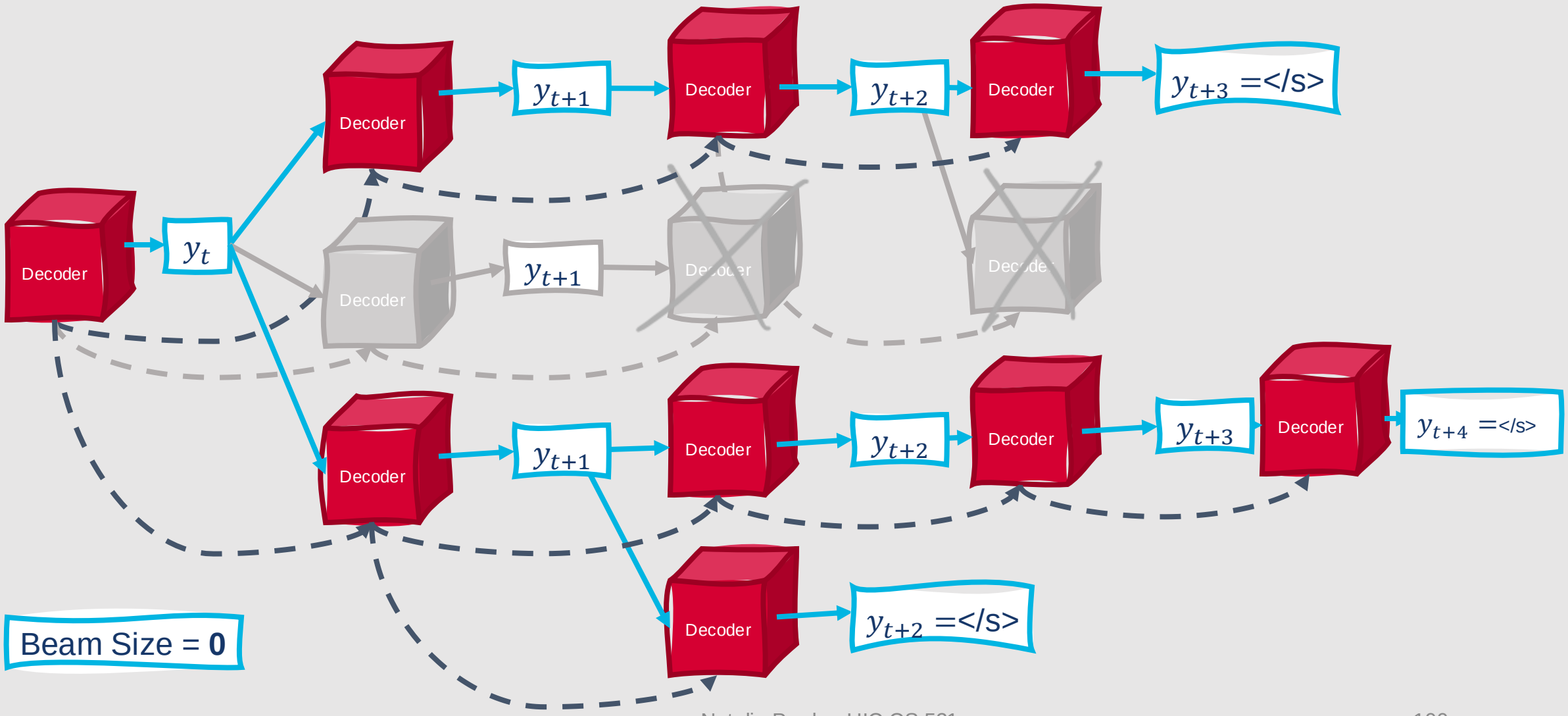
How does beam search work?



How does beam search work?



How does beam search work?

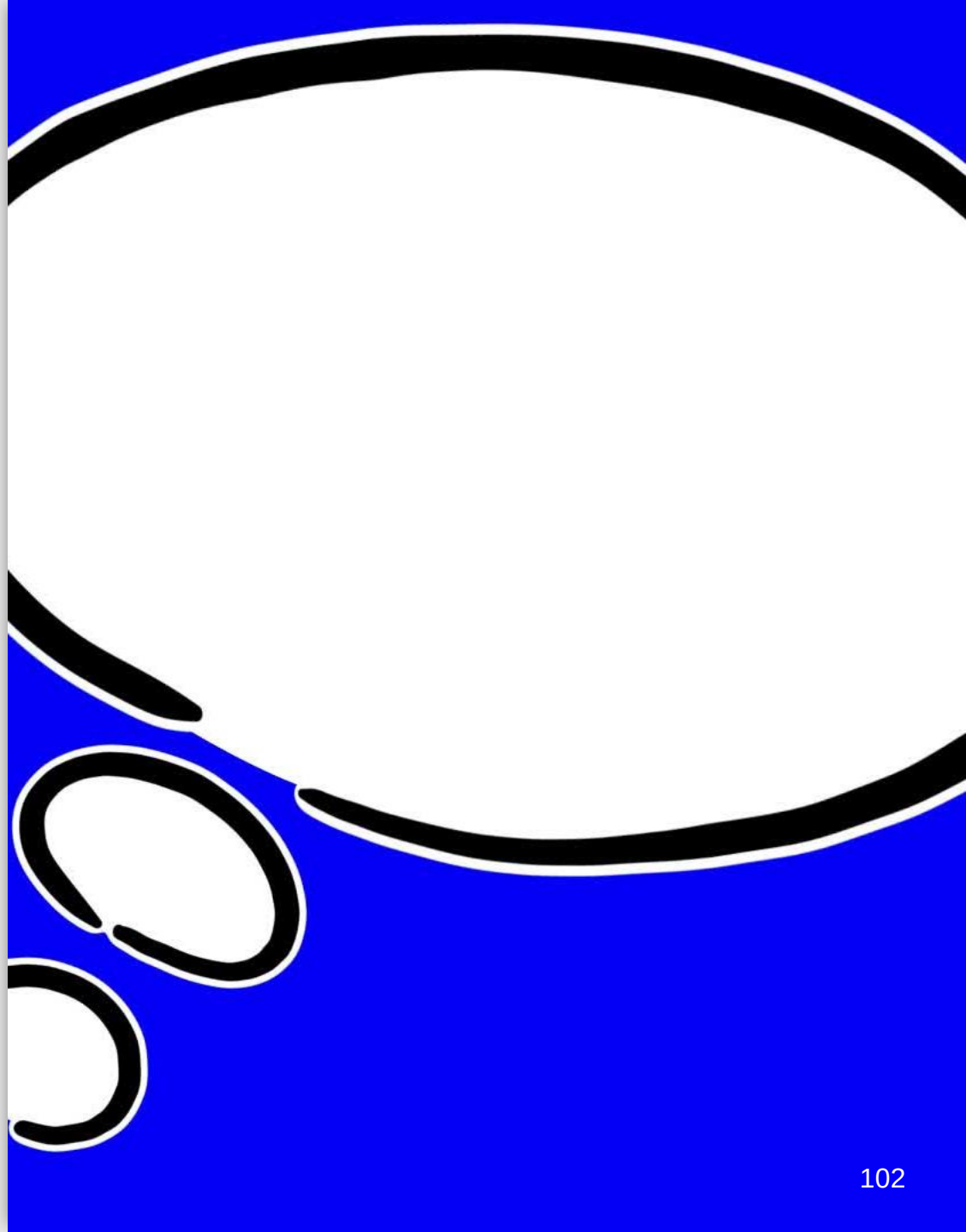


How do we choose a best hypothesis?

- Probabilistic scoring scheme
- Pass all or a subset of hypotheses to a downstream application

So far, the encoder context vectors we've seen have been simple and static.

- Can we do better?
  - Yes!



Attention Mechanism

- Takes entire encoder context into account
- Can be embodied in a fixed-size vector

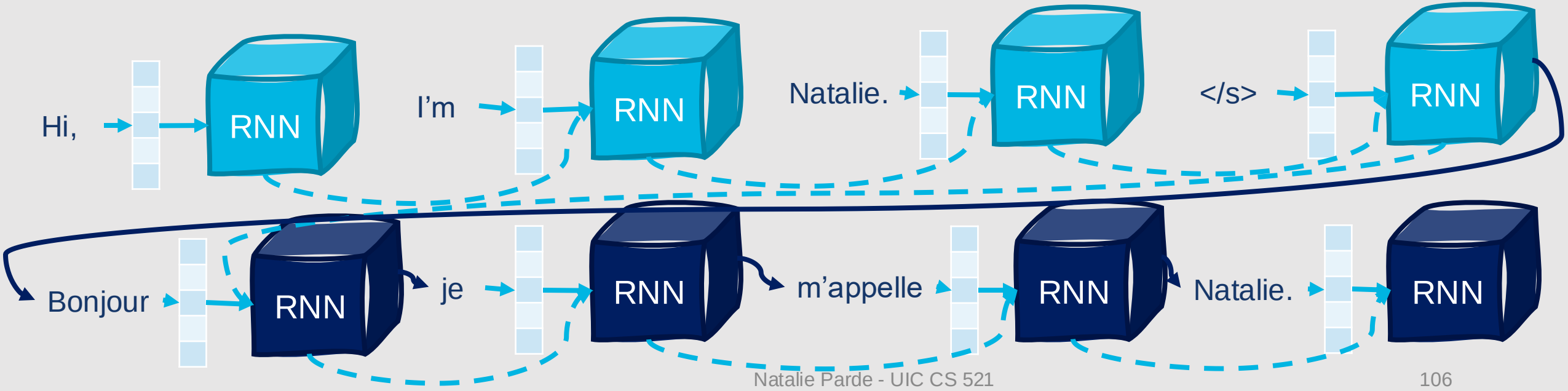
Recall….

- We've already made our context vector available at each timestep when decoding
  - $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d, c)$
- The first step in creating our attention mechanism is to update our hidden state such that it is conditioned on an updated context vector with each decoding step
  - $h_t^d = g(\widehat{y_{t-1}}, h_{t-1}^d, c_t)$

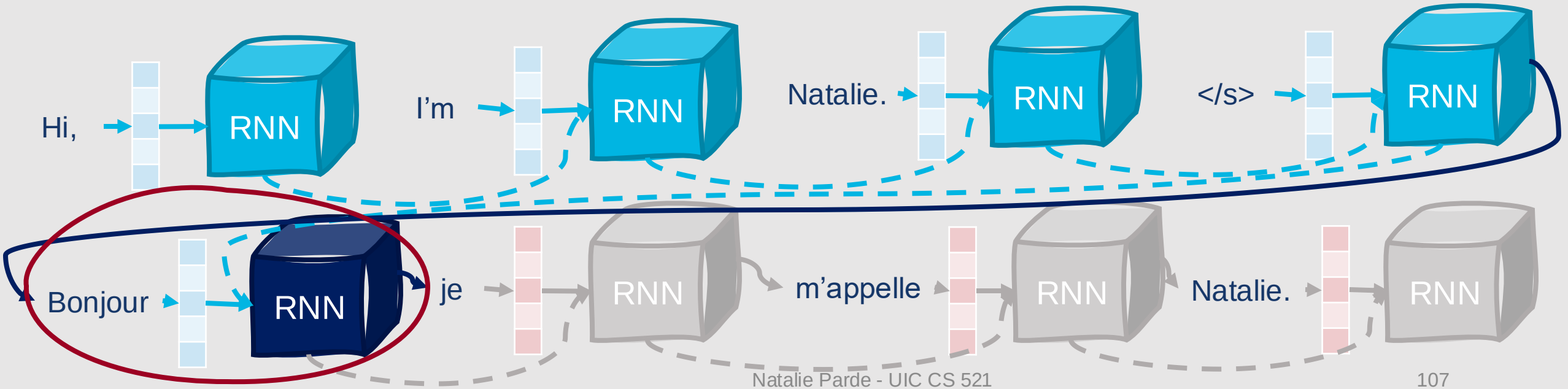
How do we dynamically create a new context vector at each step?

- Compute a vector of scores that capture the relevance of each encoder hidden state to the decoder hidden state, h_{i-1}^{d}
  - $score\left(h_{i-1}^{d}, h_{j}^{e}\right) = h_{i-1}^{d} \cdot h_{j}^{e}$

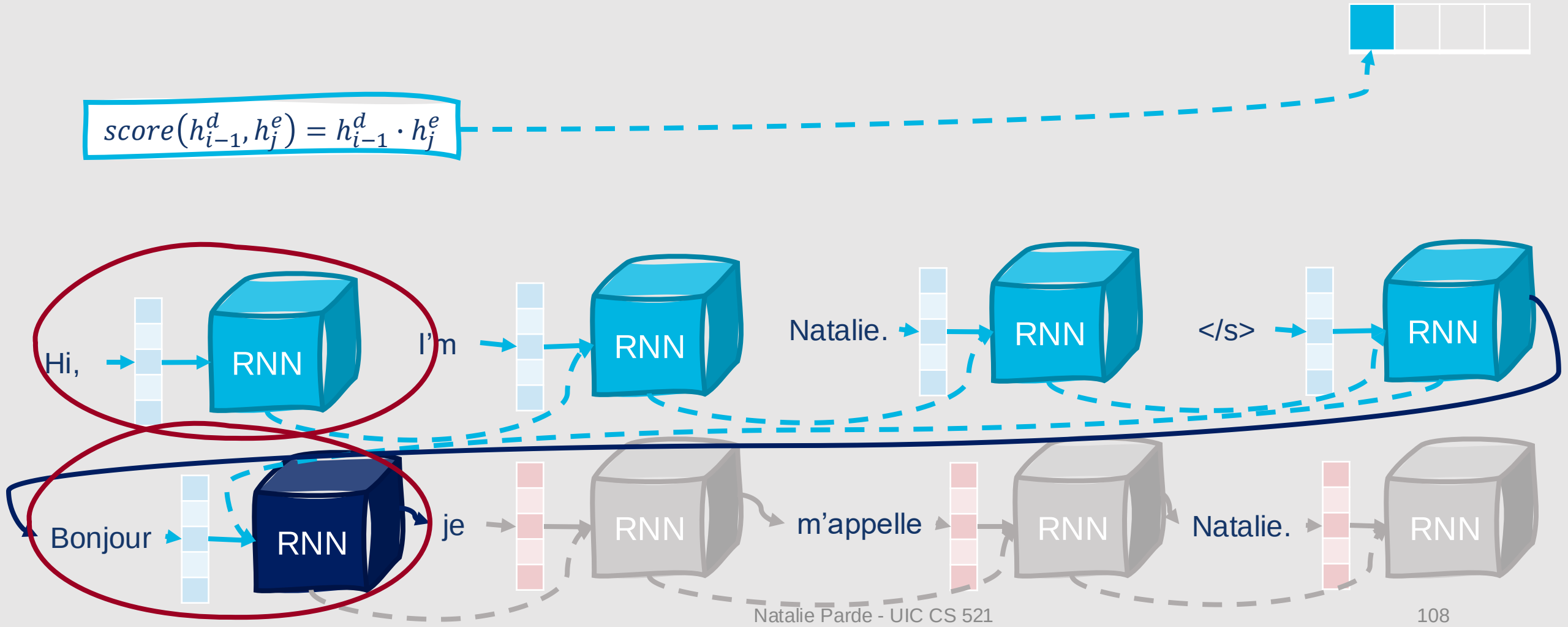
Vector of Context Scores



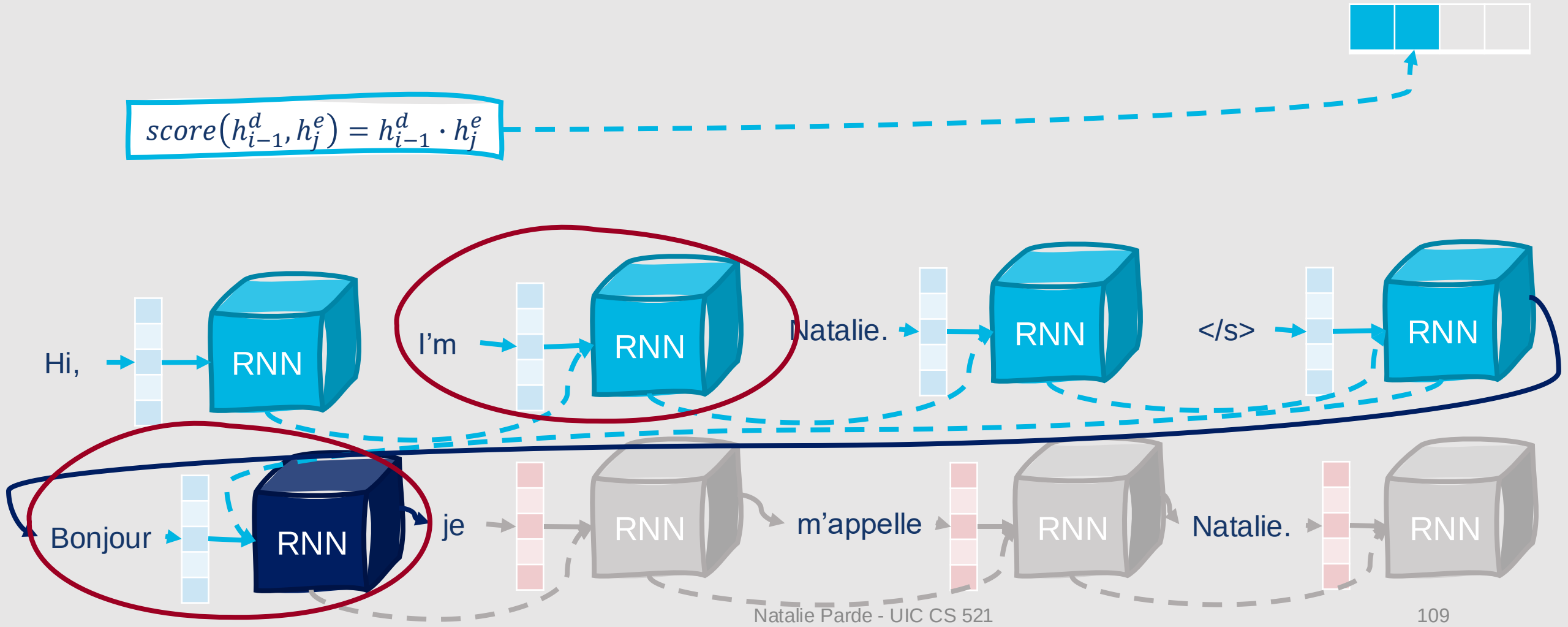
Vector of Context Scores



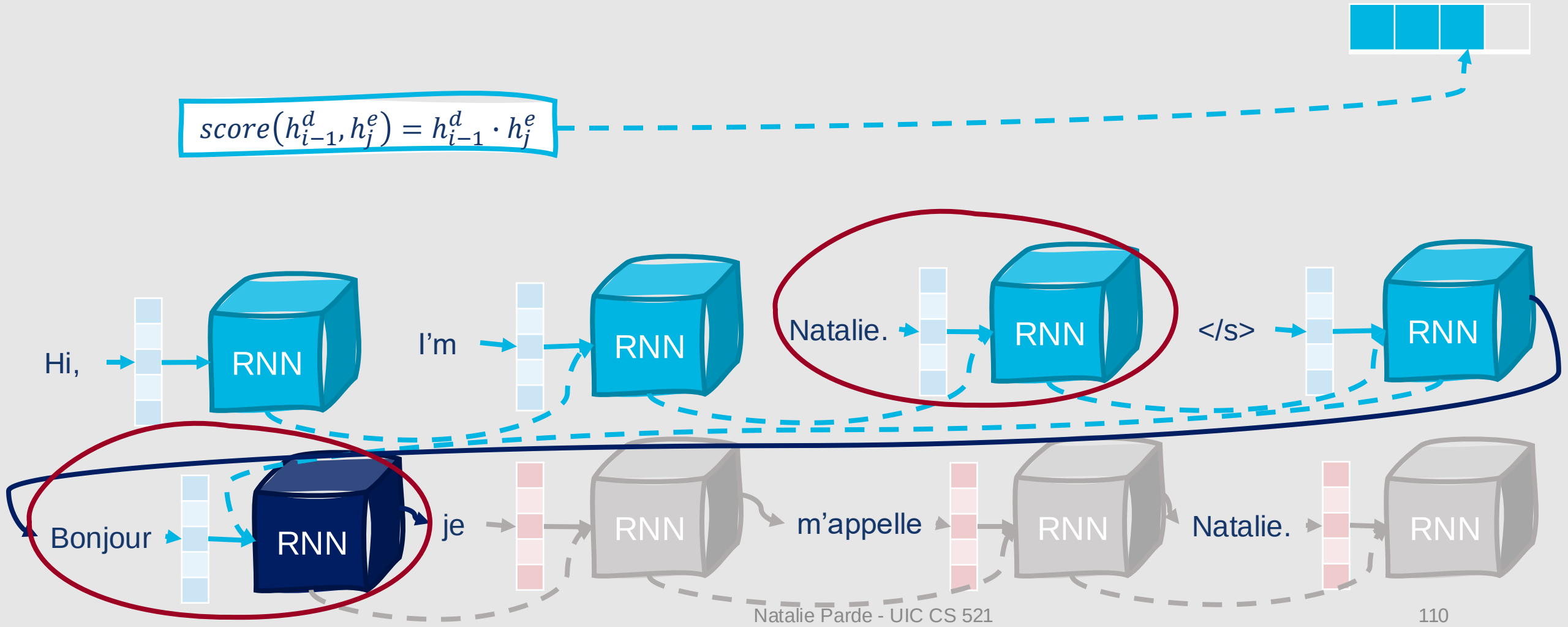
Vector of Context Scores



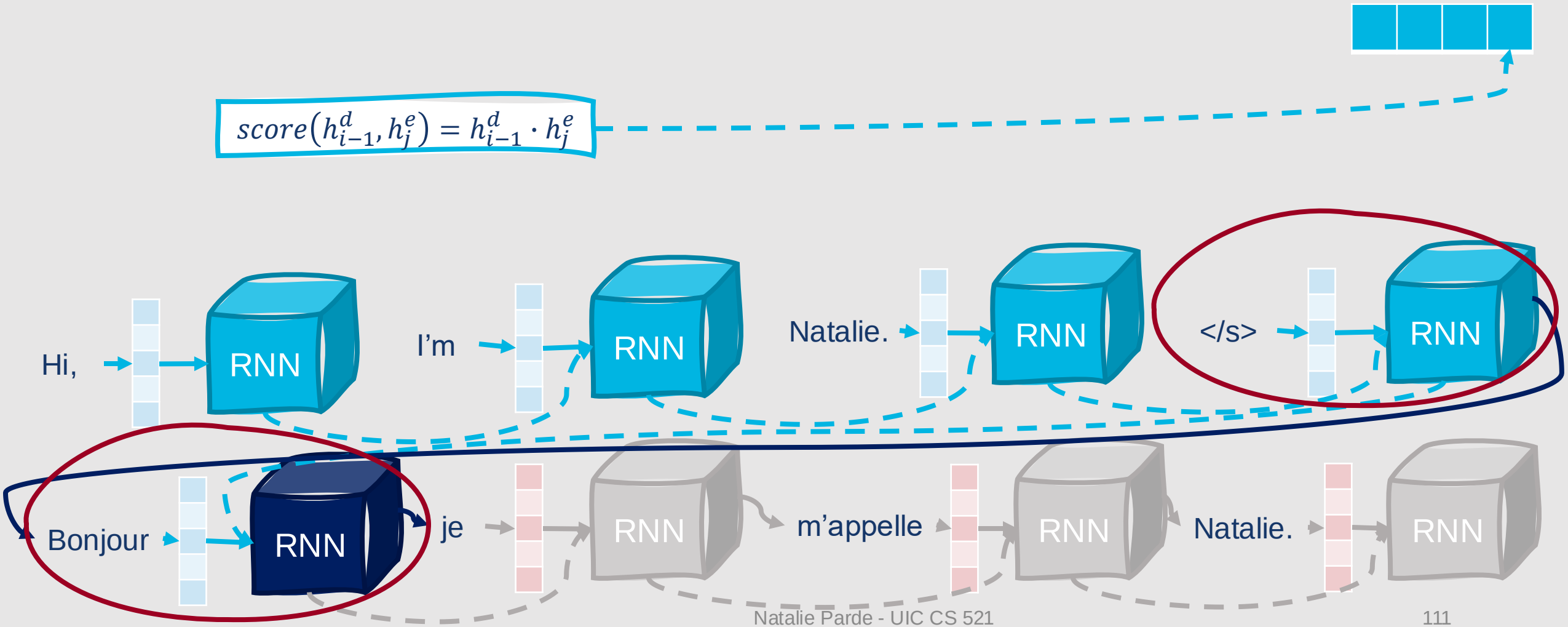
Vector of Context Scores



Vector of Context Scores



Vector of Context Scores



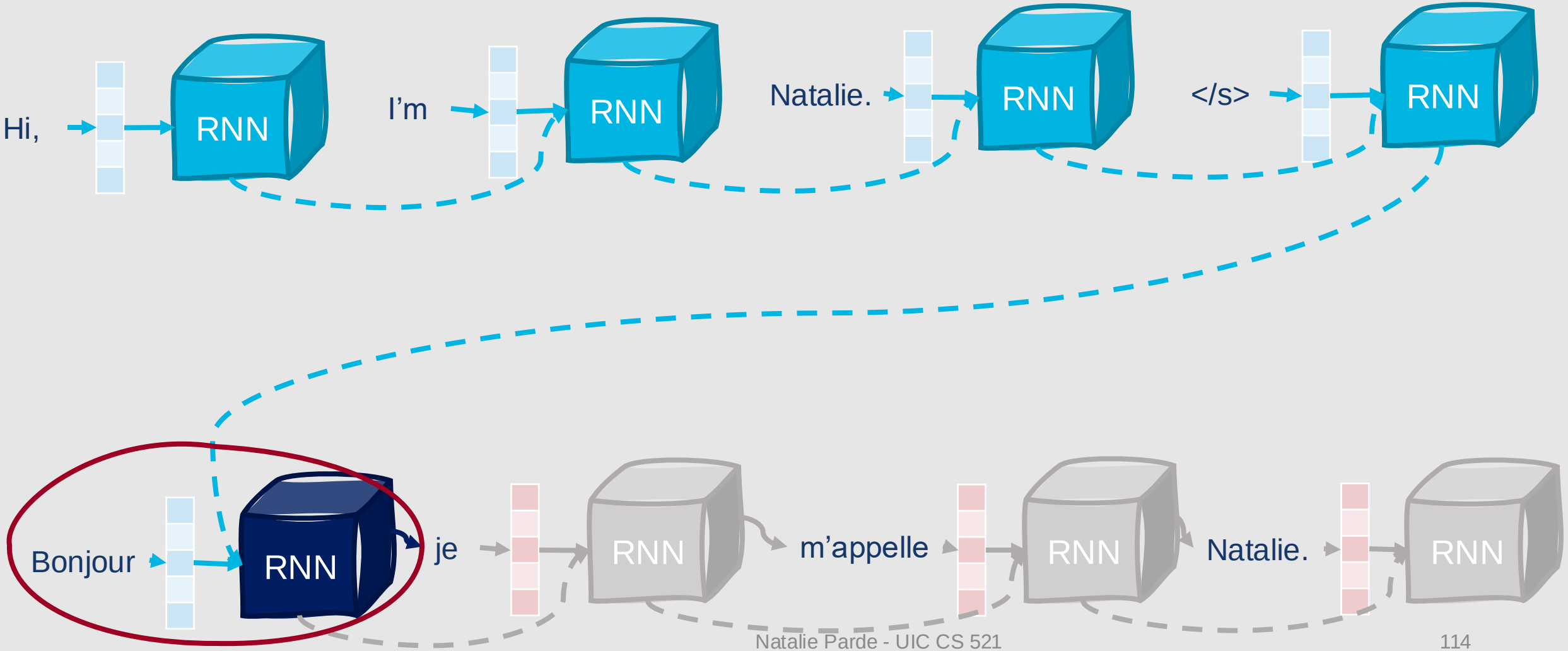
How can we make use of context scores?

- Parameterize these scores with weights
- This allows the model to learn which aspects of similarity between the encoder and decoder states are important

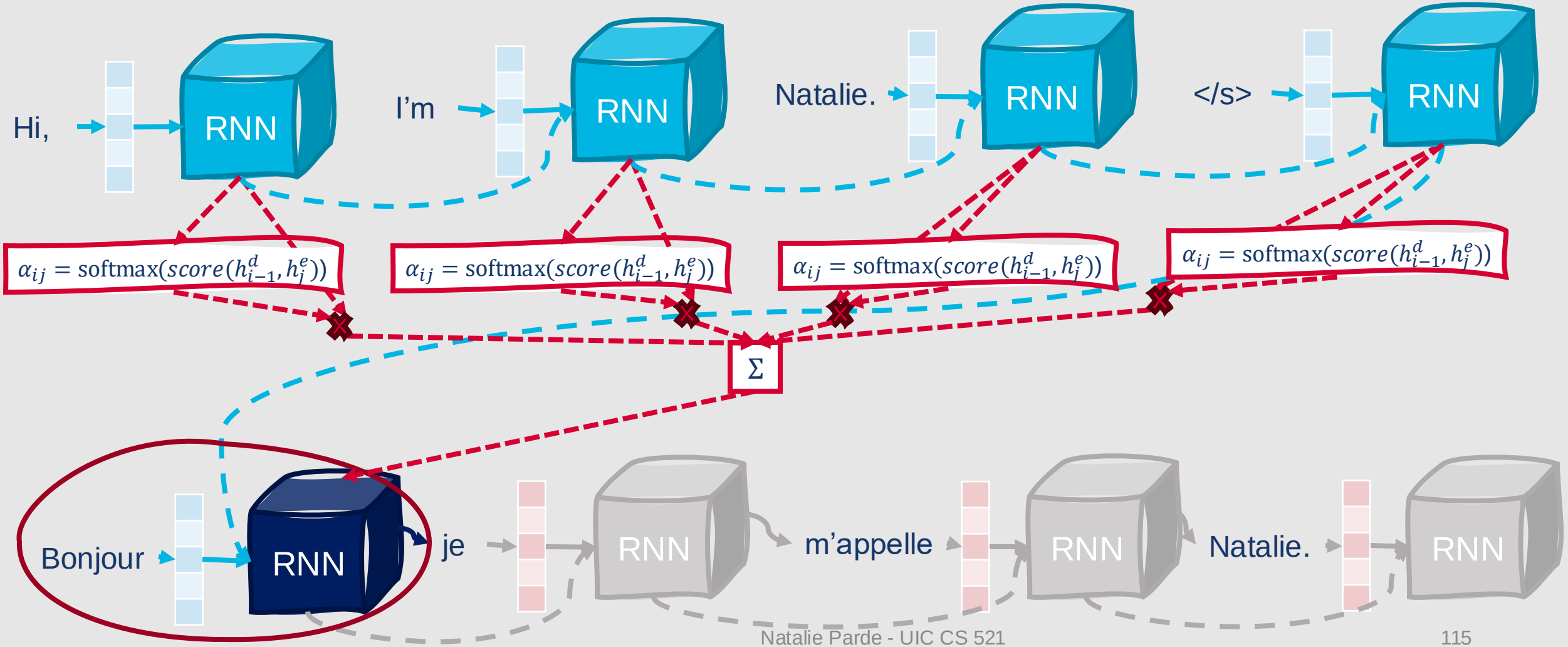
Attention Weights

- **Normalize context scores** to create a vector of weights, α_{ij}
  - $\alpha_{ij} = \text{softmax}(score(h_{i-1}^{d}, h_{j}^{e}) \forall j \in e)$
  - Provides the proportional relevance of each encoder hidden state j to the current decoder state i
- Finally, **take a weighted average over all the encoder hidden states** to create a fixed-length context vector for the current decoder state
  - $c_i = \sum_j \alpha_{ij} h_j^e$

Thus, we finally have an encoder-decoder model with attention!



Thus, we finally have an encoder-decoder model with attention!



Other Attention Weights

- More sophisticated scoring functions can be used as well
- Common: Parameterize the attention score with its own set of trainable weights
  - $\text{score}\left(\mathbf{h}_{i-1}^{d}, \mathbf{h}_{j}^{e}\right) = \mathbf{h}_{t-1}^{d} \mathbf{W}_{s} \mathbf{h}_{j}^{e}$
  - Advantage: Allows the encoder and decoder to use vectors with different dimensionality (dot-product attention requires the encoder and decoder hidden states to have the same dimensionality)

Practical Details for Building MT Systems

Vocabulary

- MT systems typically use a fixed vocabulary generated using byte pair encoding or other wordpiece algorithms
- Vocabulary is usually shared across the source and target languages

Corpora

- Parallel corpora with the same content communicated in multiple languages
- Common sources:
  - Government documents for nations with multiple official languages
  - Subtitles for movies and TV shows
- Often, text from the source and target language(s) is aligned at the sentence level

What if we don't have much training data?

- Parallel corpora are difficult to find, especially for lower-resource language pairs
- **Backtranslation:**
  1. Train an intermediate target-to-source MT system on a small parallel corpus
  2. Translate additional monolingual data from the target language to the source language using this intermediate system
  3. Consider this new, synthetic parallel data as additional training data
  4. Train a source-to-target MT system on the expanded training dataset

How do we evaluate machine translation models?

- Translation quality tends to be very subjective!
- Two common approaches:
  - **Human ratings**
  - **Automated metrics**

Evaluating Machine Translation Using Human Ratings

- Typically evaluated along multiple dimensions
- Tend to check for both **fluency** and **adequacy**
- **Fluency:**
  - Clarity
  - Naturalness
  - Style
- **Adequacy:**
  - Fidelity
  - Informativeness

Evaluating Machine Translation Using Human Ratings

- How to get quantitative measures of fluency?
  - Ask humans to rate different aspects of fluency along a scale
  - Measure how long it takes humans to read a segment of text
  - Ask humans to guess the identity of the missing word
    - “After such a late night working on my project, it was hard to wake up this _____!”

Evaluating Machine Translation Using Human Ratings

- How to get quantitative measures of adequacy?
  - Ask bilingual raters to rate how much information was preserved in the translation
  - Ask monolingual raters to do the same, given access to a gold standard reference translation
  - Ask raters to answer multiple-choice questions about content present in a translation

Another set of human evaluation metrics considers post-editing.

- Ask a human to **post-edit** or “fix” a translation
- Compute the number of edits required to correct the output to an acceptable level
  - Can be measured via number of word changes, number of keystrokes, amount of time taken, etc.

Automated Metrics

- Less accurate than human evaluation, but:
  - Useful for iteratively testing system improvements
  - Can be used as an automatic loss function
- Two main families:
  - Character- or word-overlap
  - Embedding similarity

Popular Lexical Overlap Metrics

- **BLEU**
  - Measure of word overlap
- **METEOR**
  - Measure of word overlap, considering stemming and synonymy
- **Character F-Score (chrF)**
  - Measure of character n-gram overlap

BLEU

- Weighted average of the number of n-gram overlaps with human translations
- **Precision-based metric**
  - What percentage of words in the candidate translation also occur in the gold standard translation(s)?
- To compute BLEU:
  - Count how many times each n-gram is used in the candidate translation, $c(\text{ngram})$
  - Clip that amount so that the highest it can be is $c_{max}(\text{ngram})$, defined as the maximum number of times it is used in a reference translation
  - Compute precision for each word in the candidate translation:
    - Divide (a) summed across each candidate, the sum of n-gram frequencies for n-grams appearing in both the candidate and reference, by (b) summed across each candidate, the sum of n-gram frequences for n-grams appearing in the candidate
  - Take the geometric mean of the modified n-gram precisions for unigrams, bigrams, trigrams, and 4-grams

Then, add a penalty for translation brevity….

- Otherwise, extremely short translations (e.g., “the”) could receive perfect scores!
- The penalty is based on:
  - The sum of the lengths of the reference sentences, *r*
  - The sum of the lengths of the candidate translations, *c*
- Formally, the penalty is set to:
  - $BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$
- The full BLEU score for a set of translations is then:
  - $BLEU = BP\ * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$

Example: Computing BLEU

Mina no dió una bofetada a la bruja verde. Source Sentence

Mina didn’t slap the green witch. Reference Translation

Mina did not give a slap to the green witch. Candidate Translation

Example: Computing BLEU

Mina no dió una bofetada a la bruja verde. Source Sentence

Mina didn't slap the green witch. Reference Translation

Mina did not give a slap to the green witch. Candidate Translation

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

Example: Computing BLEU

Mina didn't slap the green witch.

Mina did not give a slap to the green witch.

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

Unigram	Unigram Frequency (Candidate)	Unigram Frequency (Reference)
Mina	1	1
did	1	0
not	1	0
give	1	0
a	1	0
slap	1	1
to	1	0
the	1	1
green	1	1
witch	1	1
.	1	1

Example: Computing BLEU

Mina didn't slap the green witch.

Mina did not give a slap to the green witch.

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

Unigram	Unigram Frequency (Candidate)	Unigram Frequency (Reference)
Mina	1	1
did	1	0
not	1	0
give	1	0
a	1	0
slap	1	1
to	1	0
the	1	1
green	1	1
witch	1	1
.	1	1

$$p_1 = \frac{1+0+0+0+0+1+0+1+1+1+1}{1+1+1+1+1+1+1+1+1+1+1} = \frac{6}{11}$$

Example: Computing BLEU

Mina didn't slap the green witch.

Mina did not give a slap to the green witch.

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

Bigram	Bigram Frequency (Candidate)	Bigram Frequency (Reference)
Mina did	1	0
did not	1	0
not give	1	0
give a	1	0
a slap	1	0
slap to	1	0
to the	1	0
the green	1	1
green witch	1	1
witch .	1	1

$$p_1 = \frac{1+0+0+0+0+1+0+1+1+1+1}{1+1+1+1+1+1+1+1+1+1+1} = \frac{6}{11}$$

$$p_2 = \frac{0+0+0+0+0+0+0+1+1+1}{1+1+1+1+1+1+1+1+1+1} = \frac{3}{10}$$

Example: Computing BLEU

Mina didn't slap the green witch.

Mina did not give a slap to the green witch.

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

Trigram	Trigram Frequency (Candidate)	Trigram Frequency (Reference)
Mina did not	1	0
did not give	1	0
not give a	1	0
give a slap	1	0
a slap to	1	0
slap to the	1	0
to the green	1	0
the green witch	1	1
green witch .	1	1

$$p_1 = \frac{6}{11} \qquad p_2 = \frac{3}{10}$$

$$p_3 = \frac{0+0+0+0+0+0+0+1+1}{1+1+1+1+1+1+1+1+1} = \frac{2}{9}$$

Example: Computing BLEU

Mina didn't slap the green witch.

Mina did not give a slap to the green witch.

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

4-gram	4-gram Frequency (Candidate)	4-gram Frequency (Reference)
Mina did not give	1	0
did not give a	1	0
not give a slap	1	0
give a slap to	1	0
a slap to the	1	0
slap to the green	1	0
to the green witch	1	0
the green witch .	1	1

$$p_1 = \frac{6}{11} \qquad p_2 = \frac{3}{10} \qquad p_3 = \frac{2}{9}$$

$$p_4 = \frac{0+0+0+0+0+0+0+1}{1+1+1+1+1+1+1+1} = \frac{1}{8}$$

Example: Computing BLEU

Mina didn't slap the green witch.

Mina did not give a slap to the green witch.

$c = 11$

$r = 7$

$$\text{prec}_n = \frac{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \min(\text{c}(\text{ngram}), \text{c}_{\max}(\text{ngram}))}{\sum_{c \in \{\text{Candidates}\}} \sum_{\text{ngram} \in C} \text{c}(\text{ngram})}$$

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-\frac{r}{c})} & if\ c \leq r \end{cases}$$

$$BLEU = BP * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}}$$

$$p_1 = \frac{6}{11} \quad p_2 = \frac{3}{10} \quad p_3 = \frac{2}{9} \quad p_4 = \frac{1}{8}$$

$$BP = 1$$

$$BLEU = 1 * (\prod_{n=1}^{4} \text{prec}_n)^{\frac{1}{4}} = 1 * (\frac{6}{11} * \frac{3}{10} * \frac{2}{9} * \frac{1}{8})^{\frac{1}{4}} = 1 * 0.00454545454^{\frac{1}{4}} = 1 * 0.25965358893 = 0.26$$

What are good BLEU scores?

- No formal score ranges, but in general:
  - < 10: Very poor
  - 10-19: Difficult to understand
  - 20-29: Understandable but with many grammatical errors
  - 30-39: Understandable and reasonable quality
  - 40-49: High quality
  - 50-59: Very high quality
  - >60: May exceed human translators

Limitations of BLEU

- Word or phrase order is of minimal importance
  - When computing unigram precision, a word can exist anywhere in the translation!
- Does not consider word similarity
- Relatively low correlation with human ratings
- Nonetheless, BLEU is reasonable to use in cases when a quick, automated metric is needed to assess translation performance

Character F-Score (chrF)

- Same intuition as BLEU: Good machine translations tend to contain the same words and characters as human translations
- Ranks a candidate translation by a function of the number of character n-gram overlaps with a human reference translation
- Less sensitive to differences in wordform than BLEU (e.g., “gives” versus “is giving”)

How is chrF computed?

- Similarly to "regular" F-score
  - **chrP:** averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that also occur in the reference
  - **chrR:** averaged % of character unigrams, bigrams, …, k-grams in the reference that also occur in the hypothesis
  - **β:** weighting parameter (similarly to F-score) that determines the relative impacts of chrP and chrR on the overall F-score
  - $\text{chrF}\beta = (1+\beta^2)\frac{\text{chrP} \times \text{chrR}}{\beta^2 \times \text{chrP} + \text{chrR}}$, or for example, $\text{chrF2} = \frac{5 \times \text{chrP} \times \text{chrR}}{4 \times \text{chrP} + \text{chrR}}$

Example: Computing chrF

Example: Computing chrF

Example: Computing chrF

C	S	5	2	1	i	s	t	h	e	b	e	s	t

C	S	5	2	1	i	s	g	r	e	a	t

Example: Computing chrF

C	S	5	2	1	i	s	t	h	e	b	e	s	t

14 unigrams, 13 bigrams, ….

C	S	5	2	1	i	s	g	r	e	a	t

12 unigrams, 11 bigrams, ….

Example: Computing chrF

C	S	5	2	1	i	s	t	h	e	b	e	s	t

14 unigrams, 13 bigrams

C	S	5	2	1	i	s	g	r	e	a	t

12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

k=3

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

Example: Computing chrF

C	S	5	2	1	i	s	t	h	e	b	e	s	t

14 unigrams, 13 bigrams

C	S	5	2	1	i	s	g	r	e	a	t

12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

9 unigrams are in both the hypothesis and the reference

Example: Computing chrF

CS	S5	52	21	1i	is	st	th	he	eb	be	es	st

← 14 unigrams, 13 bigrams

CS	S5	52	21	1i	is	sg	gr	re	ea	at

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

9 unigrams are in both the hypothesis and the reference

6 bigrams are in both the hypothesis and the reference

Example: Computing chrF

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

9 unigrams are in both the hypothesis and the reference

6 bigrams are in both the hypothesis and the reference

5 trigrams are in both the hypothesis and the reference

Example: Computing chrF

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

9 unigrams are in both the hypothesis and the reference

Unigram chrP: $^{9}/_{12} = 0.75$

Unigram chrR: $^{9}/_{14} = 0.64$

6 bigrams are in both the hypothesis and the reference

5 trigrams are in both the hypothesis and the reference

Example: Computing chrF

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

9 unigrams are in both the hypothesis and the reference

Unigram chrP: $^{9}/_{12} = 0.75$

Unigram chrR: $^{9}/_{14} = 0.64$

6 bigrams are in both the hypothesis and the reference

Bigram chrP: $^{6}/_{11} = 0.55$

Bigram chrR: $^{6}/_{13} = 0.46$

5 trigrams are in both the hypothesis and the reference

Example: Computing chrF

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

9 unigrams are in both the hypothesis and the reference

Unigram chrP: $^{9}/_{12} = 0.75$

Unigram chrR: $^{9}/_{14} = 0.64$

6 bigrams are in both the hypothesis and the reference

Bigram chrP: $^{6}/_{11} = 0.55$

Bigram chrR: $^{6}/_{13} = 0.46$

5 trigrams are in both the hypothesis and the reference

Trigram chrP: $^{5}/_{10} = 0.5$

Trigram chrR: $^{5}/_{12} = 0.42$

Example: Computing chrF

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

Unigram chrP: $^{9}/_{12} = 0.75$

Bigram chrP: $^{6}/_{11} = 0.55$

Trigram chrP: $^{5}/_{10} = 0.5$

chrP: $\frac{0.75+0.55+0.5}{3} = 0.6$

Unigram chrR: $^{9}/_{14} = 0.64$

Bigram chrR: $^{6}/_{13} = 0.46$

Trigram chrR: $^{5}/_{12} = 0.42$

Example: Computing chrF

CS 521 is the best 

CS 521 is great 

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

Unigram chrP: $^{9}/_{12} = 0.75$

Bigram chrP: $^{6}/_{11} = 0.55$

Trigram chrP: $^{5}/_{10} = 0.5$

chrP: $\frac{0.75+0.55+0.5}{3} = 0.6$

Unigram chrR: $^{9}/_{14} = 0.64$

Bigram chrR: $^{6}/_{13} = 0.46$

Trigram chrR: $^{5}/_{12} = 0.42$

chrR: $\frac{0.64+0.46+0.42}{3} = 0.51$

Example: Computing chrF

CS5	S52	521	21i	1is	ist	sth	the	heb	ebe	bes	est

← 14 unigrams, 13 bigrams

CS5	S52	521	21i	1is	isg	sgr	gre	rea	eat

← 12 unigrams, 11 bigrams

chrP: averaged % of character unigrams, bigrams, …, k-grams in the hypothesis that are also in the reference

chrR: averaged % of character unigrams, bigrams, …, k-grams in the reference that are also in the hypothesis

k=3

chrP: $\frac{0.75+0.55+0.5}{3} = 0.6$

chrR: $\frac{0.64+0.46+0.42}{3} = 0.51$

$$\text{chrF2} = \frac{5 * \text{chrP} * \text{chrR}}{4 * \text{chrP} + \text{chrR}} = \frac{5 * 0.6 * 0.51}{4 * 0.6 + 0.51} = 0.53$$

Limitations of chrF

- Focuses on differences at a very local scale (i.e., character n-grams)
- Doesn’t measure discourse coherence
- Best at measuring performance for different versions of the same system, rather than comparing different systems

Embedding-Based Evaluation Methods

- Measuring exact word- or character-level overlap might be overly strict
  - Good translations may use words that are synonymous to those in the reference!
- Embedding-based methods measure the *semantic* overlap between reference and hypothesis translations

Popular Embedding-Based Methods for Evaluating MT Systems

COMET

- https://github.com/Unbabel/COMET

BLEURT

- https://github.com/google-research/bleurt

BERTScore

- https://github.com/Tiiiger/bert_score

Summary: Encoder-Decoder Models and Evaluating MT Systems

- **Encoder-decoder models** draw upon similar techniques for autoregressive language modeling to convert input to an intermediate vector representation and then convert that intermediate representation to output
- **Attention** scores help the encoder-decoder model's **context vector** focus on the most relevant information from the input for a given decoder timestep
- MT systems are commonly evaluated using both **human ratings** and **automated metrics**
- Popular automated metrics include **BLEU**, **chrF**, and **embedding-based measures**