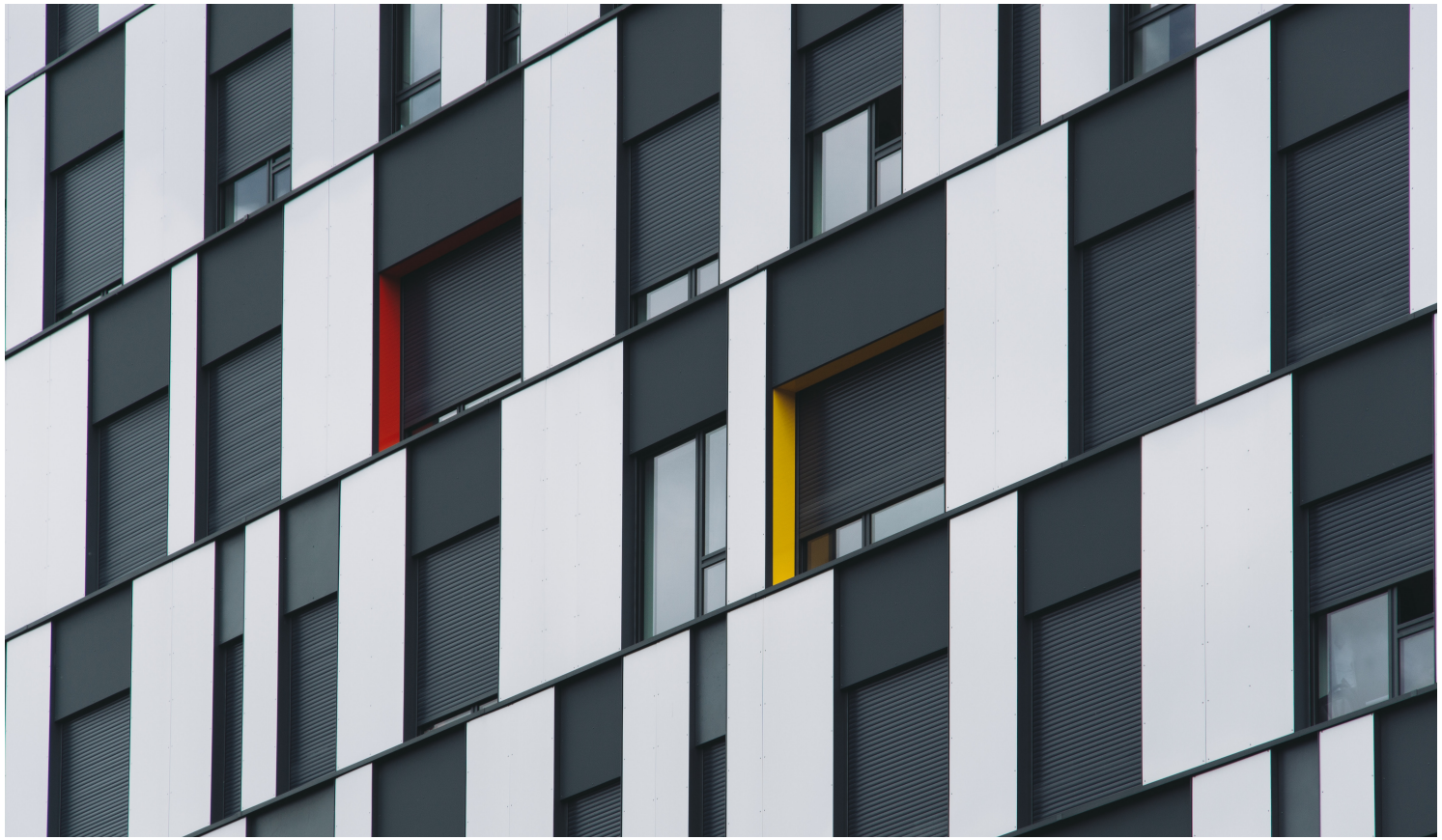


Backpropagation and Convolutional Neural Networks

Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois at Chicago

CS 521: Statistical Natural Language Processing

Spring 2020

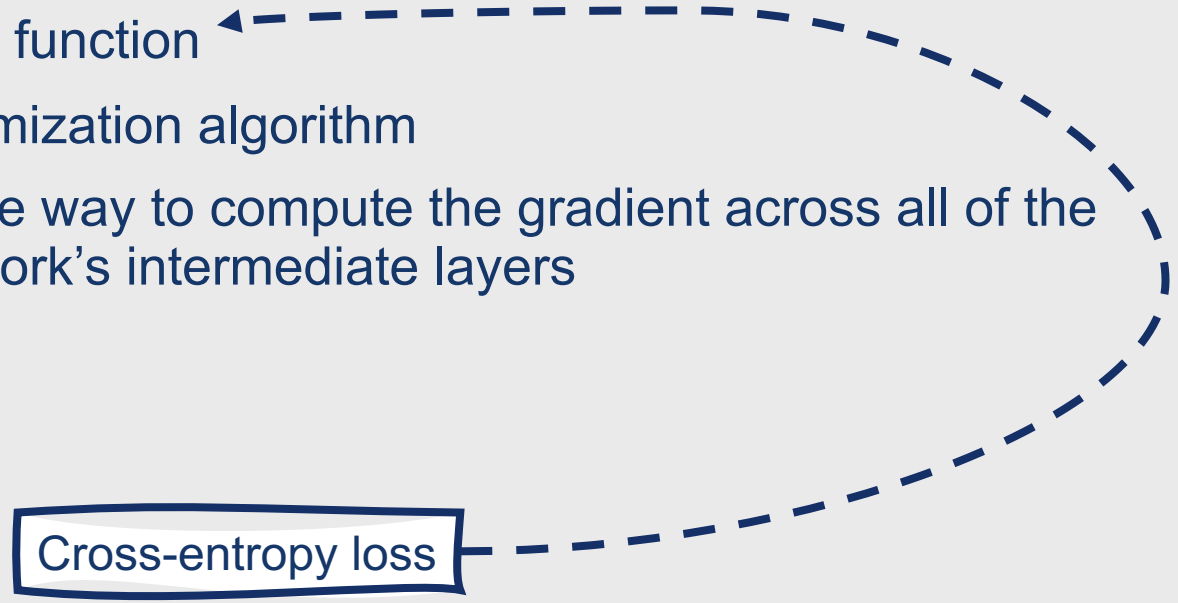
Many slides adapted from Jurafsky and Martin (https://web.stanford.edu/~jurafsky/slp3/).

How do we train neural networks?

- Loss function
- Optimization algorithm
- Some way to compute the gradient across all of the network's intermediate layers

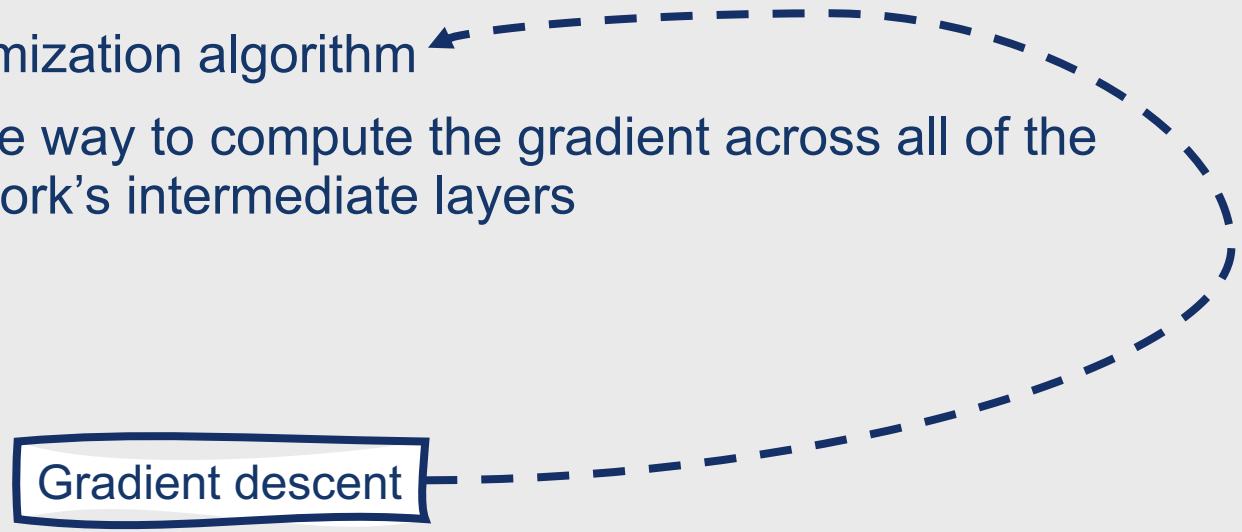
How do we train neural networks?

- ✓ Loss function
- ❑ Optimization algorithm
- ❑ Some way to compute the gradient across all of the network's intermediate layers



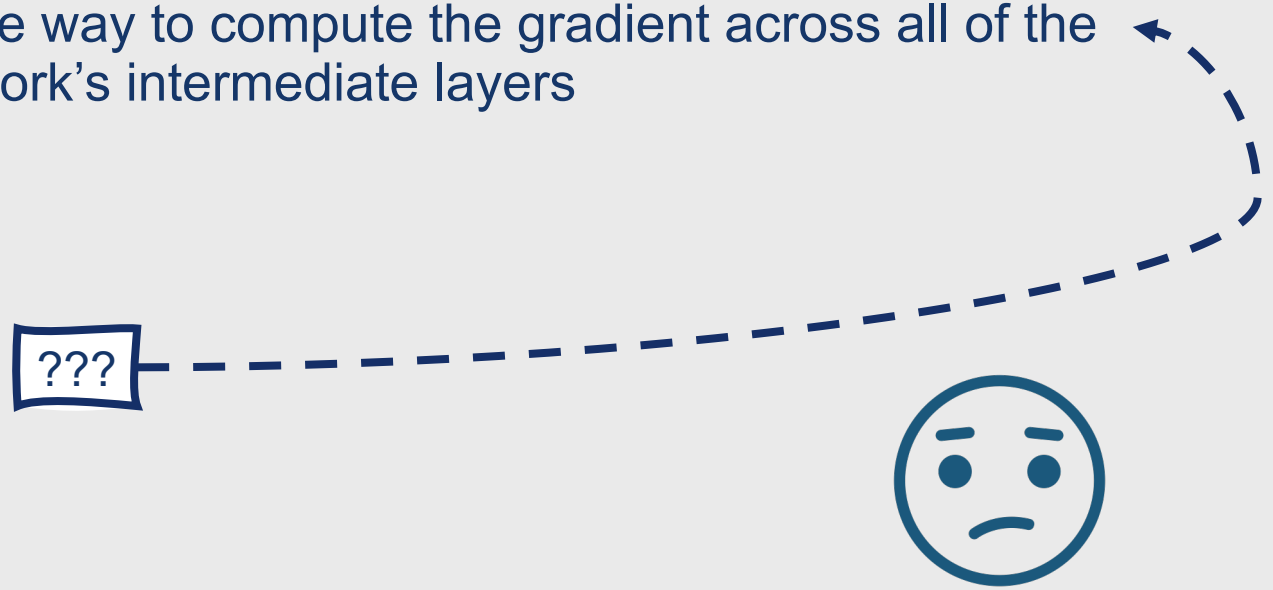
How do we train neural networks?

- ✓ Loss function
- ✓ Optimization algorithm
- ❑ Some way to compute the gradient across all of the network's intermediate layers



How do we train neural networks?

- ✓ Loss function
- ✓ Optimization algorithm
- ❑ Some way to compute the gradient across all of the network's intermediate layers



Backpropagation

- A method for propagating loss values all the way back to the beginning of a deep neural network, even though it's only computed at the end of the network

Recall….

- For a “neural network” with just one weight layer containing a single computation unit + sigmoid activation (i.e., a logistic regression classifier), we can compute the gradient of our loss function by just taking its derivative:
  - $\frac{\partial L_{CE}(w,b)}{\partial w_j} = (\hat{y} - y)x_j = (\sigma(w \cdot x + b) - y)x_j$

Recall….

- For a “neural network” with just one weight layer containing a single computation unit + sigmoid activation (i.e., a logistic regression classifier), we can compute the gradient of our loss function by just taking its derivative:
  - $\frac{\partial L_{CE}(w,b)}{\partial w_j} = (\hat{y} - y)x_j = (\sigma(w \cdot x + b) - y)x_j$

Difference between true and estimated *y*

Corresponding input observation

However, we can't do that with a neural network that has multiple weight layers ("hidden" layers).

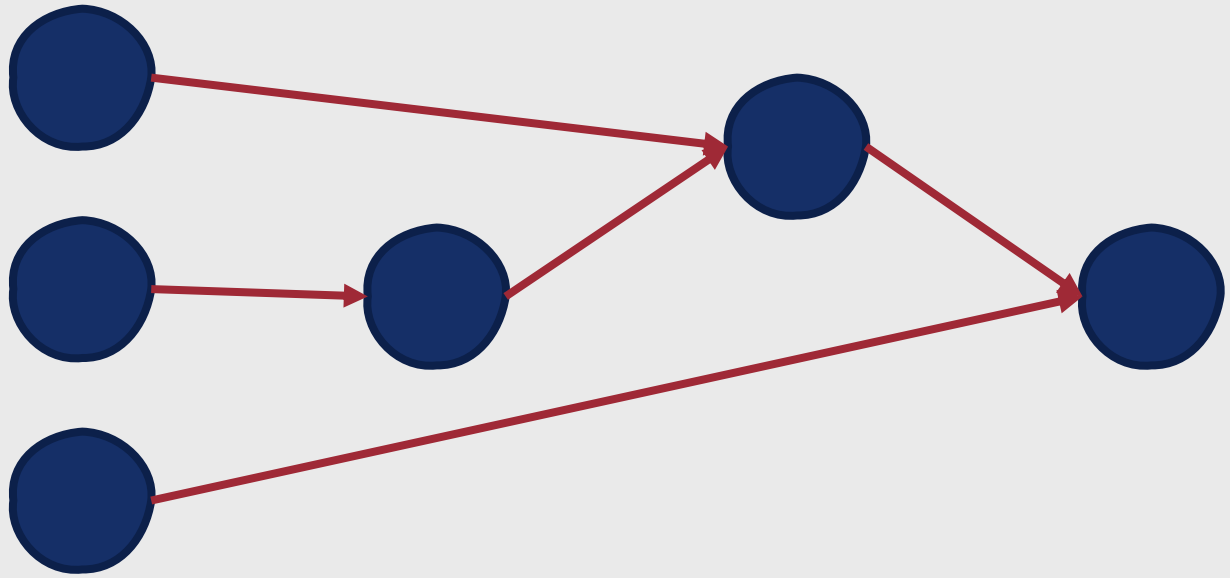
- Why?
  - Simply taking the derivative like we did for logistic regression only provides the gradient for the most recent (i.e., last) weight layer
  - What we need is a way to:
    - Compute the derivative with respect to weight parameters occurring earlier in the network as well
    - Even though we can only compute loss at a single point (the end of the network)

We do this using backward differentiation.

- Usually referred to as **backpropagation** (“**backprop**” for short) in the context of neural networks
- Frames neural networks as **computation graphs**

What are computation graphs?

- Representations of interconnected mathematical operations
- **Nodes** = Operations
- **Directed edges** = connections between output/input of nodes



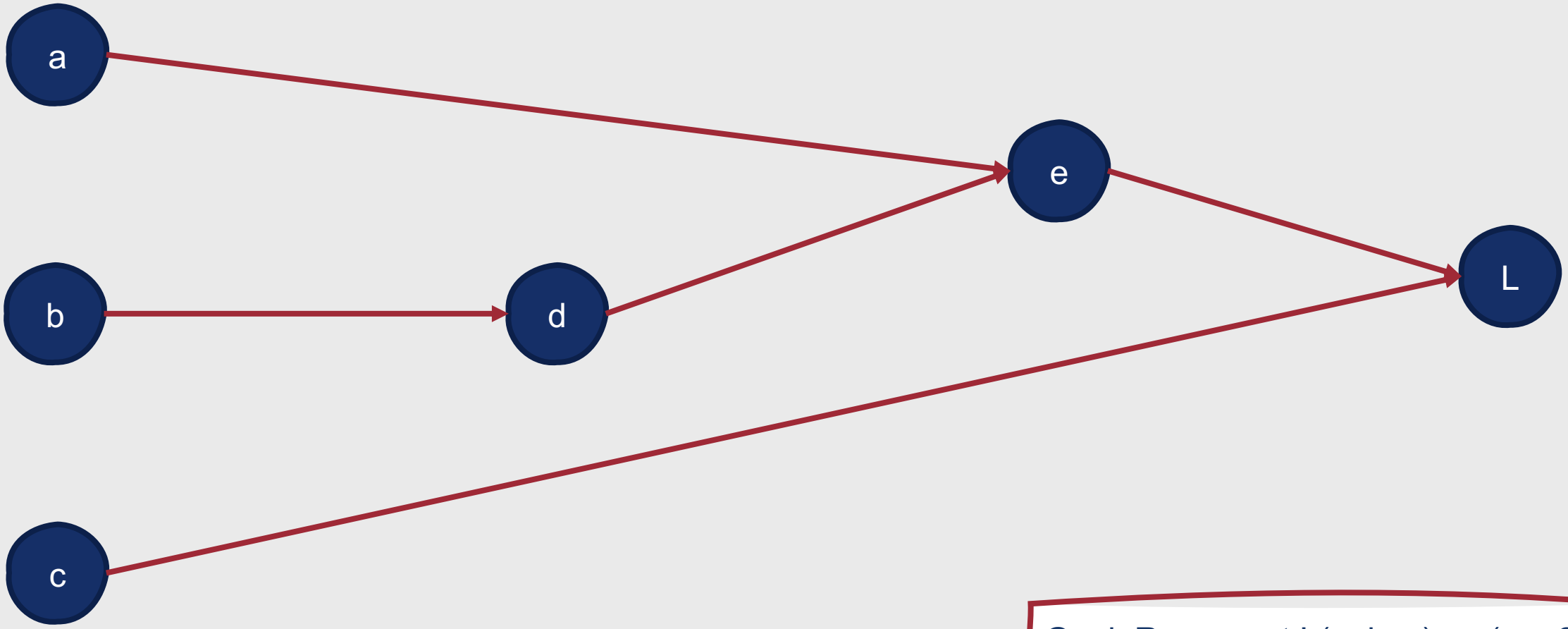
There are two different ways that we can pass information through our neural network computation graphs.

- **Forward pass**
  - Apply operations in the direction of the arrows
  - Pass the output of one computation as the input to the next
- **Backward pass**
  - Compute partial derivatives in the opposite direction of the arrows
  - Multiply them by the partial derivatives passed down from the previous step

Example: Forward Pass

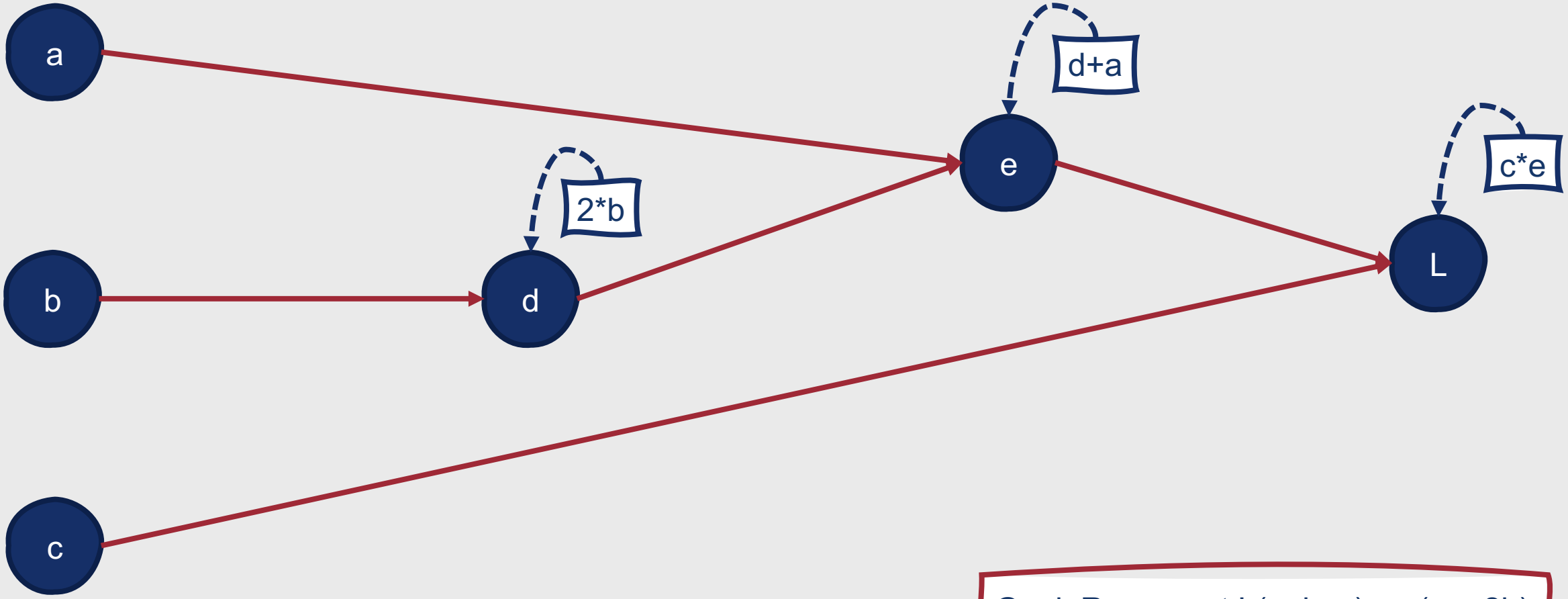
Goal: Represent L(a, b, c) = c(a + 2b)

Example: Forward Pass



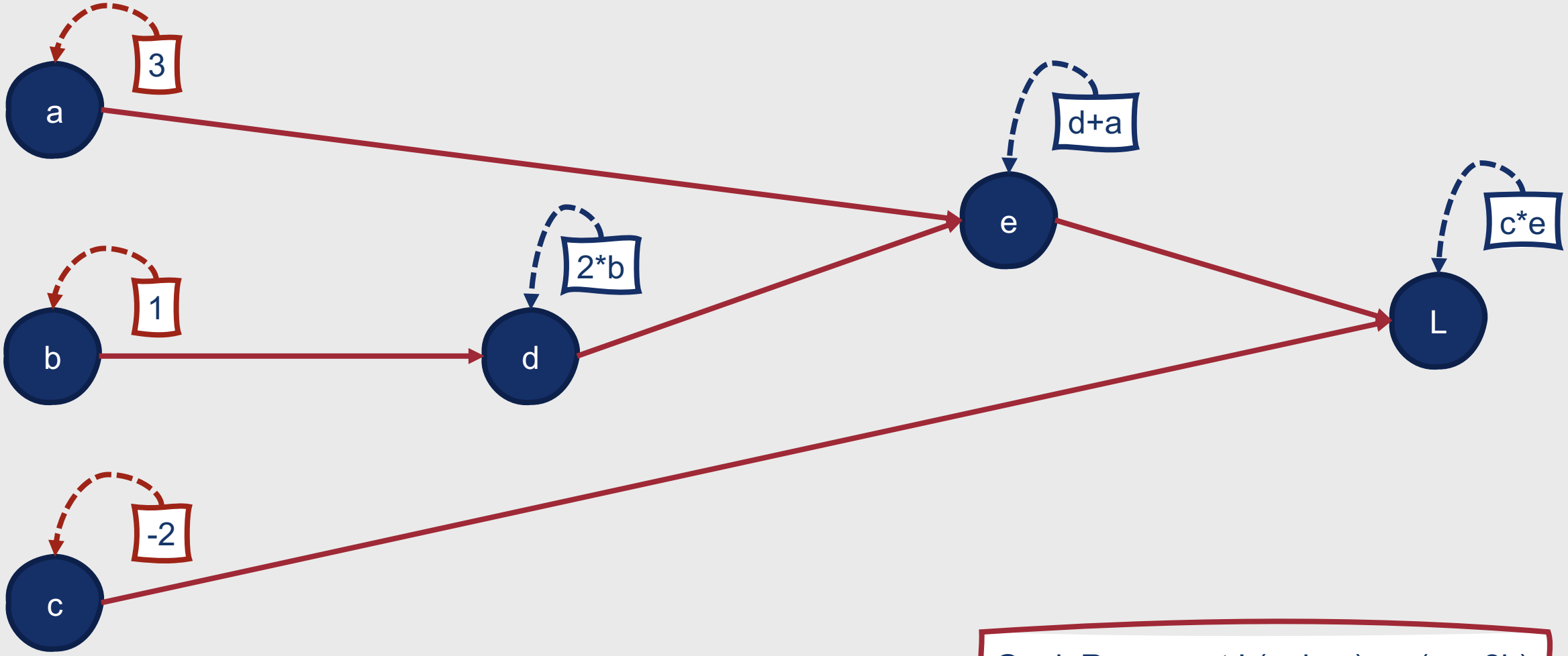
Goal: Represent L(a, b, c) = c(a + 2b)

Example: Forward Pass



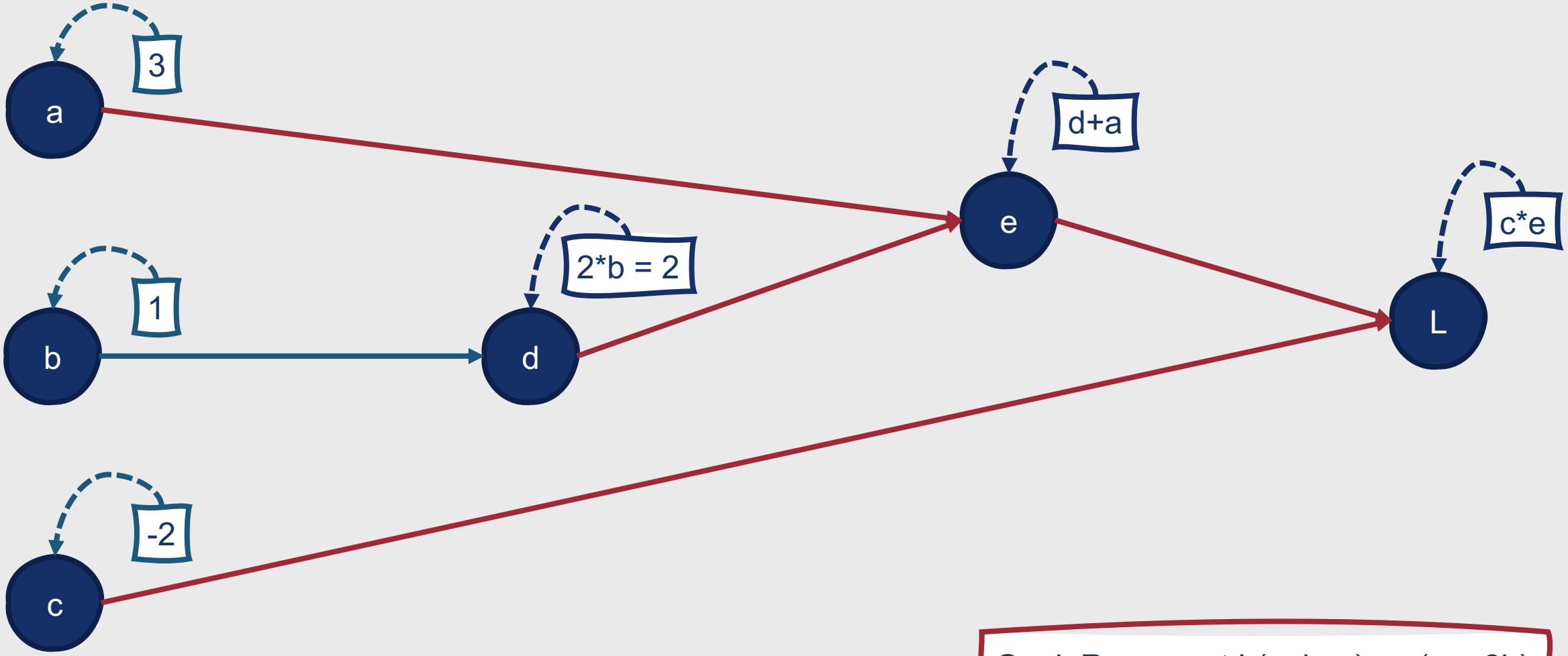
Goal: Represent L(a, b, c) = c(a + 2b)

Example: Forward Pass



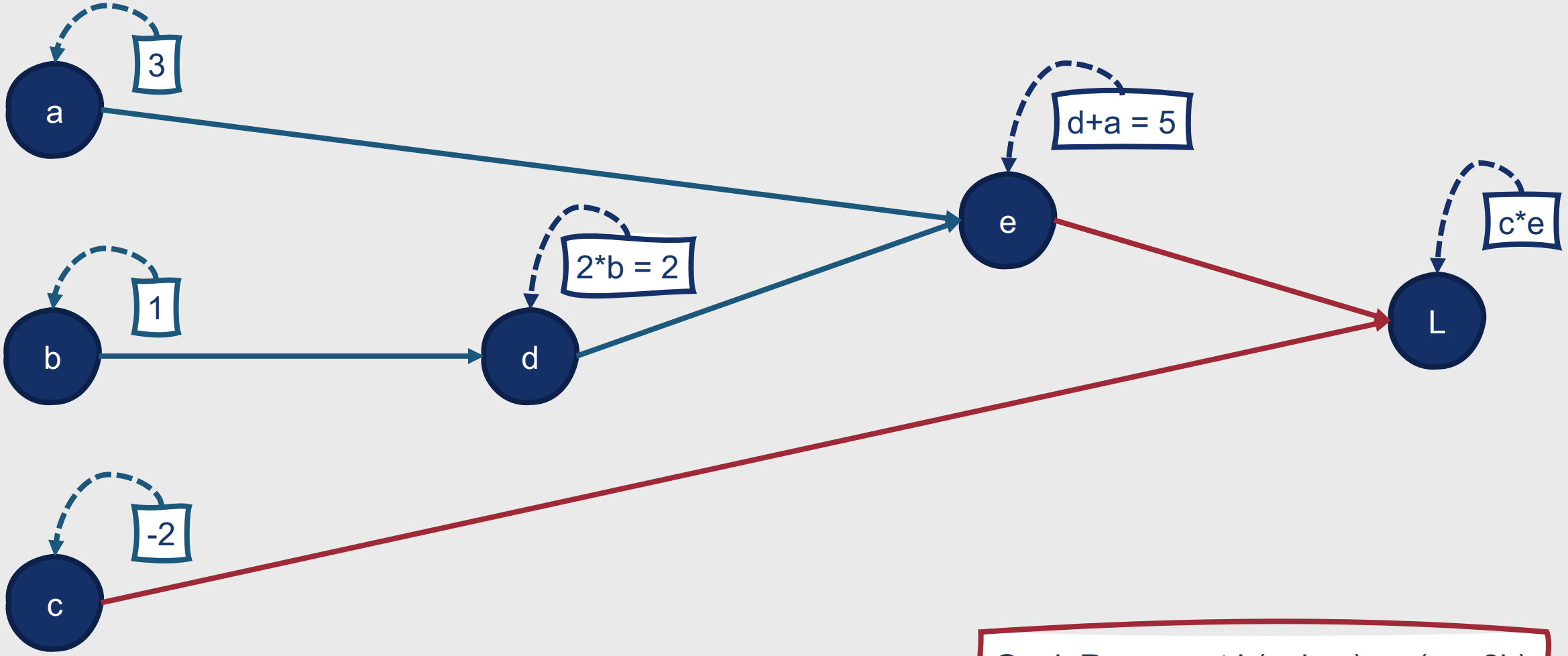
Goal: Represent L(a, b, c) = c(a + 2b)

Example: Forward Pass

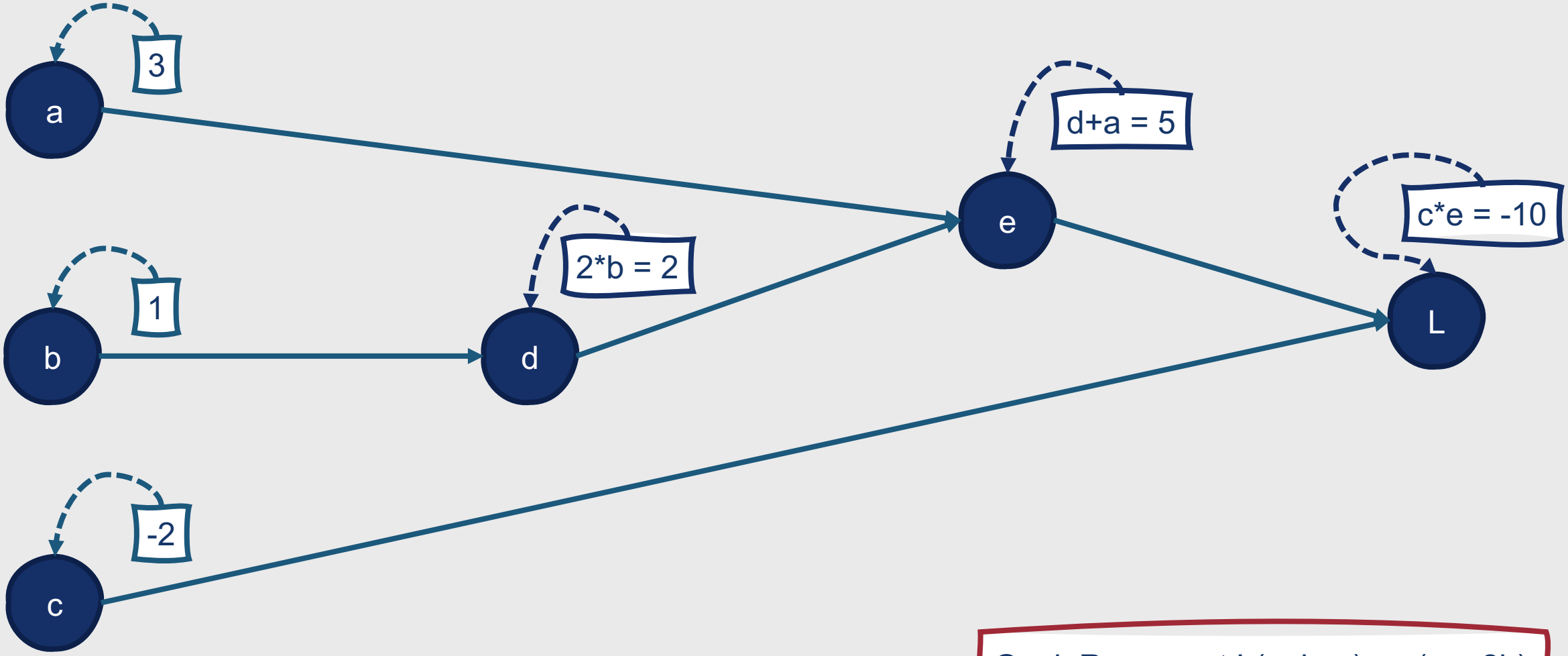


Goal: Represent L(a, b, c) = c(a + 2b)

Example: Forward Pass

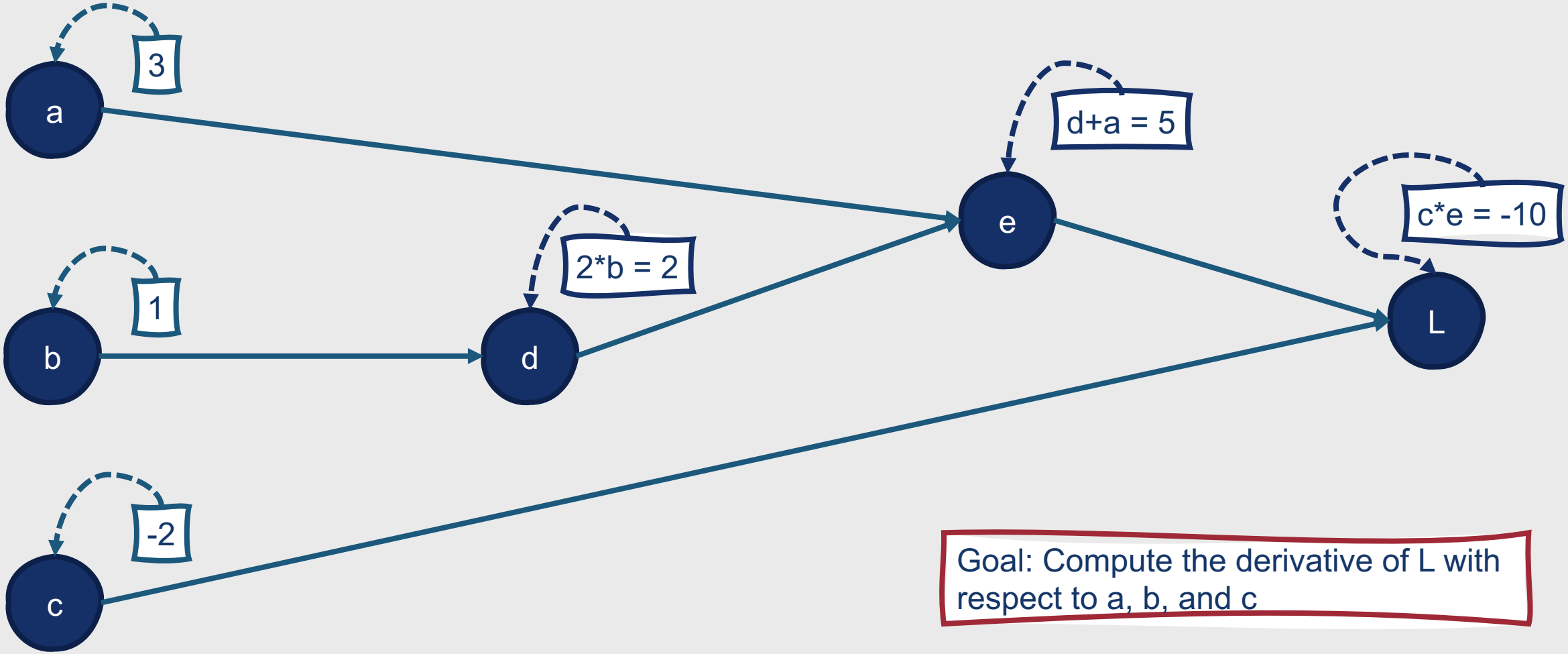


Example: Forward Pass



Goal: Represent L(a, b, c) = c(a + 2b)

Example: Backward Pass

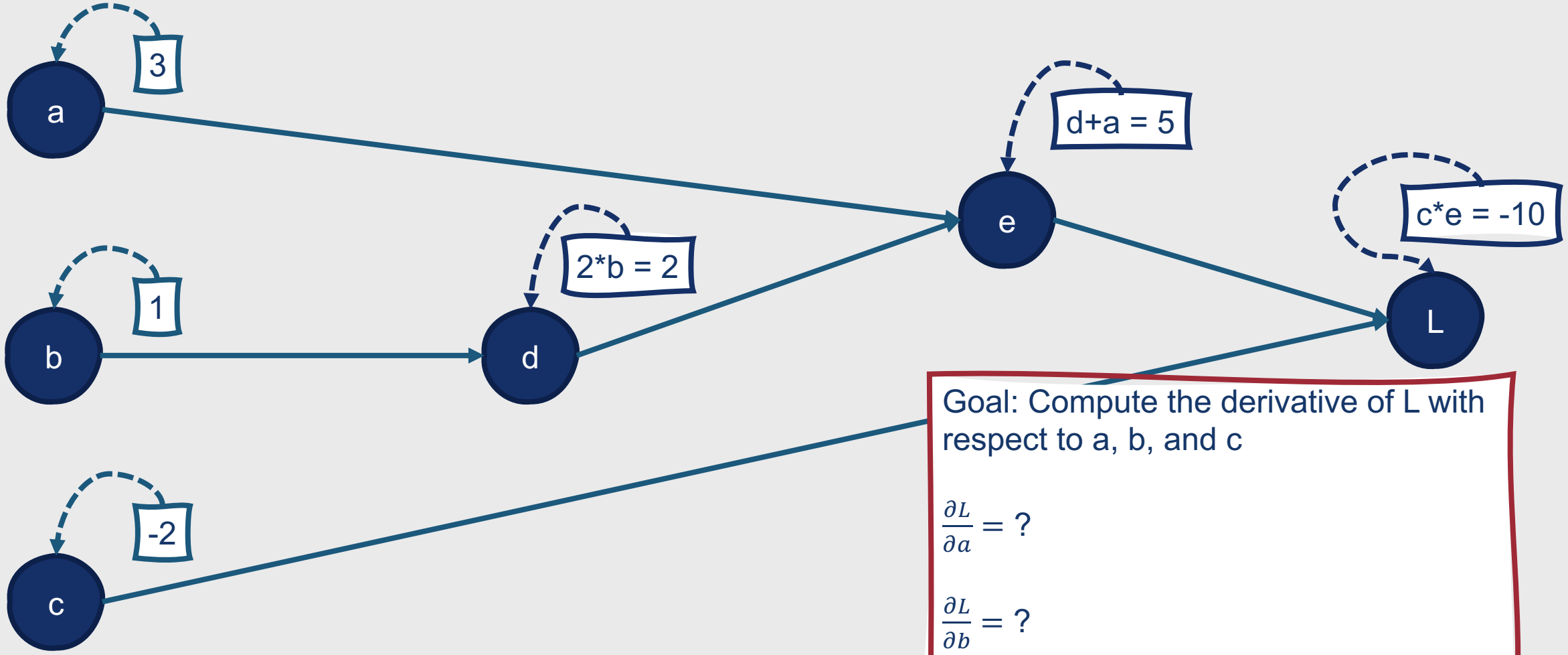


Goal: Compute the derivative of L with respect to a, b, and c

How do we get from L all the way back to a, b, and c?

- Chain rule!
  - Given a function f(x) = u(v(x)):
    - Find the derivative of u(x) with respect to v(x)
    - Find the derivative of v(x) with respect to x
    - Multiply the two together
    - $\frac{df}{dx} = \frac{du}{dv} * \frac{dv}{dx}$

Example: Backward Pass



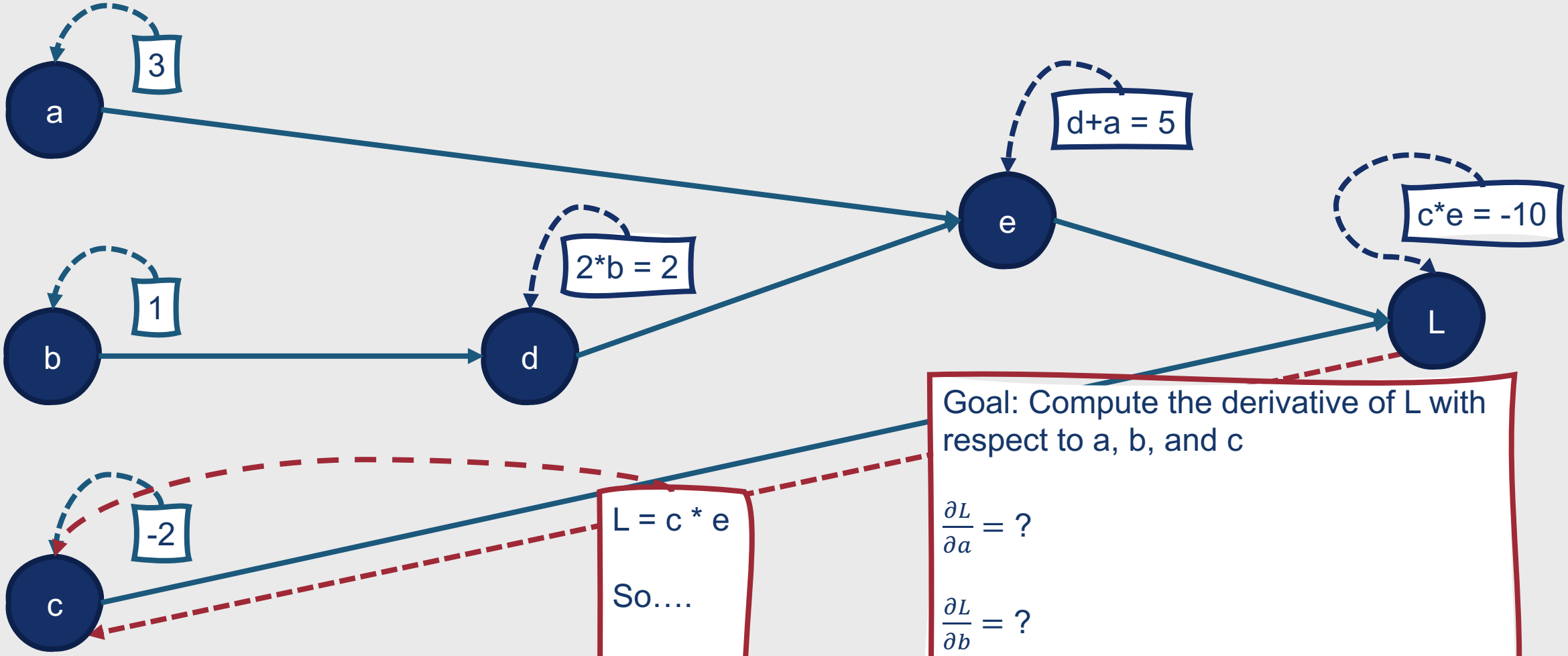
Goal: Compute the derivative of L with respect to a, b, and c

$\frac{\partial L}{\partial a} = ?$

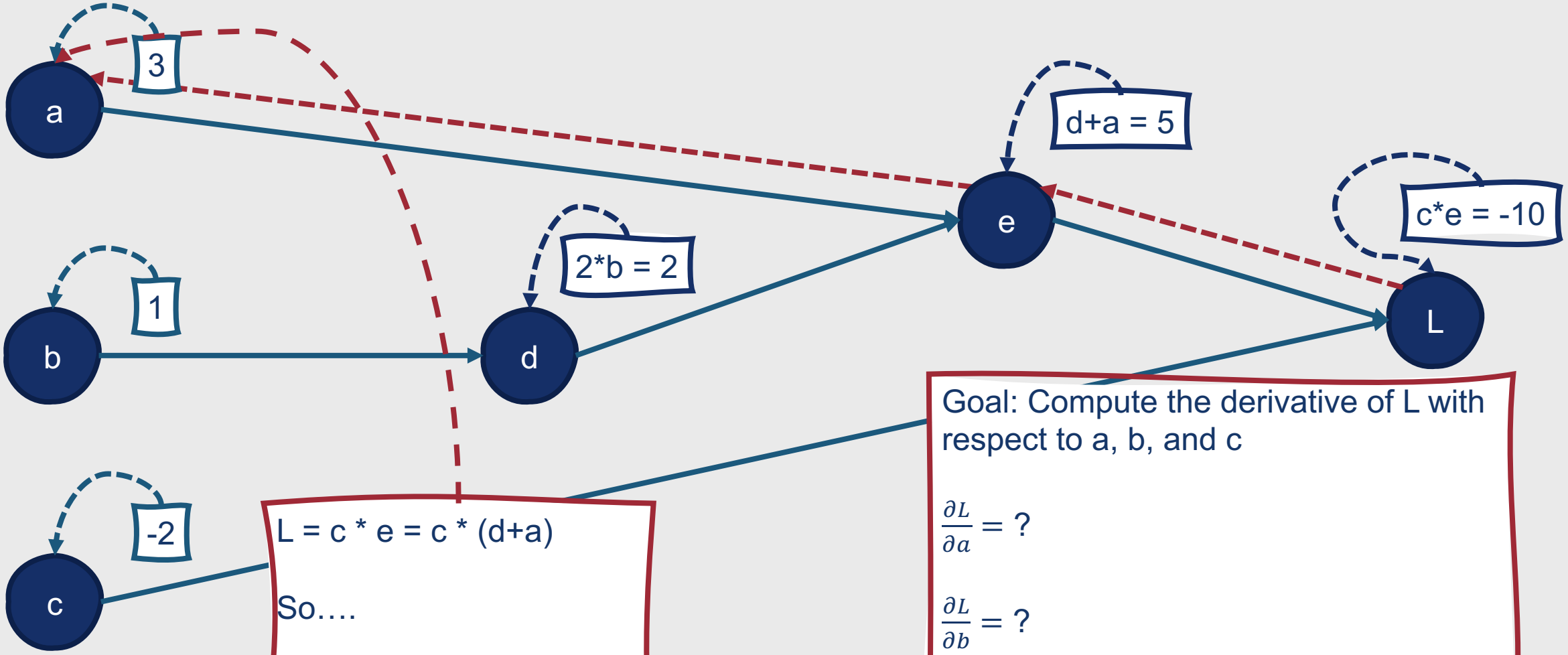
$\frac{\partial L}{\partial b} = ?$

$\frac{\partial L}{\partial c} = ?$

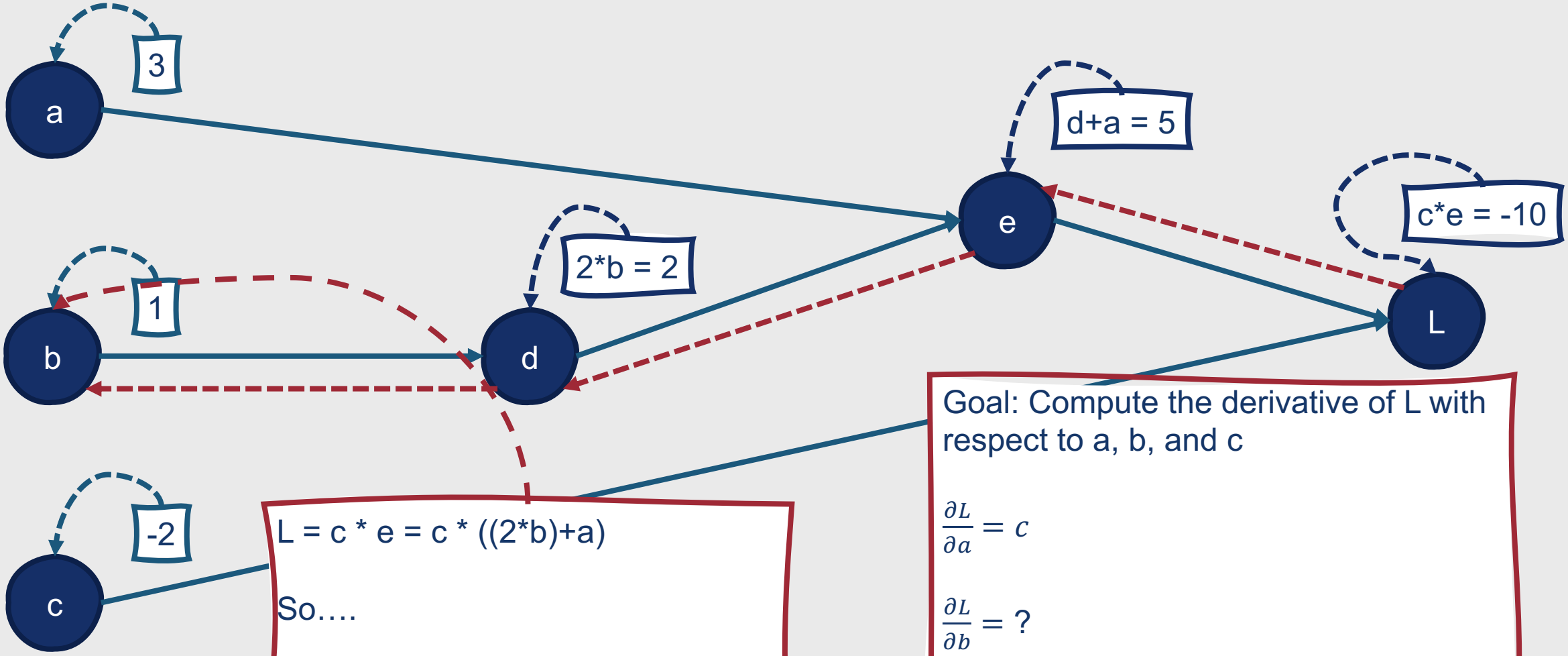
Example: Backward Pass



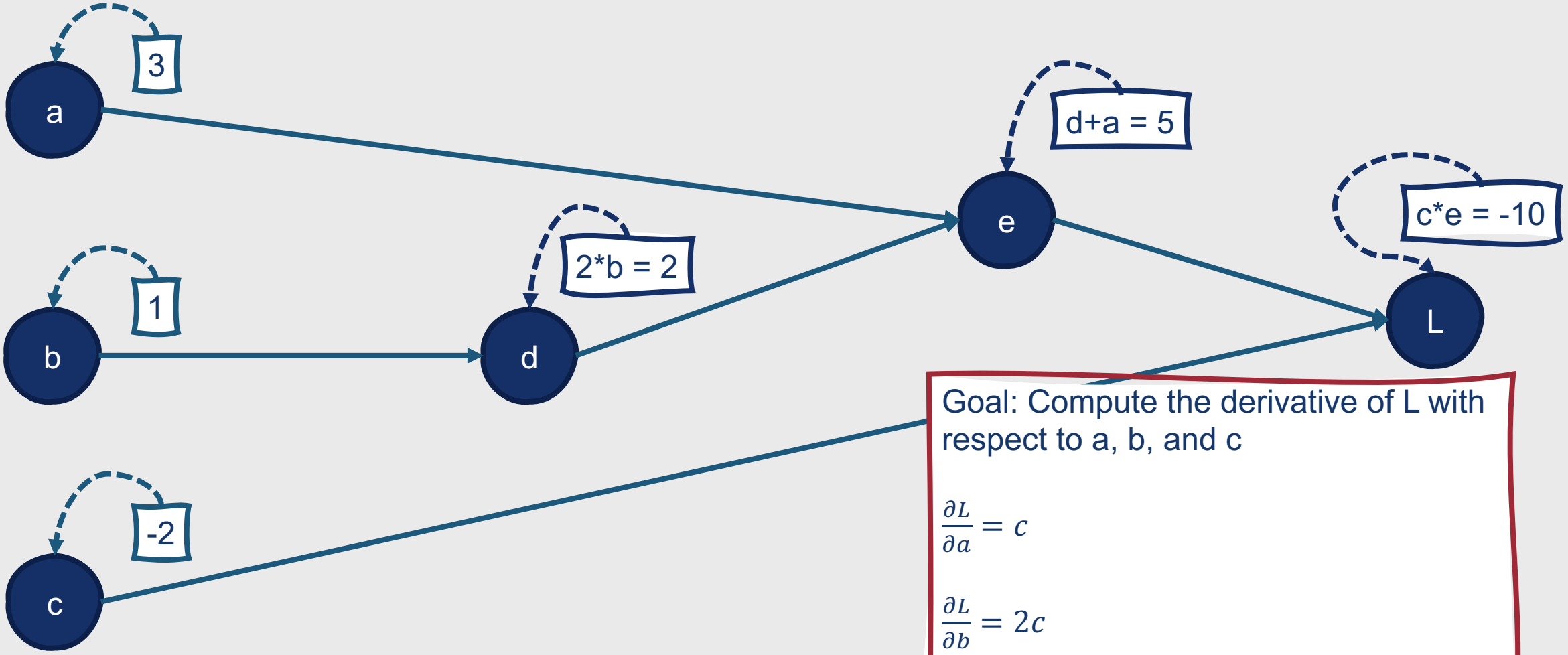
Example: Backward Pass



Example: Backward Pass



Example: Backward Pass



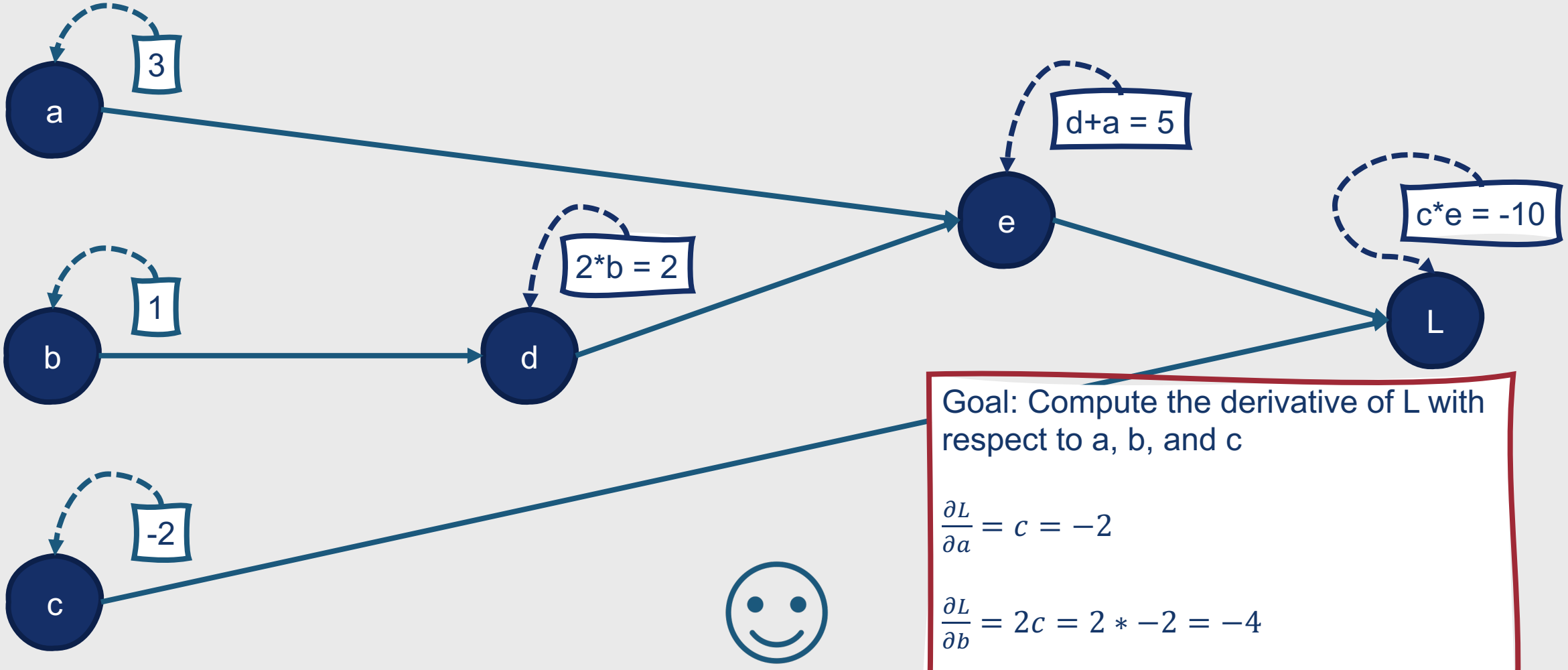
Goal: Compute the derivative of L with respect to a, b, and c

$$\frac{\partial L}{\partial a} = c$$

$$\frac{\partial L}{\partial b} = 2c$$

$$\frac{\partial L}{\partial c} = e$$

Example: Backward Pass



Goal: Compute the derivative of L with respect to a, b, and c

$$\frac{\partial L}{\partial a} = c = -2$$

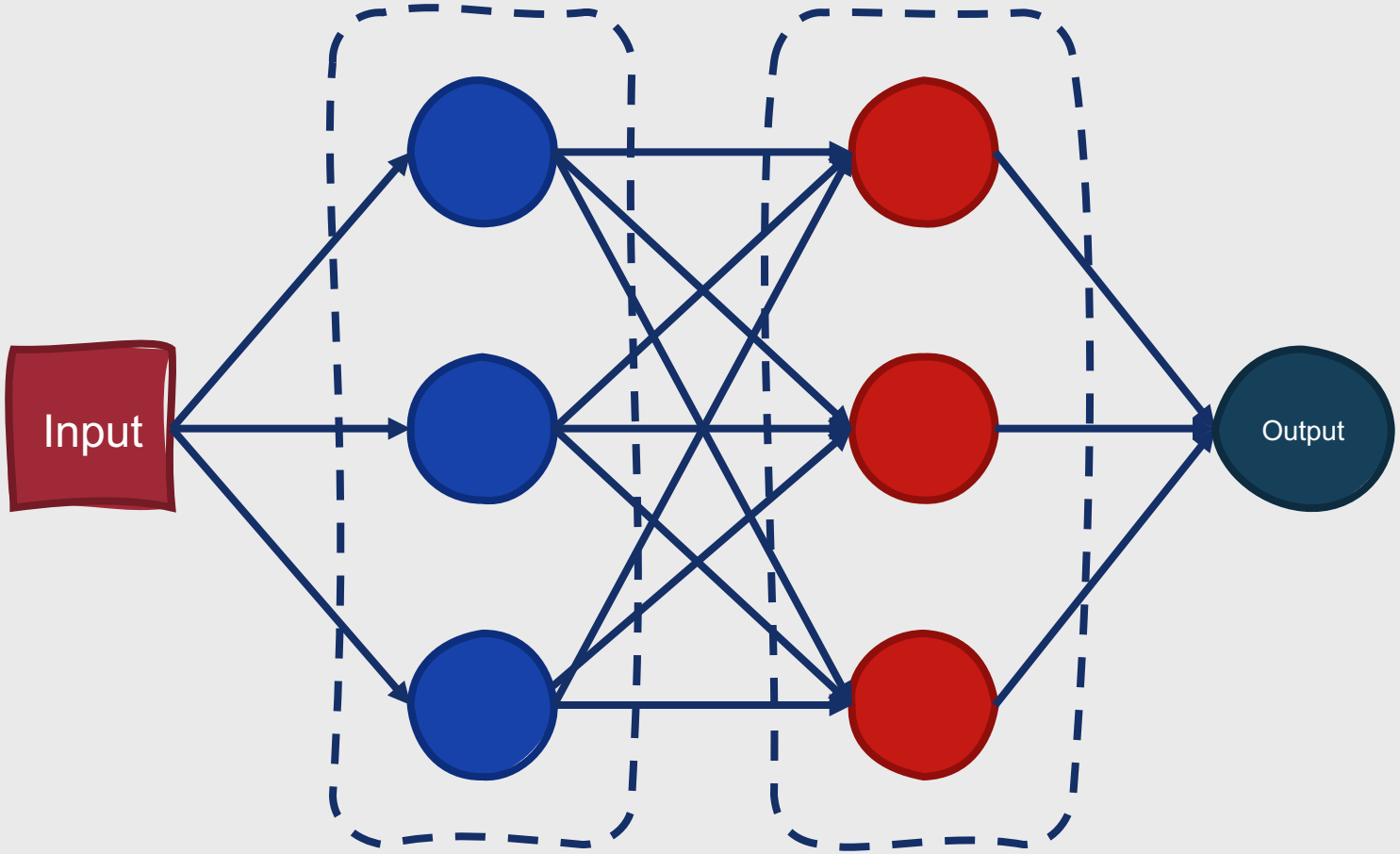
$$\frac{\partial L}{\partial b} = 2c = 2 * -2 = -4$$

$$\frac{\partial L}{\partial c} = e = 5$$

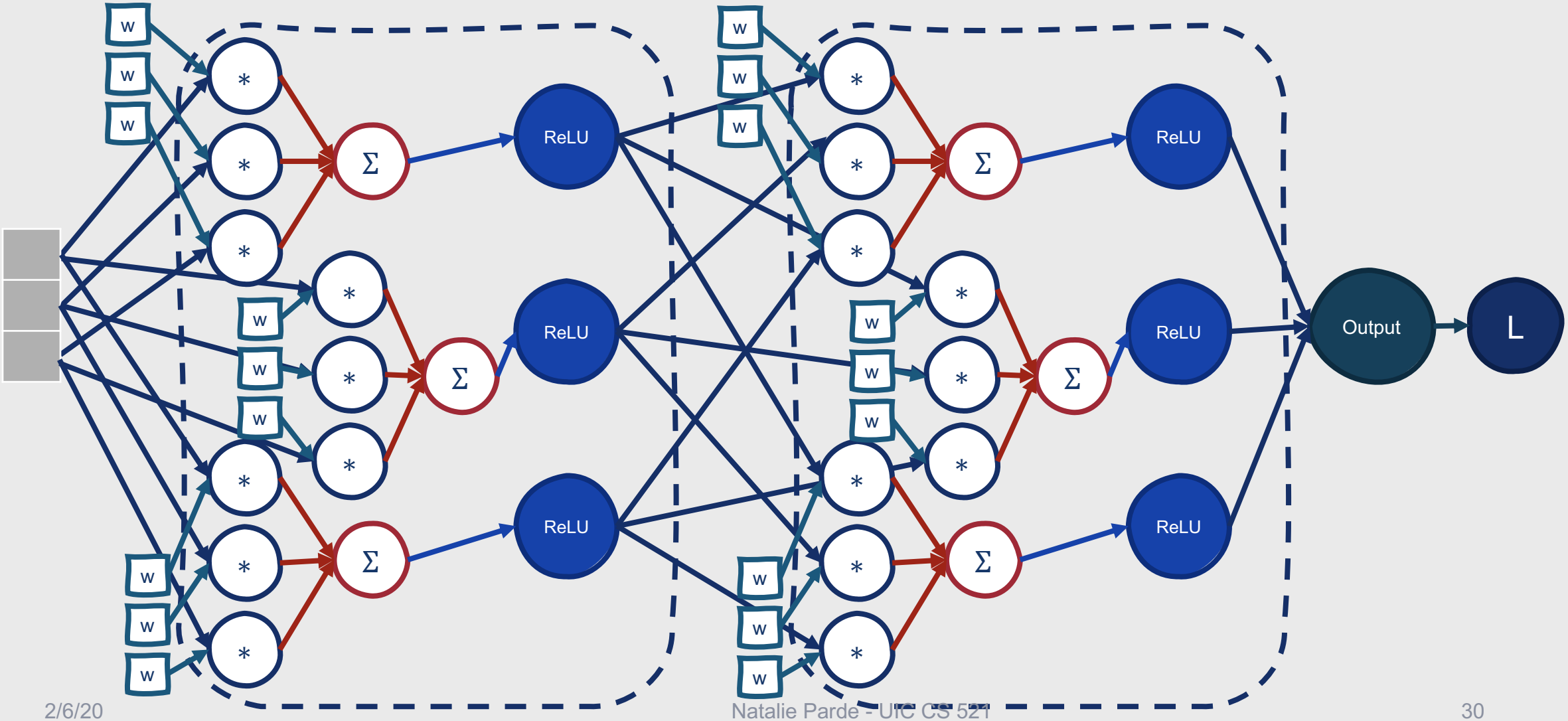
Computation graphs for neural networks are a bit more complex than the previous example.

- More operations:
  - Products (input * weight)
  - Summations (of weighted inputs)
  - Activation functions

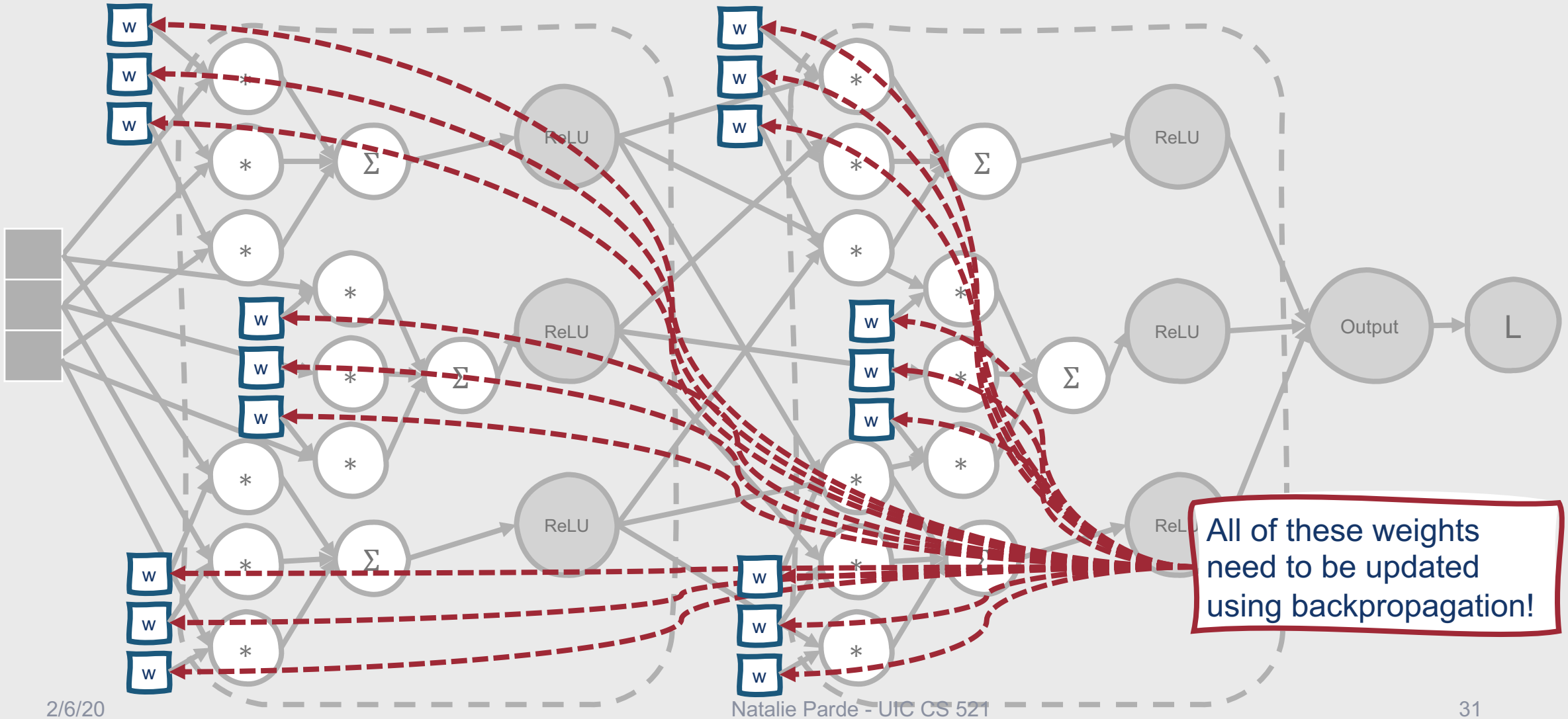
What would a computation graph look like for a simple neural network?



What would a computation graph look like for a simple neural network?



What would a computation graph look like for a simple neural network?



What are the derivatives of some other common activation functions?

- tanh
  - $\frac{\partial \tanh(z)}{\partial z} = 1 - \tanh^2(z)$
- ReLU
  - $\frac{\partial \text{ReLU}(z)}{\partial z} = \begin{cases} 0 \text{ for } z < 0 \\ 1 \text{ for } z \geq 0 \end{cases}$

General Tips for Improving Neural Network Performance

- **Normalize input values** to have a mean of 0
- **Initialize weights** with small random numbers
- Randomly drop some units and their connections from the network during training (**dropout**)
- **Tune hyperparameters**
  - Learning rate
  - Number of layers
  - Number of units per layer
  - Type of activation function
  - Type of optimization function

Fortunately, you shouldn't need to build your computation graphs from scratch!

TensorFlow

- *https://www.tensorflow.org/*

Keras

- *https://keras.io/*

PyTorch

- *https://pytorch.org/*

DL4J

- *https://deeplearning4j.org/*

Neural Language Models

- Popular application of neural networks
- Advantages over *n*-gram language models:
  - Can handle longer histories
  - Can generalize over contexts of similar words
- Disadvantage:
  - Slower to train
- Neural language models have higher predictive accuracy than n-gram language models trained on datasets of similar sizes

Neural Language Models

- Neural language models are used to boost performance for many modern NLP tasks
  - Machine translation
  - Dialogue systems
  - Language generation

Sample Generated by a Neural Language Model (GPT-2)

- Link to article: https://openai.com/blog/better-language-models/

System Prompt (Human-Written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

Sample Generated by a Neural Language Model (GPT-2)

Model Completion (Machine-Written, 10 Tries): The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

Feedforward Neural Language Model

- Input: Representation of some number of previous words
  - w_{t-1}, w_{t-2}, etc.
- Output: Probability distribution over possible next words
- Goal: Approximate the probability of a word given the entire prior context $P(w_t|w_1^{t-1})$ based on the *n* previous words
  - $P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-n+1}^{t-1})$

Neural language models represent prior context using embeddings of the previous words.

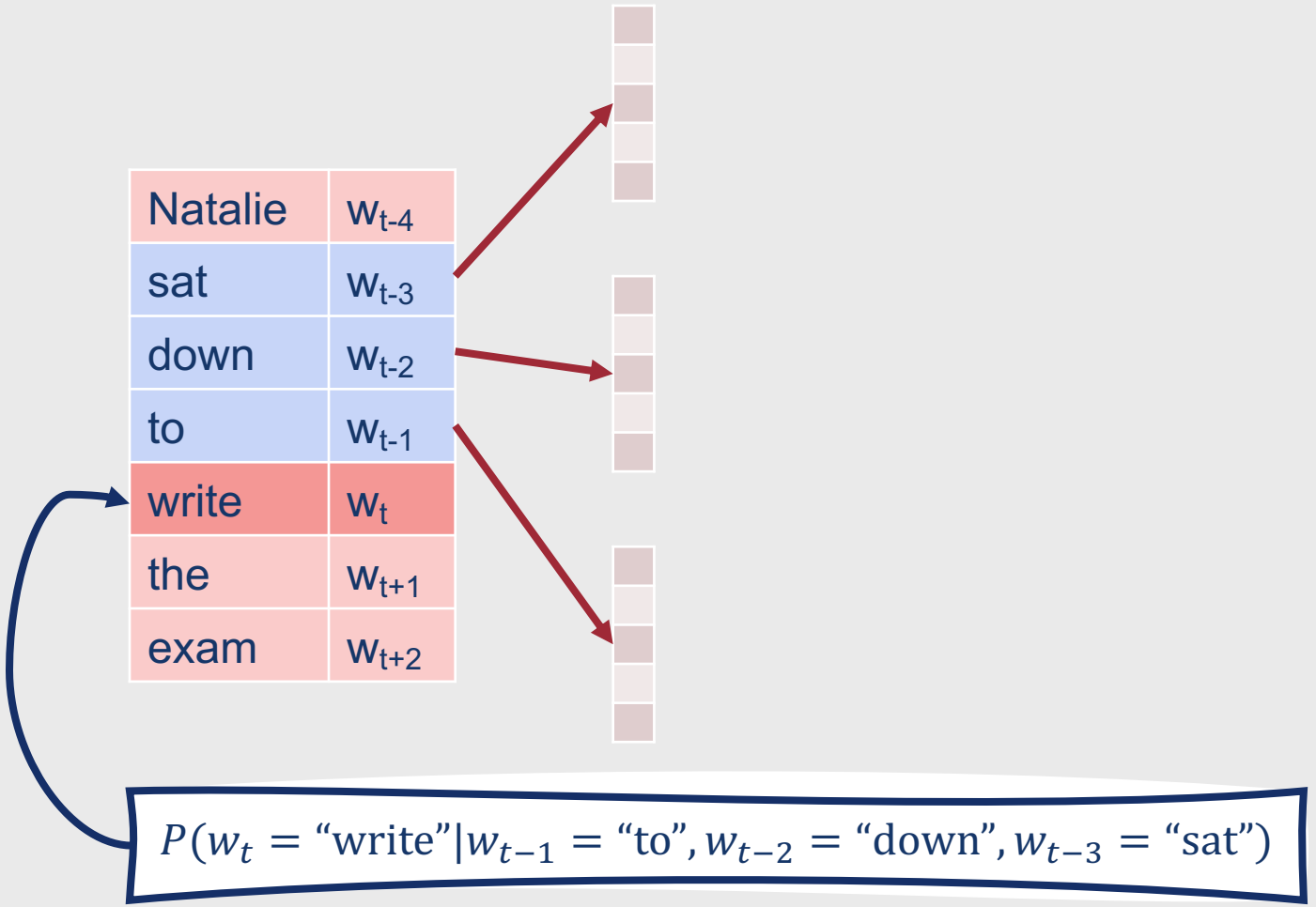
- Allows them to generalize to unseen data better than *n*-gram models
- Embeddings can come from various sources
  - E.g., pretrained Word2Vec embeddings

Neural Language Model

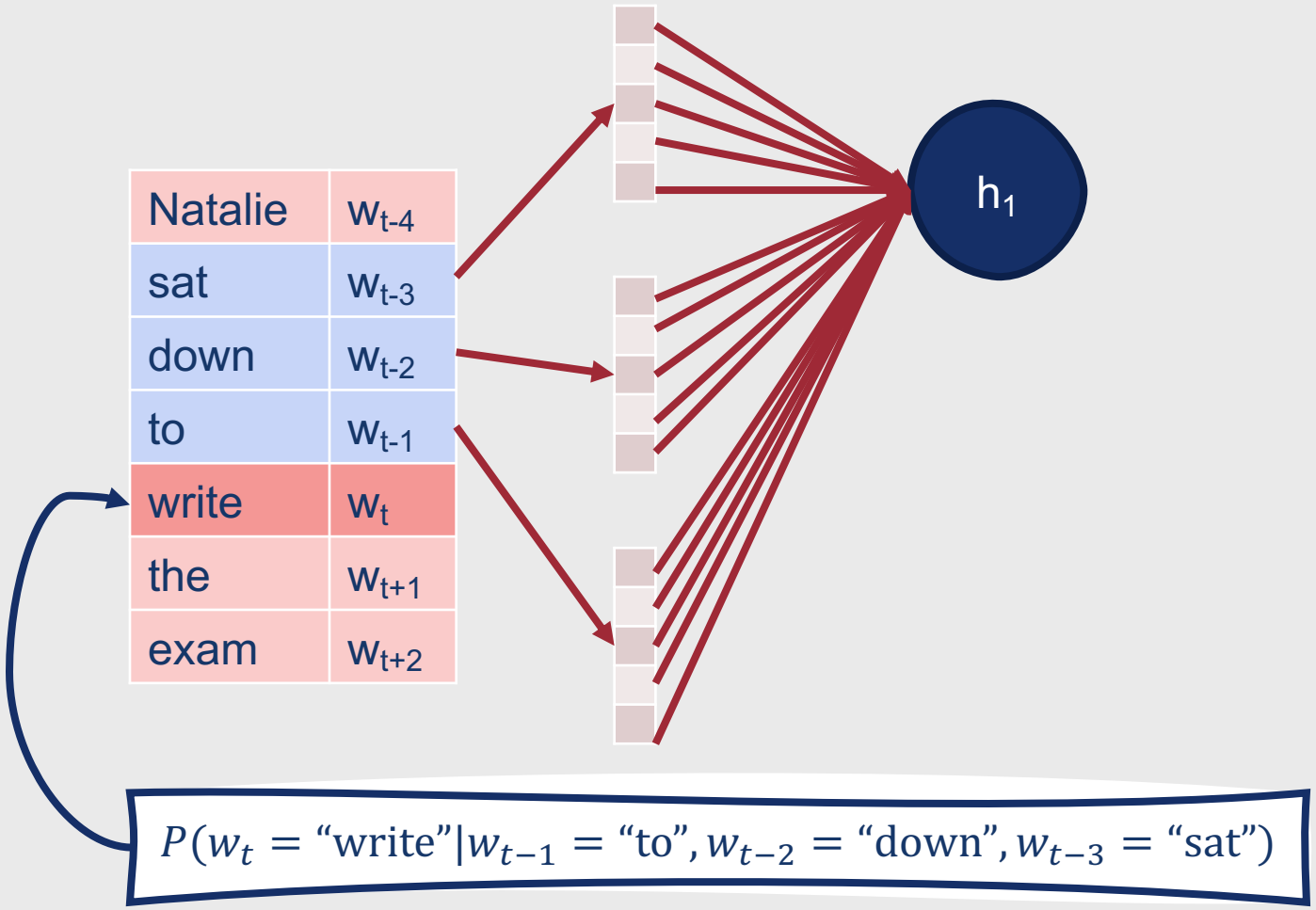
Natalie	w_{t-4}
sat	w_{t-3}
down	w_{t-2}
to	w_{t-1}
write	w_t
the	w_{t+1}
exam	w_{t+2}

$$P(w_t = \text{“write”} | w_{t-1} = \text{“to”}, w_{t-2} = \text{“down”}, w_{t-3} = \text{“sat”})$$

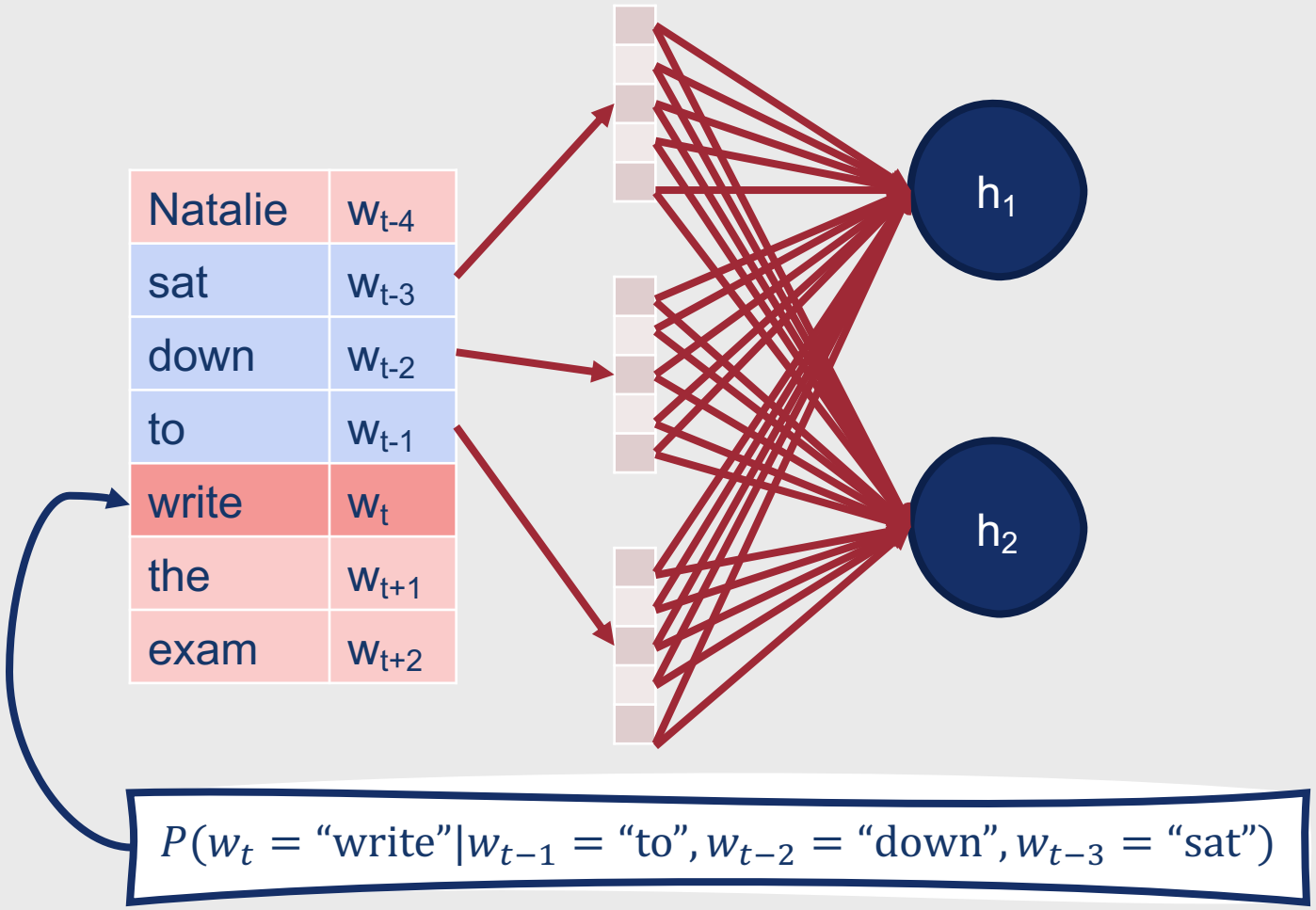
Neural Language Model



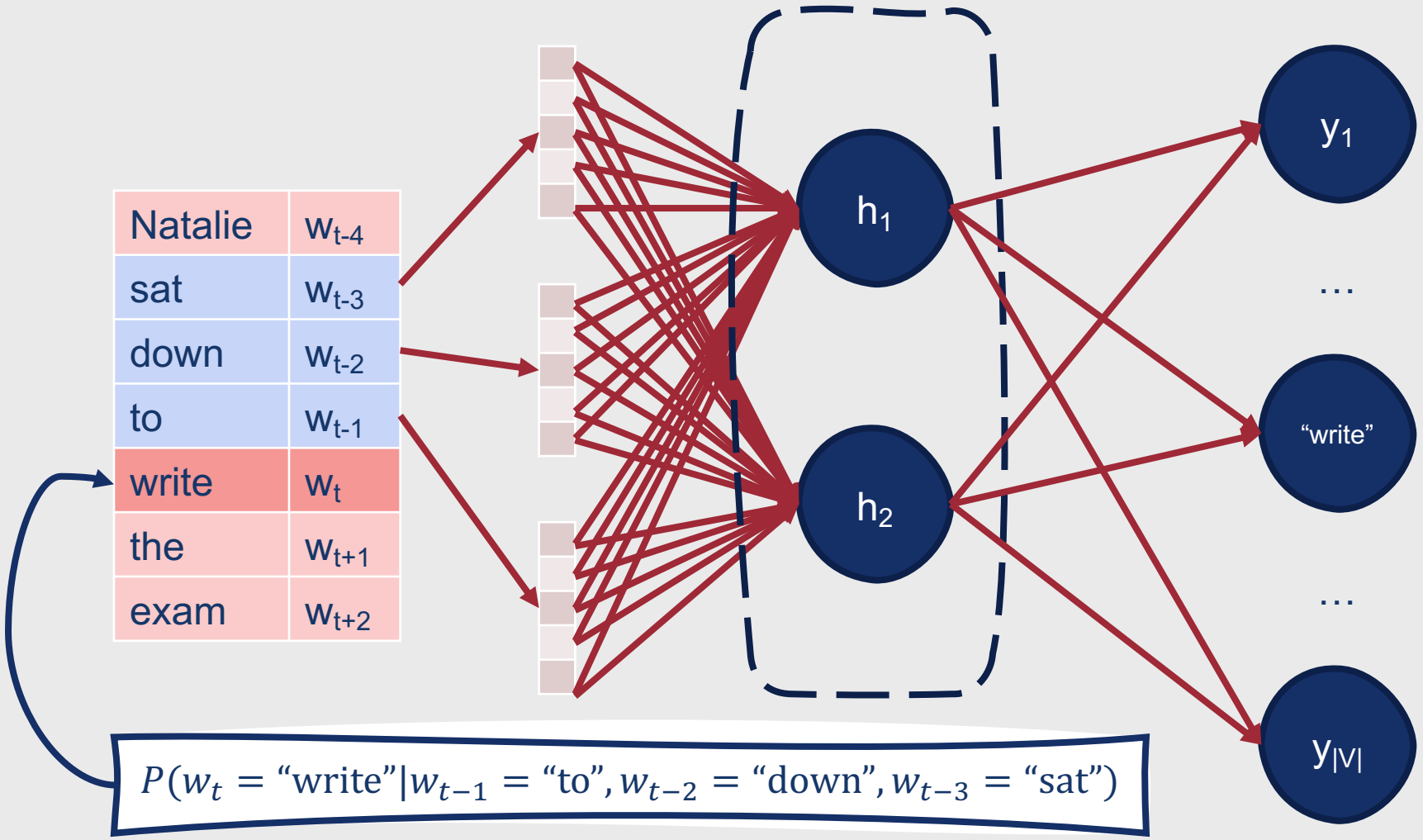
Neural Language Model



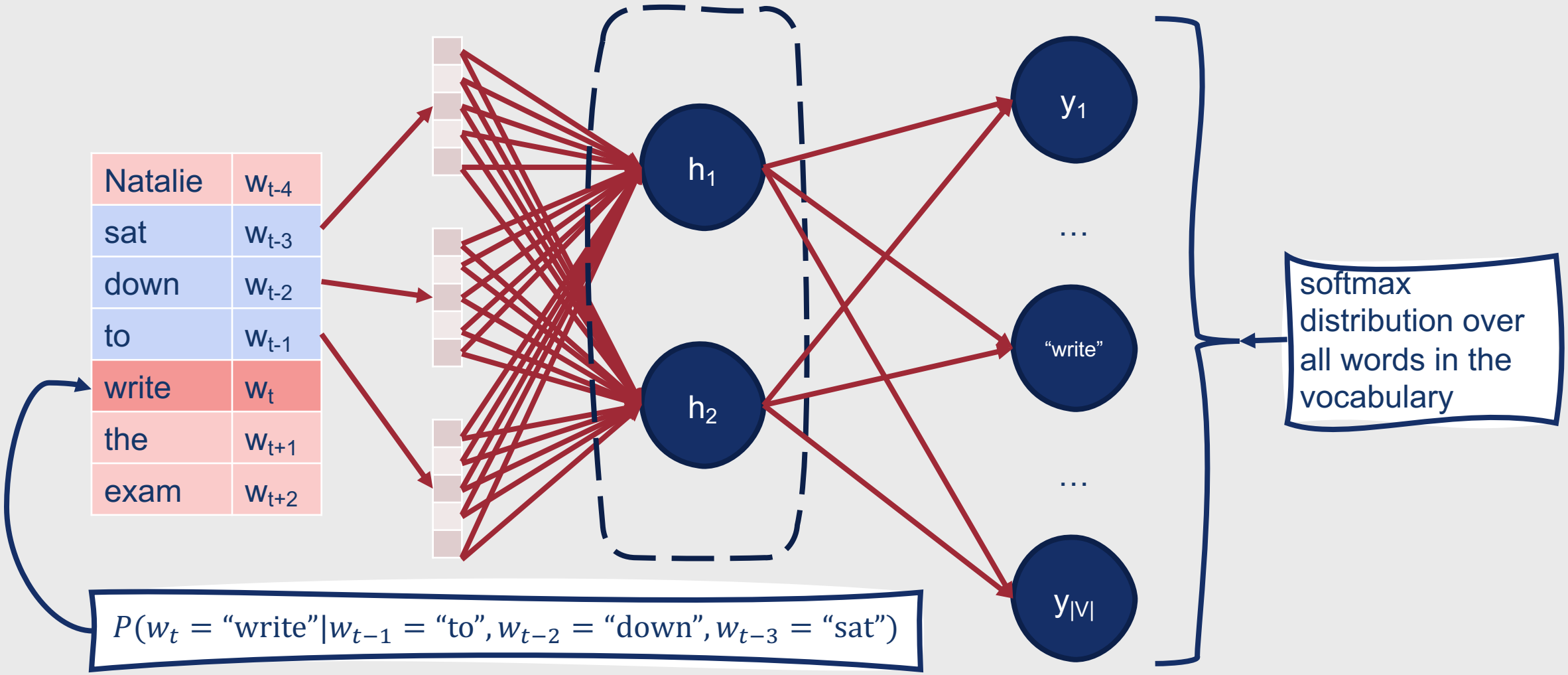
Neural Language Model



Neural Language Model



Neural Language Model



- When we use another algorithm to learn the embeddings for our input words, this is called **pretraining**
- However, sometimes it's preferable to learn embeddings while training the network, rather than using **pretrained embeddings**
  - E.g., if the desired application places strong constraints on what makes a good representation

What if we don't already have dense word embeddings?

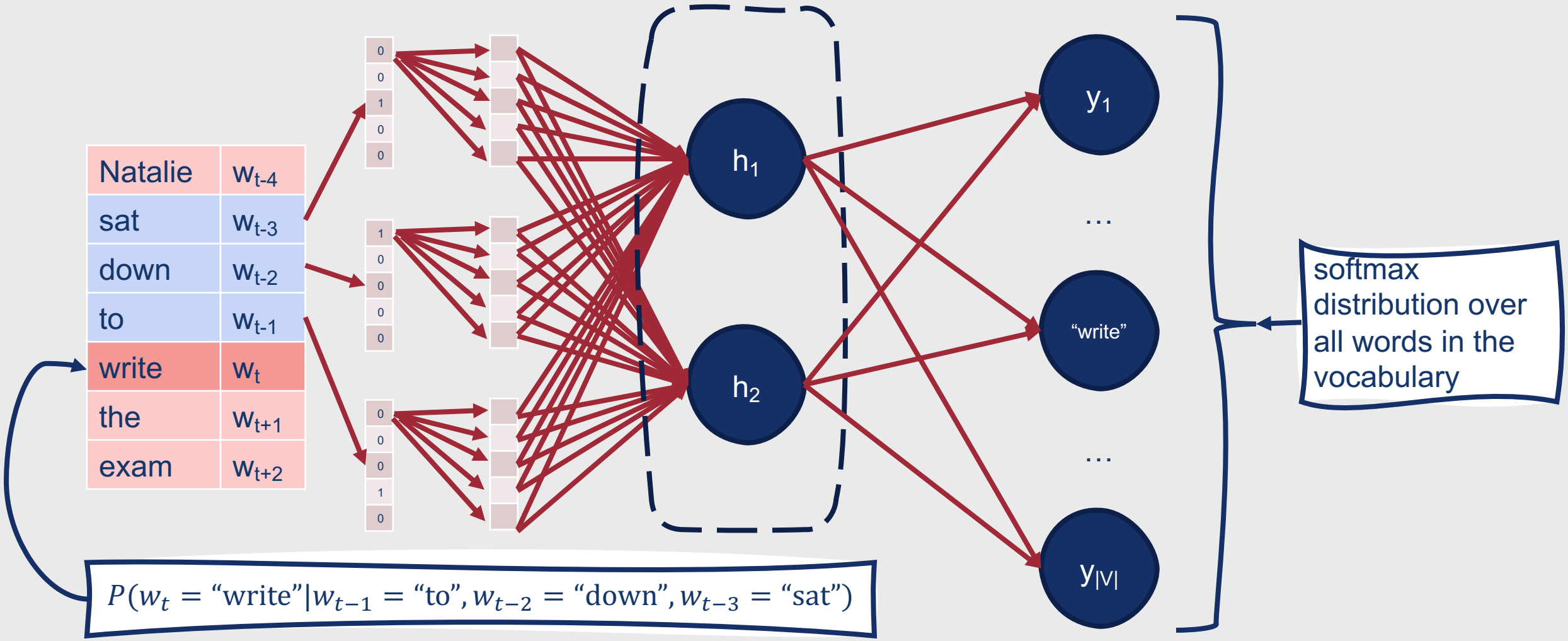
Learning New Embeddings

- Start with a **one-hot vector** for each word in the vocabulary
  - Element for a given word is set to 1
  - All other elements are set to 0
- Randomly initialize the hidden (weight/embedding) layer
- Maintain a separate vector of weights for that layer, for each vocabulary word

Formal Definition: Learning New Embeddings

- Letting *E* be an embedding matrix of dimensionality *d*, with one row for each word in the vocabulary:
  - $\mathbf{e} = (E_{x_1}, E_{x_2}, \ldots, E_{x_n})$
  - $\mathbf{h} = \sigma(W\mathbf{e} + \mathbf{b})$
  - $\mathbf{z} = U\mathbf{h}$
  - $y = \text{softmax}(\mathbf{z})$
- Optimizing this network using the same techniques discussed for other neural networks will result in both
  - A model that predicts words
  - A new set of word embeddings that can be used for other tasks

Neural Language Model

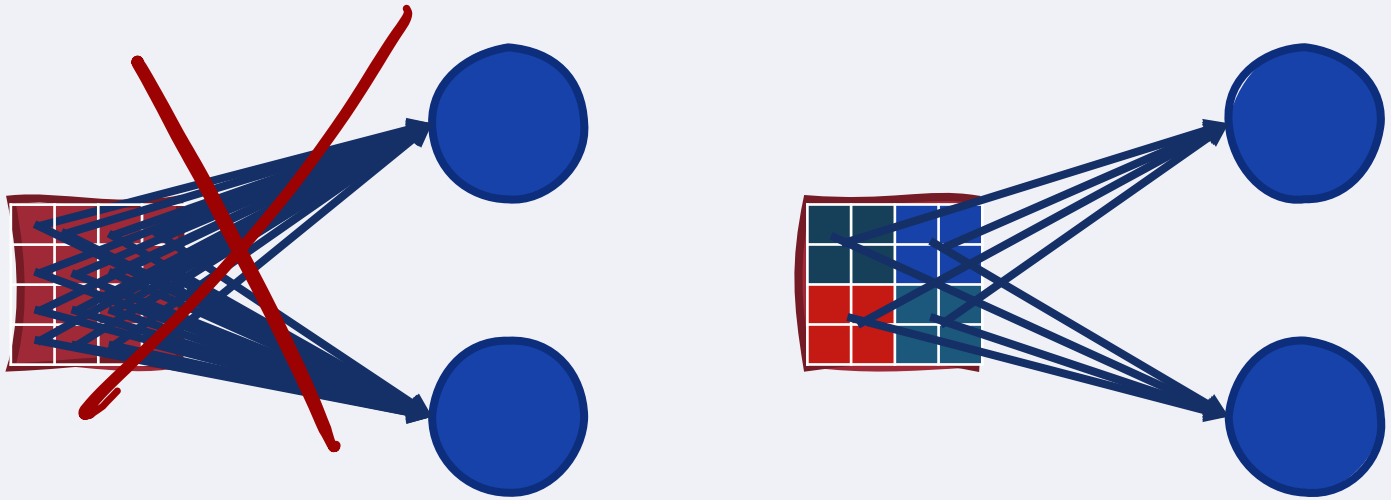


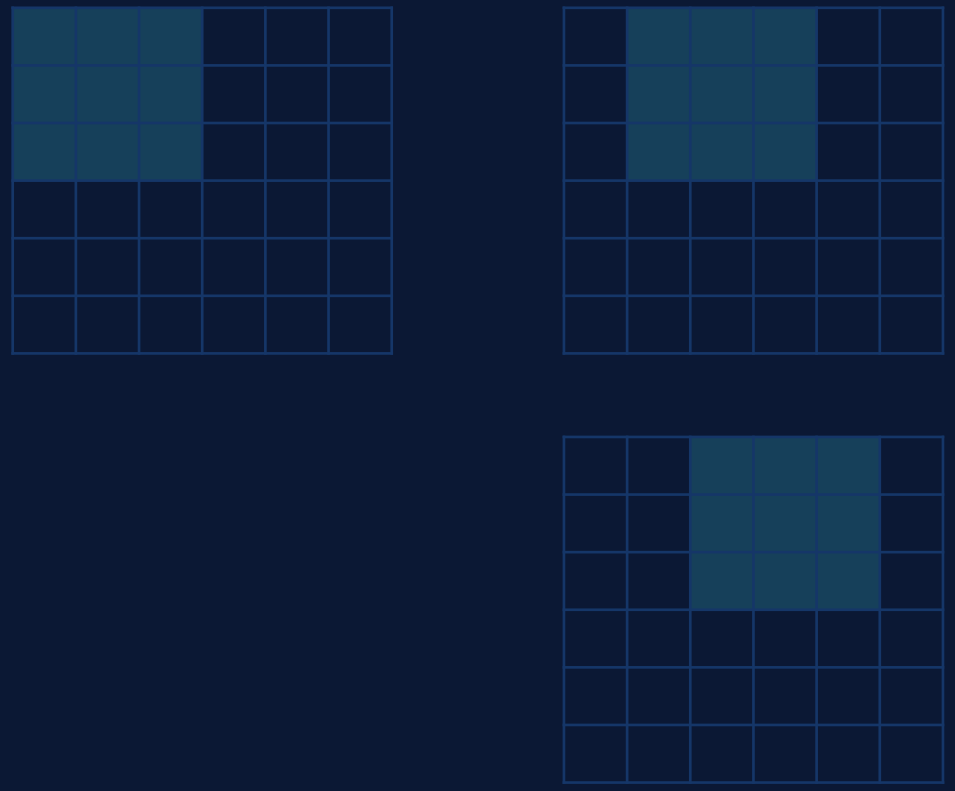
Convolutional Neural Networks

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

What are convolutional layers?

- **Sliding windows** that perform matrix operations on subsets of the input
- Compute products between those subsets of input and a corresponding weight matrix

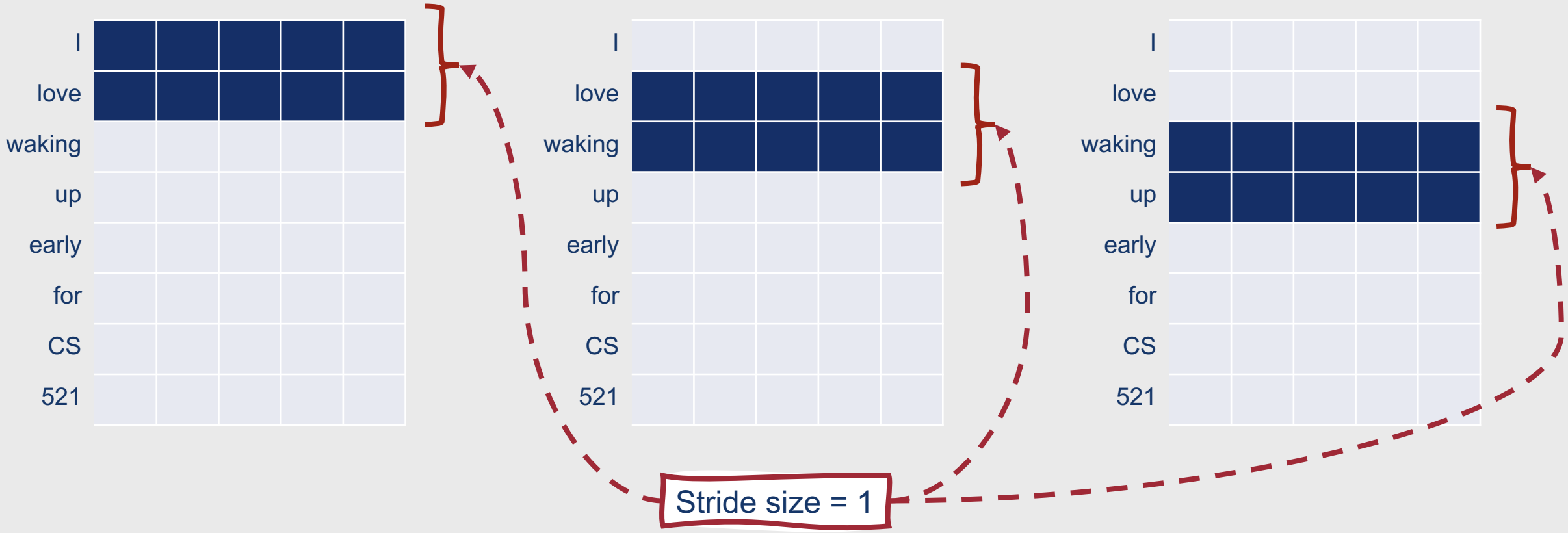




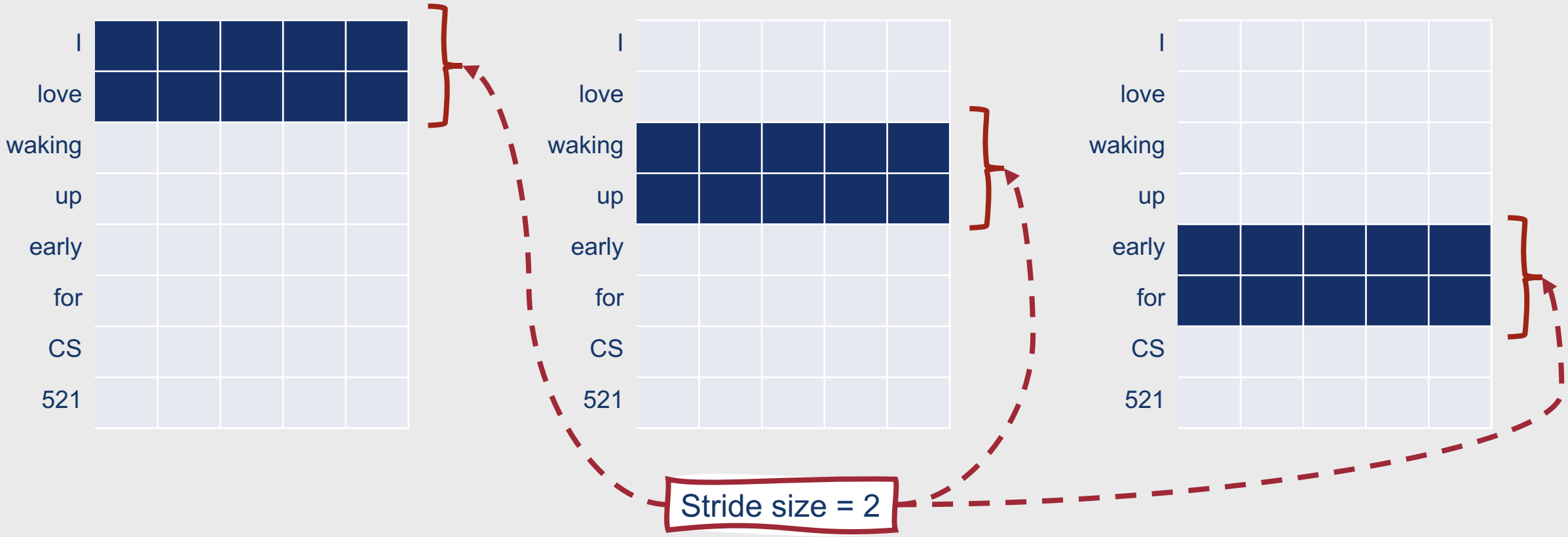
- First layer(s): low-level features
  - Color, gradient orientation
  - N-grams
- Higher layer(s): high-level features
  - Objects
  - Phrases

Convolutional Layers

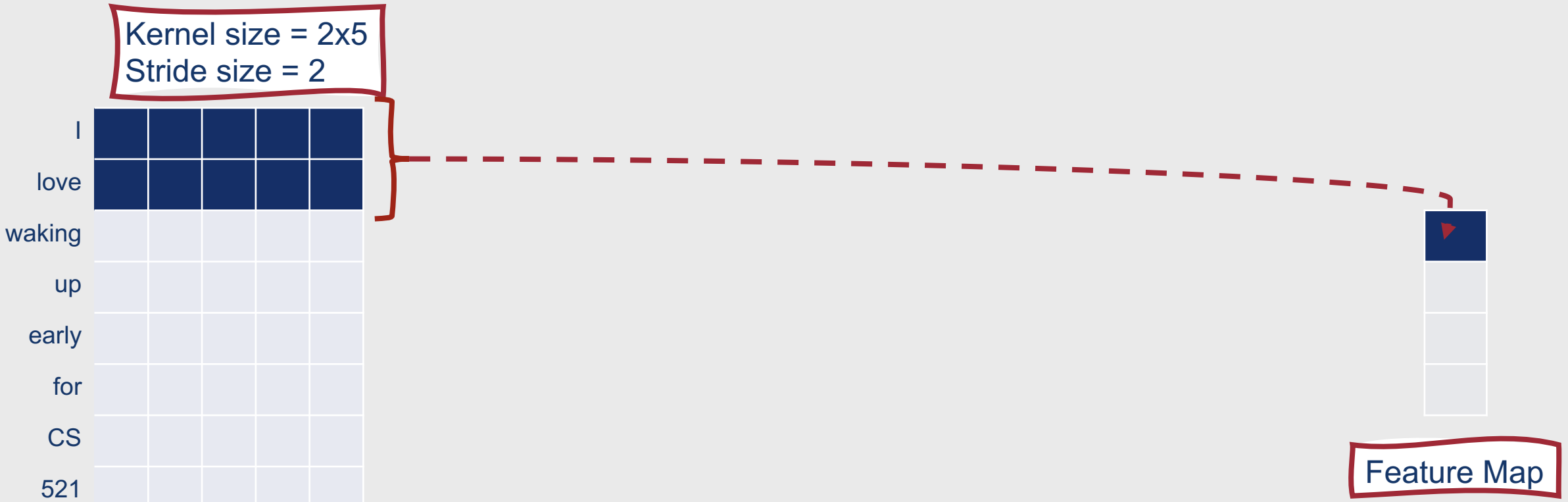
In NLP, convolutions are typically performed on entire rows of an input matrix, where each row corresponds to a word.



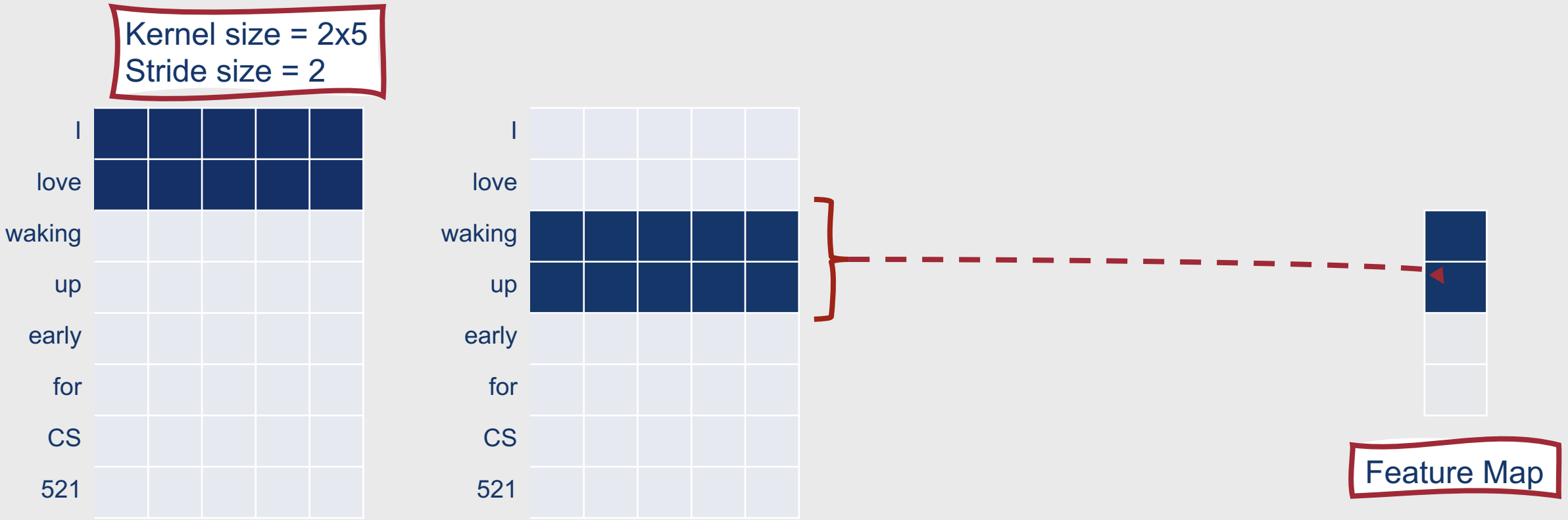
In NLP, convolutions are typically performed on entire rows of an input matrix, where each row corresponds to a word.



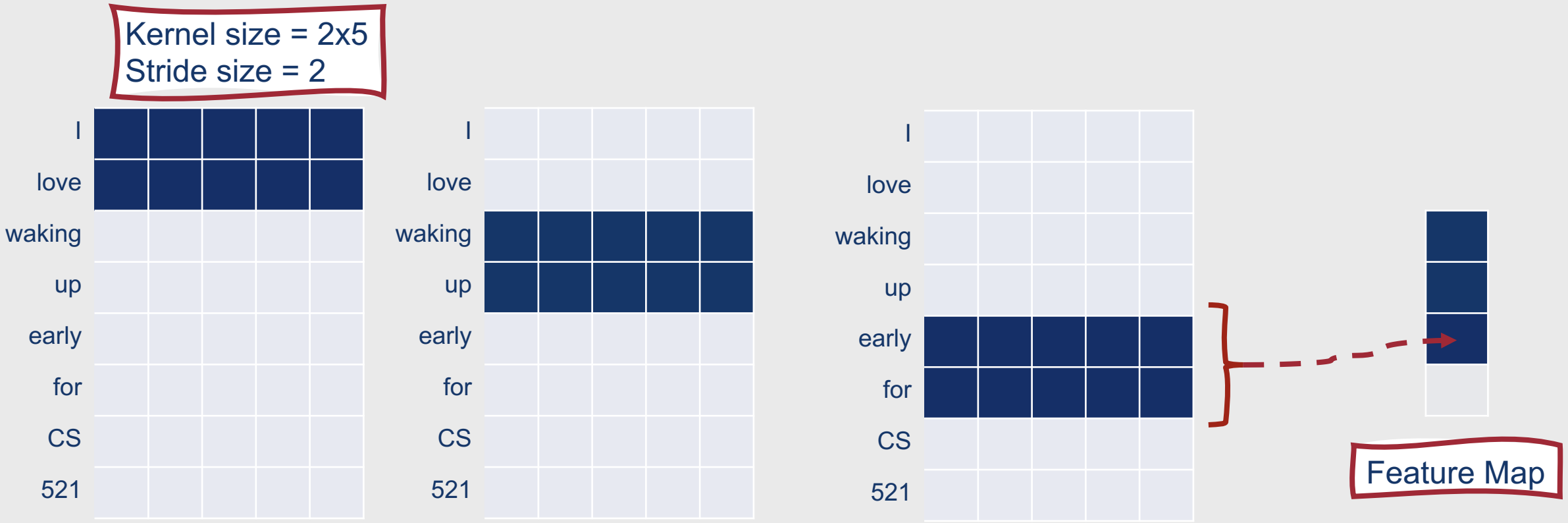
After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.



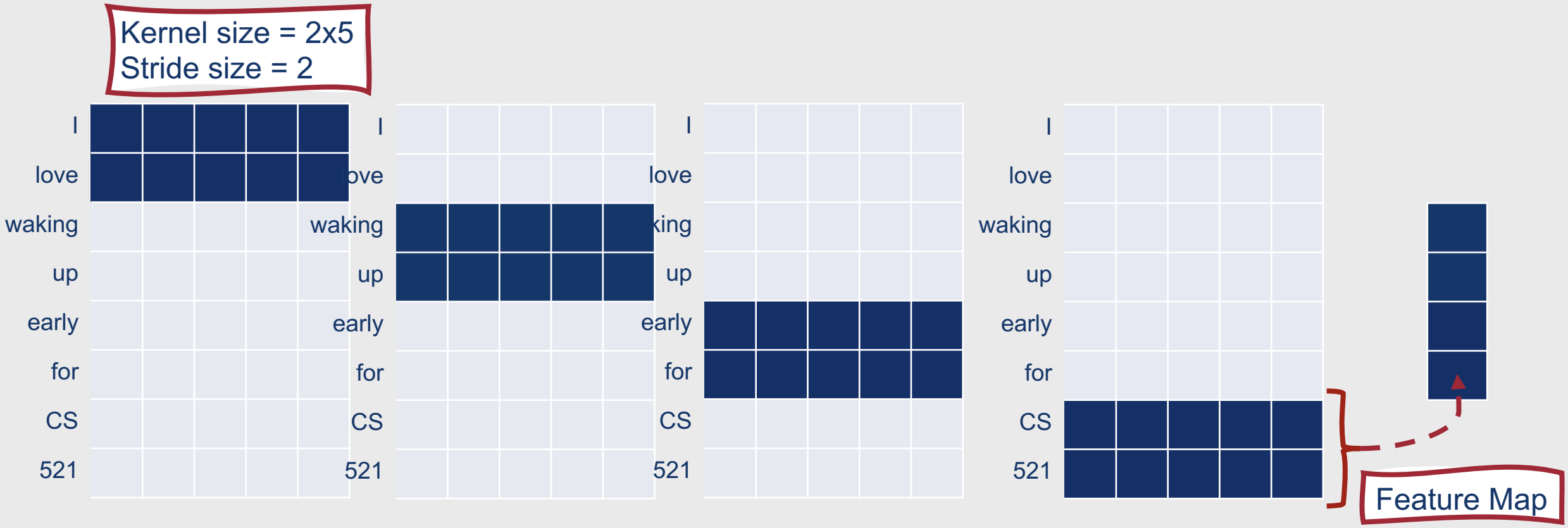
After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.

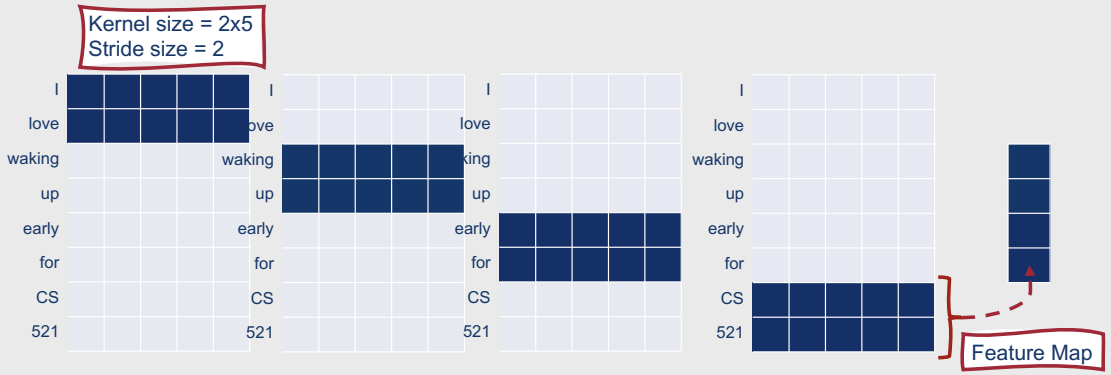


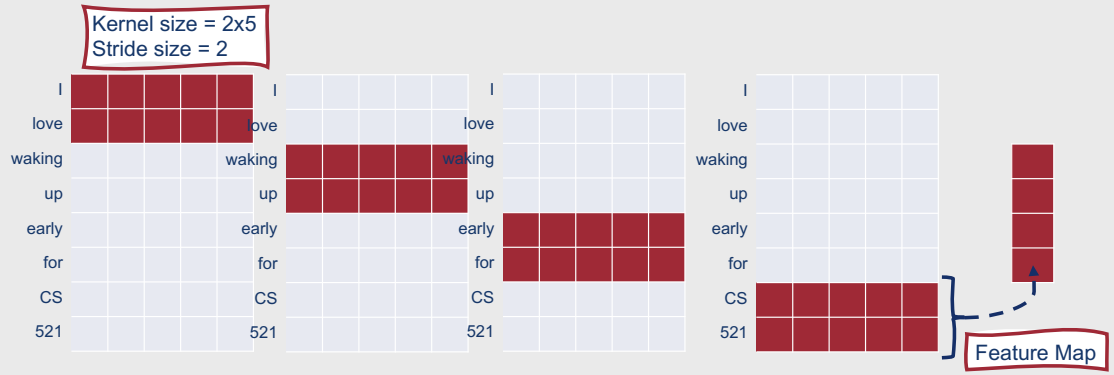
After applying a convolution with specific region (kernel) and stride sizes to an input matrix, we end up with a feature map.



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







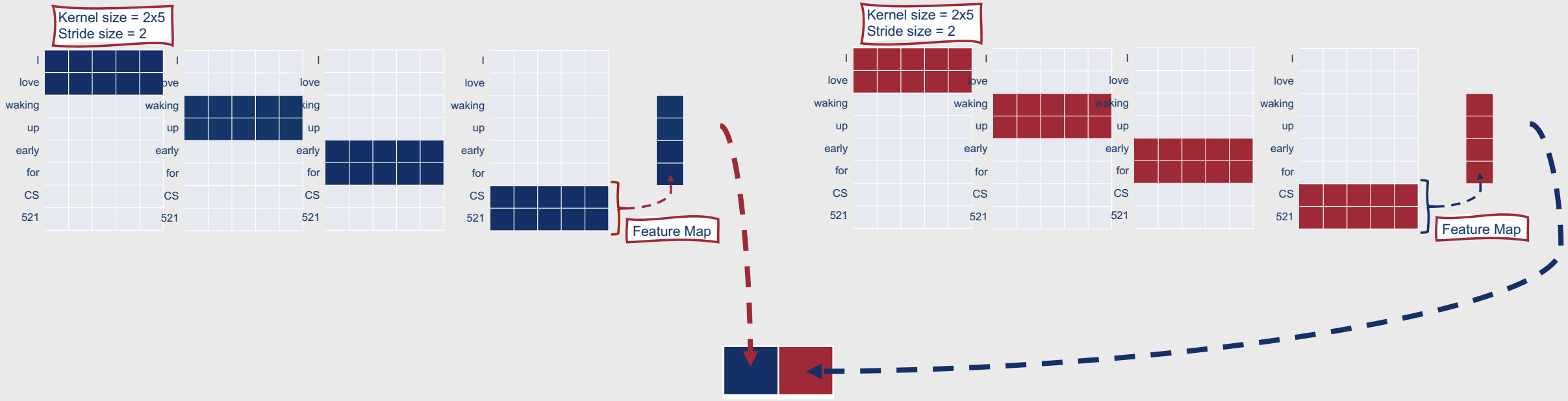
It’s common to extract multiple different feature maps from the same input.

After extracting feature maps from the input, convolutional neural networks ("CNNs" or "convnets") utilize pooling layers.

- **Pooling layers:** Layers that reduce the dimensionality of input feature maps by pooling all of the values in a given region

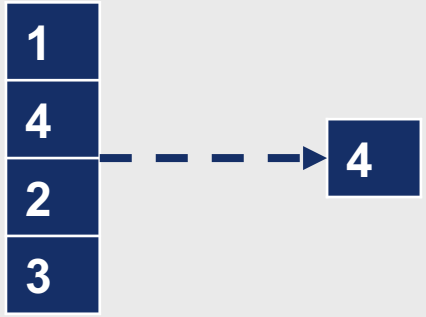
- Why use pooling layers?
  - Further increase **efficiency**
  - Improve the model's ability to be **invariant to small changes**

Pooling Layers



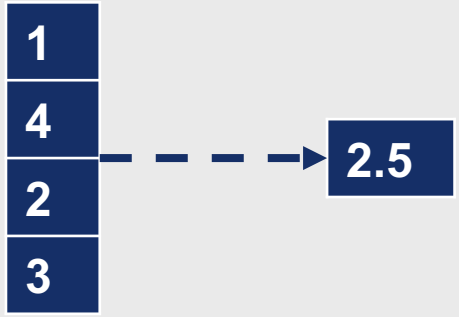
Common Techniques for Pooling

- **Max pooling**
  - Take the maximum of all values computed in a given window
- **Average pooling**
  - Take the average of all values computed in a given window

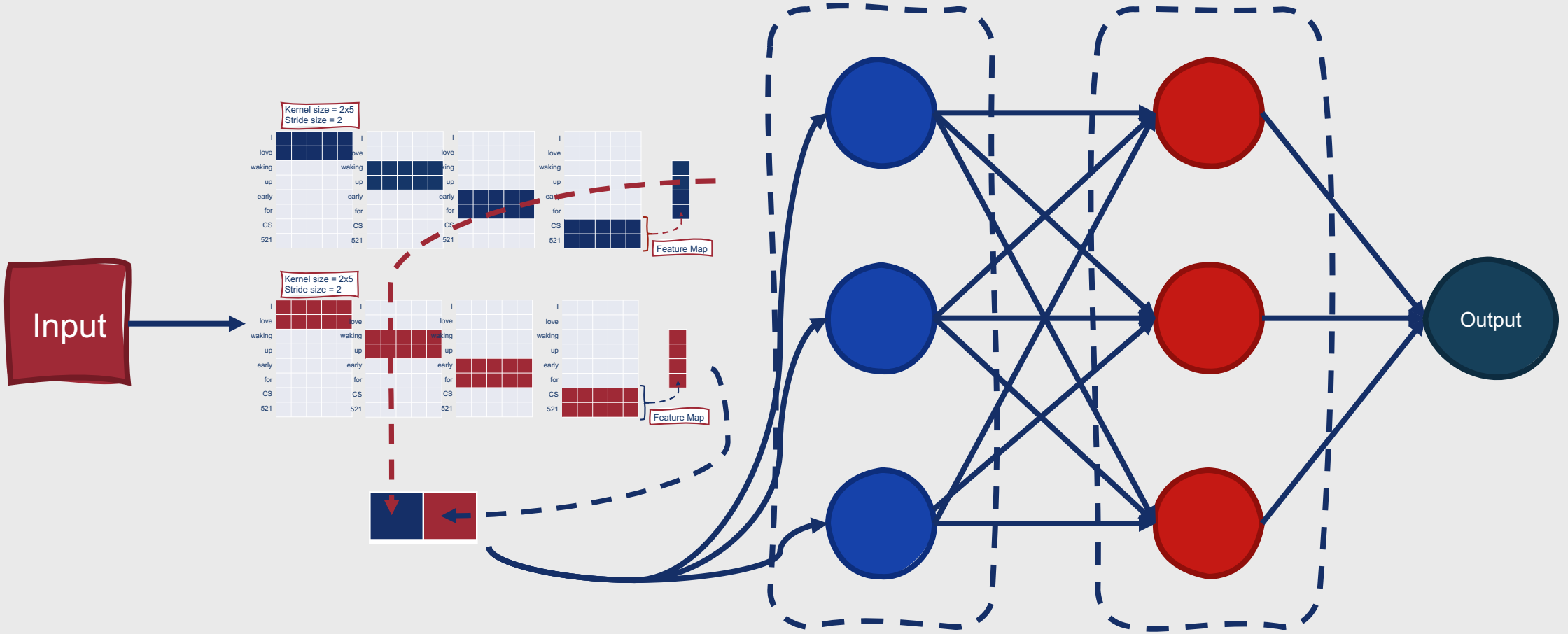


Common Techniques for Pooling

- **Max pooling**
  - Take the maximum of all values computed in a given window
- **Average pooling**
  - Take the average of all values computed in a given window



The output from pooling layers is typically then passed along as input to one or more feedforward layers.

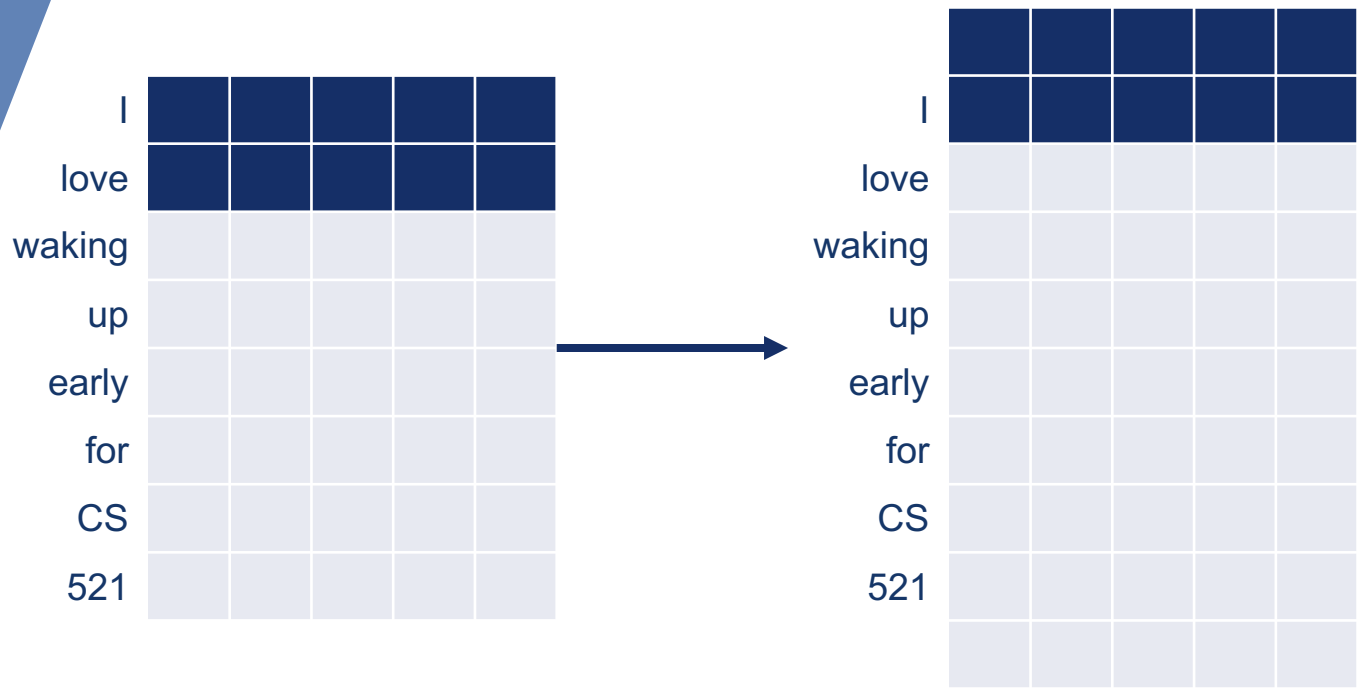


Convolutional neural network architectures can vary greatly!

- Additional hyperparameters:
  - Kernel size
  - Padding
  - Stride size
  - Number of channels
  - Pooling technique

Padding?

- Add empty vectors to the beginning and end of your text input
- Why do this?
  - Allows you to apply a filter to every element of the input matrix



Channels?

For images, generally corresponds to color channels

- Red, green, blue

For text, can mean:

- Different types of word embeddings
  - Word2Vec, GloVe, etc.
- Other feature types
  - POS tags, word length, etc.

The big question …why use CNNs at all?

- Traditionally for image classification!
- However, offer unique advantages for NLP tasks:
  - CNNs inherently extract meaningful local structures from input
  - In NLP → implicitly-learned, useful n-grams!

Summary: Backpropagation and Convolutional Neural Networks

- Weights are optimized in deep neural networks using **backpropagation**
- Backpropagation works by **propagating loss values backward** from the end of a neural network to the beginning, using the **chain rule** to eventually compute the gradient at the first layer
- This is done by framing networks as complex **computation graphs**
- **Convolutional neural networks** incorporate one or more **convolutional layers**
- After **feature maps** are produced using those convolutional layers, **pooling** techniques are used to reduce their dimensionality
- CNN architectures can vary greatly, and require additional hyperparameters:
  - **Kernel size**
  - **Padding**
  - **Stride size**
  - **# channels**
  - **Pooling technique**
- A key advantage of CNNs is that they allow a model to **automatically learn useful n-grams** from the training data