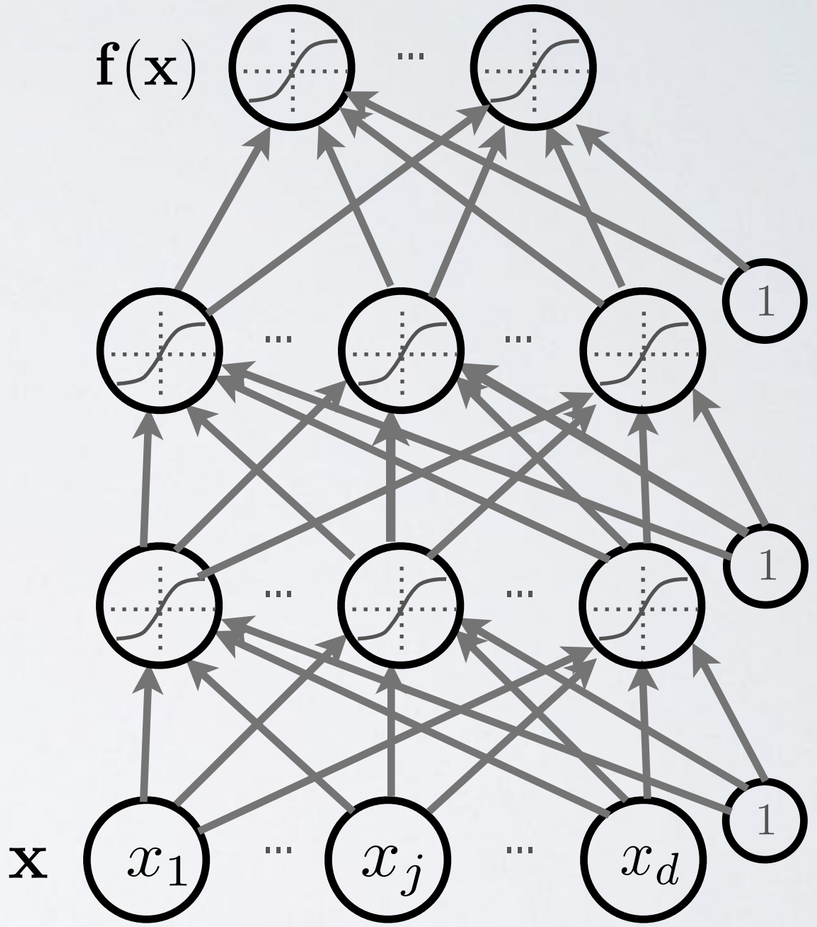
Neural Networks

Hugo Larochelle (@hugo_larochelle)
Google Brain

NEURAL NETWORKS

- What we'll cover
  - computer vision architectures
    - convolutional networks
    - data augmentation
    - residual networks
  - natural language processing architectures
    - word embeddings
    - recurrent neural networks
    - long short-term memory networks (LSTMs)



Neural Networks

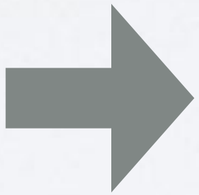
Computer vision

COMPUTER VISION

Topics: computer vision, object recognition

- Computer vision is the design of computers that can process visual data and accomplish some given task
  - we will focus on object recognition: given some input image, identify which object it contains

Caltech 101 dataset



"sun flower"

COMPUTER VISION

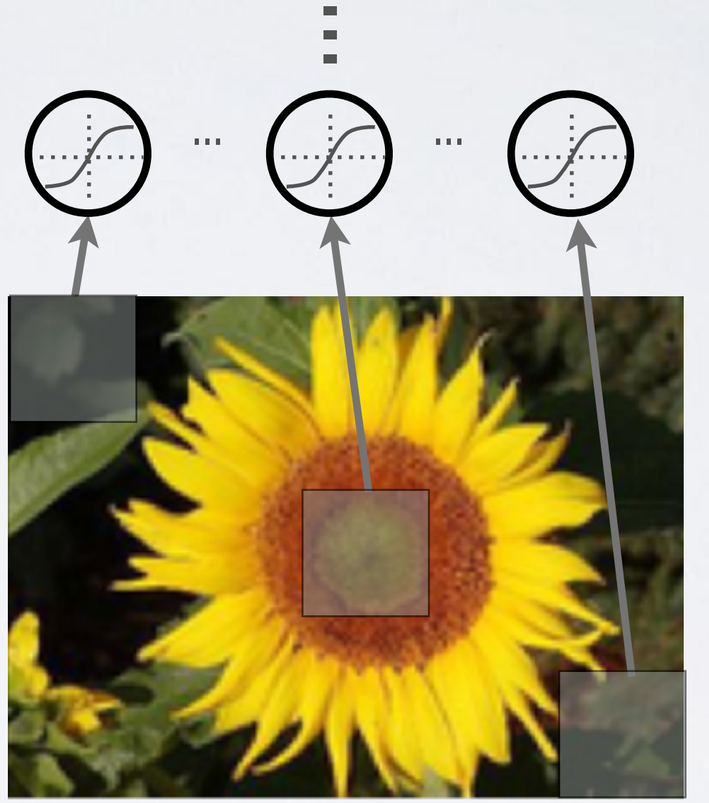
Topics: computer vision

- We can design neural networks that are specifically adapted for such problems
  - must deal with very high-dimensional inputs
    - 150 x 150 pixels = 22500 inputs, or 3 x 22500 if RGB pixels
  - can exploit the 2D topology of pixels (or 3D for video data)
  - can build in invariance to certain variations we can expect
    - translations, illumination, etc.
- Convolutional networks leverage these ideas
  - local connectivity
  - parameter sharing
  - pooling / subsampling hidden units

COMPUTER VISION

Topics: local connectivity

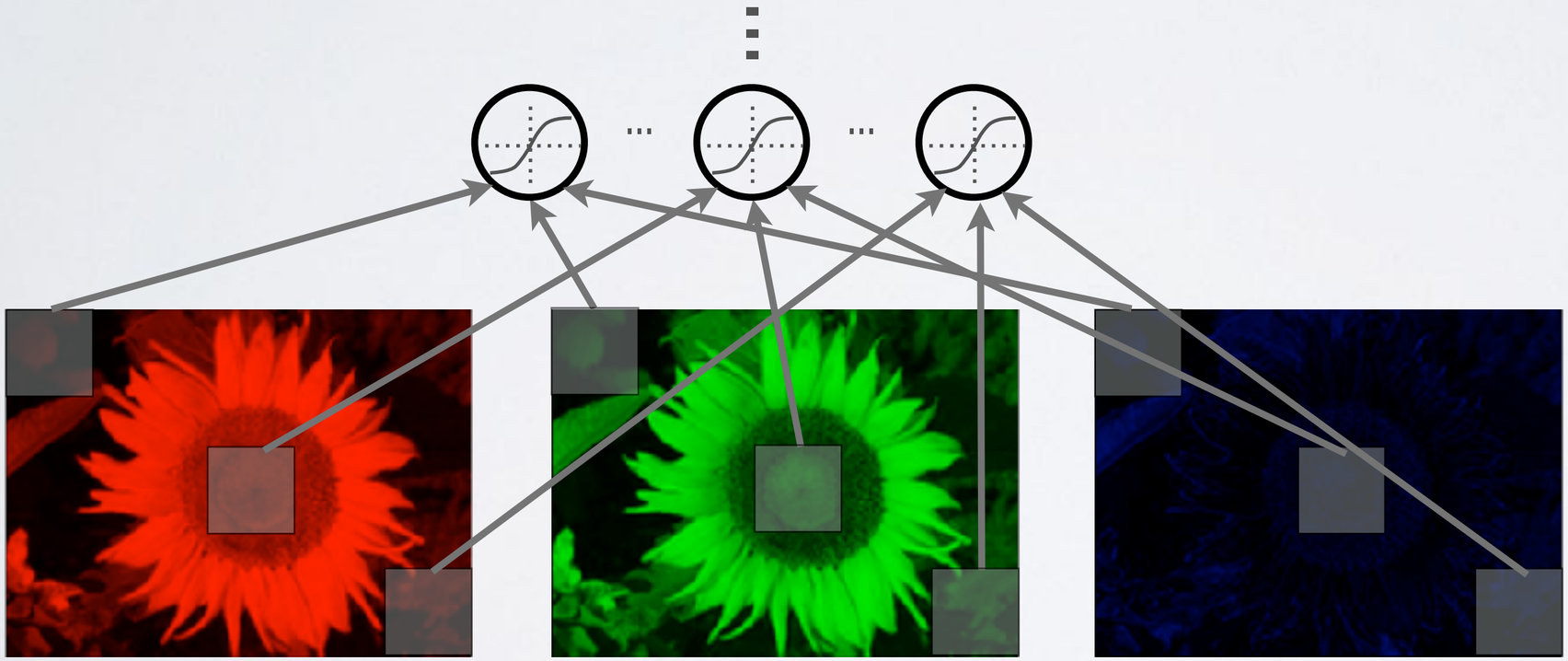
- First idea: use a local connectivity of hidden units
  - each hidden unit is connected only to a subregion (patch) of the input image
  - it is connected to all channels
    - 1 if greyscale image
    - 3 (R, G, B) for color image
- Solves the following problems:
  - fully connected hidden layer would have an unmanageable number of parameters
  - computing the linear activations of the hidden units would be very expensive



COMPUTER VISION

Topics: local connectivity

- Units are connected to all channels:
  - 1 channel if grayscale image, 3 channels (R, G, B) if color image



COMPUTER VISION

Topics: parameter sharing

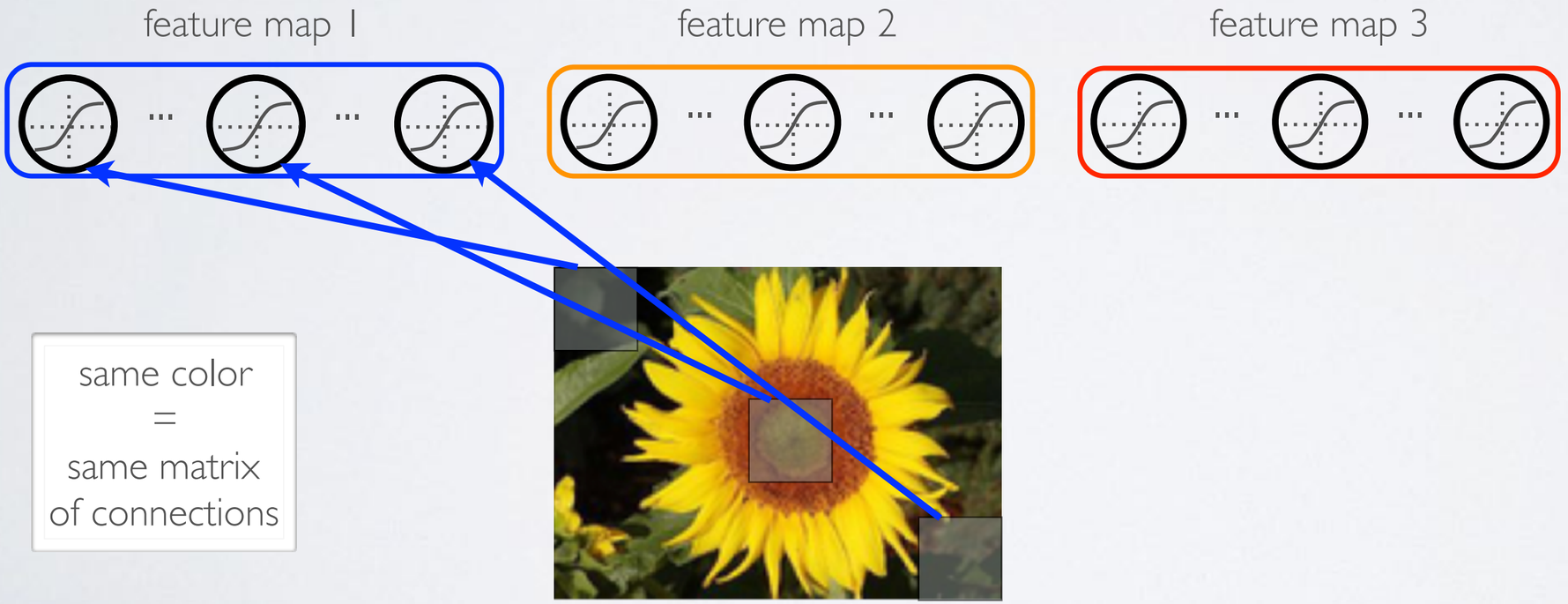
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image

same color
=
same matrix
of connections

COMPUTER VISION

Topics: parameter sharing

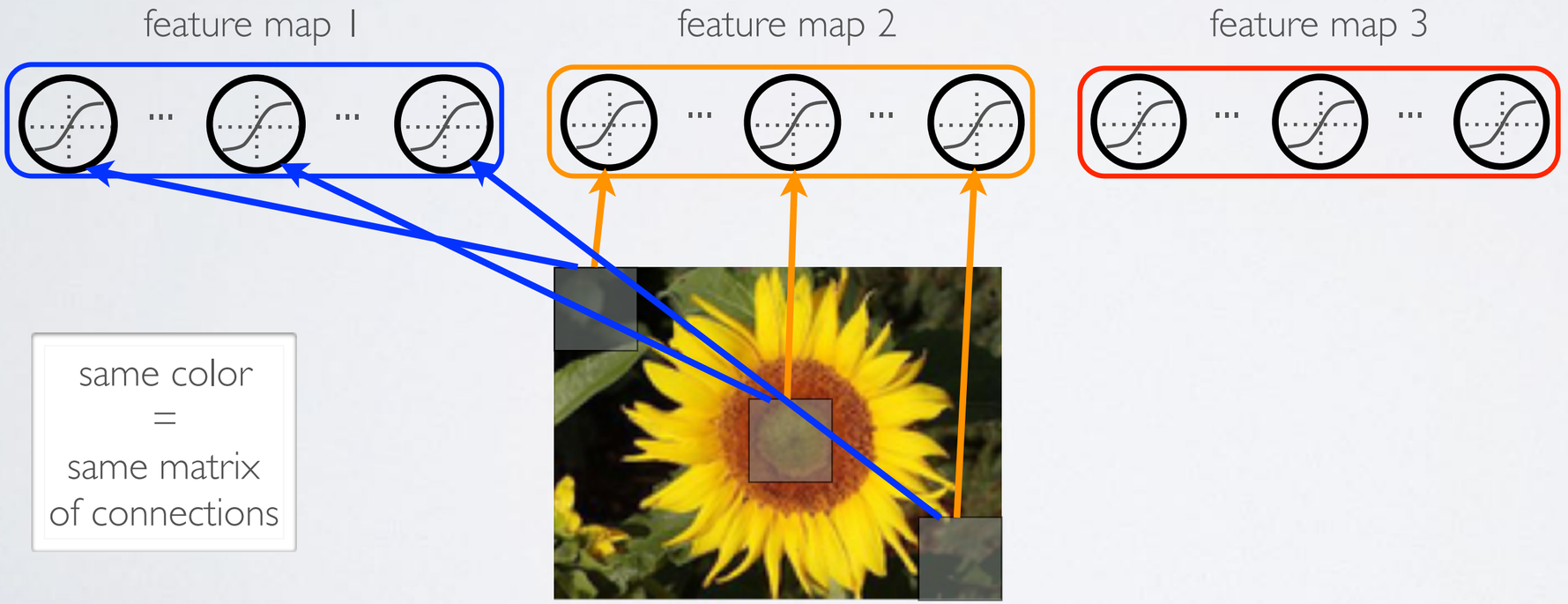
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



COMPUTER VISION

Topics: parameter sharing

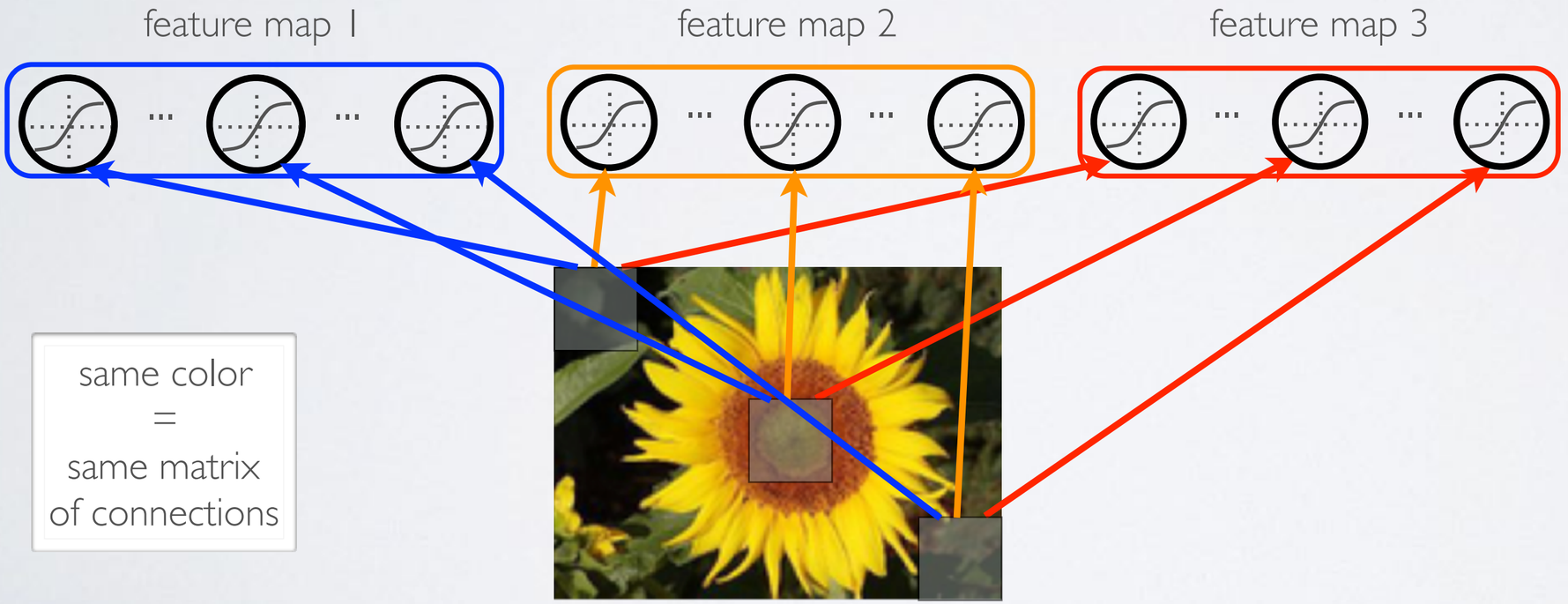
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



COMPUTER VISION

Topics: parameter sharing

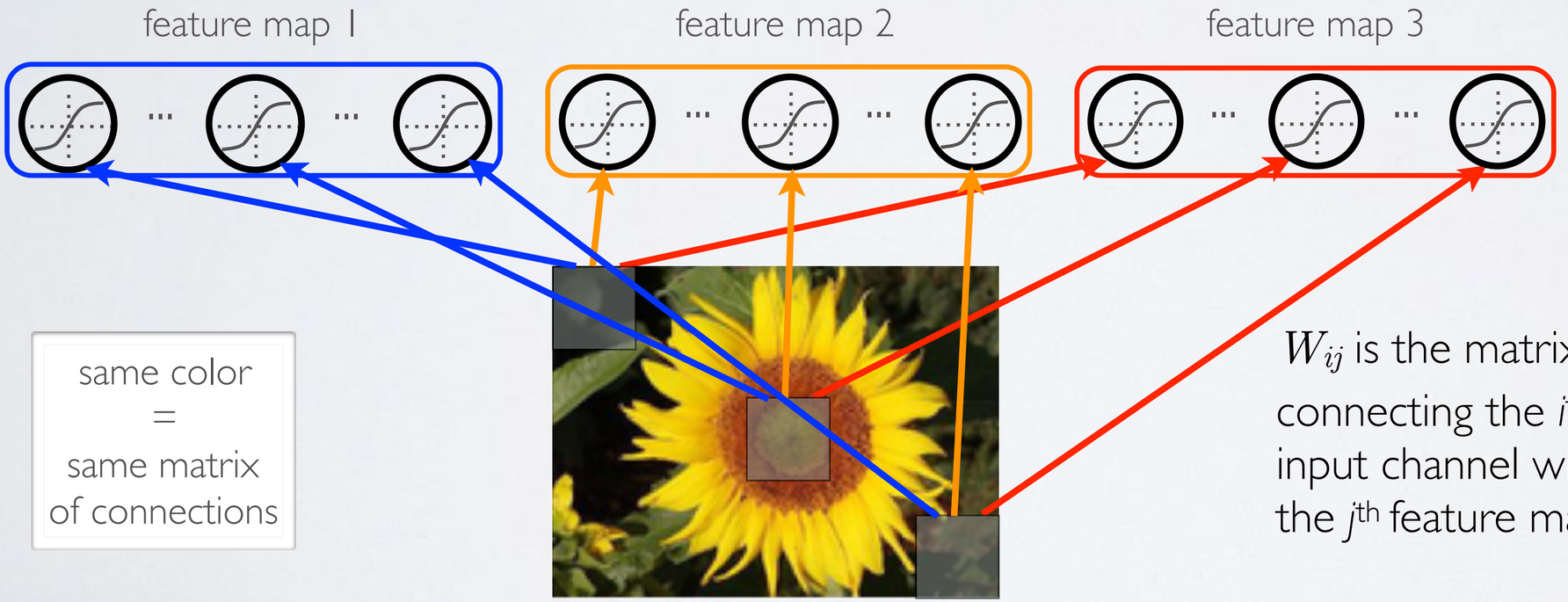
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



COMPUTER VISION

Topics: parameter sharing

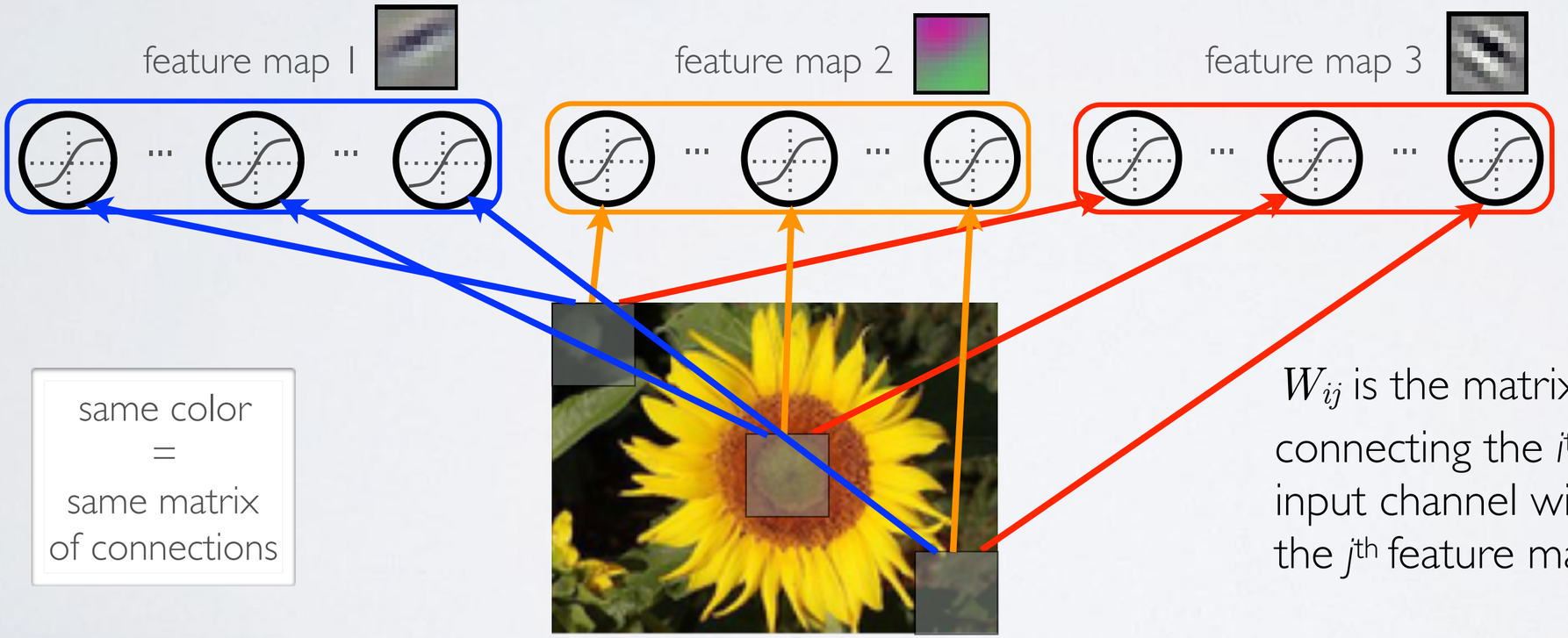
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



COMPUTER VISION

Topics: parameter sharing

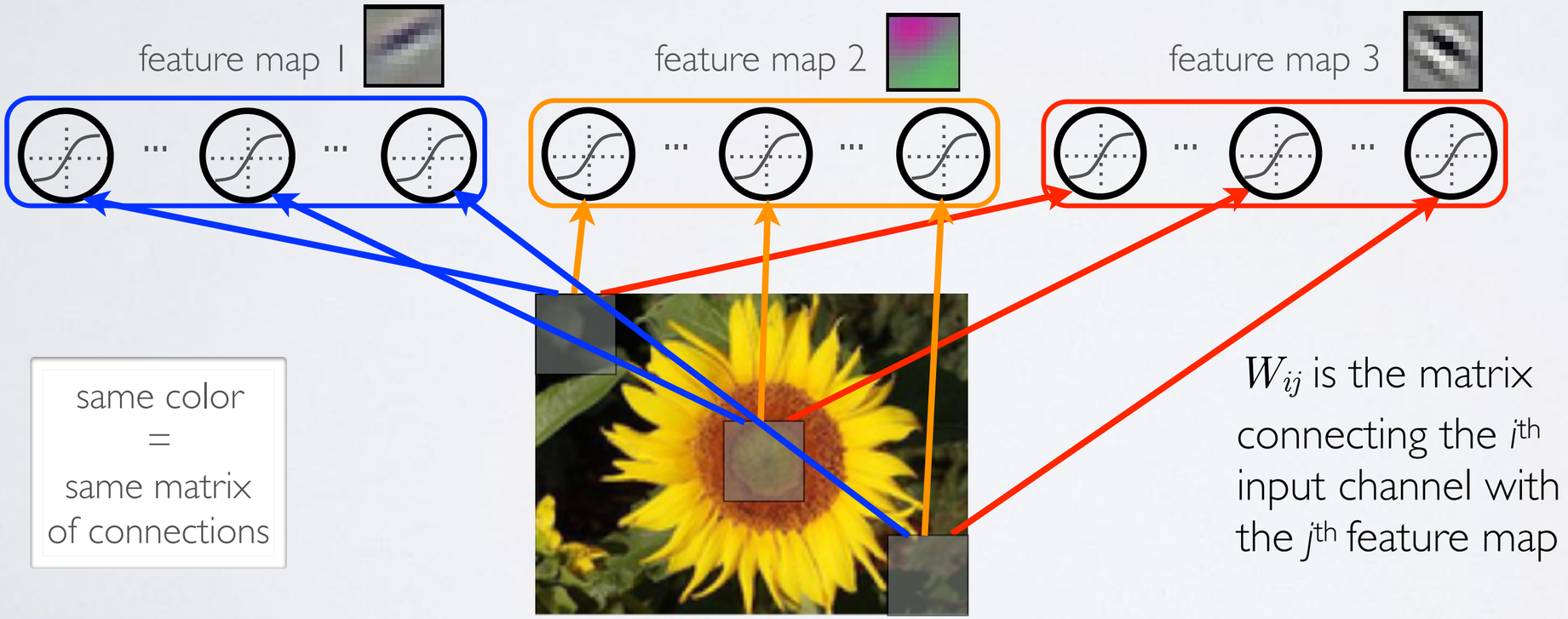
- Second idea: share matrix of parameters across certain units
  - units organized into the same "feature map" share parameters
  - hidden units within a feature map cover different positions in the image



COMPUTER VISION

Topics: parameter sharing

- Solves the following problems:
  - reduces even more the number of parameters
  - will extract the same features at every position (features are "equivariant")

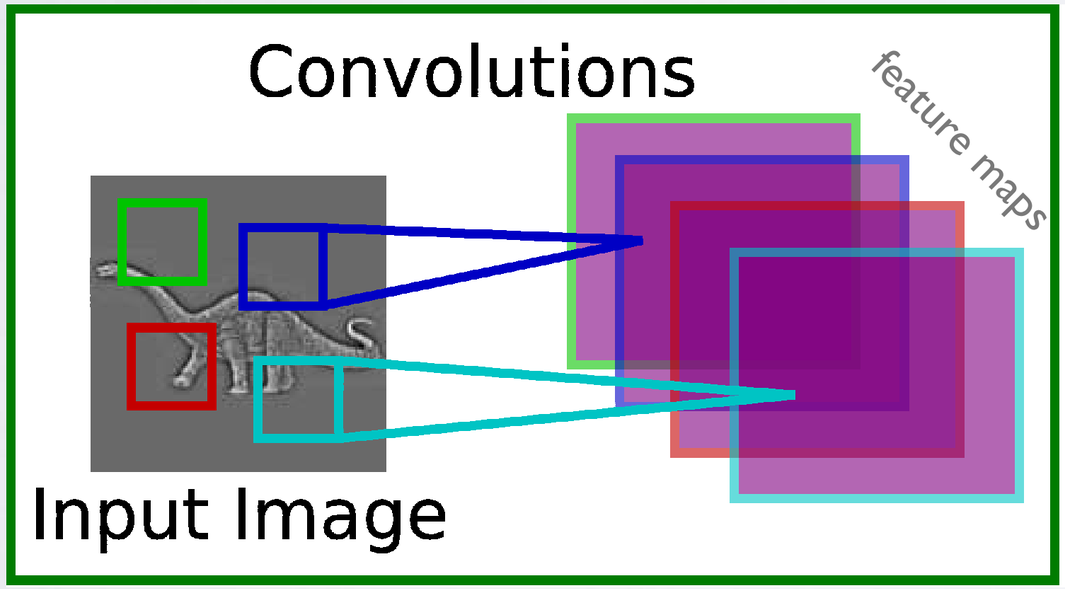


COMPUTER VISION

Topics: parameter sharing

Jarret et al. 2009

- Each feature map forms a 2D grid of features
  - can be computed with a discrete convolution ($*$) of a kernel matrix k_{ij} which is the hidden weights matrix W_{ij} with its rows and columns flipped



- x_i is the i[th] channel of input
- k_{ij} is the convolution kernel
- g_j is a learned scaling factor
- y_j is the hidden layer

(could have added a bias)

$$y_j = g_j \tanh(\sum_i k_{ij} * x_i)$$

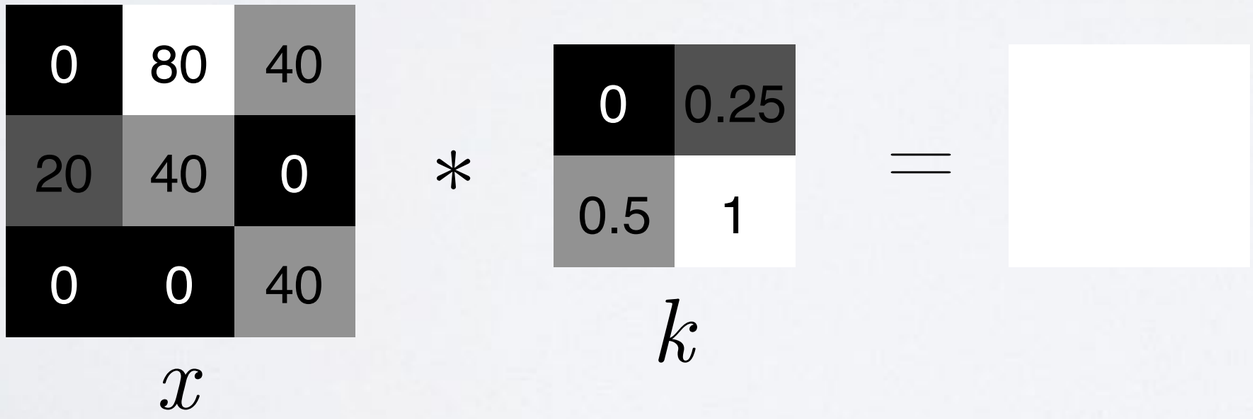
COMPUTER VISION

Topics: discrete convolution

- The convolution of an image x with a kernel k is computed as follows:

$$(x * k)_{ij} = \sum x_{i+p,j+q} \, k_{r-p,r-q}$$

- Example:



COMPUTER VISION

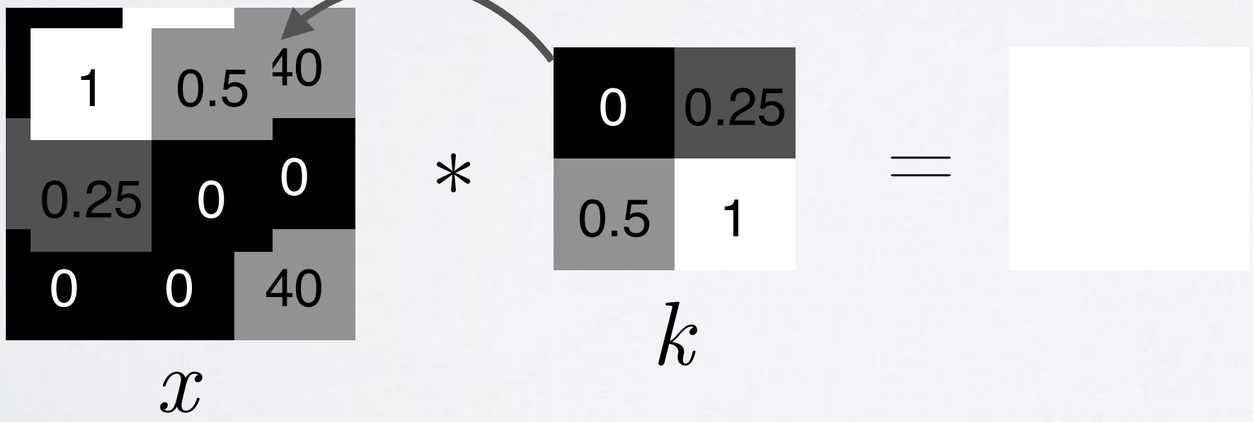
Topics: discrete convolution

- The convolution of an image x with a kernel k is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:

$\tilde{k} = k$ with rows and columns flipped



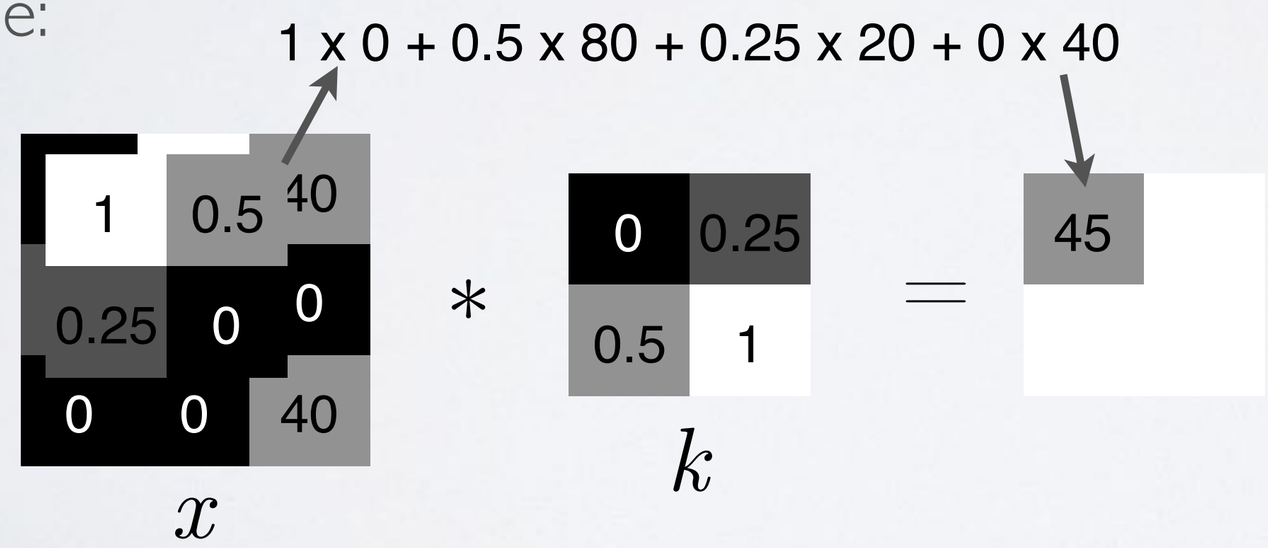
COMPUTER VISION

Topics: discrete convolution

- The convolution of an image x with a kernel k is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



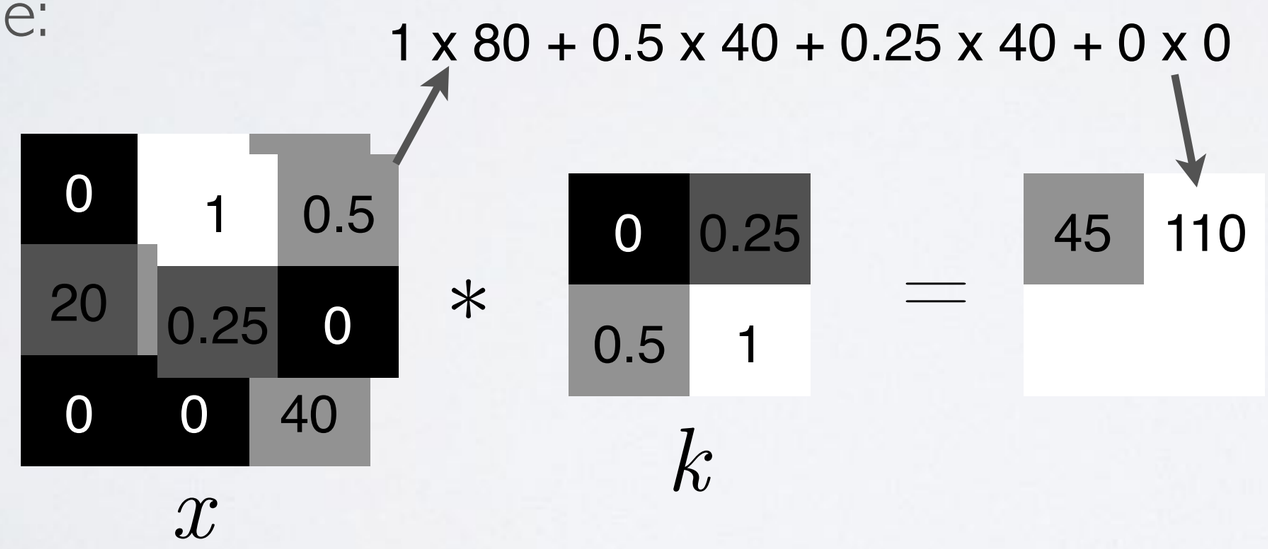
COMPUTER VISION

Topics: discrete convolution

- The convolution of an image x with a kernel k is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



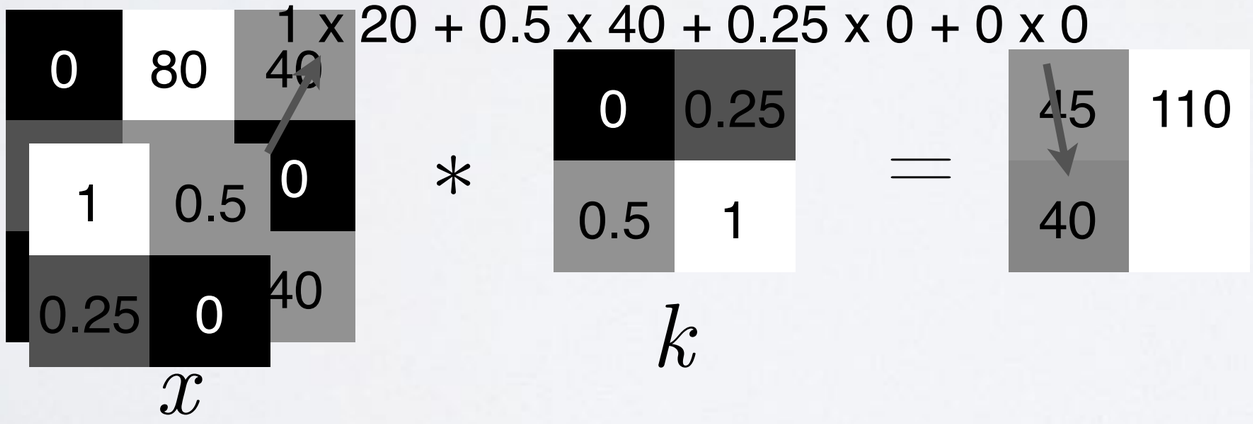
COMPUTER VISION

Topics: discrete convolution

- The convolution of an image x with a kernel k is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



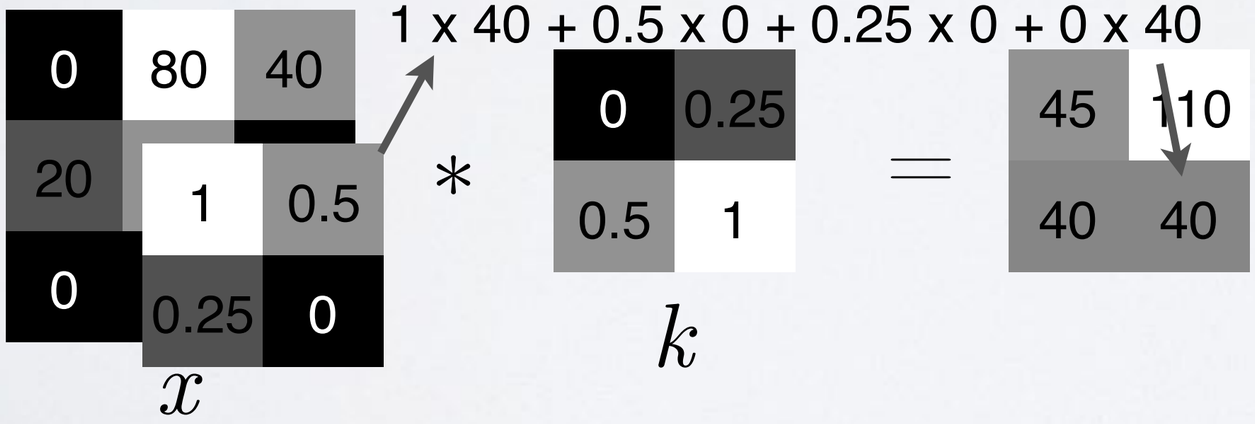
COMPUTER VISION

Topics: discrete convolution

- The convolution of an image x with a kernel k is computed as follows:

$$(x * k)_{ij} = \sum_{pq} x_{i+p,j+q} \; k_{r-p,r-q}$$

- Example:



COMPUTER VISION

Topics: discrete convolution

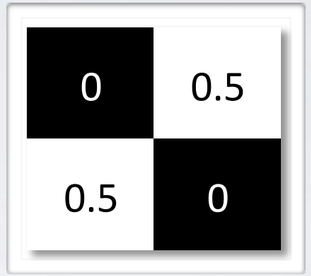
- Pre-activations from channel x_i into feature map y_j can be computed by:
  - getting the convolution kernel where $k_{ij} = \widetilde{W}_{ij}$ from the connection matrix W_{ij}
  - applying the convolution $x_i * k_{ij}$

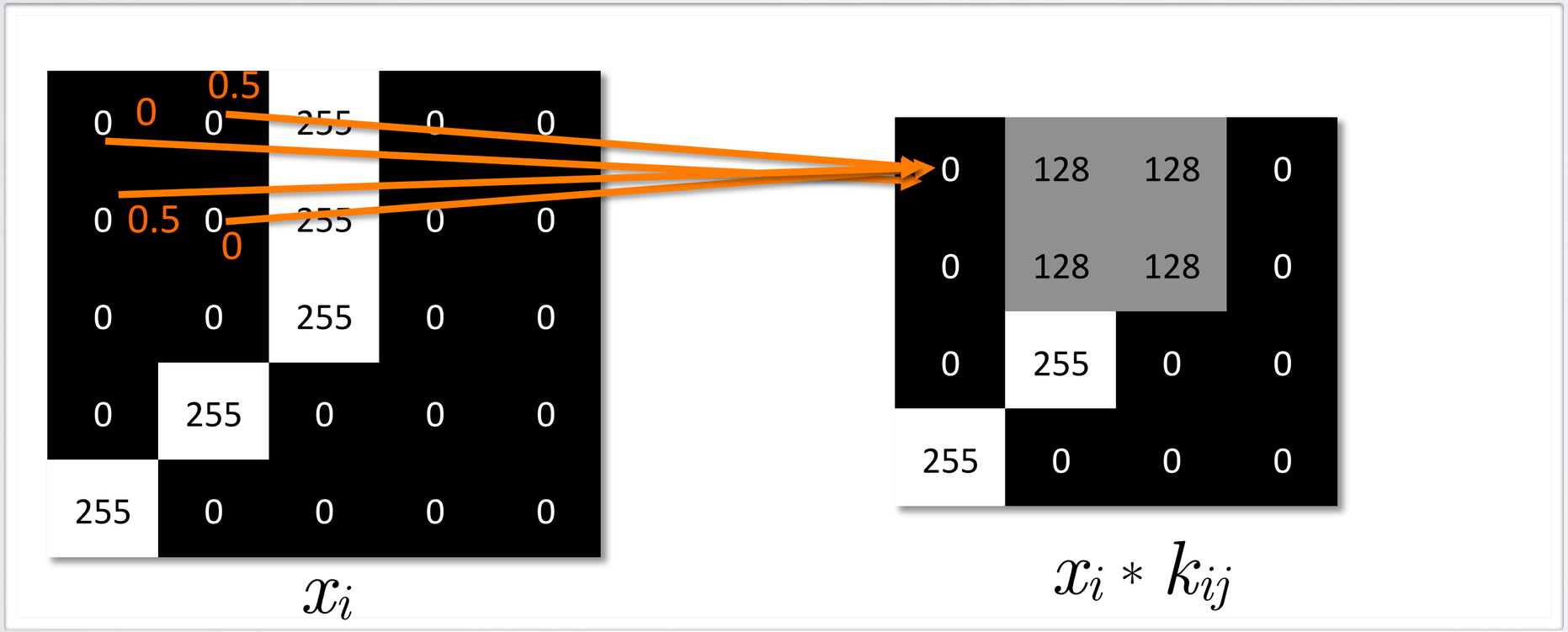
- This is equivalent to computing the discrete correlation of x_i with W_{ij}

COMPUTER VISION

Topics: discrete convolution

- Simple illustration: $x_i * k_{ij}$ where $W_{ij} = \tilde{W}_{ij}$

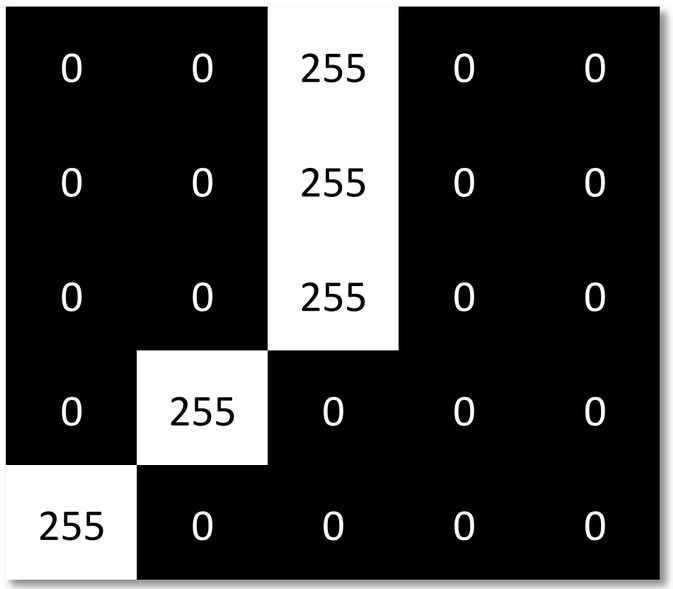




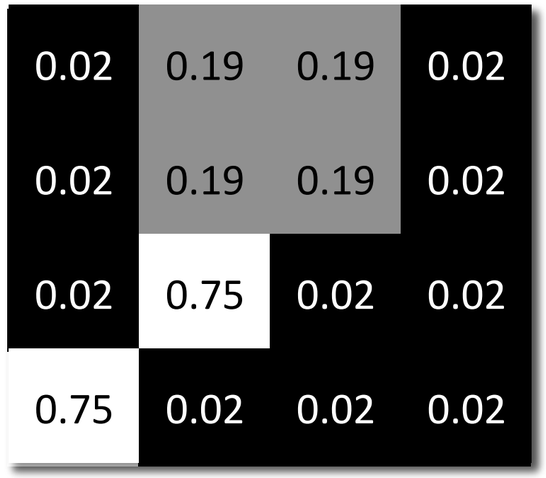
COMPUTER VISION

Topics: discrete convolution

- With a non-linearity, we get a detector of a feature at any position in the image



x_i

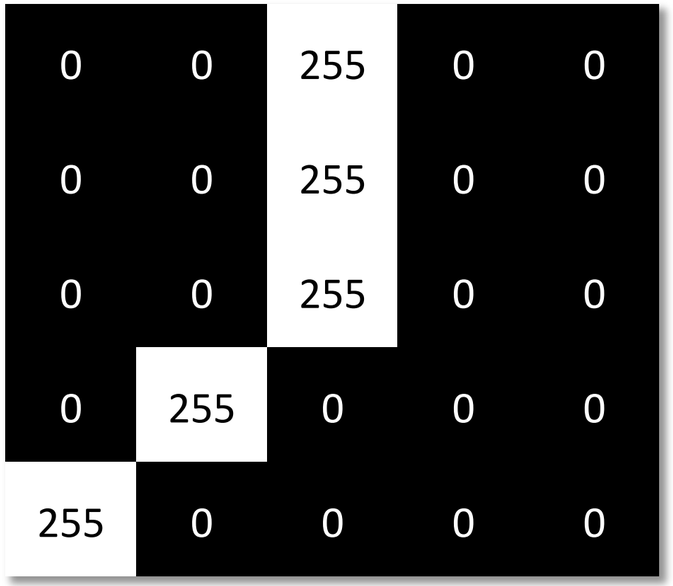


$$\mathrm{sigm}(0.02\ x_i * k_{ij} \text{-}4)$$

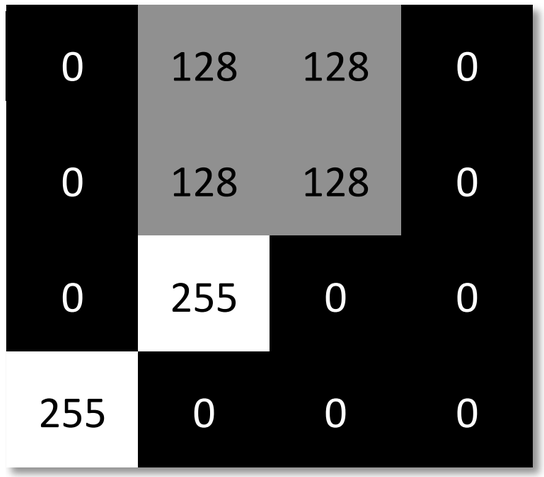
COMPUTER VISION

Topics: discrete convolution

- Can use "zero padding" to allow going over the borders ($\underline{*}$)



x_i

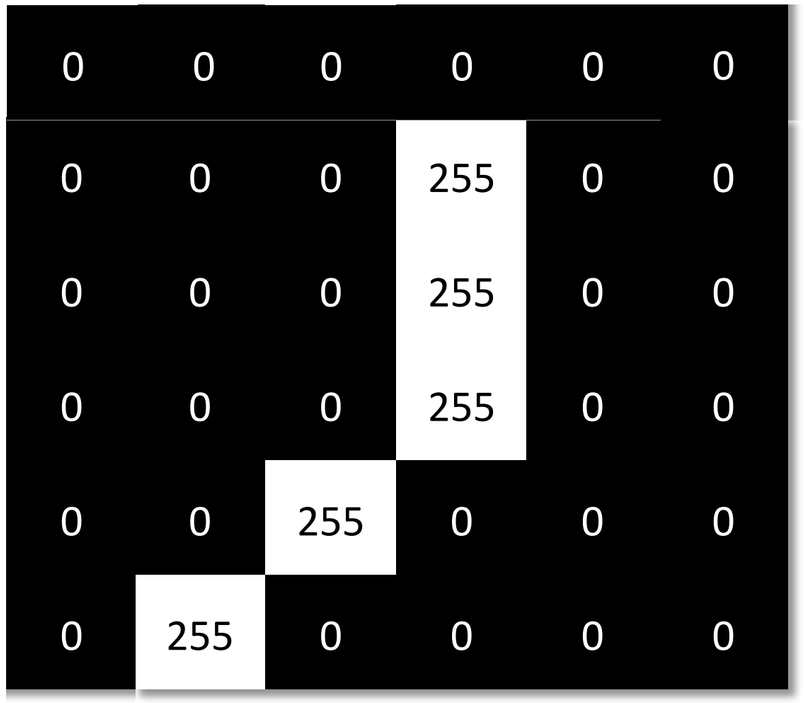


$x_i \underline{*} k_{ij}$

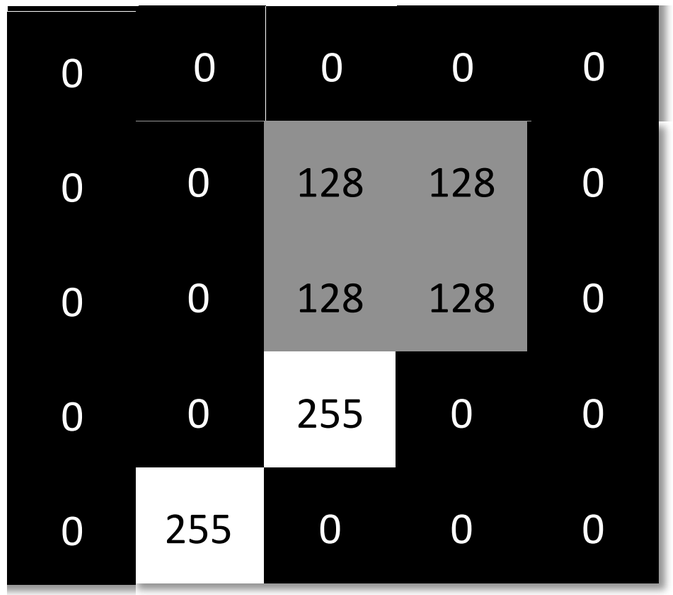
COMPUTER VISION

Topics: discrete convolution

- Can use "zero padding" to allow going over the borders ($\underline{*}$)



x_i

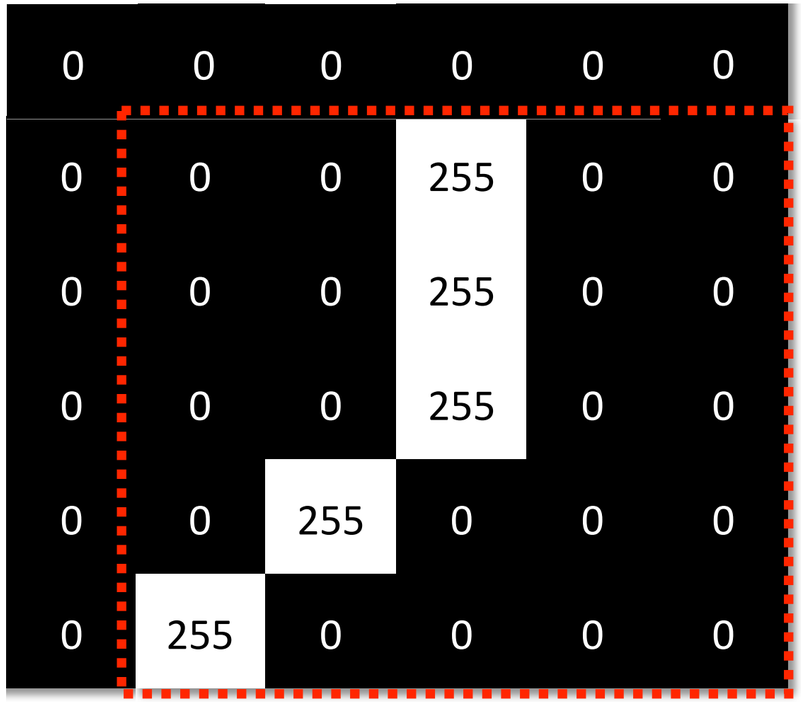


$x_i \underline{*} k_{ij}$

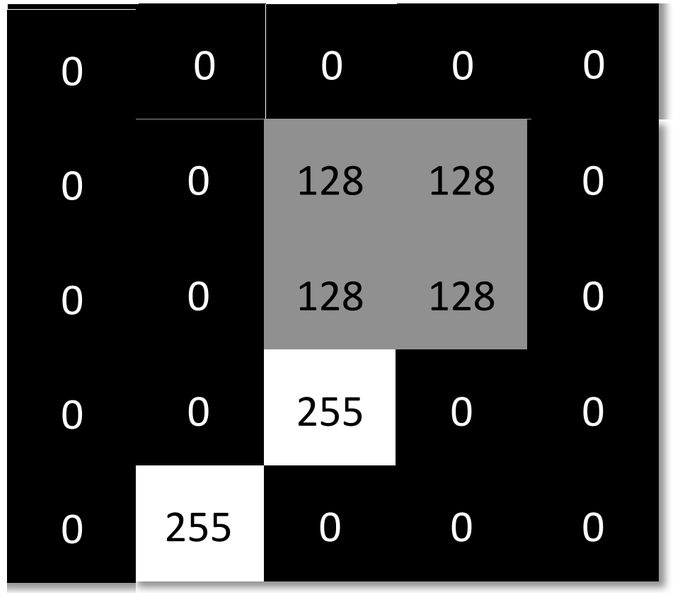
COMPUTER VISION

Topics: discrete convolution

- Can use "zero padding" to allow going over the borders ($\underline{*}$)



x_i

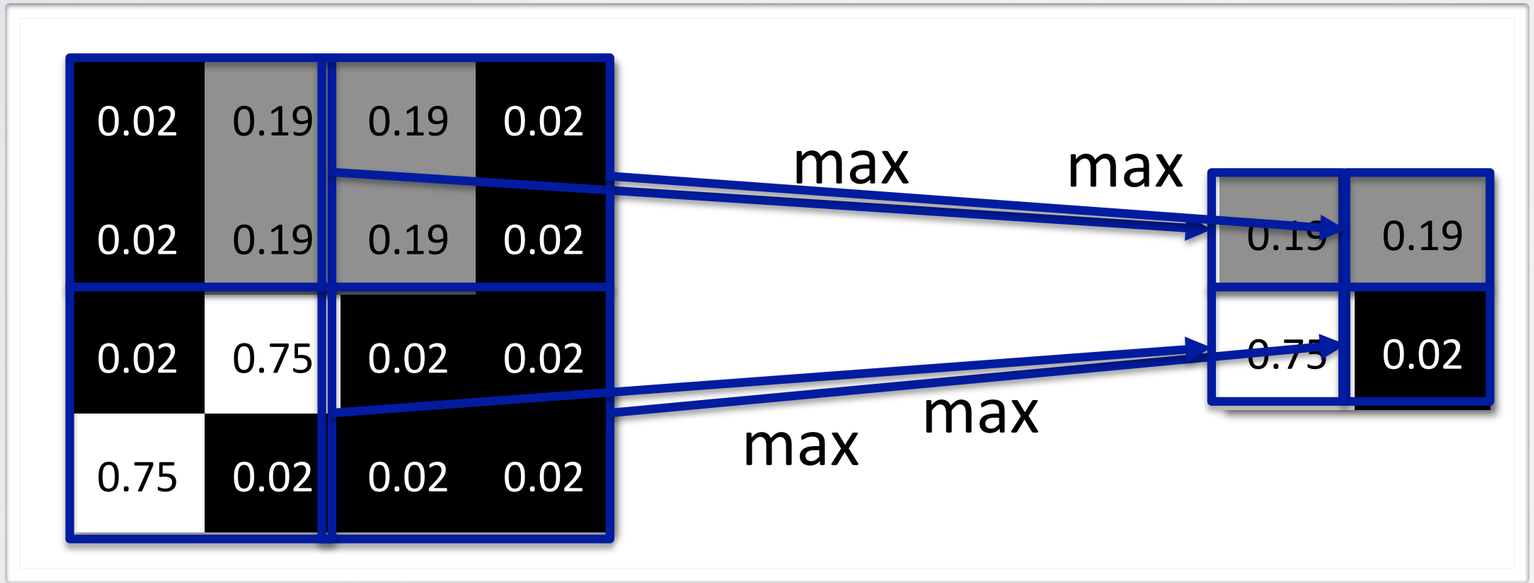


$x_i \underline{*} k_{ij}$

COMPUTER VISION

Topics: pooling, stride

- Illustration of pooling (2x2) + subsampling using stride (2)

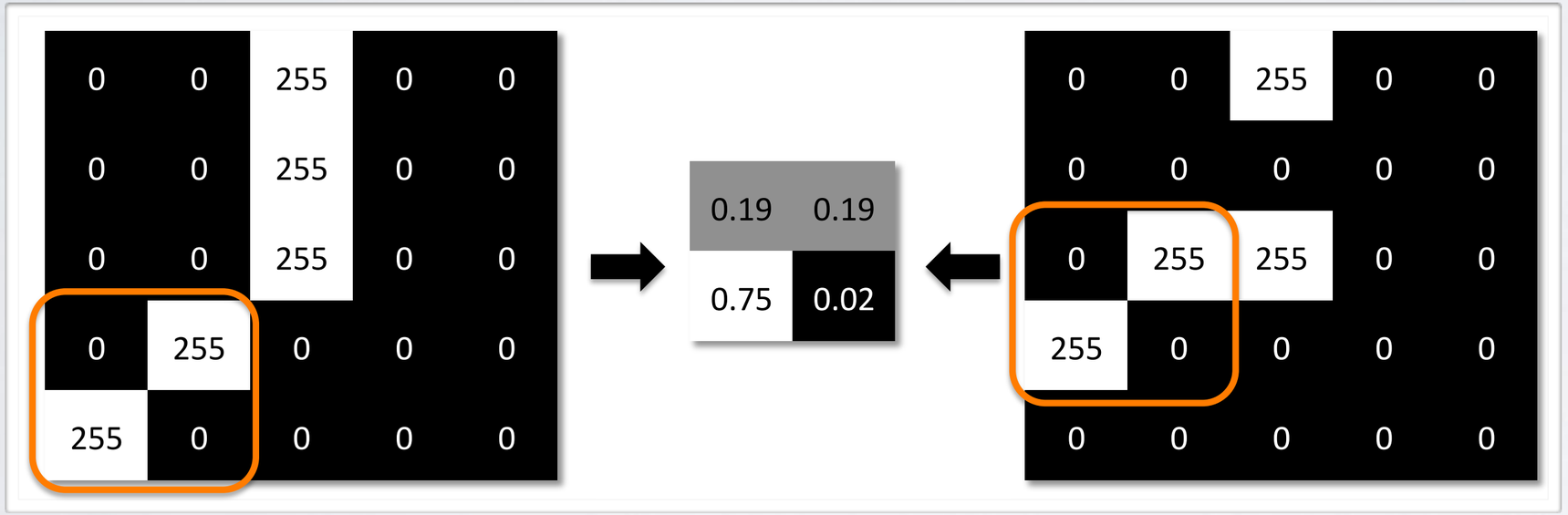


- Solves the following problems:
  - introduces invariance to local translations
  - reduces the number of hidden units in hidden layer

COMPUTER VISION

Topics: pooling and subsampling

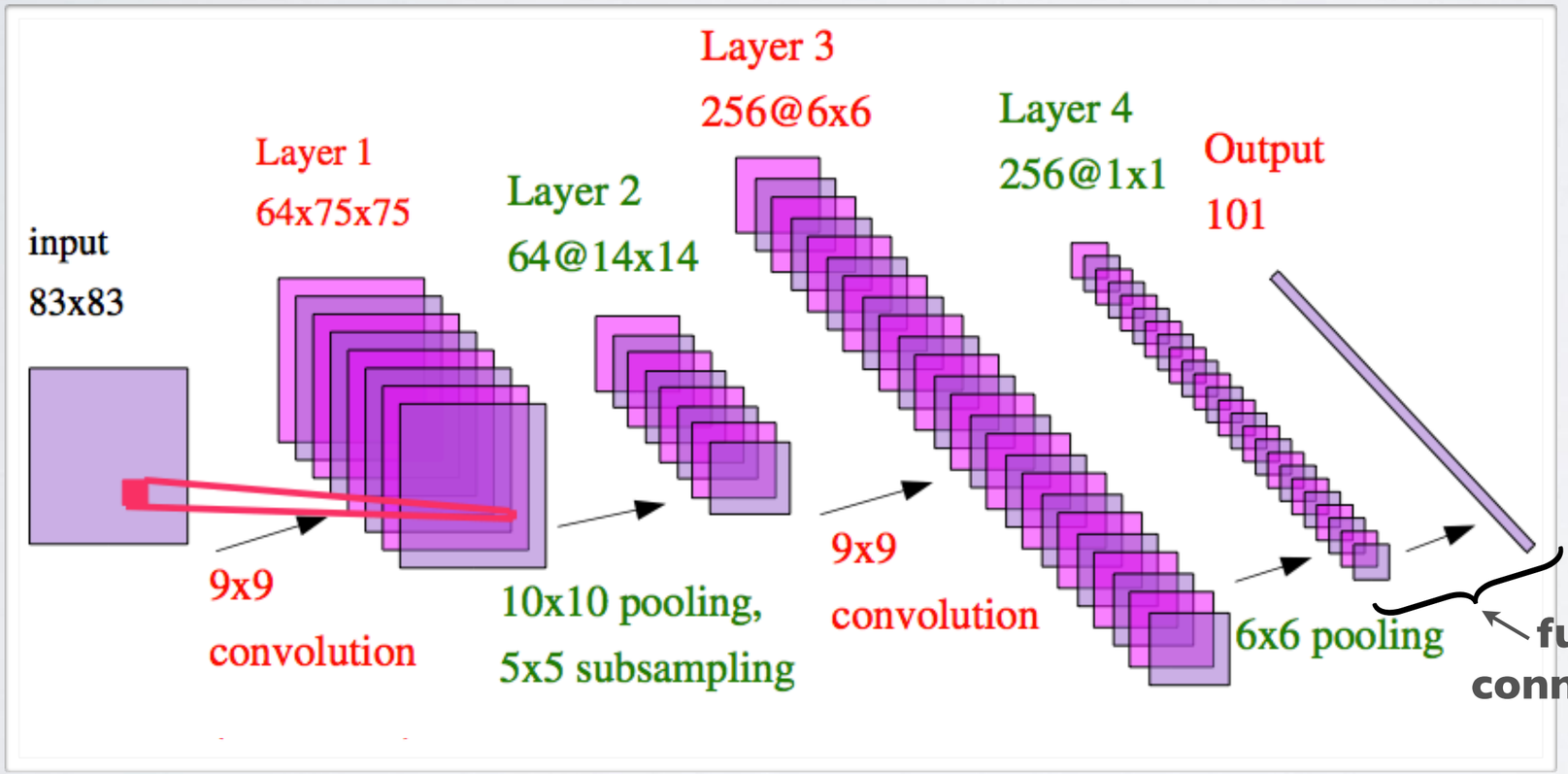
- Illustration of local translation invariance
  - both images given the same feature map after pooling/subsampling



CONVOLUTIONAL NETWORK

Topics: convolutional network

- Convolutional neural network alternates between the convolutional and pooling layers



(from Yann Lecun)

CONVOLUTIONAL NETWORK

Topics: convolutional network

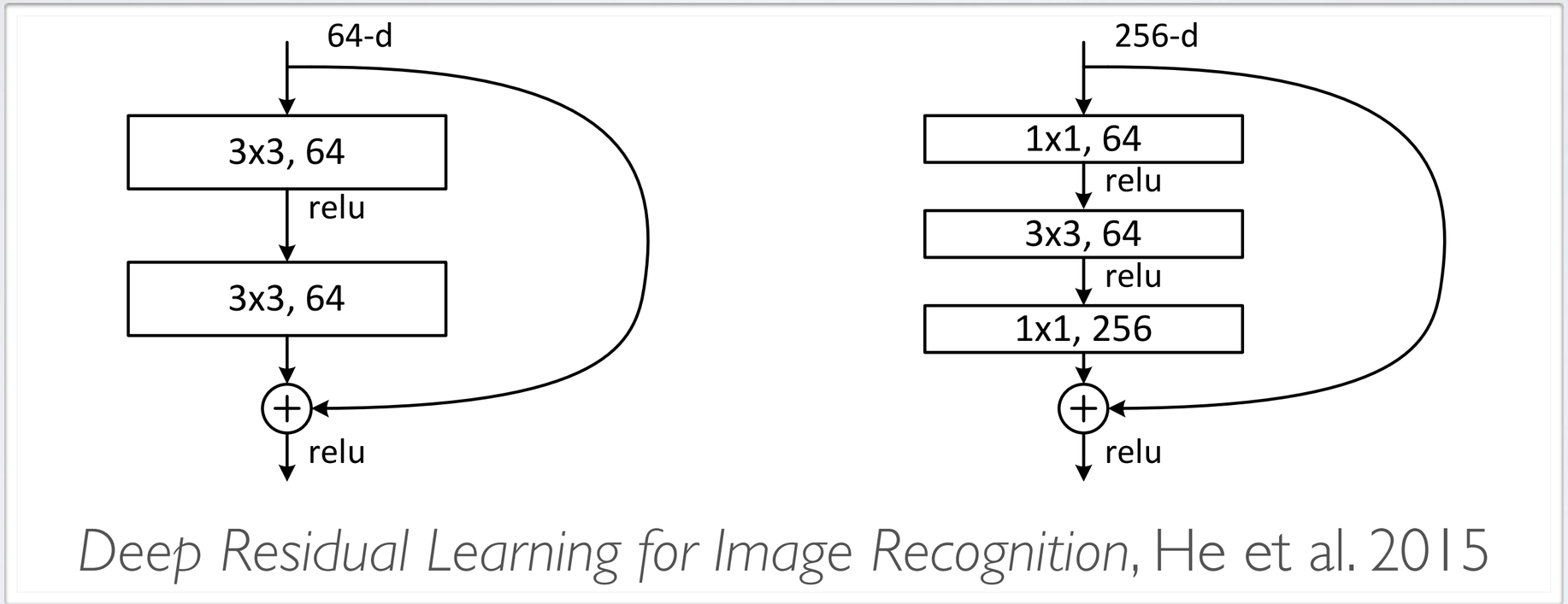
- Output layer is a regular, fully connected layer with softmax non-linearity
  - output provides an estimate of the conditional probability of each class

- The network is trained by stochastic gradient descent
  - backpropagation is used similarly as in a fully connected network
  - we have seen how to pass gradients through element-wise activation function
  - we also need to pass gradients through the convolution operation and the pooling operation

CONVOLUTIONAL NETWORK

Topics: residual networks, bottleneck feature maps, batch normalization

- Very deep models are often used, with residual connections and bottlenecks maps
  - reduces potential problems with vanishing gradients



Deep Residual Learning for Image Recognition, He et al. 2015

- Batch normalization is adapted to also normalize across spatial locations

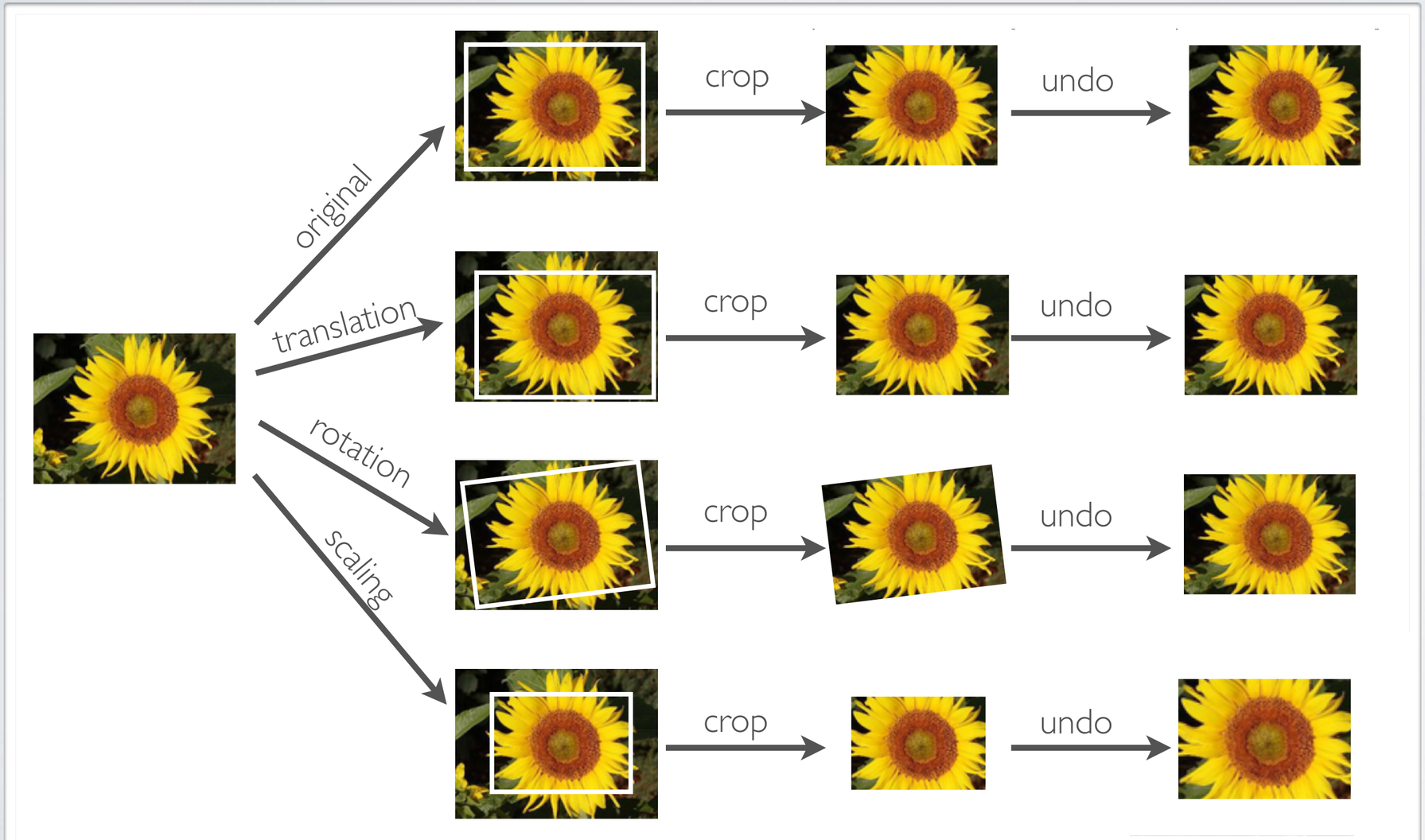
INVARIANCE BY DATA AUGMENTATION

Topics: generating additional examples

- Invariances built-in in convolutional network:
  - small translations: due to convolution and max pooling
  - small illumination changes: due to local contrast normalization
- It is not invariant to other important variations such as rotations and scale changes
- However, it's easy to artificially generate data with such transformations
  - could use such data as additional training data
  - neural network will learn to be invariant to such transformations

INVARIANCE BY DATA AUGMENTATION

Topics: generating additional examples



Neural Networks

Natural language processing

NEURAL NETWORKS FOR NLP

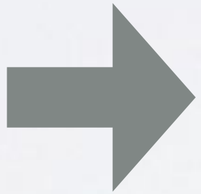
- What we'll cover
  - how to feed text data to neural networks
    - preprocessing
    - word representations (embeddings) with lookup table
    - neural network language modeling
  - how to classify text data with neural networks
    - average word embedding
    - recurrent neural networks (RNNs)
    - long short-term memory (LSTM) networks

NATURAL LANGUAGE PROCESSING

Topics: tokenization

- Typical preprocessing steps of text data
  - tokenize text (from a long string to a list of token strings)

"He's spending 7 days in San Francisco."



"He"
"'s"
"spending"
"7"
"days"
"in"
"San Francisco"
"."

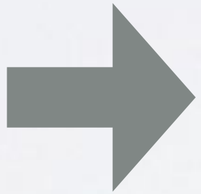
  - for many datasets, this has already been done for you
  - splitting into tokens based on spaces and separating punctuation is good enough in English or French

NATURAL LANGUAGE PROCESSING

Topics: lemmatization

- Typical preprocessing steps of text data
  - lemmatize tokens (put into standard form)

"He"
"'s"
"spending"
"7"
"days"
"in"
"San Francisco"
"."



"he"
"be"
"spend"
"NUMBER"
"day"
"in"
"San Francisco"
"."

  - the specific lemmatization will depend on the problem we want to solve
    - we can remove variations of words that are not relevant to the task at hand

NATURAL LANGUAGE PROCESSING

Topics: vocabulary

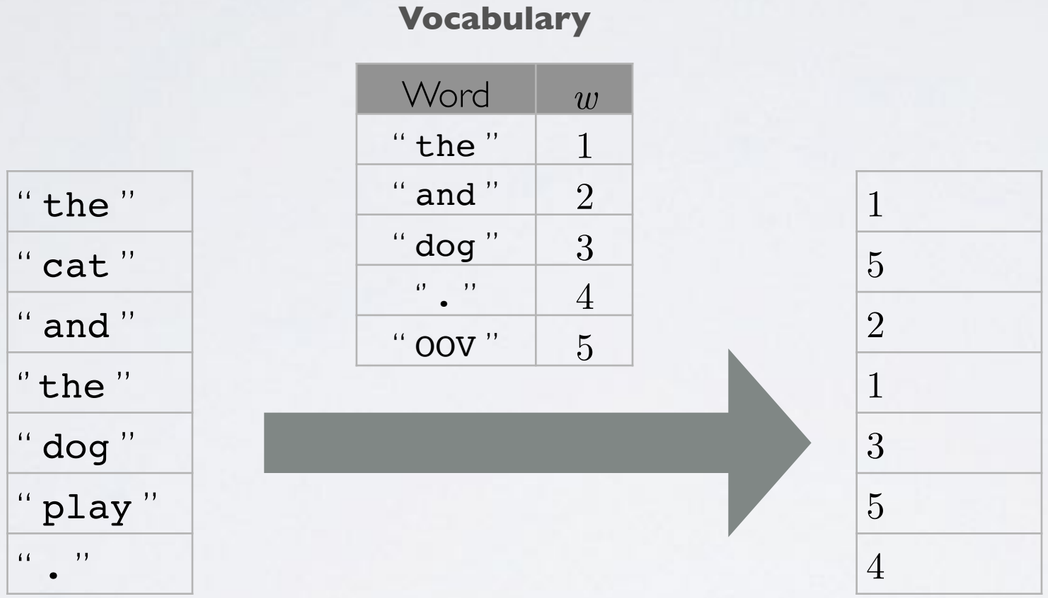
- Typical preprocessing steps of text data
  - form vocabulary of words that maps lemmatized words to a unique ID (position of word in vocabulary)
  - different criteria can be used to select which words are part of the vocabulary
    - pick most frequent words
    - ignore uninformative words from a user-defined short list (ex.: " `the` ", " `a` ", etc.)
  - all words not in the vocabulary will be mapped to a special "out-of-vocabulary" ID

- Typical vocabulary sizes will vary between 10 000 and 250 000

NATURAL LANGUAGE PROCESSING

Topics: vocabulary

- Example:



- We will note word IDs with the symbol w
  - can think of w as a categorical feature for the original word
  - we will sometimes refer to w as a word, for simplicity

NATURAL LANGUAGE PROCESSING

Topics: one-hot encoding

- From its word ID, we get a basic representation of a word through the one-hot encoding of the ID
  - the one-hot vector of an ID is a vector filled with 0s, except for a 1 at the position associated with the ID
    - ex.: for vocabulary size $D=10$, the one-hot vector of word ID $w=4$ is

$$\mathrm{e}(w) = [\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ ]$$

  - a one-hot encoding makes no assumption about word similarity
    - $||\mathrm{e}(w) - \mathrm{e}(w')||^2 = 0$ if $w = w'$
    - $||\mathrm{e}(w) - \mathrm{e}(w')||^2 = 2$ if $w \neq w'$
    - all words are equally different from each other
  - this is a natural representation to start with, though a poor one

NATURAL LANGUAGE PROCESSING

Topics: one-hot encoding

- The major problem with the one-hot representation is that it is very high-dimensional
  - the dimensionality of $\mathrm{e}(w)$ is the size of the vocabulary
  - a typical vocabulary size is $\approx 100\ 000$
  - a window of 10 words would correspond to an input vector of at least $1\ 000\ 000$ units!
- This has 2 consequences:
  - vulnerability to overfitting
    - millions of inputs means millions of parameters to train in a regular neural network
  - computationally expensive
    - not all computations can be sparsified (ex.: reconstruction in autoencoder)

WORD REPRESENTATIONS

Topics: continuous word representation

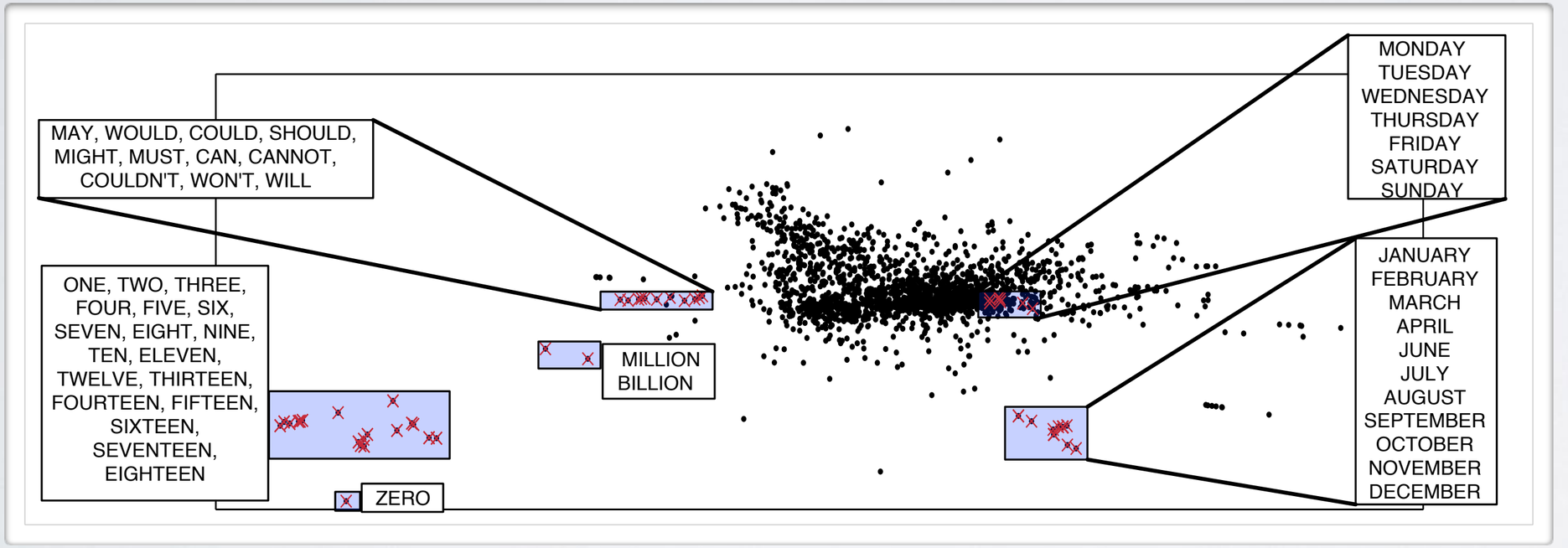
- Idea: learn a continuous representation of words
  - each word w is associated with a real-valued vector $C(w)$

Word	w	$C(w)$
"the"	1	[0.6762, -0.9607, 0.3626, -0.2410, 0.6636]
"a"	2	[0.6859, -0.9266, 0.3777, -0.2140, 0.6711]
"have"	3	[0.1656, -0.1530, 0.0310, -0.3321, -0.1342]
"be"	4	[0.1760, -0.1340, 0.0702, -0.2981, -0.1111]
"cat"	5	[0.5896, 0.9137, 0.0452, 0.7603, -0.6541]
"dog"	6	[0.5965, 0.9143, 0.0899, 0.7702, -0.6392]
"car"	7	[-0.0069, 0.7995, 0.6433, 0.2898, 0.6359]
...	...	...

WORD REPRESENTATIONS

Topics: continuous word representation

- Idea: learn a continuous representation of words
  - we would like the distance $||C(w)-C(w')||$ to reflect meaningful similarities between words



(from Blitzer et al. 2004)

WORD REPRESENTATIONS

Topics: continuous word representation

- Idea: learn a continuous representation of words
  - we could then use these representations as input to a neural network
  - to represent a window of 10 words $[w_1, \ldots, w_{10}]$, we concatenate the representations of each word

$$\mathbf{x} = [C(w_1)^\top, \ldots, C(w_{10})^\top]^\top$$

- We learn these representations by gradient descent
  - we don't only update the neural network parameters
  - we also update each representation $C(w)$ in the input $\mathbf{x}$ with a gradient step

$$C(w) \Longleftarrow C(w) - \alpha \nabla_{C(w)} l$$

where l is the loss function optimized by the neural network

WORD REPRESENTATIONS

Topics: word representations as a lookup table

- Let $\mathbf{C}$ be a matrix whose rows are the representations $C(w)$
  - obtaining $C(w)$ corresponds to the multiplication $\mathbf{e}(w)^\top \mathbf{C}$
  - viewed differently, we are projecting $\mathbf{e}(w)$ onto the columns of C
    - this is a reduction of the dimensionality of the one-hot representations $\mathbf{e}(w)$
  - this is a continuous transformation, through which we can propagate gradients

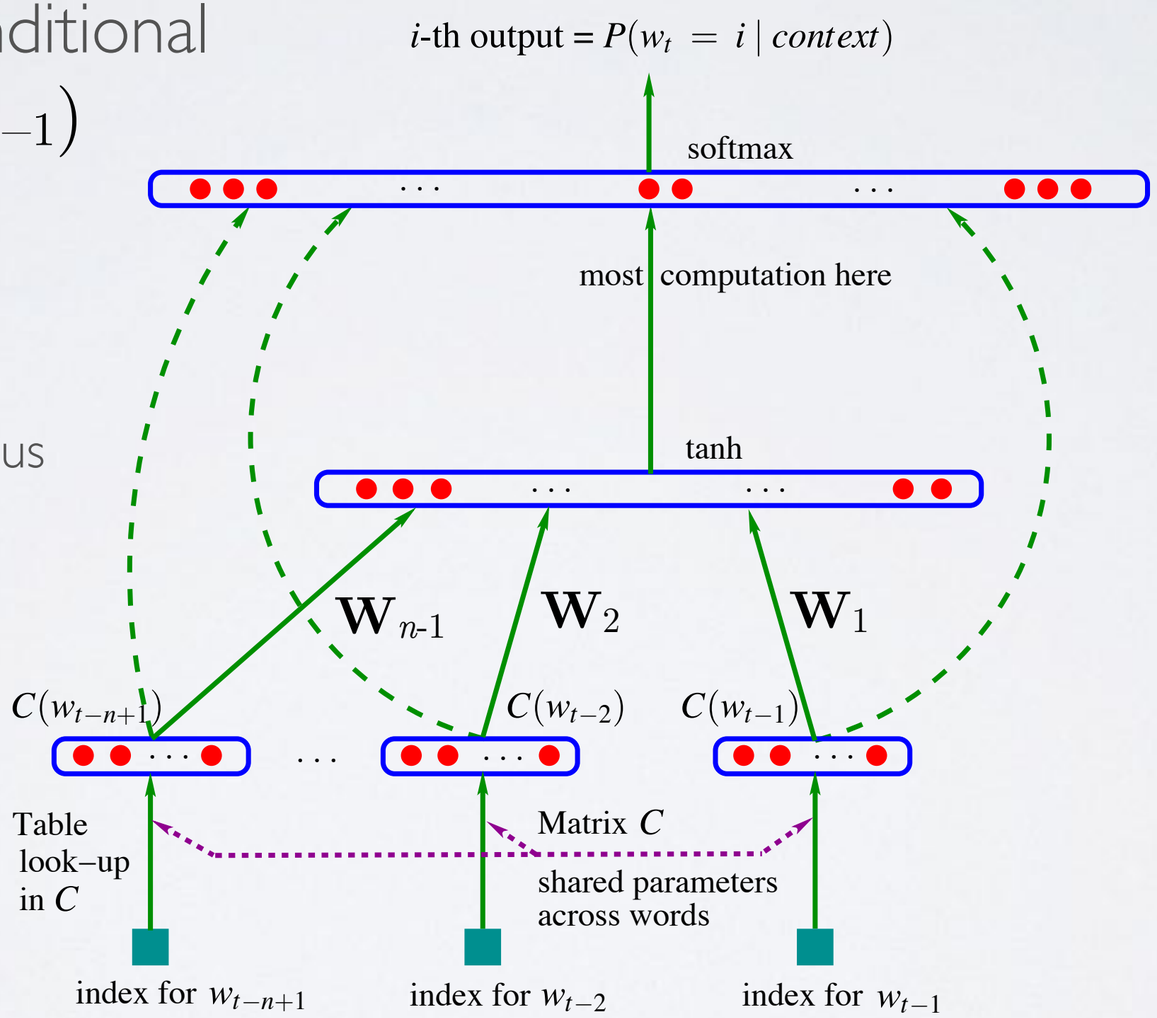
- In practice, we implement $C(w)$ with a lookup table, not with a multiplication
  - $C(w)$ returns an array pointing to the w^{th} row of $\mathbf{C}$

NEURAL NETWORK LANGUAGE MODEL

Topics: neural network language model

- Solution: model the conditional $p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$ with a neural network
  - learn word representations to allow transfer to n-grams not observed in training corpus

Bengio, Ducharme, Vincent and Jauvin, 2003

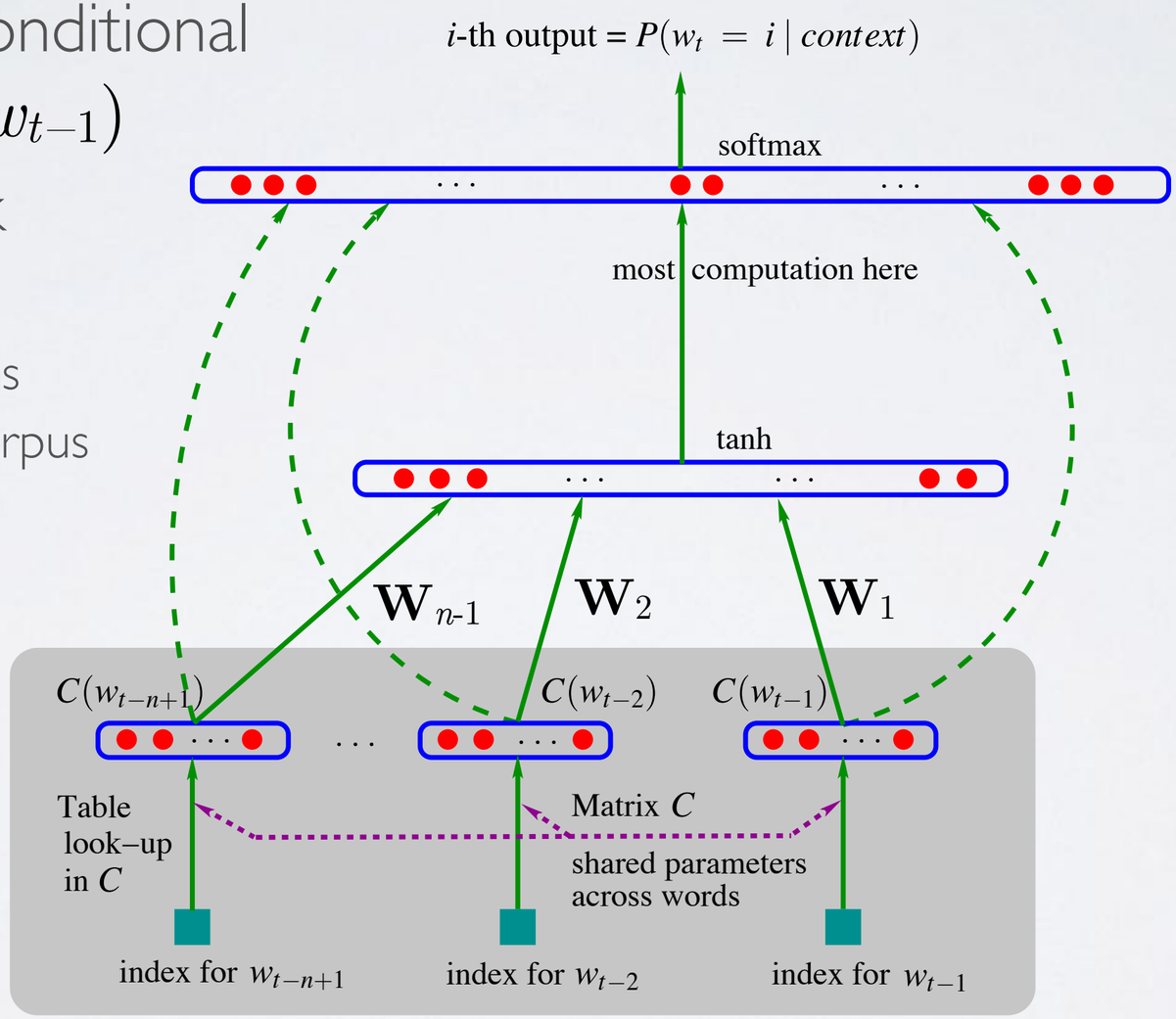


NEURAL NETWORK LANGUAGE MODEL

Topics: neural network language model

- Solution: model the conditional $p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$ with a neural network
  - learn word representations to allow transfer to n-grams not observed in training corpus

Bengio, Ducharme, Vincent and Jauvin, 2003

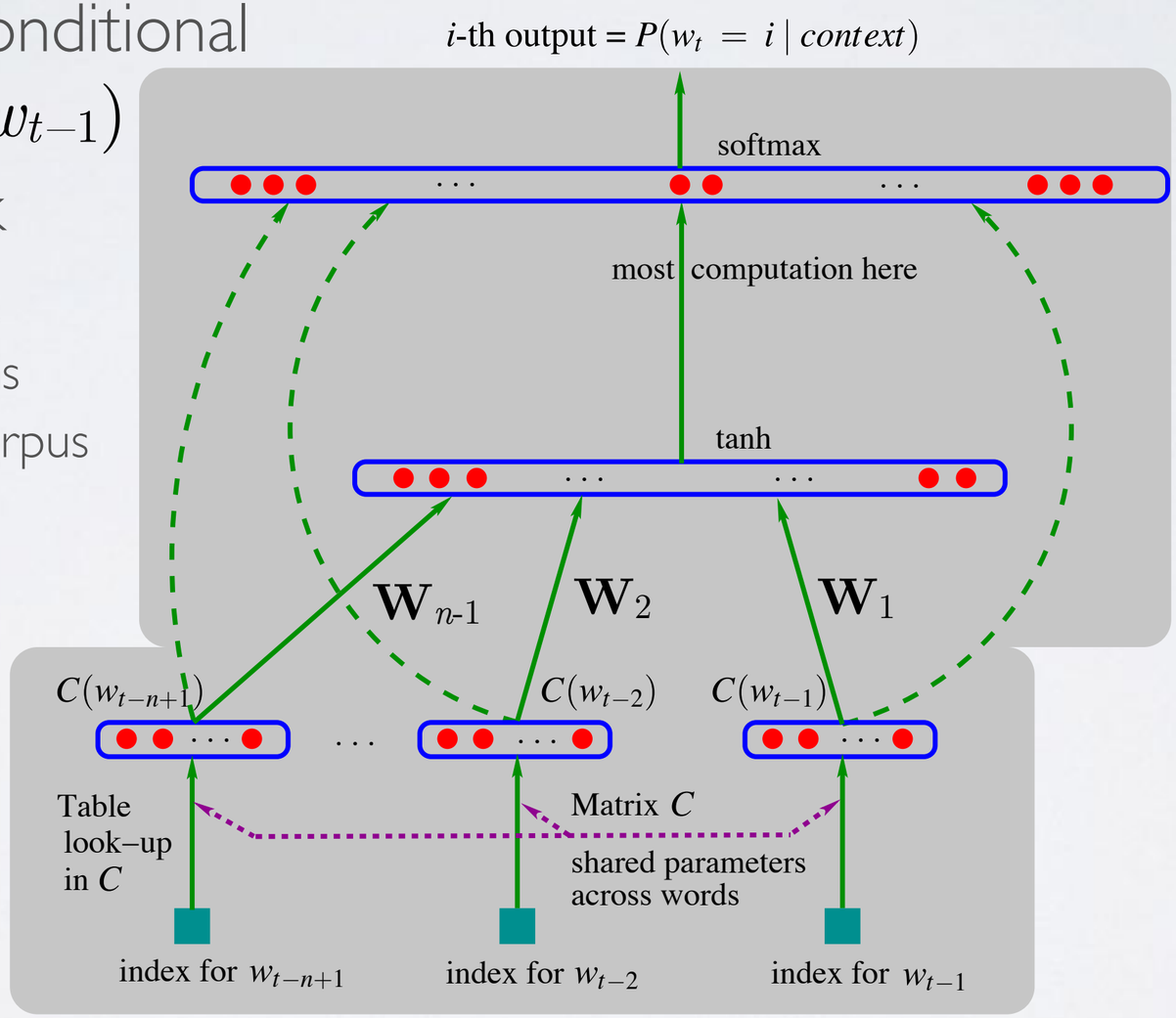


NEURAL NETWORK LANGUAGE MODEL

Topics: neural network language model

- Solution: model the conditional $p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$ with a neural network
  - learn word representations to allow transfer to n-grams not observed in training corpus

Bengio, Ducharme, Vincent and Jauvin, 2003

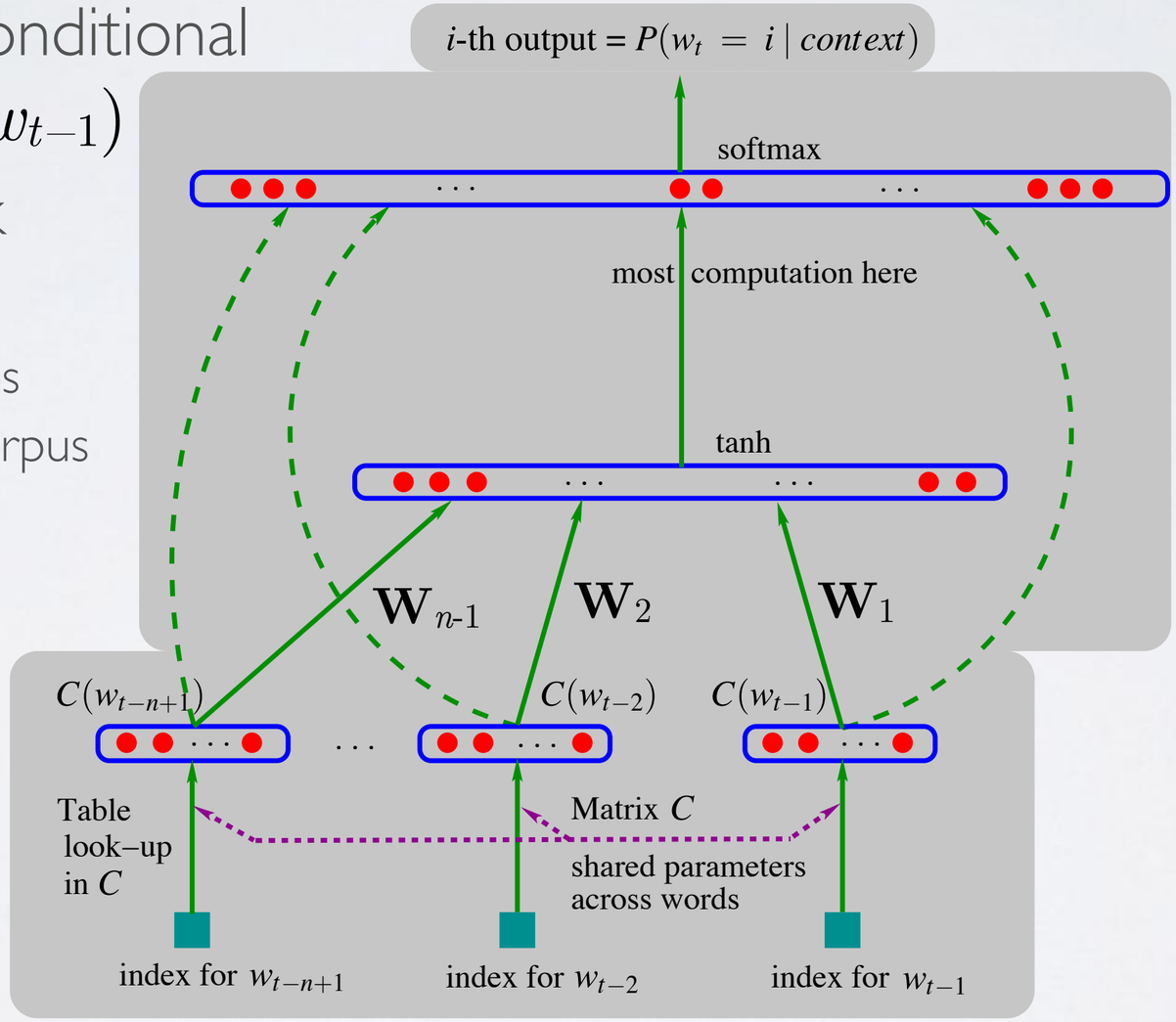


NEURAL NETWORK LANGUAGE MODEL

Topics: neural network language model

- Solution: model the conditional $p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$ with a neural network
  - learn word representations to allow transfer to n-grams not observed in training corpus

Bengio, Ducharme, Vincent and Jauvin, 2003



NEURAL NETWORK LANGUAGE MODEL

Topics: neural network language model

- Can potentially generalize to contexts not seen in training set
  - example: $p(\text{“}\texttt{eating}\text{”} \mid \text{“}\texttt{the}\text{”}, \text{“}\texttt{cat}\text{”}, \text{“}\texttt{is}\text{”})$
    - imagine 4-gram [“`the`”,“`cat`”,“`is`”, “`eating`”] is not in training corpus, but [“`the`”,“`dog`”,“`is`”, “`eating`”] is
    - if the word representations of “`cat`” and “`dog`” are similar, then the neural network will be able to generalize to the case of “`cat`”
    - neural network could learn similar word representations for those words based on other 4-grams:

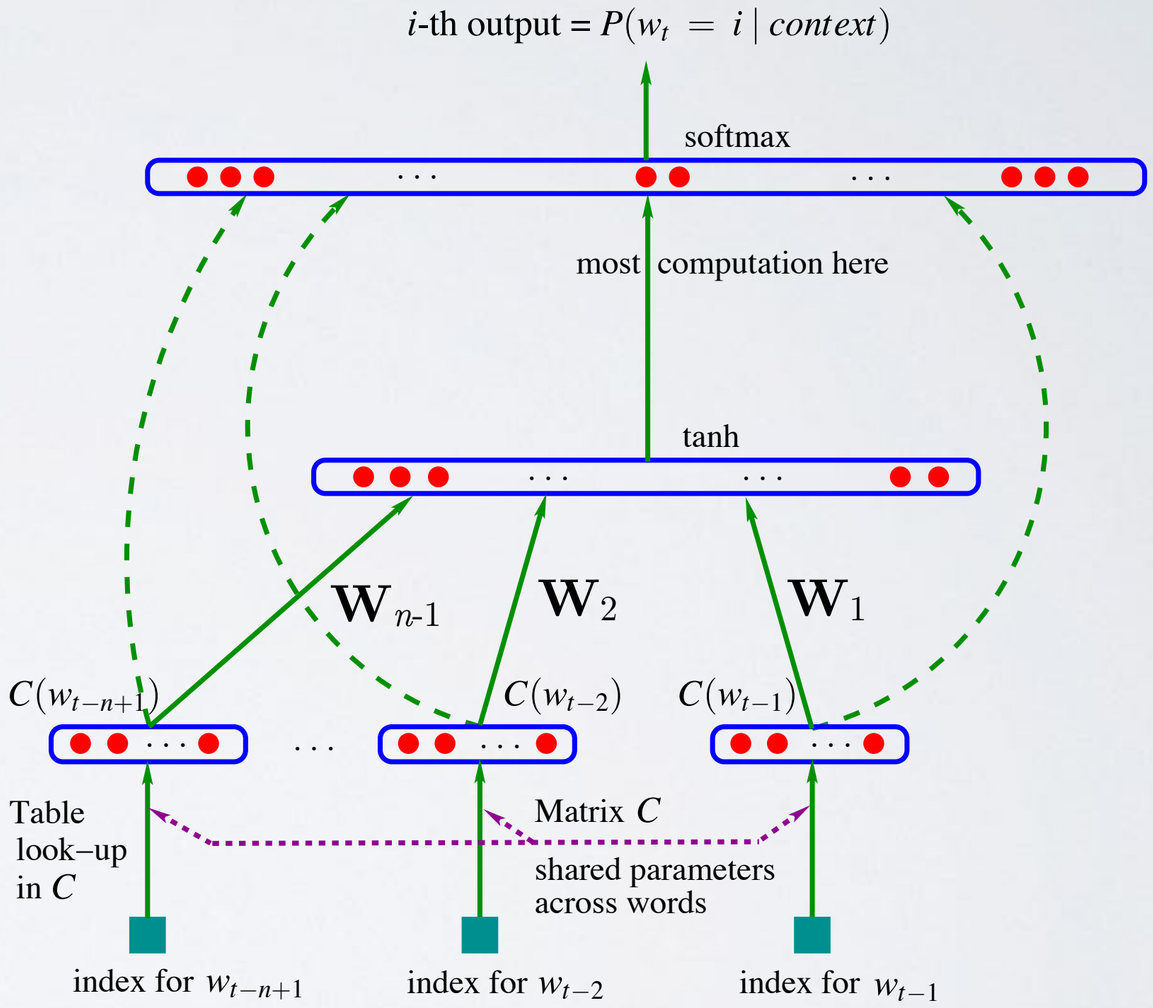
[“`the`”,“`cat`”,“`was`”, “`sleeping`”]
[“`the`”,“`dog`”,“`was`”, “`sleeping`”]

NEURAL NETWORK LANGUAGE MODEL

Topics: word representation gradients

- We know how to propagate gradients in such a network
  - we know how to compute the gradient for the linear activation of the hidden layer $\nabla_{\mathbf{a}(\mathbf{x})} l$
  - let's note the submatrix connecting w_{t-i} and the hidden layer as $\mathbf{W}_i$
- The gradient wrt $C(w)$ for any w is

$$\nabla_{C(w)} l = \sum_{i=1}^{n-1} 1_{(w_{t-i}=w)} \, \mathbf{W}_i^\top \, \nabla_{\mathbf{a}(\mathbf{x})} l$$



NEURAL NETWORK LANGUAGE MODEL

Topics: word representation gradients

- Example: [“ `the` ”,“ `dog` ”,“ `and` ”, “ `the` ”, “ `cat` ”]

$$\begin{array}{ccccccccc} & w_3 & & w_4 & & w_5 & & w_6 & & w_7 \\ & \searrow & & \searrow & & \searrow & & \searrow & & \\ & 21 & & 3 & & 14 & & 21 & & \end{array}$$

  - the loss is $l = -\log p(\text{“ } \texttt{cat} \text{ ”} \mid \text{“ } \texttt{the} \text{ ”},\text{“ } \texttt{dog} \text{ ”},\text{“ } \texttt{and} \text{ ”}, \text{“ } \texttt{the} \text{ ”})$
  - $\nabla_{C(3)} l = \mathbf{W}_3^\top \nabla_{\mathbf{a}(\mathbf{x})} l$
  - $\nabla_{C(14)} l = \mathbf{W}_2^\top \nabla_{\mathbf{a}(\mathbf{x})} l$
  - $\nabla_{C(21)} l = \mathbf{W}_1^\top \nabla_{\mathbf{a}(\mathbf{x})} l + \mathbf{W}_4^\top \nabla_{\mathbf{a}(\mathbf{x})} l$
  - $\nabla_{C(w)} l = 0$ for all other words w

- Only need to update the representations $C(3)$, $C(14)$ and $C(21)$,

CLASSIFYING TEXT DATA

Topics: neural network architectures for text classification

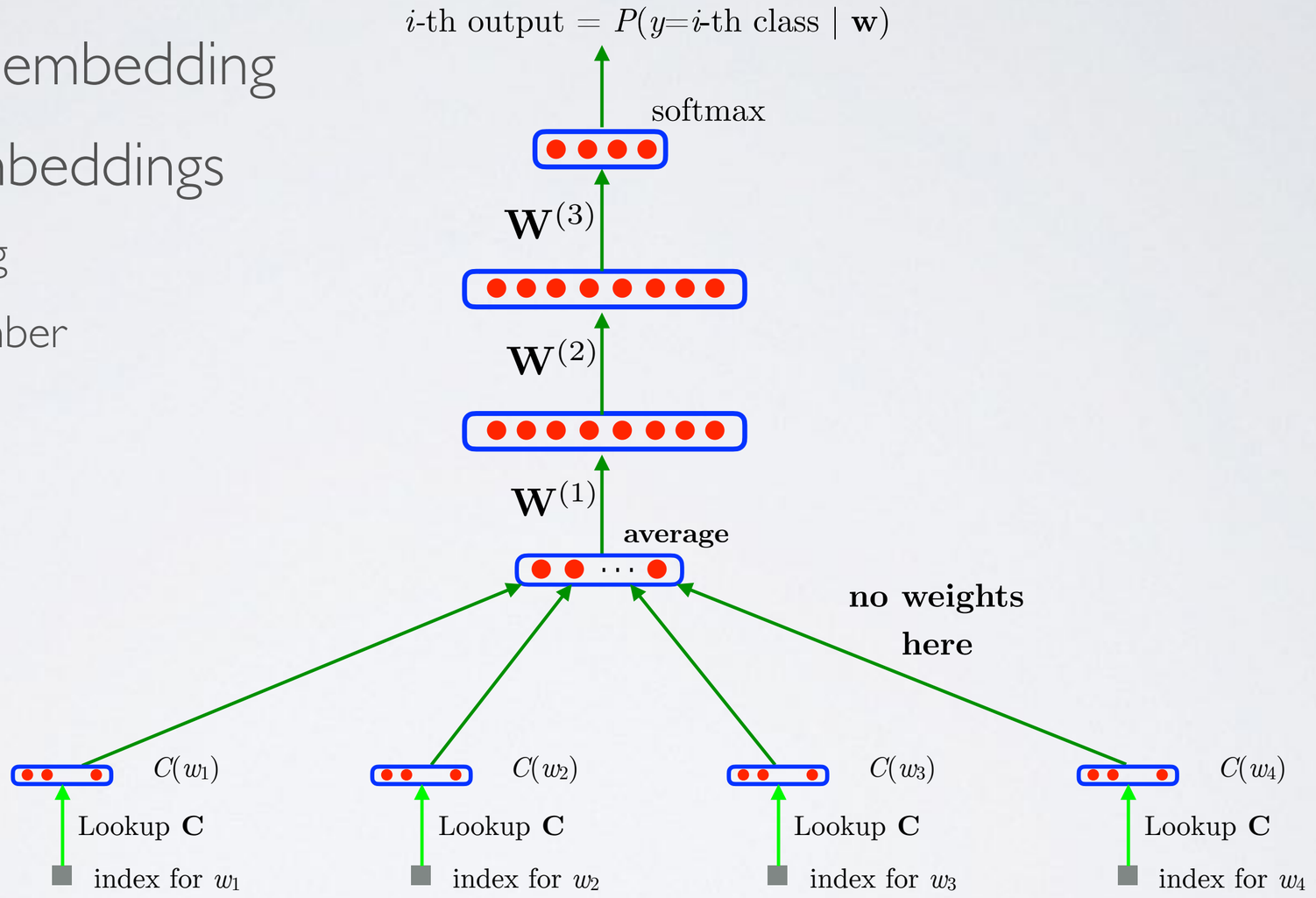
- Need to go from word representations to text representation (e.g. sentences, documents, etc.)
  - from text representation, can feed to (multiple) feed-forward representations
  - how to go from sequence of word embeddings to single text embedding?

- Depending on the complexity of the problem, best architecture will vary
  - average word embedding
  - recurrent neural networks (RNNs)
  - long short-term memory (LSTM) networks

AVERAGE WORD EMBEDDING

Topics: average word embedding

- Simply average the embeddings
  - referred to as mean pooling
  - may use the sum if the number of words is important
  - may use a TF-IDF weighting within the average



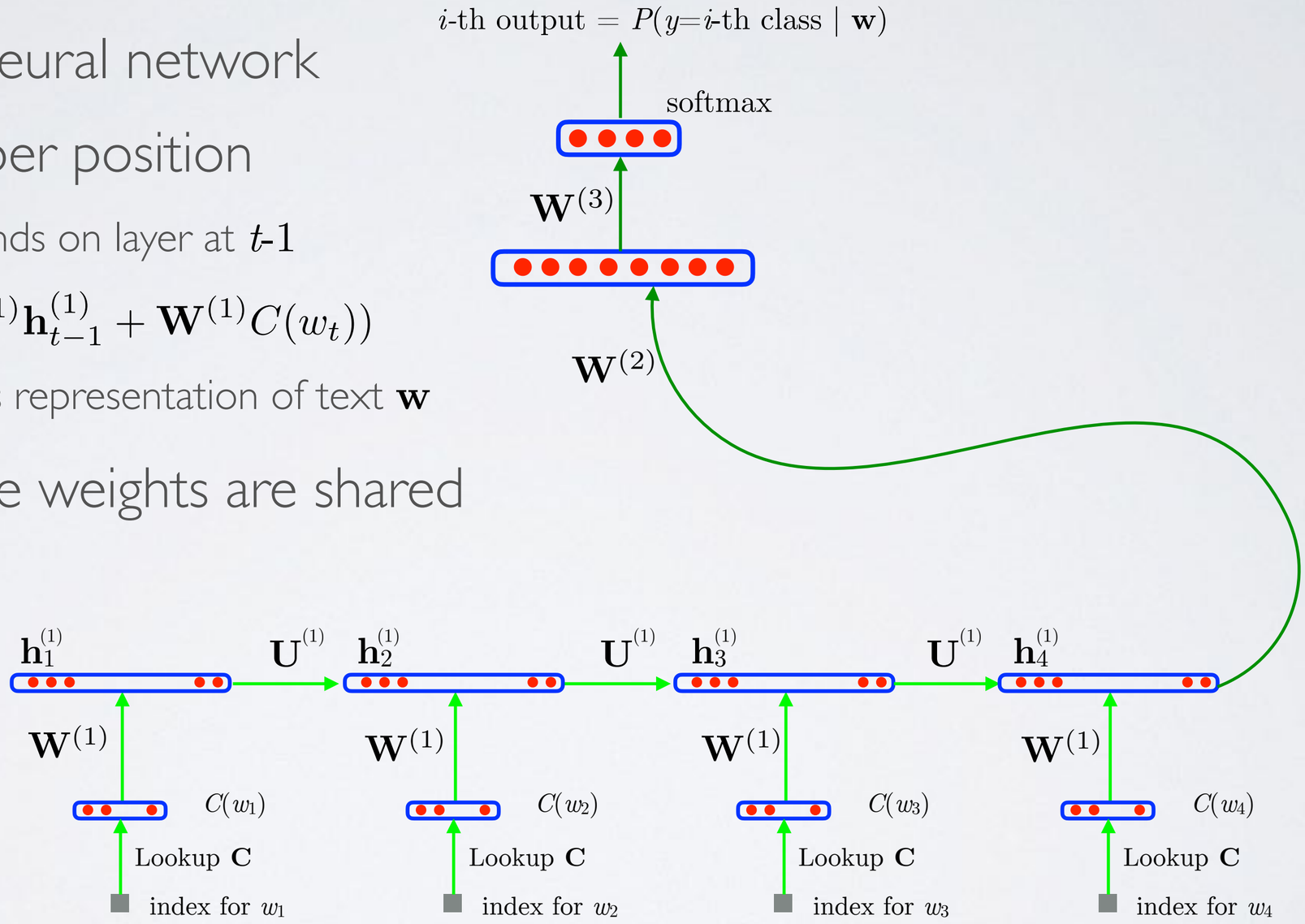
RECURRENT NEURAL NETWORK

Topics: recurrent neural network

- Have hidden layer per position
  - layer at position t depends on layer at t-1

    $$\mathbf{h}_t^{(1)} = \tanh(\mathbf{b}^{(1)} + \mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}C(w_t))$$

  - use last position layer as representation of text $\mathbf{w}$
- "Recurrent" because weights are shared
  - may initialize $\mathbf{h}_0^{(1)}=0$



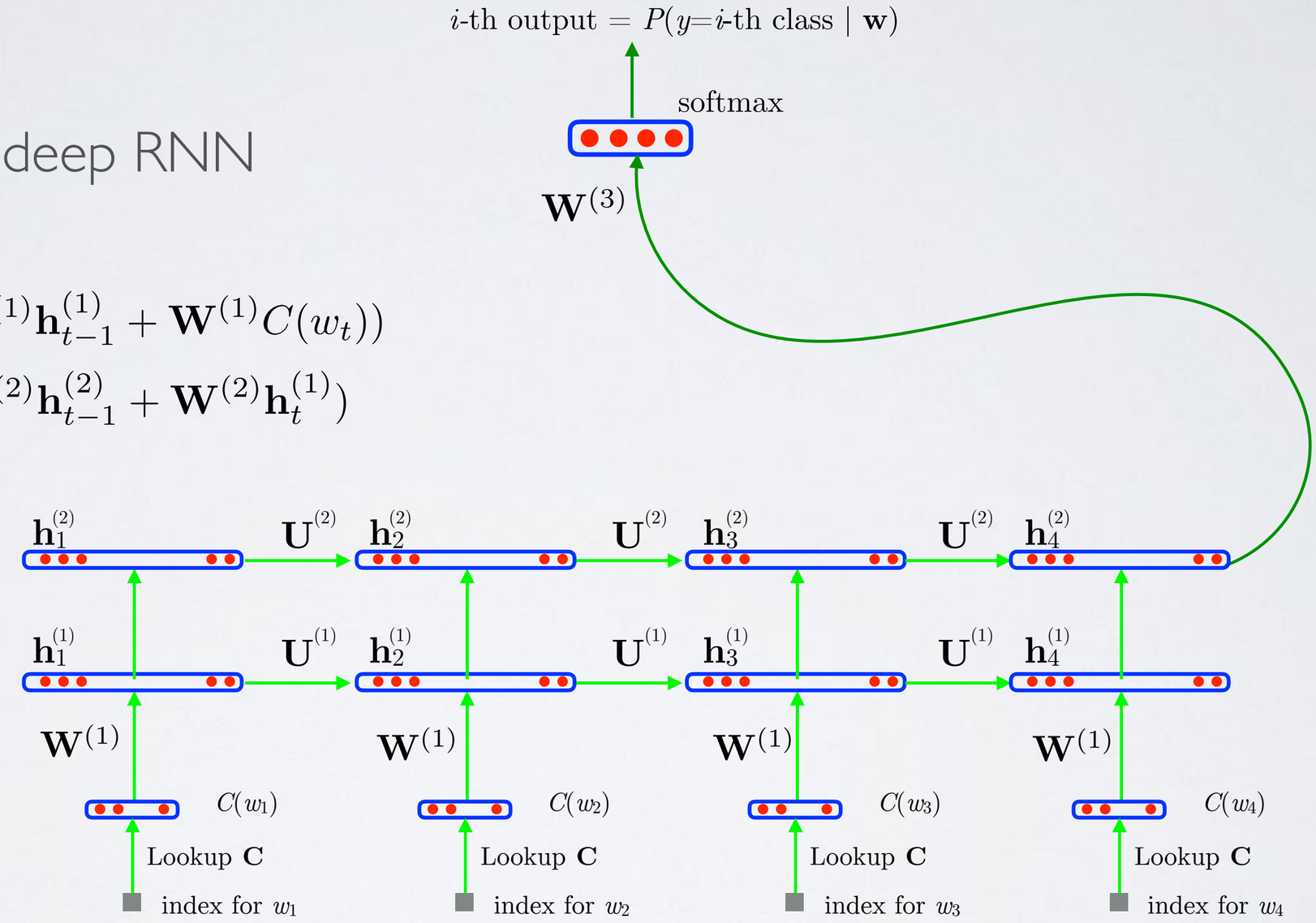
RECURRENT NEURAL NETWORK

Topics: deep RNN

- Easy to turn into a deep RNN
  - example with depth 2

$$\mathbf{h}_t^{(1)} = \tanh(\mathbf{b}^{(1)} + \mathbf{U}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}^{(1)}C(w_t))$$

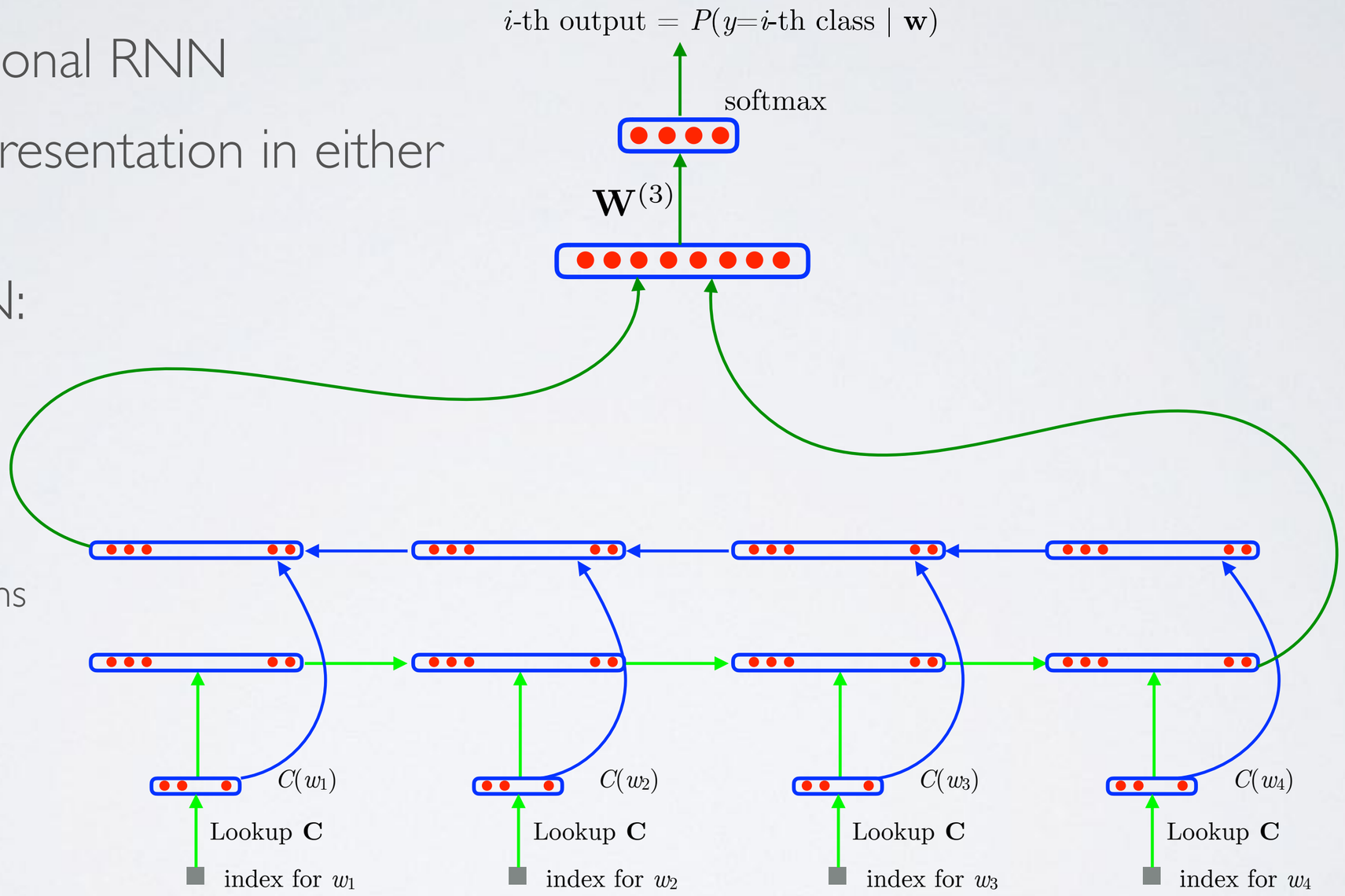
$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{b}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{W}^{(2)}\mathbf{h}_t^{(1)})$$



RECURRENT NEURAL NETWORK

Topics: bidirectional RNN

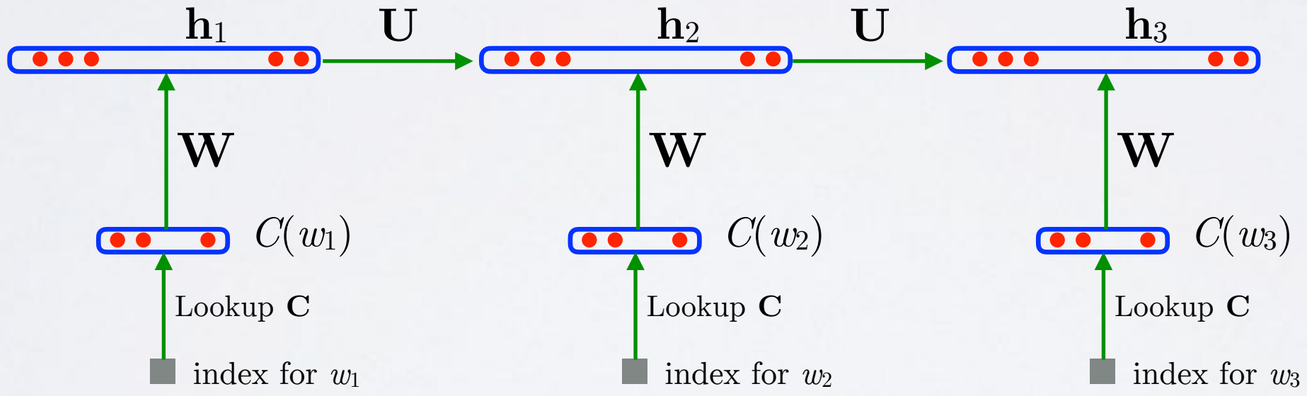
- Can extract representation in either direction
- Bidirection RNN:
  - have 2 RNNs, one in each direction
  - concatenation representation from both directions



RECURRENT NEURAL NETWORK

Topics: recurrent neural network

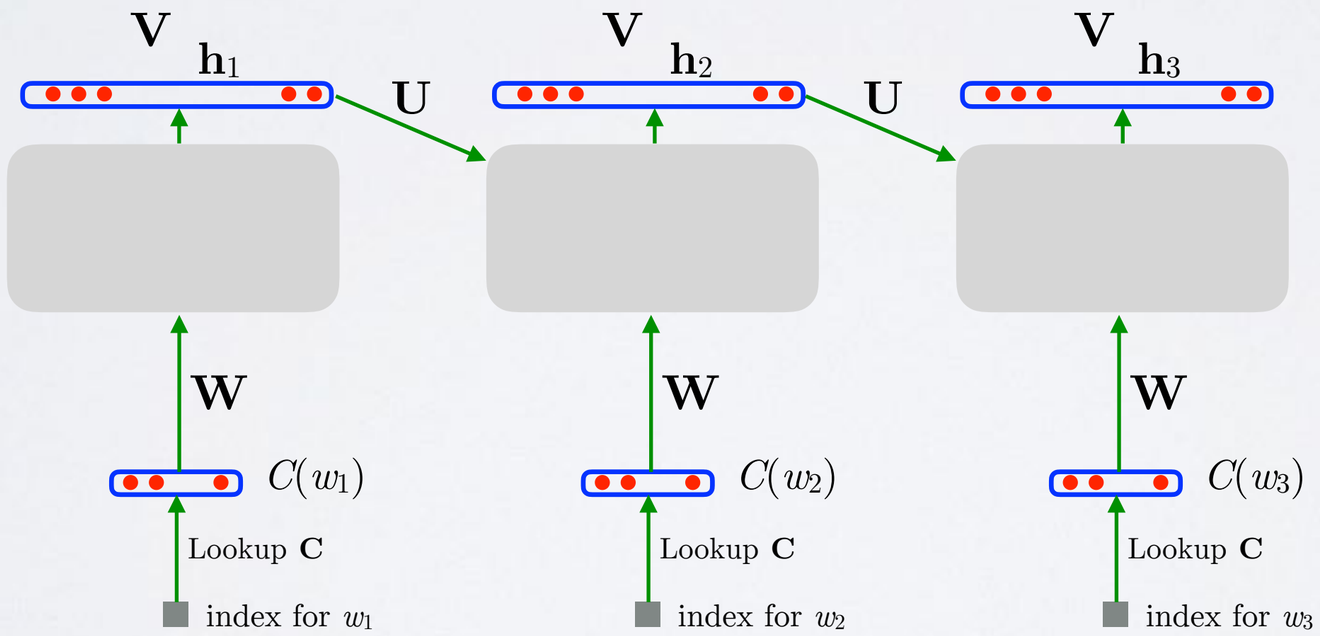
- RNNs easily suffer from the vanishing gradient problem
- Long short-term memory (LSTM) network address this issue



LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**



Hochreiter, Schmidhuber 1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

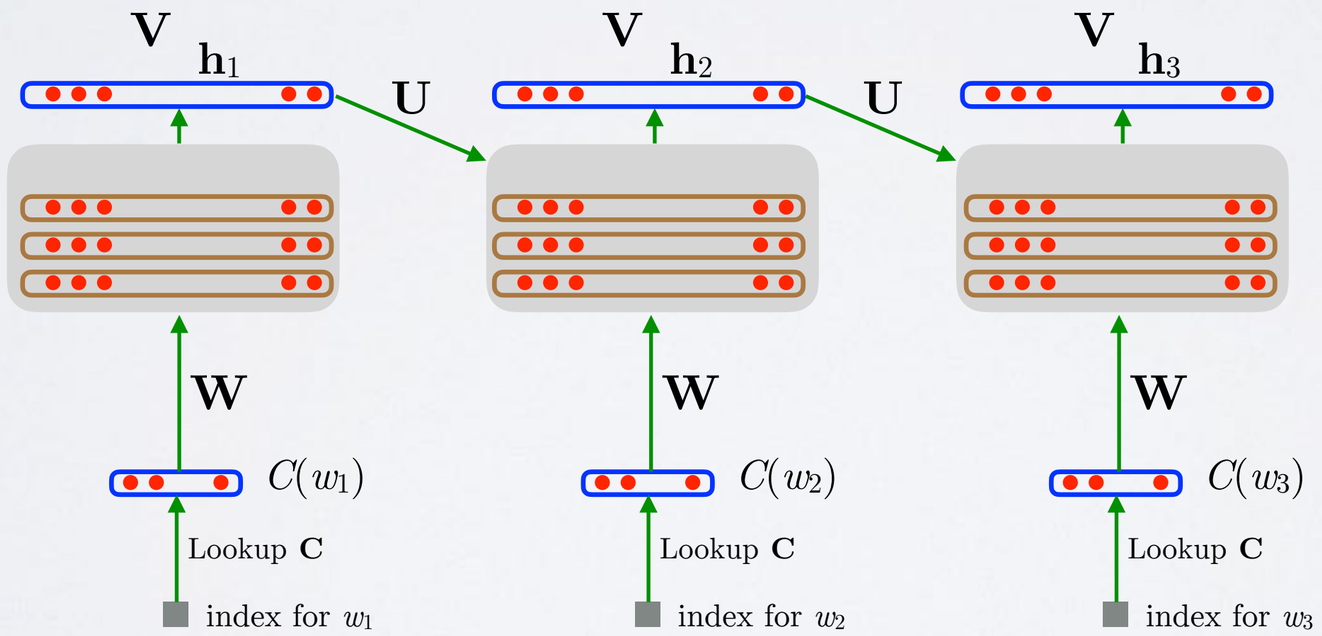
- Layer $\mathbf{h}_t$ is a function of **memory cells**

Input, forget, output gates:

$$\mathbf{i}_t = \text{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \text{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \text{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

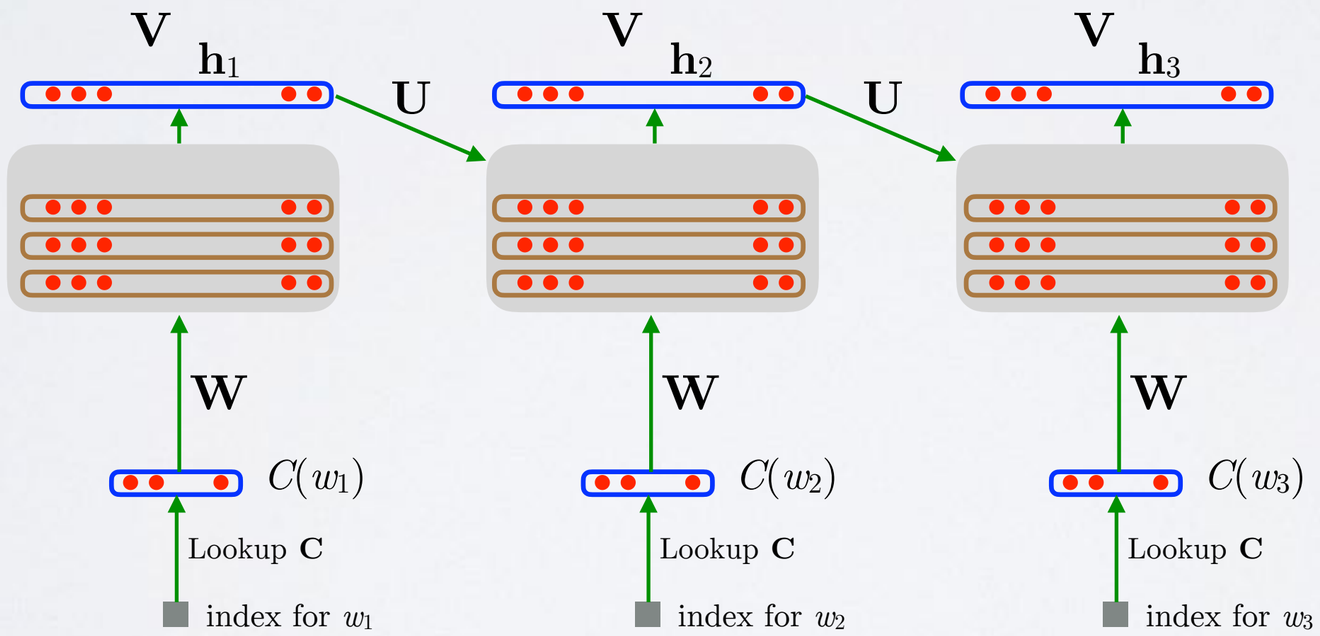


Hochreiter, Schmidhuber 1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**



Hochreiter, Schmidhuber 1995

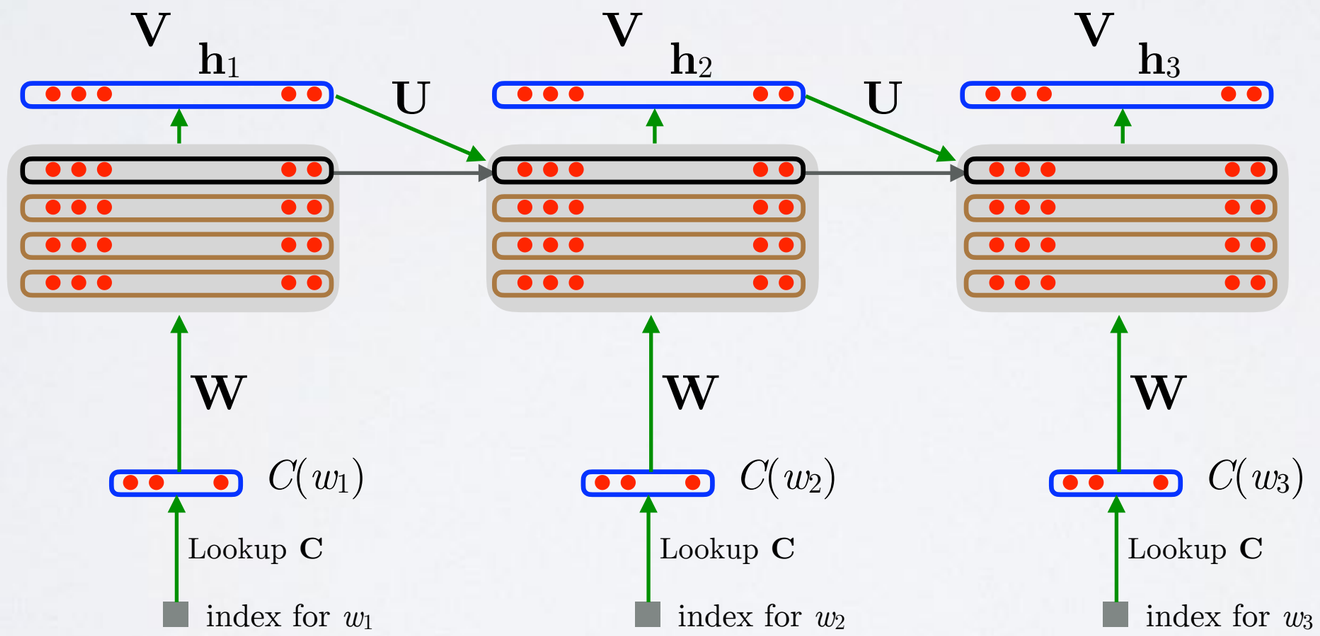
LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

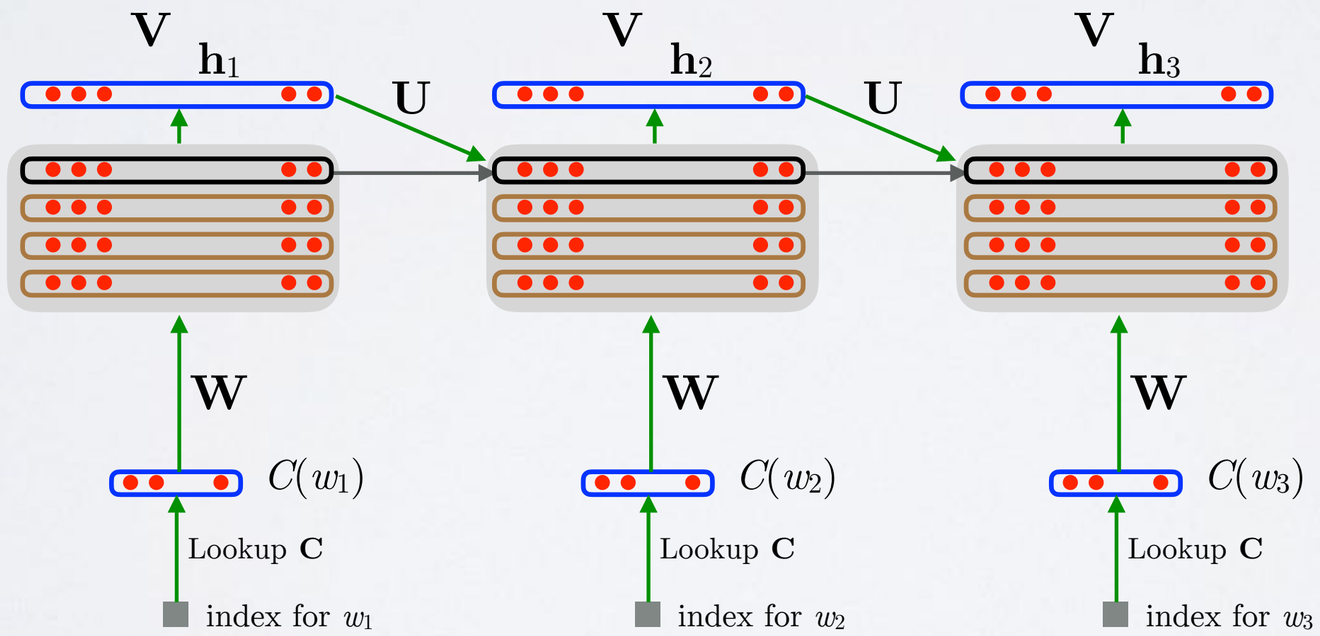


Hochreiter, Schmidhuber 1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**



Hochreiter, Schmidhuber 1995

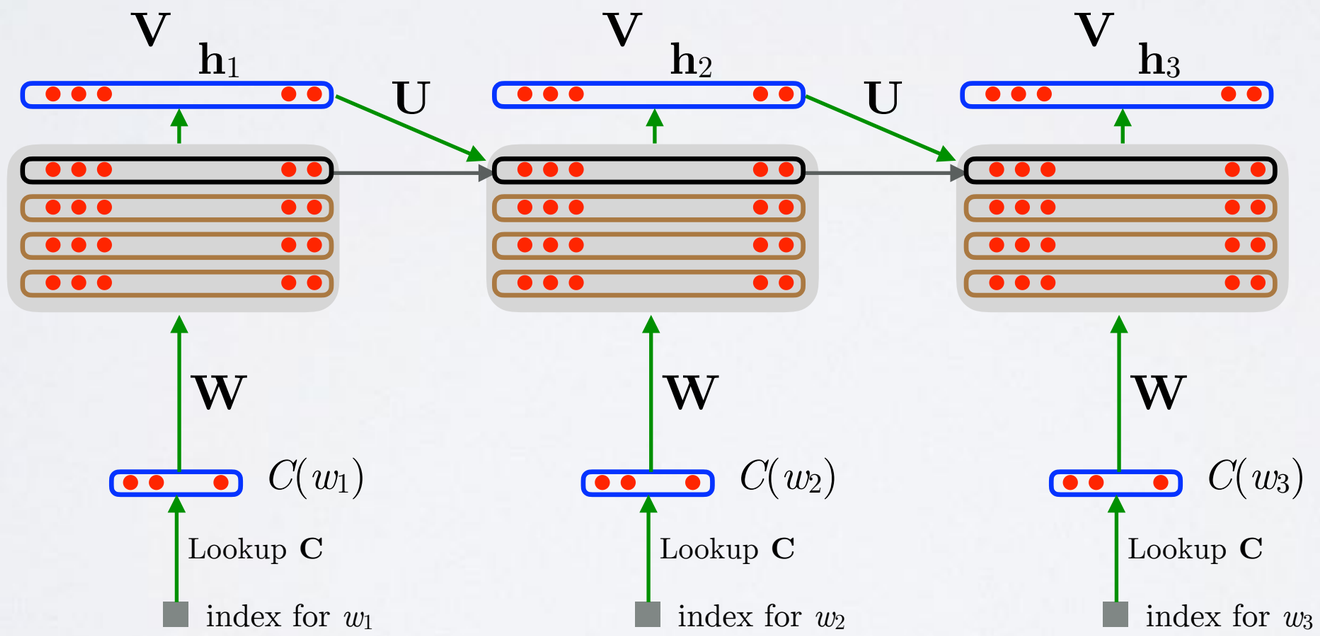
LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- Layer $\mathbf{h}_t$ is a function of **memory cells**

Hidden layer:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$



Hochreiter, Schmidhuber 1995

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \text{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \text{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \text{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

Hidden layer:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \text{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$

$$\mathbf{f}_t = \text{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$

$$\mathbf{o}_t = \text{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

The **gates** control the flow of information in ($\mathbf{i}_t$, $\mathbf{f}_t$,) and out ($\mathbf{o}_t$) of the cell

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

Hidden layer:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \text{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$
$$\mathbf{f}_t = \text{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$
$$\mathbf{o}_t = \text{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

The **gates** control the flow of information in ($\mathbf{i}_t$, $\mathbf{f}_t$,) and out ($\mathbf{o}_t$) of the cell

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

The **cell state** maintains information on the input

Hidden layer:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long short-term memory (LSTM) network

- To sum up:

Input, forget, output gates:

$$\mathbf{i}_t = \text{sigm}(\mathbf{b}_{[i]} + \mathbf{U}_{[i]}\mathbf{h}_{t-1} + \mathbf{W}_{[i]}C(w_t))$$
$$\mathbf{f}_t = \text{sigm}(\mathbf{b}_{[f]} + \mathbf{U}_{[f]}\mathbf{h}_{t-1} + \mathbf{W}_{[f]}C(w_t))$$
$$\mathbf{o}_t = \text{sigm}(\mathbf{b}_{[o]} + \mathbf{U}_{[o]}\mathbf{h}_{t-1} + \mathbf{W}_{[o]}C(w_t))$$

The **gates** control the flow of information in ($\mathbf{i}_t$, $\mathbf{f}_t$,) and out ($\mathbf{o}_t$) of the cell

Cell state:

$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_{[c]} + \mathbf{U}_{[c]}\mathbf{h}_{t-1} + \mathbf{W}_{[c]}C(w_t))$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

The **cell state** maintains information on the input

Hidden layer:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

The **hidden layer** sees what passes through the output gate

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{f}_{t-1} \odot \mathbf{c}_{t-2} + \mathbf{f}_t \odot \mathbf{i}_{t-1} \odot \widetilde{\mathbf{c}}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t =$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \sum_{t'=0}^{t} \mathbf{f}_t \odot \cdots \odot \mathbf{f}_{t'+1} \odot \mathbf{i}_{t'} \odot \widetilde{\mathbf{c}}_{t'}$$

LONG SHORT-TERM MEMORY NETWORK

Topics: long-term dependencies, forget bias initialization

- Why is it better at learning long-term dependencies?

$$\mathbf{c}_t = \sum_{t'=0}^{t} \mathbf{f}_t \odot \cdots \odot \mathbf{f}_{t'+1} \odot \mathbf{i}_{t'} \odot \widetilde{\mathbf{c}}_{t'}$$

- As long as forget gates are open (close to 1), gradient may pass into $\widetilde{\mathbf{c}}_t$ over long time gaps
  - saturation of forget gates doesn't stop gradient flow
  - suggests that a better initialization of forget gate bias $\mathbf{b}_{[f]}$ is $\gg 0$ (e.g. 1)
- Easy to compute gradients with automatic differentiation
  - known as backprop through time (BPTT)

MERCI!