TEXAS A&M UNIVERSITY

CSCE 633: Machine Learning

Lecture 8

Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

Deep neural networks: Motivation

Traditional recognition

But what's next?

shallower

deeper

Deep neural networks: Motivation

Deep Learning

Specialized components, domain knowledge required

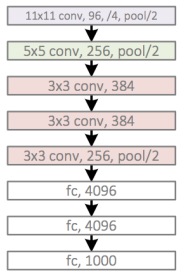
Generic components ("layers"), less domain knowledge

Repeat elementary layers => Going deeper

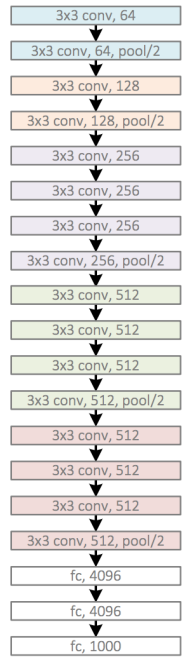
- End-to-end learning
- Richer solution space

Deep neural networks: Motivation

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)

Deep neural networks: Motivation

Why go deep?

- Deep Representations might allow for a hierarchy or representation
  - Non-local generalization
  - Comprehensibility
- Multiple levels of latent variables allow combinatorial sharing of statistical strength
- Deep architectures work well (vision, audio, NLP, etc.)!

[Contents for the following slides have been summarized from the NIPS 2010, CVPR 2012, and Stanford Deep Learning Tutorials]

Deep neural networks: Motivation

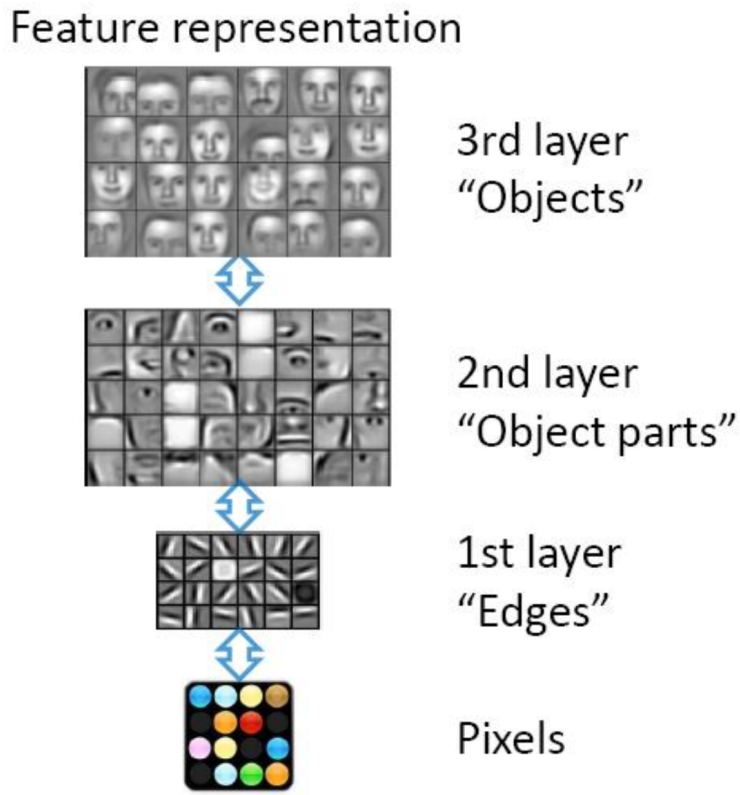
Key ideas of (deep) neural networks

- Learn features from data
- Use differentiable functions that produce features efficiently
- End-to-end learning: no distinction between feature extractor and classifier
- "Deep" architectures: cascade of simpler non-linear modules

Deep neural networks: Motivation

Different levels of abstraction:Hierarchical learning

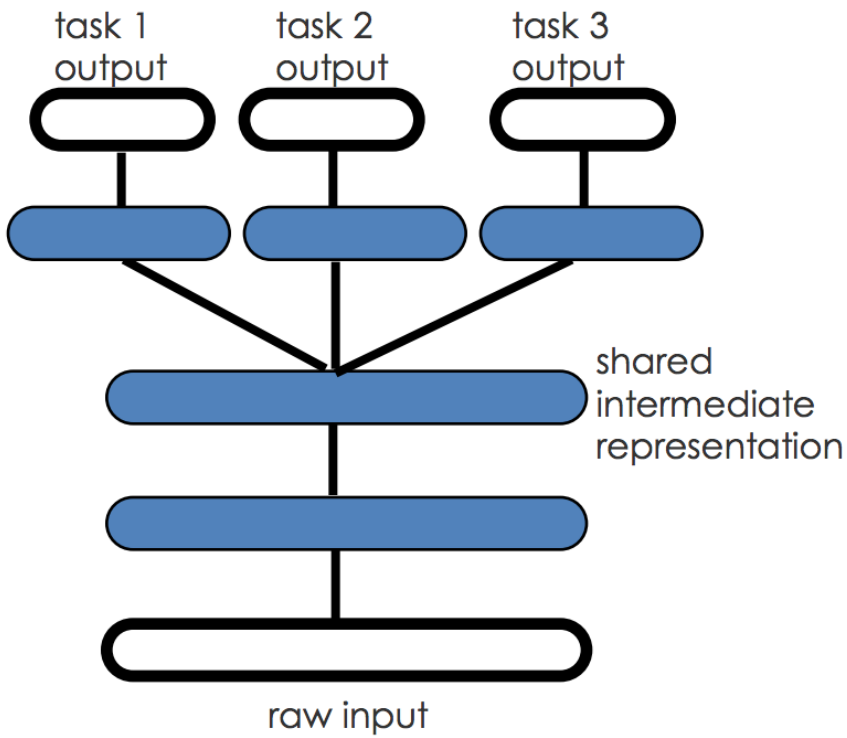
- Natural progression from low level to high level structure as seen in natural complexity
- Easier to monitor what is being learnt and to guide the machine to better subspaces
- A good lower level representation can be used for many distinct tasks



Deep neural networks: Motivation

Shared low-level representations

- Multi-task learning
- Unsupervised training



Deep neural networks: Challenges

High memory requirements

- Memory is used to store input data, weight parameters and activations as an input propagates through the network
- Activations from a forward pass must be retained until they can be used to calculate the error gradients in the backwards pass
- Example: 50-layer neural network
  - 26 million weight parameters, 16 million activations in the forward pass
  - 168MB memory (assuming 32-bit float)

Parallelize computations with GPU (graphics processing units)

Deep neural networks: Challenges

Backpropagation does not work well

- Deep networks trained with backpropagation (without unsupervised pretraining) perform worse than shallow networks
- Gradient is progressively getting more dilute
  - Weight correction is minimal after moving back a couple of layers
- High risk of getting "stuck" to local minima
- In practice, a small portion of data is labelled

Perform pretraining to mitigate this issue

Example error rates with and without pretraining

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.2%	1.2%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.7%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

(Bengio et al., NIPS 2007)

Overview

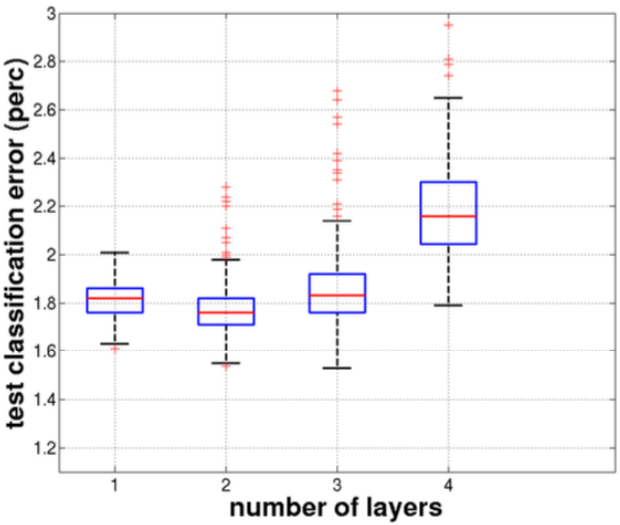
- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

Deep neural networks: Unsupervised Pretraining

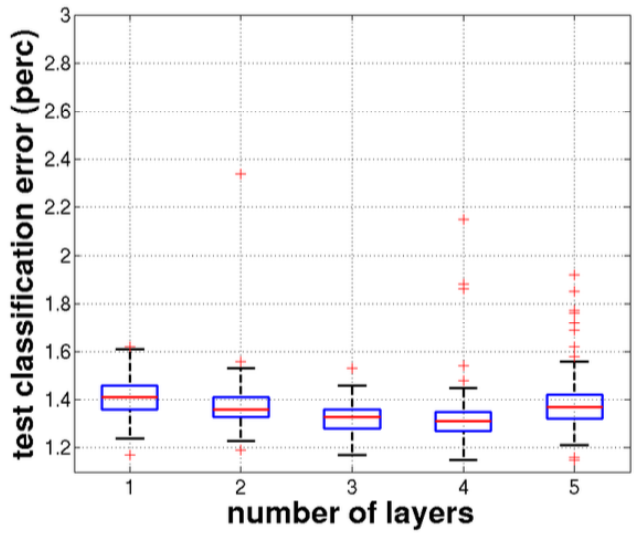
- This idea came into play when research studies found that a DNN trained on a particular task (e.g. object recognition) can be applied on another domain (e.g. object subcategorization) giving state-of-the-art results
- **1st part: Greedy layer-wise unsupervised pre-training**
  - Each layer is pre-trained with an unsupervised learning algorithm
  - Learning a nonlinear transformation that captures the main variations in its input (the output of the previous layer)
- **2nd part: Supervised fine-tuning**
  - The deep architecture is fine-tuned with respect to a supervised training criterion with gradient-based optimization
- We will examine the deep belief networks and stacked autoencoders

Unusual form of regularization: minimizing variance and introducing bias towards configurations of the parameter space that are useful for unsupervised learning

Deep neural networks: Unsupervised Pretraining



Without pre-training



With pre-training

[Source: Erhan et al., 2010]

Deep neural networks: Unsupervised Pretraining

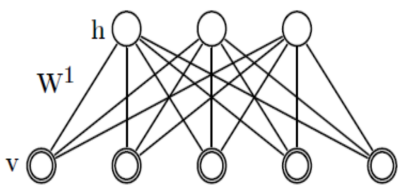
Deep Belief Networks (Hinton et al. 2006)

- Pretraining is implemented by stacking several layers of Restricted Boltzmann Machines (RBM) in a greedy manner
- Assuming joint distribution between hidden h_i and observed variables x_j with parameters $\mathbf{W}, \mathbf{b}, \mathbf{c}$
  $P(\mathbf{x}, \mathbf{h}) \propto \exp(\mathbf{h}^T \mathbf{W} \mathbf{x} + \mathbf{b}^T \mathbf{x} + \mathbf{c}^T \mathbf{h})$
  $P(\mathbf{x}|\mathbf{h}) = \prod_j P(x_j|\mathbf{h}),\ P(x_j = 1|\mathbf{h}) = \text{sigmoid}(b_j + \sum_i W_{ij} h_i)$
  $P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x}),\ P(h_i = 1|\mathbf{x}) = \text{sigmoid}(c_i + \sum_j W_{ij} x_j)$
- RBM trained by approximate stochastic gradient descent
- This representation is extended to all hidden layers
- The RBM parameters correspond to the parameters of the feed-forward multi-layer neural network

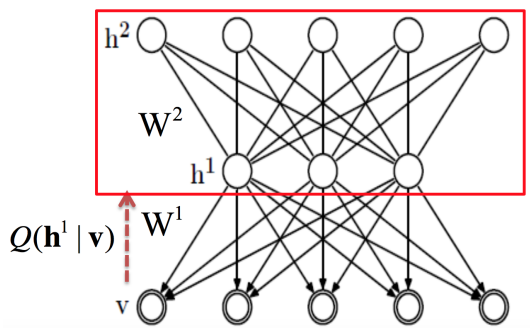
Deep neural networks: Unsupervised Pretraining

Deep Belief Networks (Hinton et al. 2006)

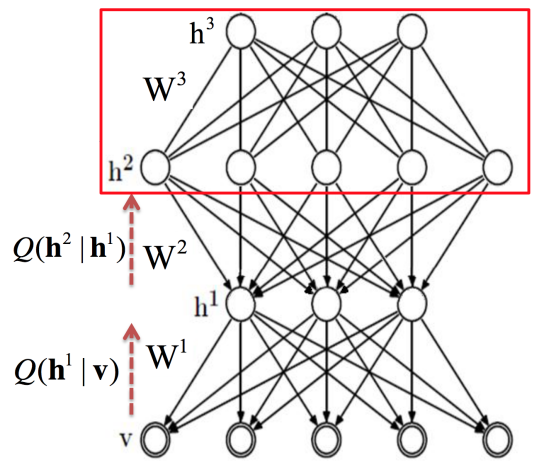
- Step 1: Construct an RBM with an input and hidden layer and train to find $\mathbf{W}^{(1)}$
- Step 2: Stack another hidden layer on top of the RBM to form a new RBM
  - Fix $\mathbf{W}^{1}$. Assume $\mathbf{h}^{(1)}$ as input. Train to find $\mathbf{W}^{(2)}$.
- Step 3: Continue to stack layers and find weights $\mathbf{W}^{(3)}$, etc.



Step 1



Step 2

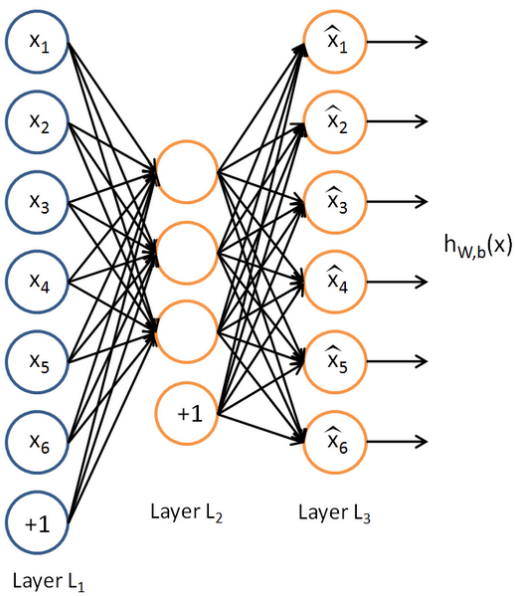


Step 3

Deep neural networks: Unsupervised Pretraining

Autencoder

- Unsupervised algorithm that tries to learn an approximation of the identity function $h_{\mathbf{W},\mathbf{b}}(x) \approx \mathbf{x}$
- Trivial problem unless we place constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data
  e.g. if some of the input features are correlated, then this algorithm will be able to discover some of those correlations

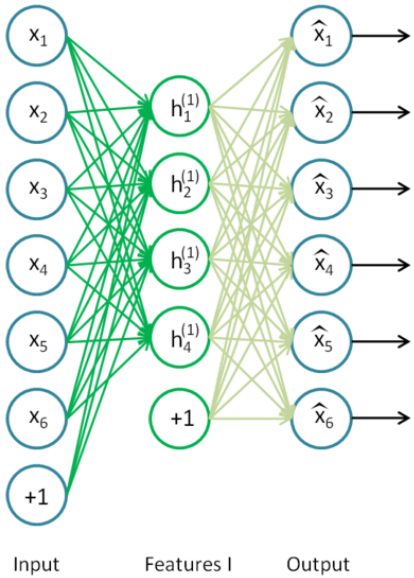


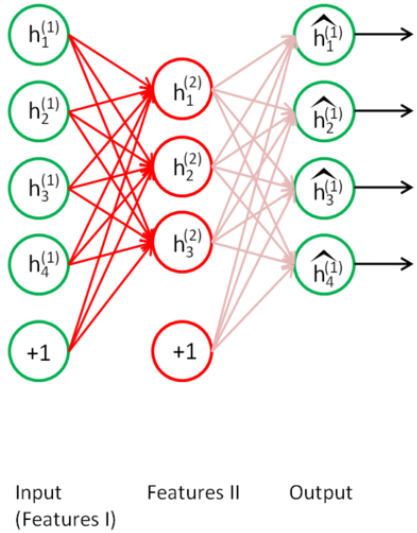
- $\alpha_j^{(i)} = f(W_{i1}^{(1)} x_1 + W_{i2}^{(1)} x_2 + \ldots + b_i^{(1)})$
- Trained using back-propagation and additional sparsity constraints
- Can be also used for feature transformation

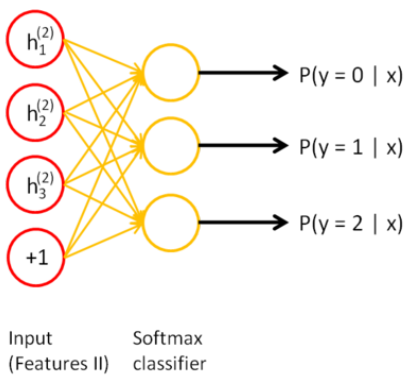
[http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity]

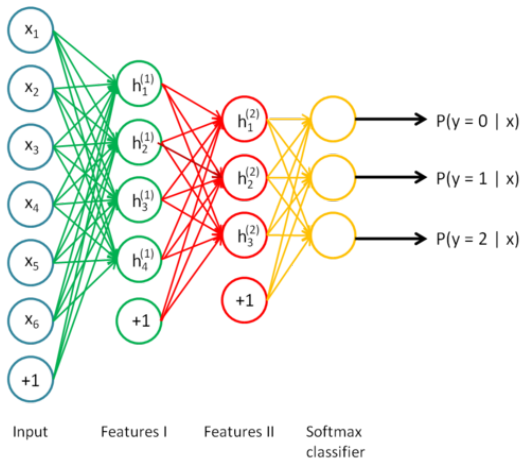
Deep neural networks: Unsupervised Pretraining

Stacked Autencoders for pretraining









Deep neural networks: Unsupervised Pretraining

Stacked Autencoders for pretraining

- Capture a "hierarchical grouping" of the input
- First layer learns a good representation of input features (e.g. edges)
- Second layer learns a good representation of the patterns in the first layer (e.g. corners), etc.

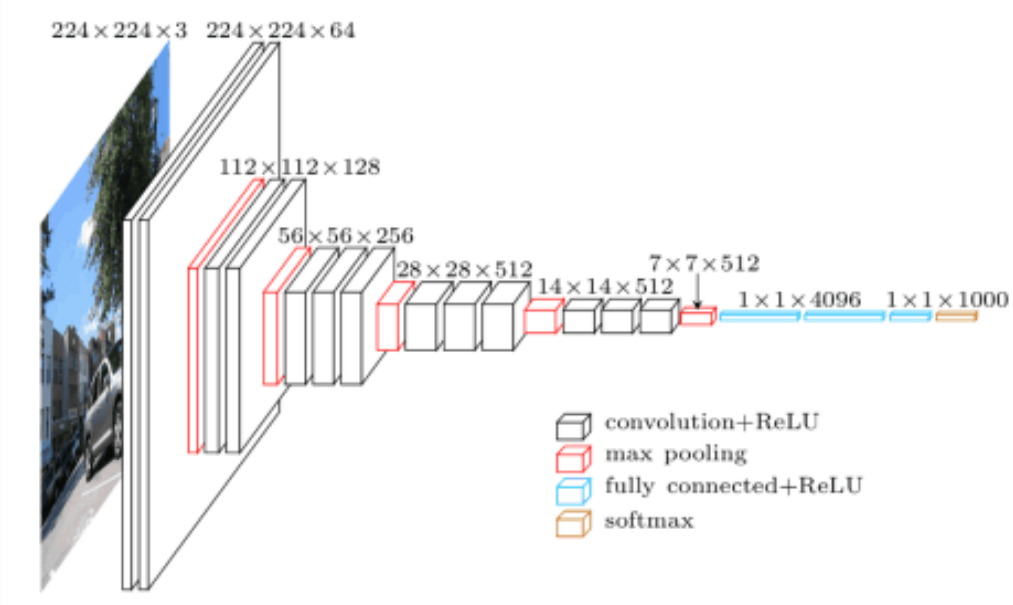
`http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders`

Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

Deep neural networks: Fine-tuning

- Taking advantage of labelled data from large (publicly available) datasets, e.g., VGG16
- Tweak the parameters of an already trained network so that it adapts to the new task at hand
- Initial layers $\rightarrow$ learn general features
- Last layers $\rightarrow$ learn features more specific to the task of interest
- Fine-tuning freezes the first layers, and relearns weights from the last



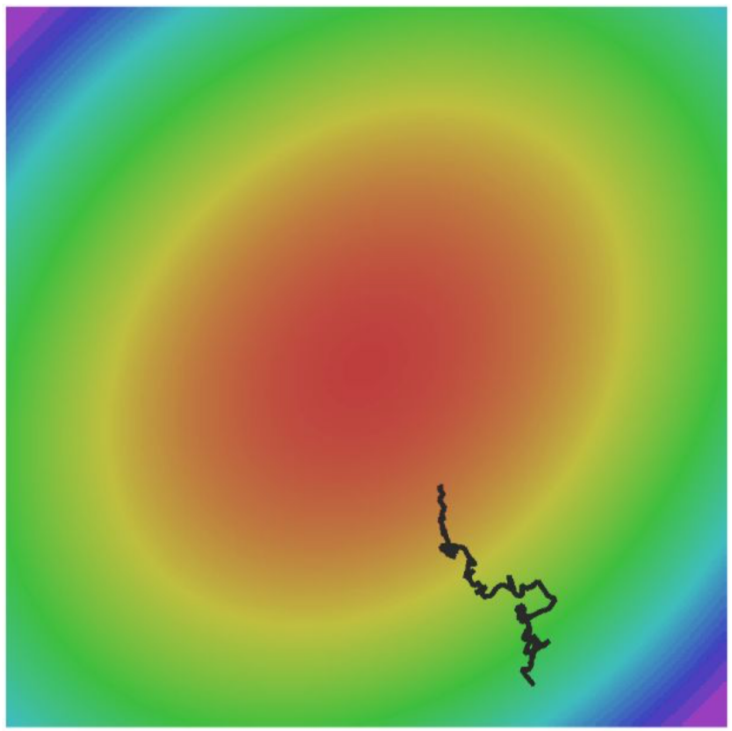
Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

Alternative optimization methods

Problems with stochastic gradient descent

- Gradient becomes zero as we increase the # layers
- Local optima and saddle points become more common in high dimensions



Alternative optimization methods

Stochastic gradient descent + Momentum

- Movement through the parameter space is averaged over multiple time steps
- Momentum speeds up movement along directions of strong improvement (loss decrease) and also helps the network avoid local minima

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

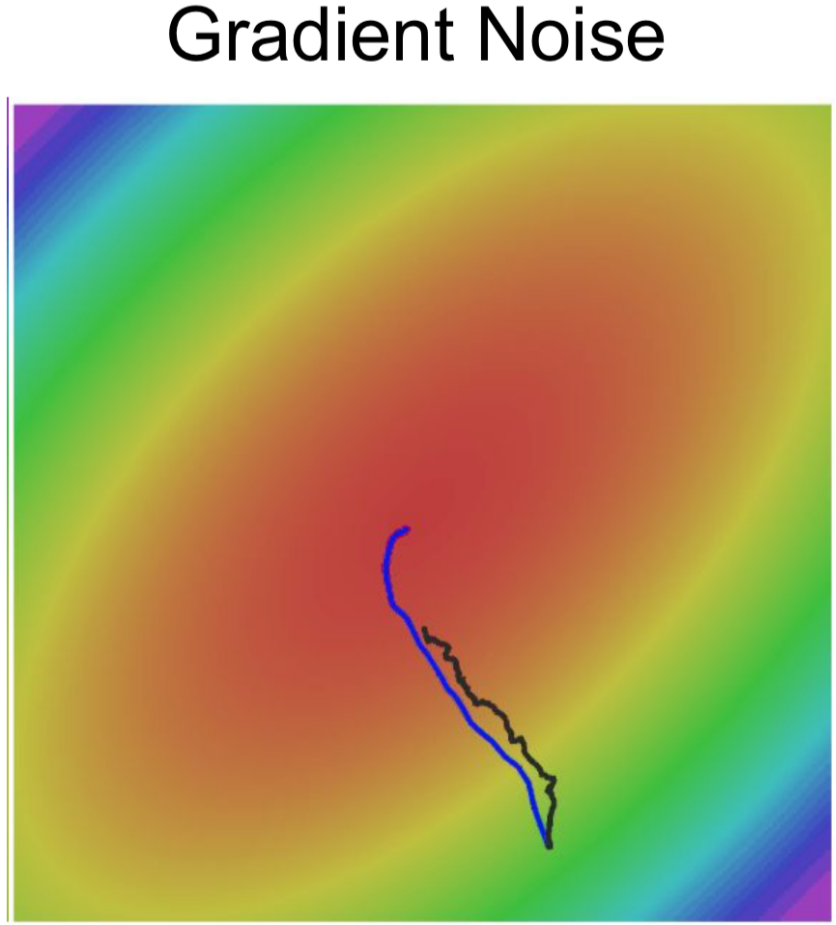
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Alternative optimization methods

Stochastic gradient descent + Momentum

Issue with noisy trajectories that diverge from optima



Alternative optimization methods

Stochastic gradient descent + Nesterov Momentum

- Gradient term is not computed from current parameter position x_t
- Gradient term is computed using the current position and momentum $x_t + \rho v_t$
- While the gradient term always points in the right direction, the momentum term may not
- If the momentum term points in the wrong direction or overshoots, the gradient can still "go back" and correct it in the same update step.

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

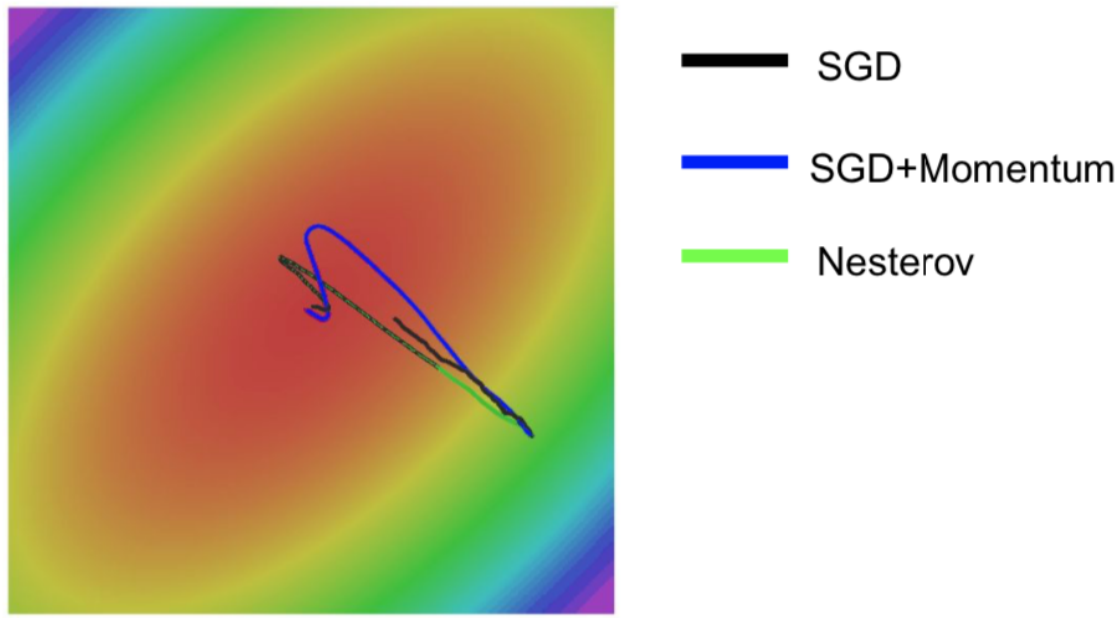
Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$
$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1}$$
$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

Alternative optimization methods

Stochastic gradient descent + Nesterov Momentum



Alternative optimization methods

AdaGrad & RMSProp

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

AdaGrad

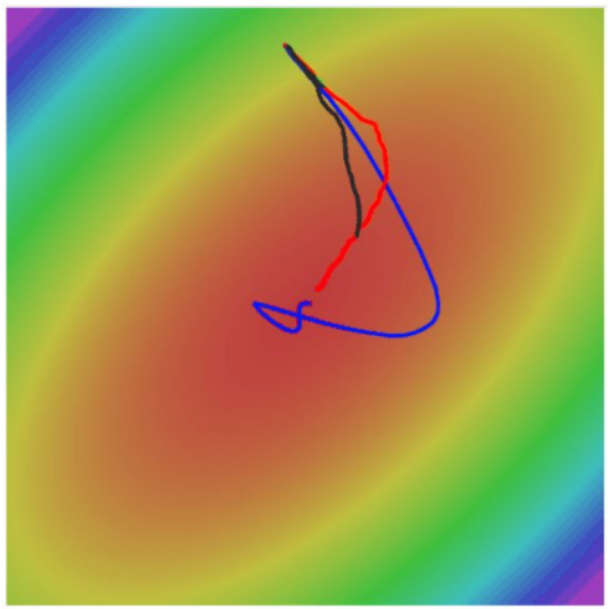
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Alternative optimization methods

AdaGrad & RMSProp



Alternative optimization methods

Adam

Combination of RMSProp and Momentum

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

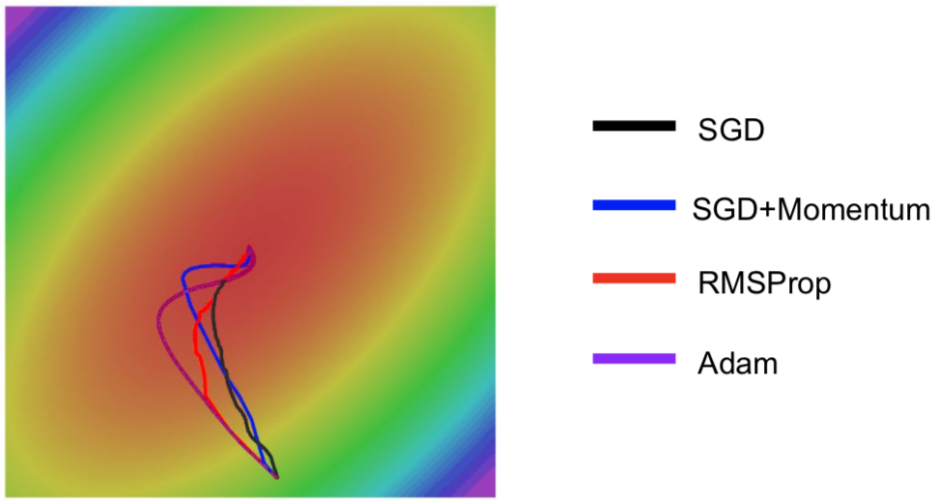
Momentum

AdaGrad / RMSProp

Alternative optimization methods

Adam

Issue with noisy trajectories that diverge from optima



Alternative optimization methods

Stochastic gradient descent + Nesterov Momentum

- Gradient term is not computed from current parameter position x_t
- Gradient term is computed using the current position and momentum $x_t + \rho v_t$
- While the gradient term always points in the right direction, the momentum term may not
- If the momentum term points in the wrong direction or overshoots, the gradient can still "go back" and correct it in the same update step.

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

Annoying, usually we want update in terms of $x_t, \nabla f(x_t)$

Change of variables $\tilde{x}_t = x_t + \rho v_t$ and rearrange:

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$
$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1}$$
$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

Alternative optimization methods

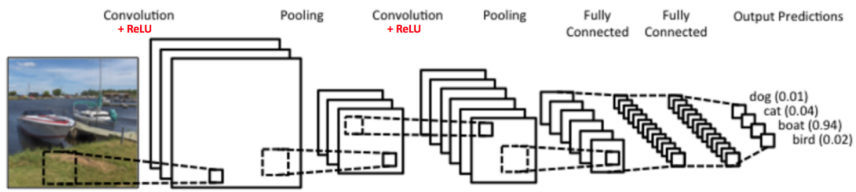
- Adam is a good default choice
- A more informed selection of the optimization method can be done through hyper-parameter tuning

Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

Convolutional neural networks

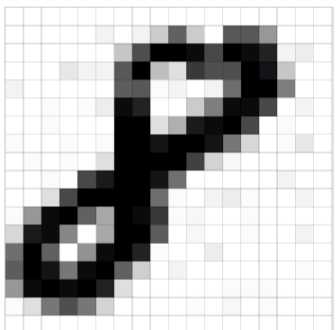
- Similar to regular neural networks
  - made up of neurons, each with an input and an activation function
  - have weights and biases to be learned
  - have a loss function on the last (fully-connected) layer
- Explicit assumption that the inputs are images
  - vastly reduce the amount of parameters in the network

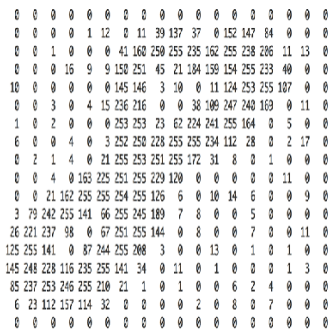


Convolutional neural networks

Image representation

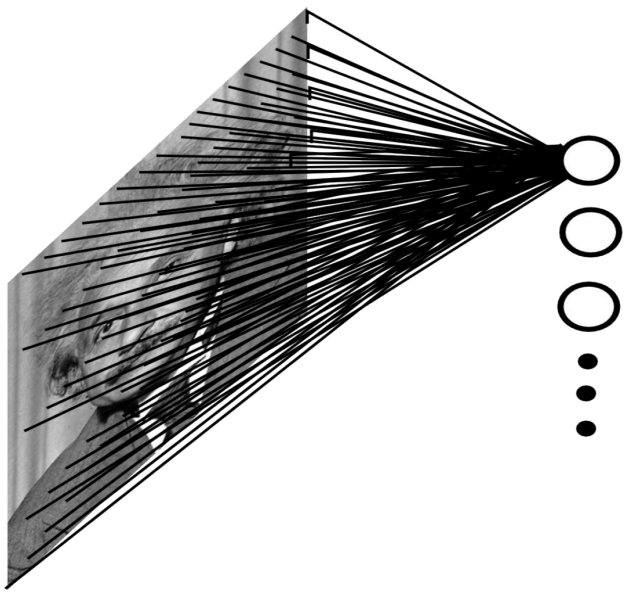
- Grayscale image (1-channel)
  - 2d-matrix
  - each pixel ranges from 0 to 255 - 0: black, 255: white
- Color image (3-channel, RGB)
  - three 2d-matrices stacked over each other
  - each with pixel values ranging between 0 and 255





Convolutional neural networks

A **fully** connected neural network with image input

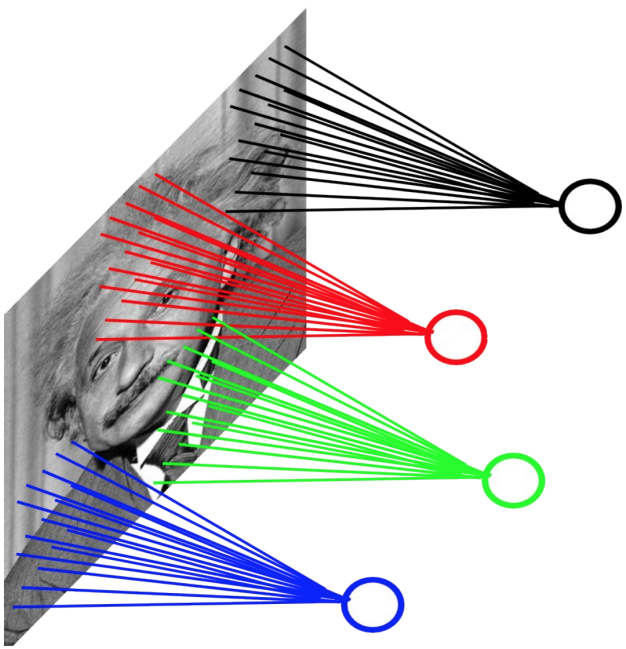


- 1000×1000 image, 1M hidden units $\rightarrow 10^{12}$ parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional neural networks

Idea 1: Convolution

A **locally** connected neural network with image input

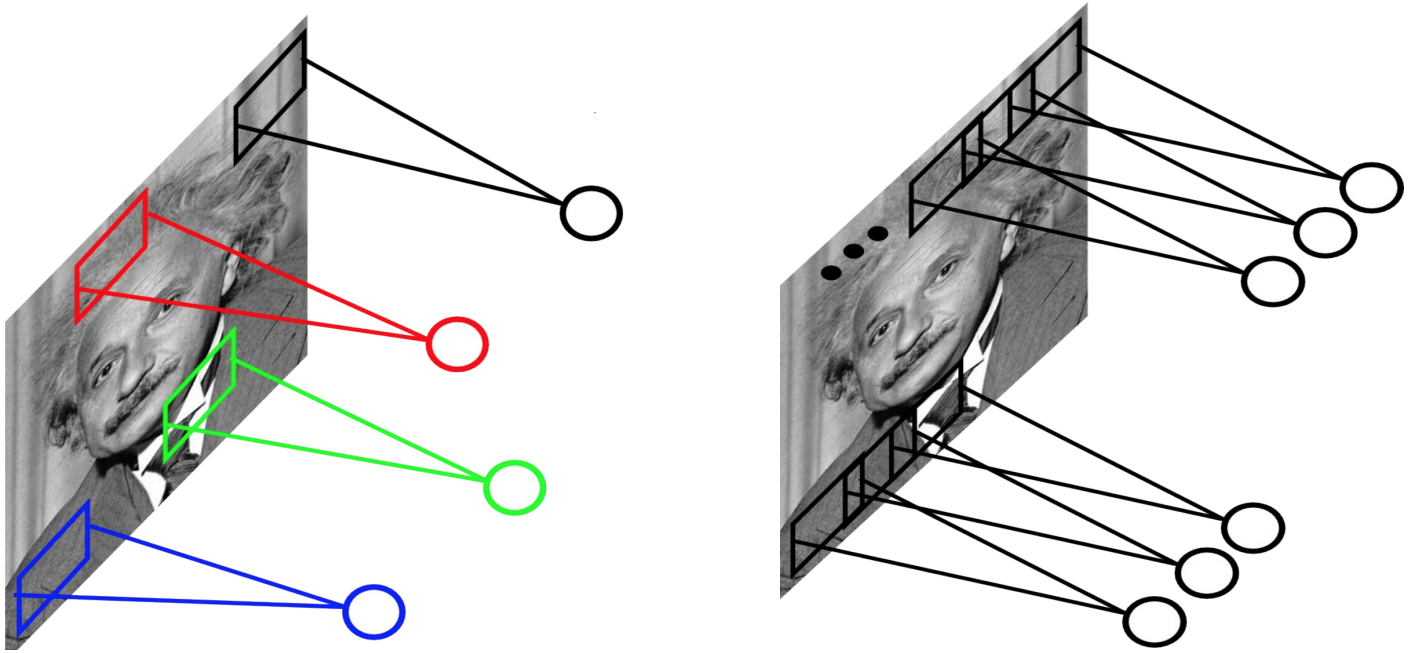


- 1000×1000 image, 1M hidden units, 10×10 filter size $\rightarrow$ 10^8 parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional neural networks

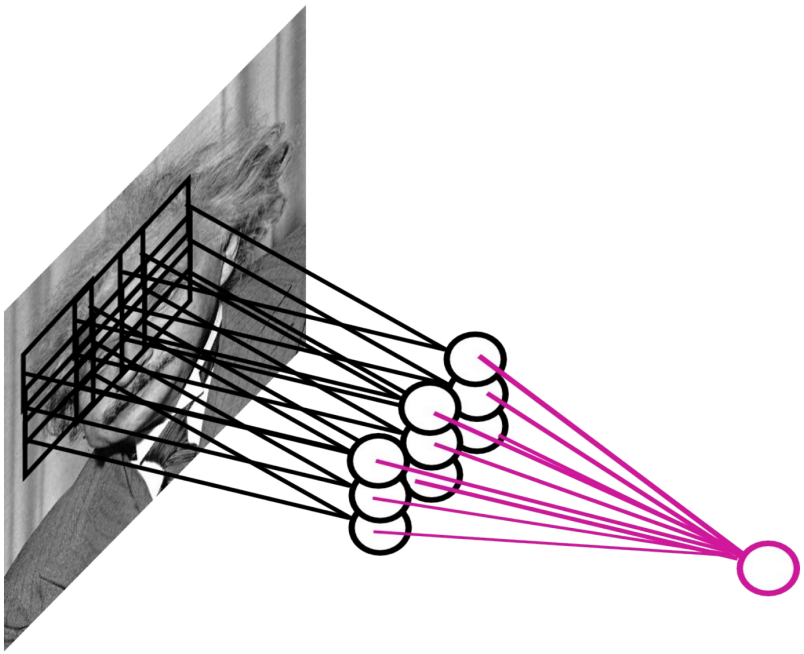
Idea 2: Weight sharing

- Stationarity: Statistics are similar at different locations
- Share the same parameters across different locations



Convolutional neural networks

Idea 3: Max-pooling



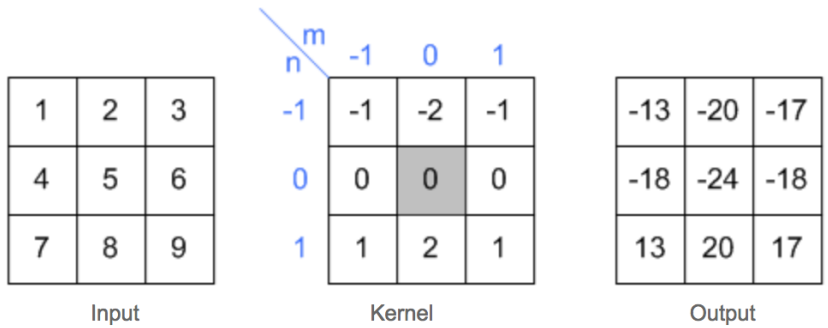
- Let us assume filter is an "eye" detector
- How can we make the detection robust to the exact location of the eye?
- By pooling (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features

Convolutional neural networks: The convolution operation

Convolution 2d-example

- Convolution is the mathematical operation that implements filtering
- Given an input image $x[m, n]$ and an impulse response $h[m, n]$ (filter or kernel), the convolution output can be written as

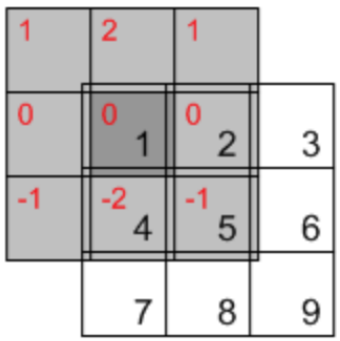
$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] h[m - i, n - j]$$



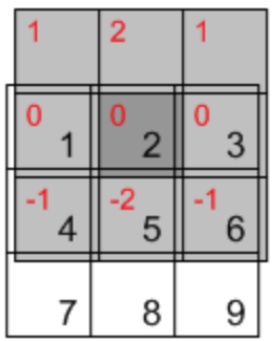
http://www.songho.ca/dsp/convolution/convolution2d_example.html

Convolutional neural networks: The convolution operation

Convolution 2d-example



$$
\begin{aligned}
y[0,0] = {} & x[-1,-1]\cdot h[1,1] + x[0,-1]\cdot h[0,1] + x[1,-1]\cdot h[-1,1] \\
& + x[-1,0]\cdot h[1,0] + x[0,0]\cdot h[0,0] + x[1,0]\cdot h[-1,0] \\
& + x[-1,1]\cdot h[1,-1] + x[0,1]\cdot h[0,-1] + x[1,1]\cdot h[-1,-1] \\
= {} & 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 0\cdot 0 + 1\cdot 0 + 2\cdot 0 + 0\cdot(-1) + 4\cdot(-2) + 5\cdot(-1) = -13
\end{aligned}
$$

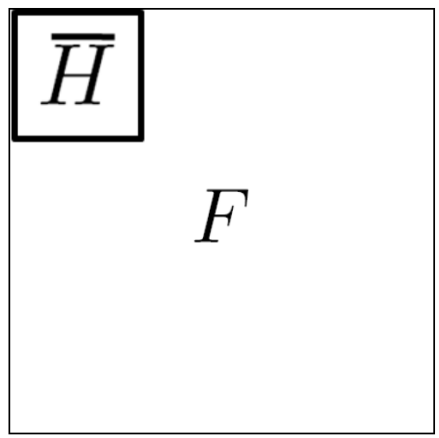


$$
\begin{aligned}
y[1,0] = {} & x[0,-1]\cdot h[1,1] + x[1,-1]\cdot h[0,1] + x[2,-1]\cdot h[-1,1] \\
& + x[0,0]\cdot h[1,0] + x[1,0]\cdot h[0,0] + x[2,0]\cdot h[-1,0] \\
& + x[0,1]\cdot h[1,-1] + x[1,1]\cdot h[0,-1] + x[2,1]\cdot h[-1,-1] \\
= {} & 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 1\cdot 0 + 2\cdot 0 + 3\cdot 0 + 4\cdot(-1) + 5\cdot(-2) + 6\cdot(-1) = -20
\end{aligned}
$$

http://www.songho.ca/dsp/convolution/convolution2d_example.html

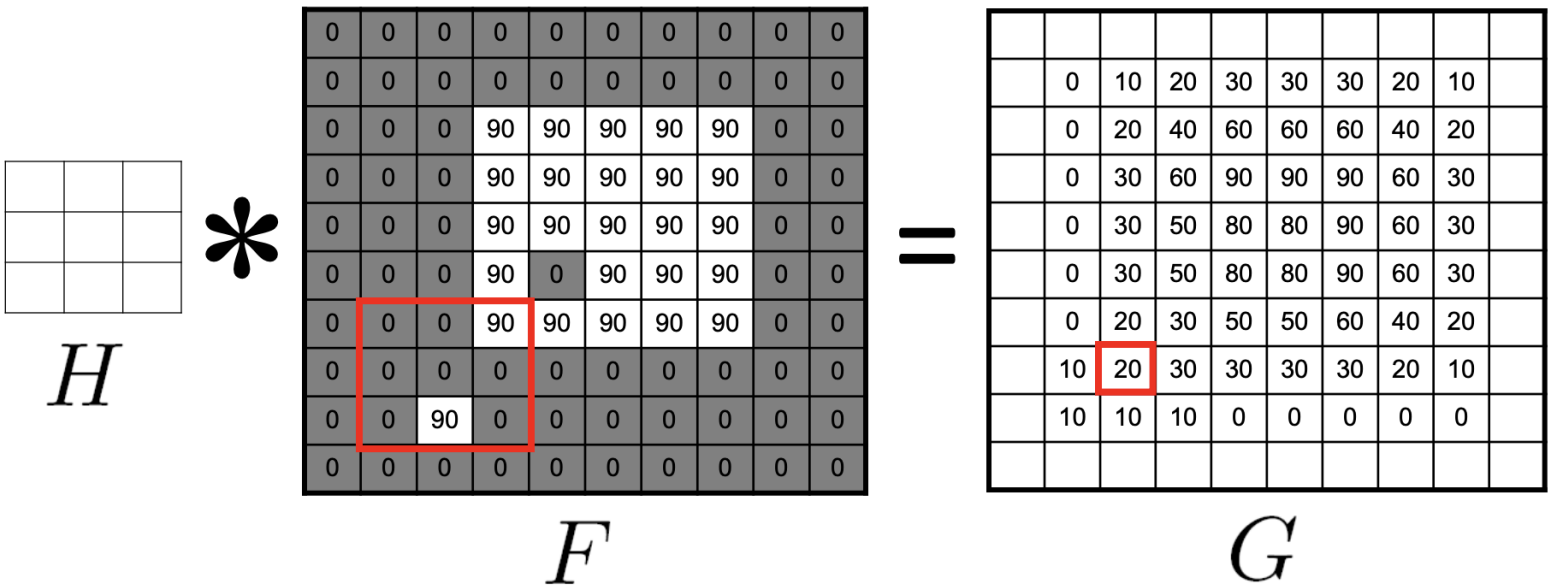
Convolutional neural networks: The convolution operation

Convolution operation



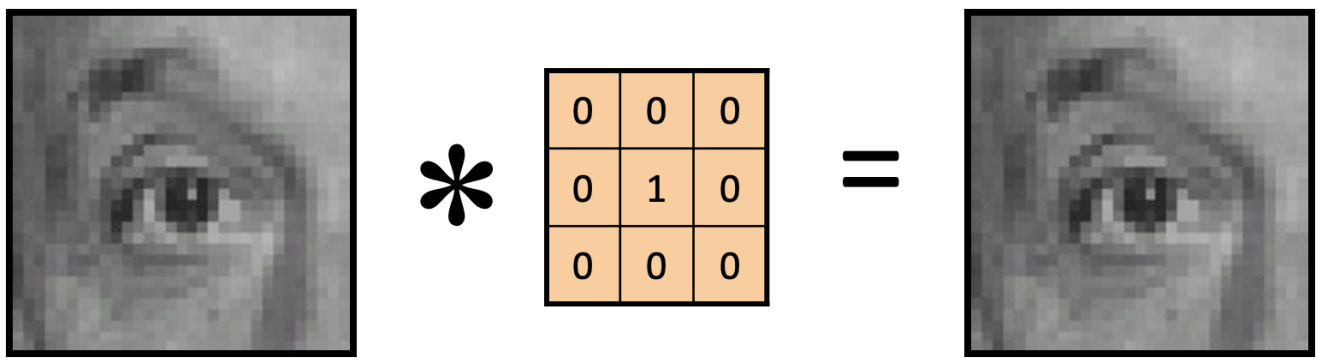
Convolutional neural networks: The convolution operation

Mean filtering



Convolutional neural networks: The convolution operation

Linear filters: Example

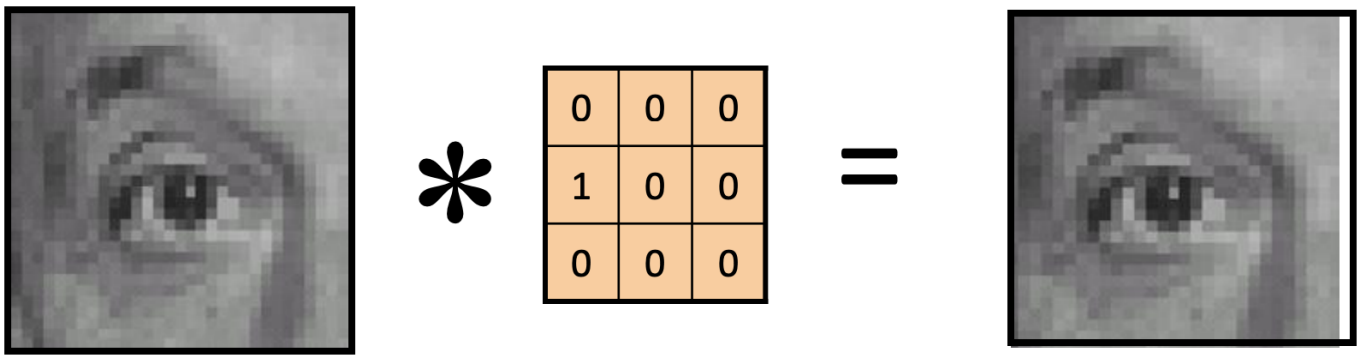


Original

Identical image

Convolutional neural networks: The convolution operation

Linear filters: Example

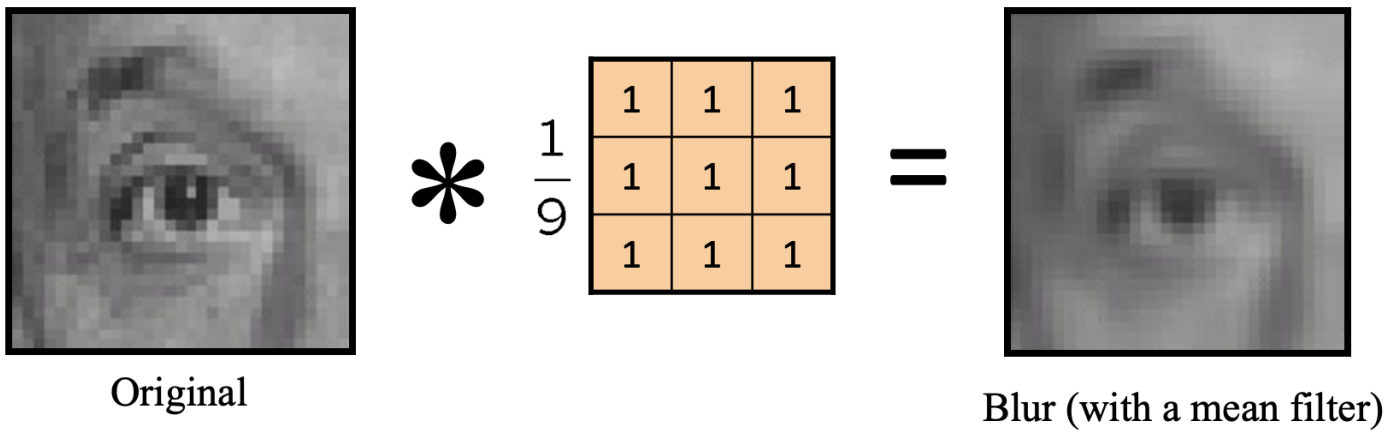


Original

Shifted left
By 1 pixel

Convolutional neural networks: The convolution operation

Linear filters: Example

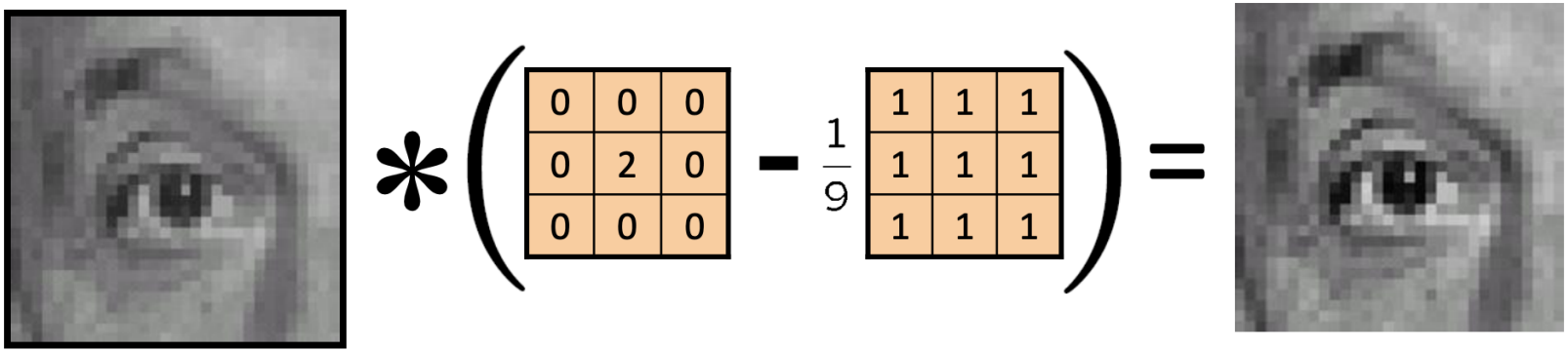


Original

Blur (with a mean filter)

Convolutional neural networks: The convolution operation

Linear filters: Example

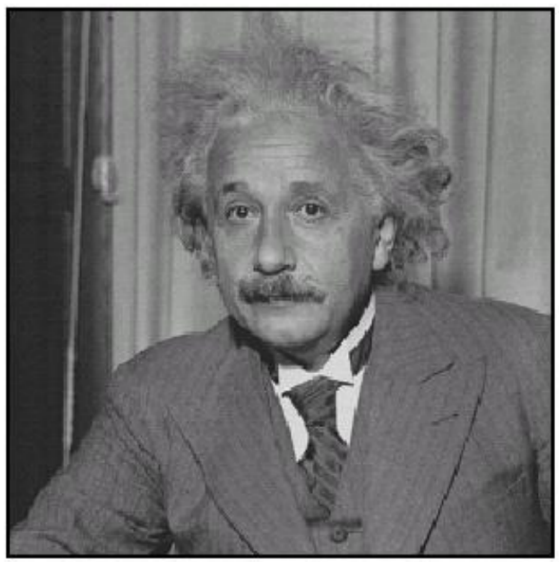


Original

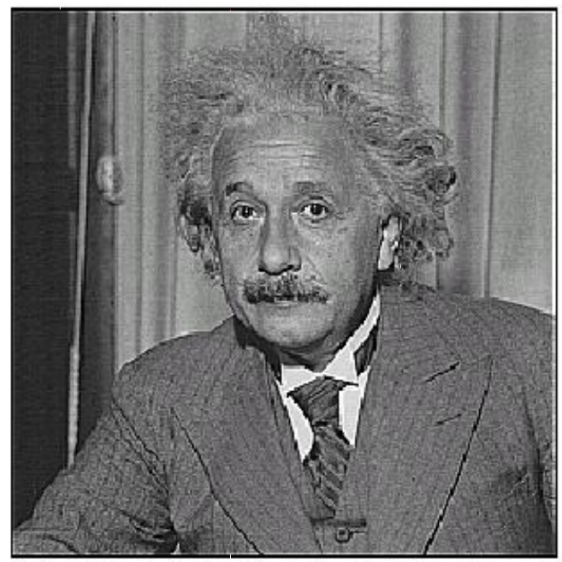
Sharpening filter
(accentuates edges)

Convolutional neural networks: The convolution operation

Linear filters: Example



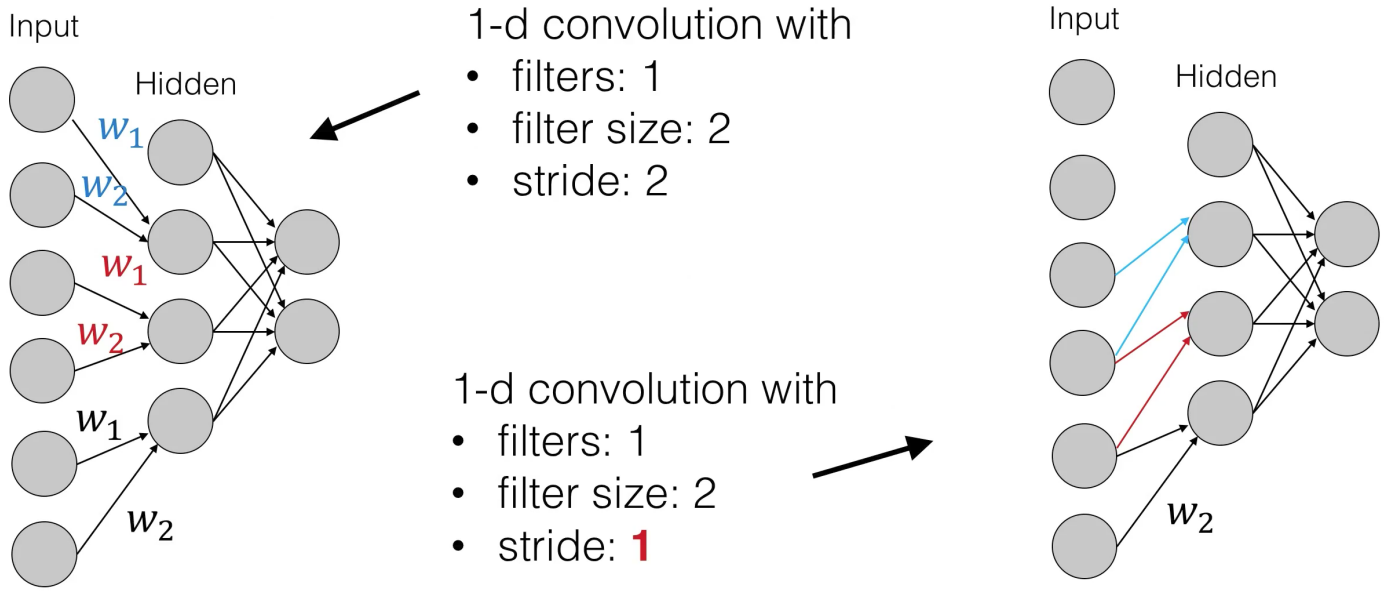
before



after

Convolutional neural networks: The convolution operation

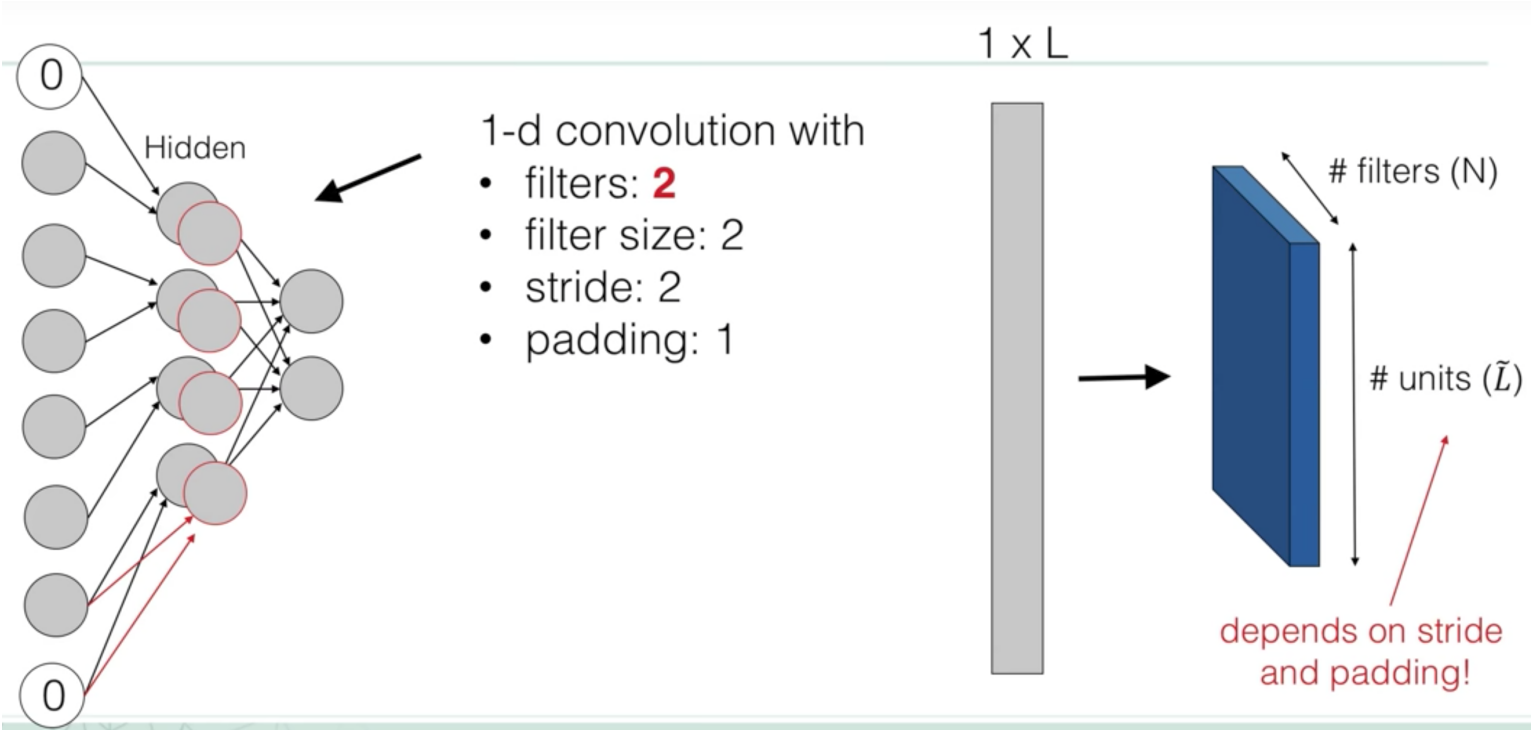
Image 1d-convolution



https://www.nervanasys.com/convolutional-neural-networks/

Convolutional neural networks: The convolution operation

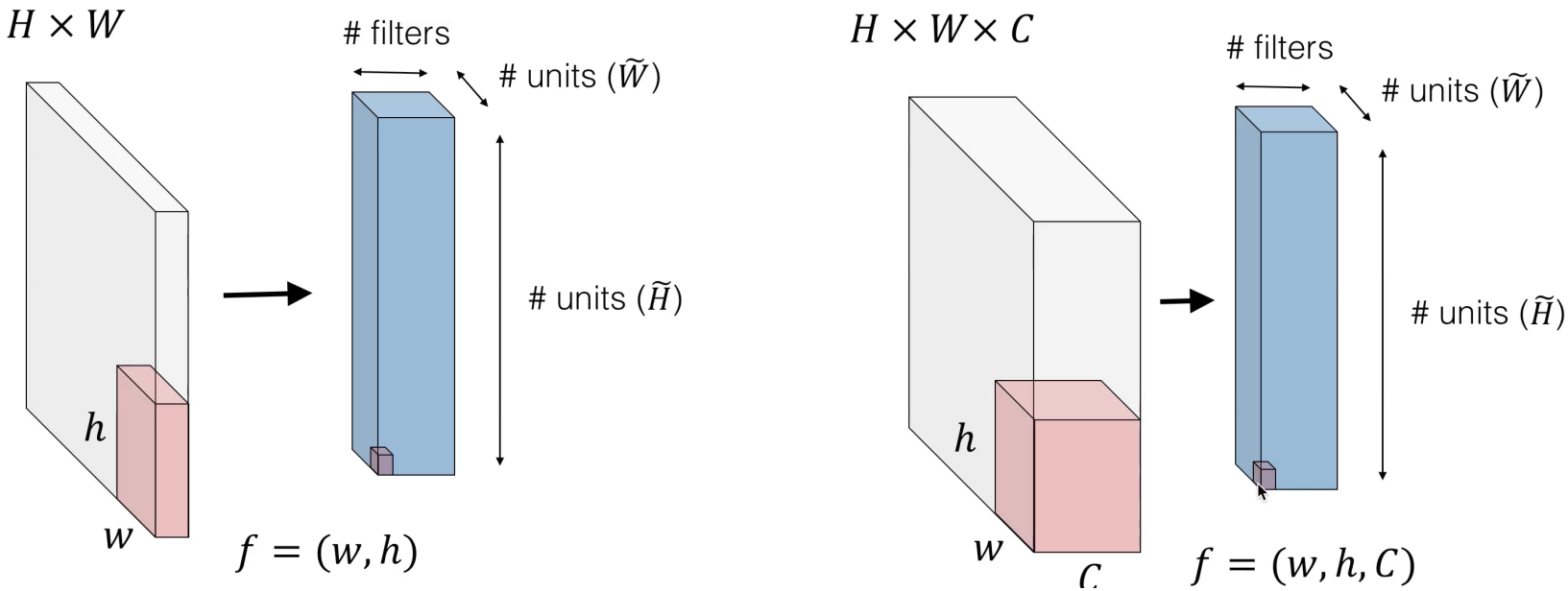
Image 1d-convolution



https://www.nervanasys.com/convolutional-neural-networks/

Convolutional neural networks: The convolution operation

Image 2d-convolution



https://www.nervanasys.com/convolutional-neural-networks/

Also check:

http://cs231n.github.io/assets/conv-demo/index.html

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (figure 6)

Convolutional neural networks: The convolution operation

Image 2d-convolution hyperparameters

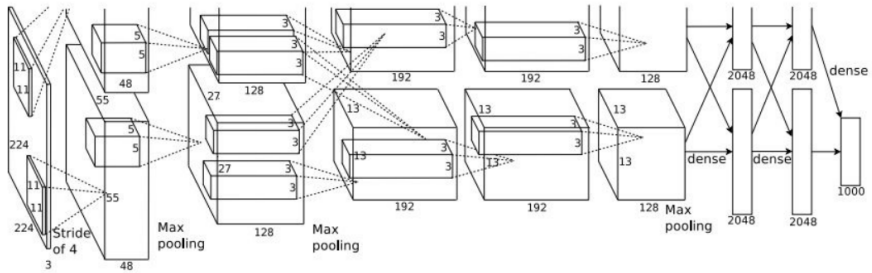
- Depth: the number of filters we use for the convolution operation
- Stride: the number of pixels by which we slide our filter matrix over the input
- Zero-padding: padding the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

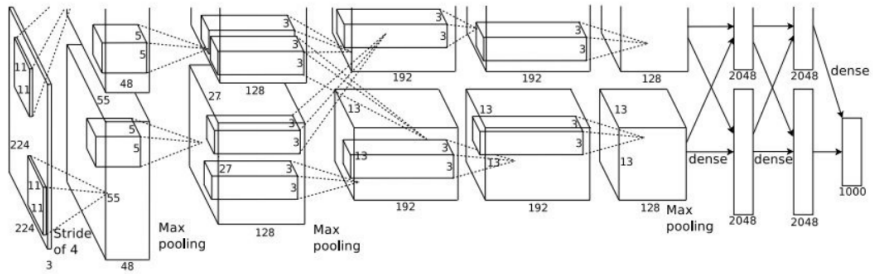


Assuming no zero-padding and weight sharing throughout the entire image

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



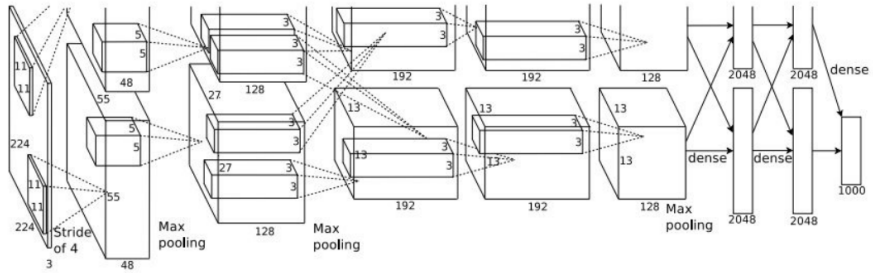
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

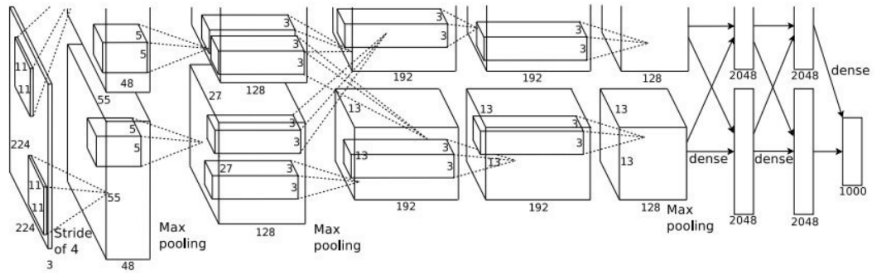
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



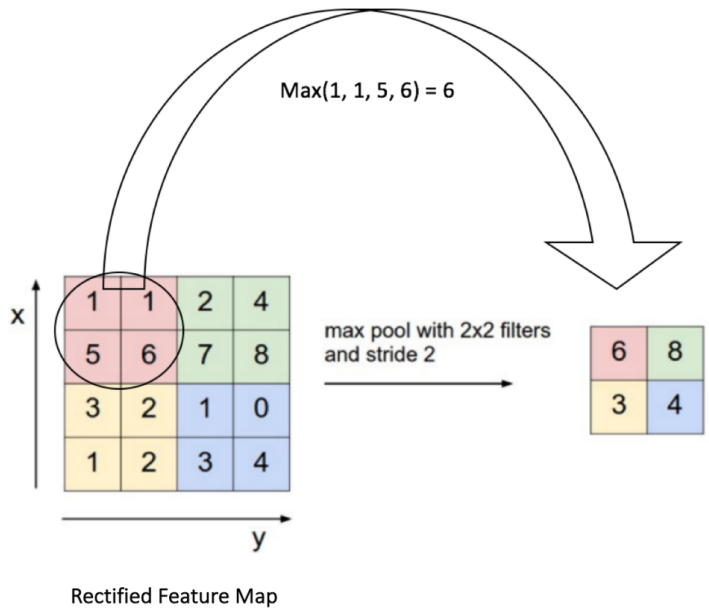
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Convolutional neural networks: Max-pooling

Spatial Pooling (also called subsampling or downsampling)

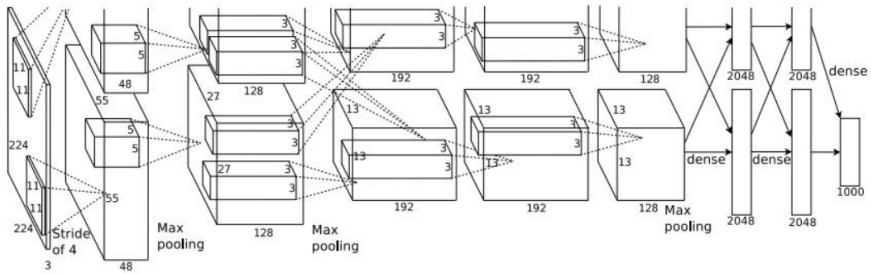
- Reduces the dimensionality of each feature map but retains the most important information
- Can be of different types: Max, Average, Sum etc.
- Makes the input representations (feature dimension) smaller and more manageable
- Promotes an almost scale invariant representation of the image



Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

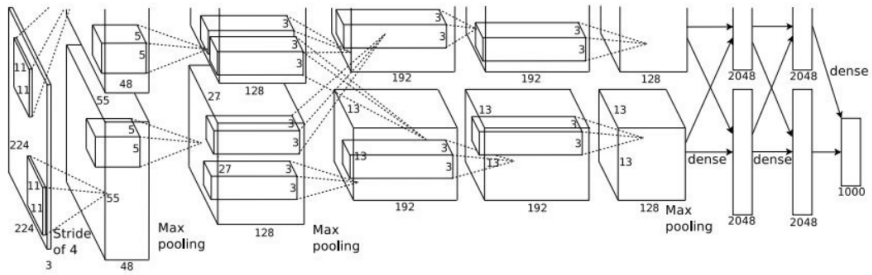
Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96

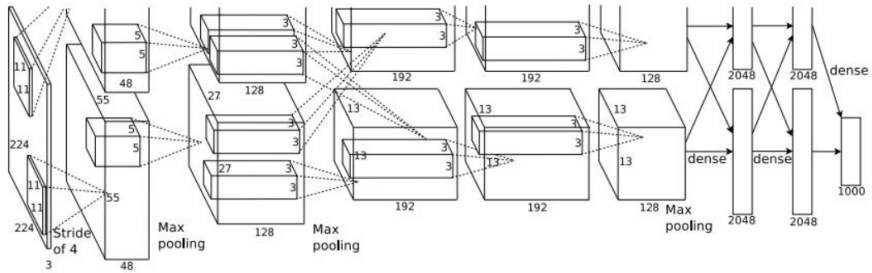
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Convolutional neural networks: Example

Case Study: AlexNet

[Krizhevsky et al. 2012]



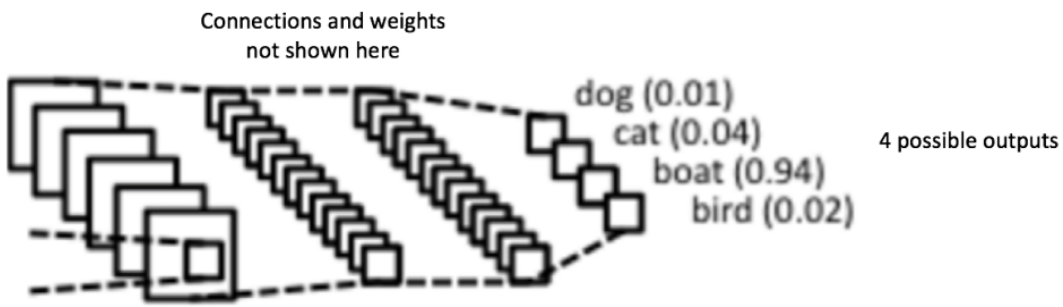
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Convolutional neural networks: Final fully connected layer

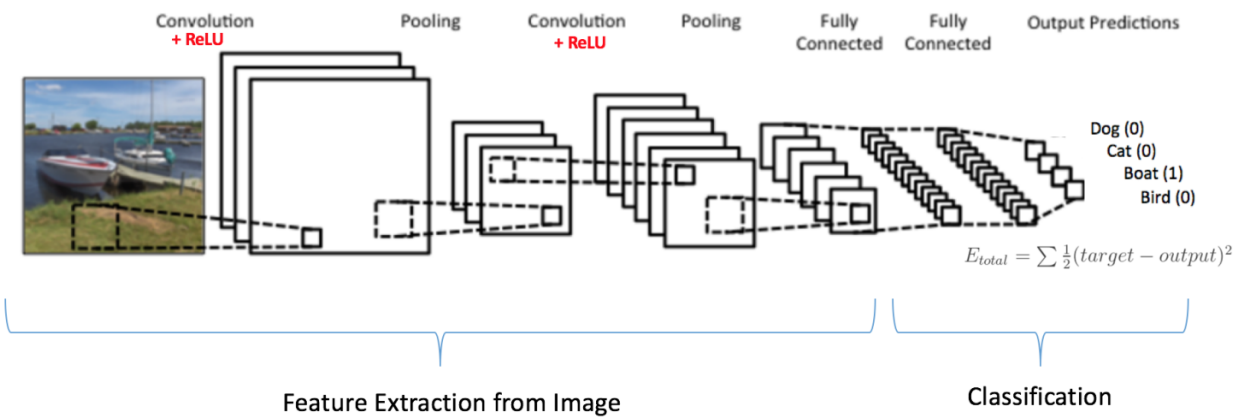
Final fully connected layer

- Traditional multilayer perceptron
- Yields the classification/regression result

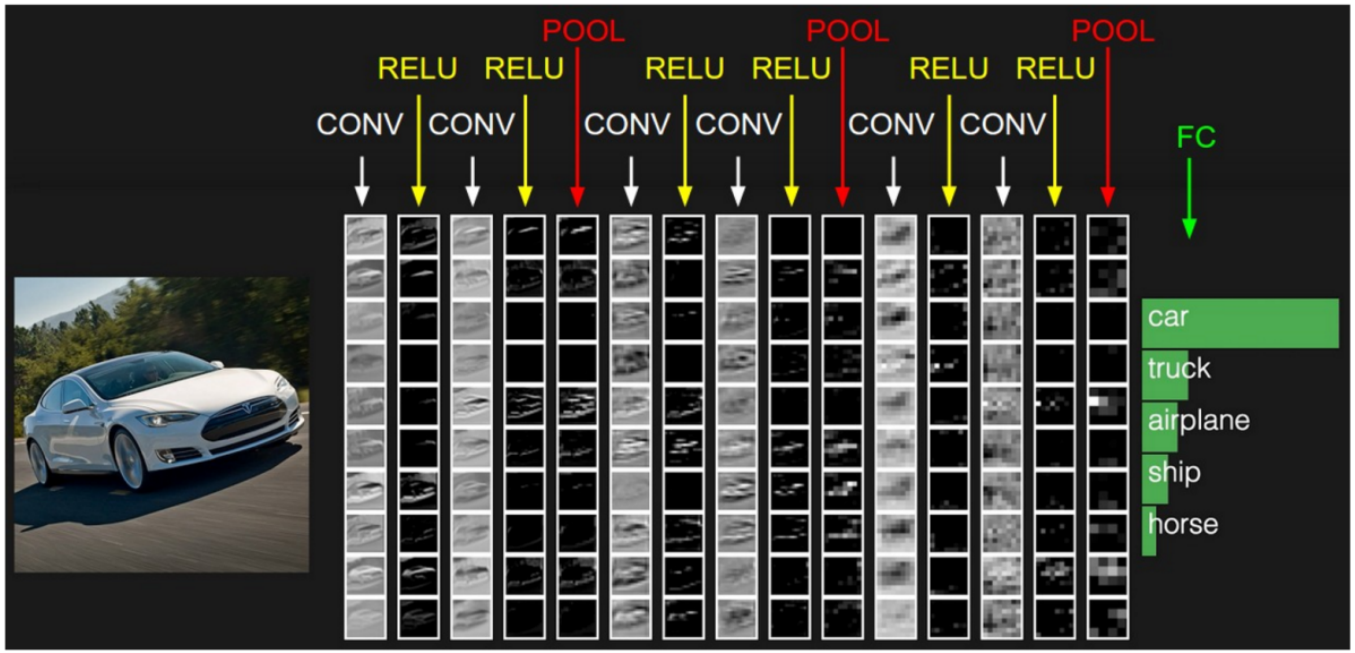


Convolutional neural networks: Putting it all together

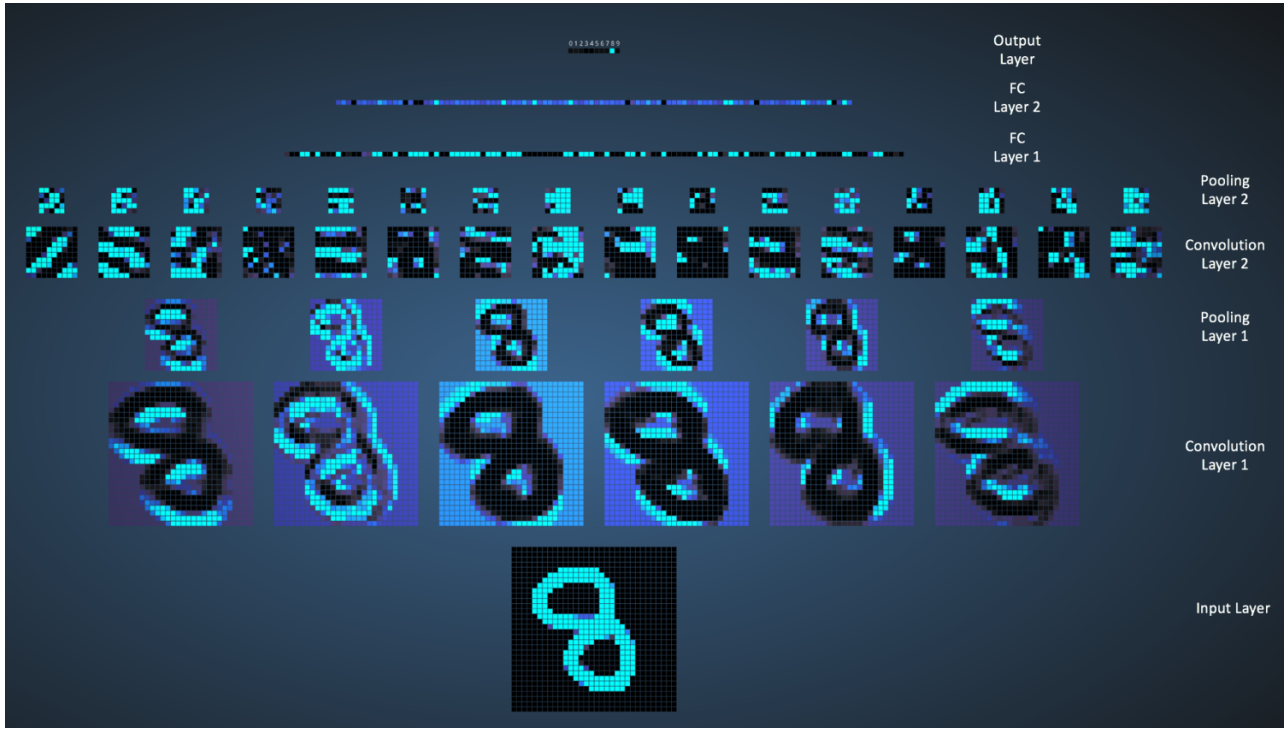
- **Step 1:** Initialize weights
- **Step 2:** Take first image as input and go through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the fully connected layer) and finds the output probabilities for each class
- **Step 3:** Calculate the total error at the output layer
- **Step 4:** Use backpropagation to update the weights, which are adjusted in proportion to their contribution to the total error
- **Step 5:** Repeat Steps 1-4 for all train images



Convolutional neural networks: Examples



Convolutional neural networks: Examples



Convolutional neural networks: Hyperparameter tuning

- Learning rate: how much to update the weight during optimization
- Number of epochs: number of times the entire training set pass through the neural network
- Batch size: the number of samples in the training set for weight update
- Activation function: the function that introduces non-linearity to the model (e.g. sigmoid, tanh, ReLU, etc.)
- Number of hidden layers and units
- Weight initialization: e.g., uniform distribution
- Dropout for regularization: probability of dropping a unit
- Optimization method: optimization method to learn the weights (e.g., Adam, RMSProp)

We can perform grid or randomized search over all parameters

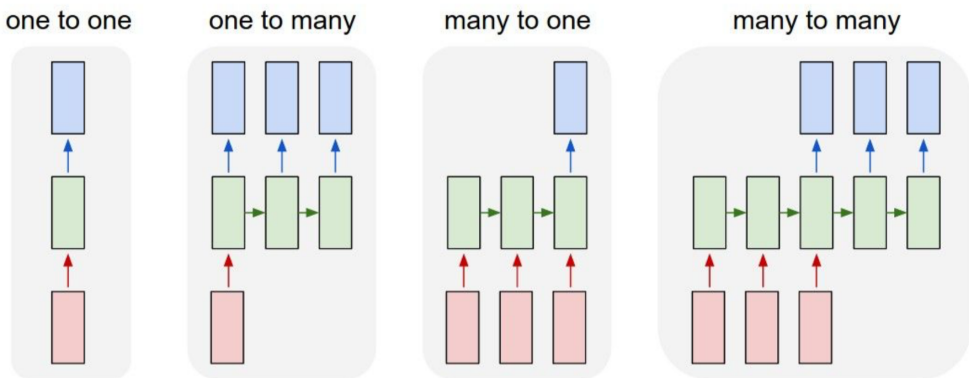
Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

[The content for the following slides have been summarized from Li, Johnson, & Yeung, Stanford CSCE 231]

Recurrent neural networks: Motivation

- Networks with feedback loops (recurrent edges)
- Output at current time step depends on current input as well as previous state (via recurrent edges)



- one-to-one: e.g., image classification (image $\rightarrow$ user ID)
- one-to-many: e.g., image captioning (image $\rightarrow$ sequence of words)
- many-to-one: e.g., sentiment classification (sequence of words $\rightarrow$ emotion)
- many-to-many: e.g., machine translation (e.g., sequence of words $\rightarrow$ sequence of words)

Recurrent neural networks: Representation

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

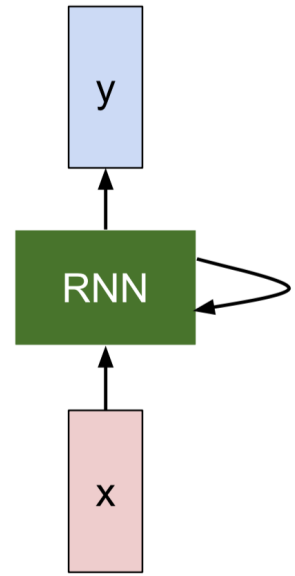
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

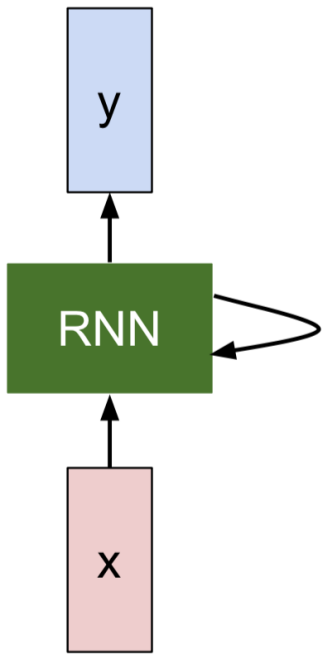
some function with parameters W



The same function and the same set of parameters are used at every time step

Recurrent neural networks: Representation

The state consists of a single *"hidden"* vector **h**:



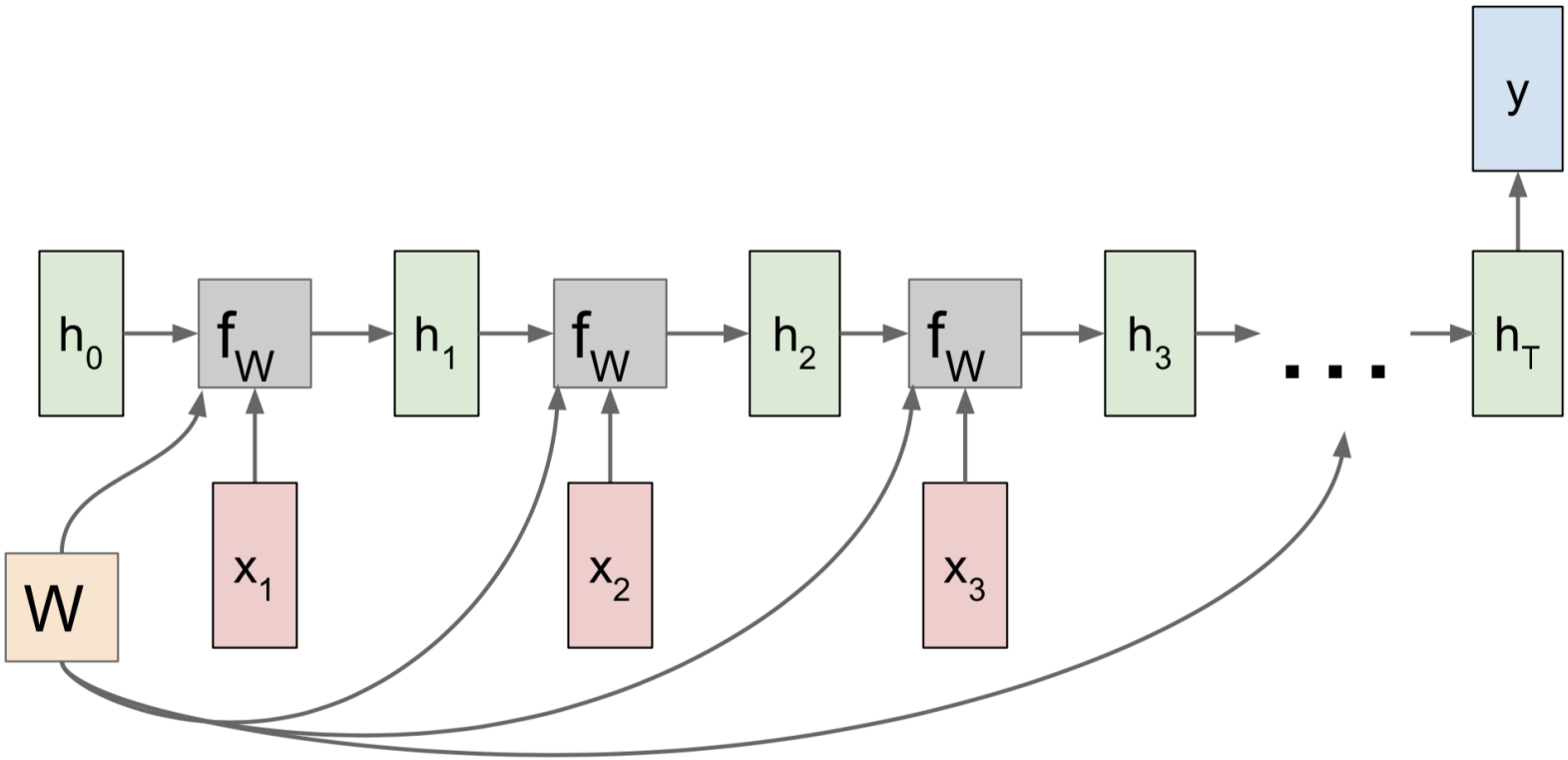
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Recurrent neural networks: Representation

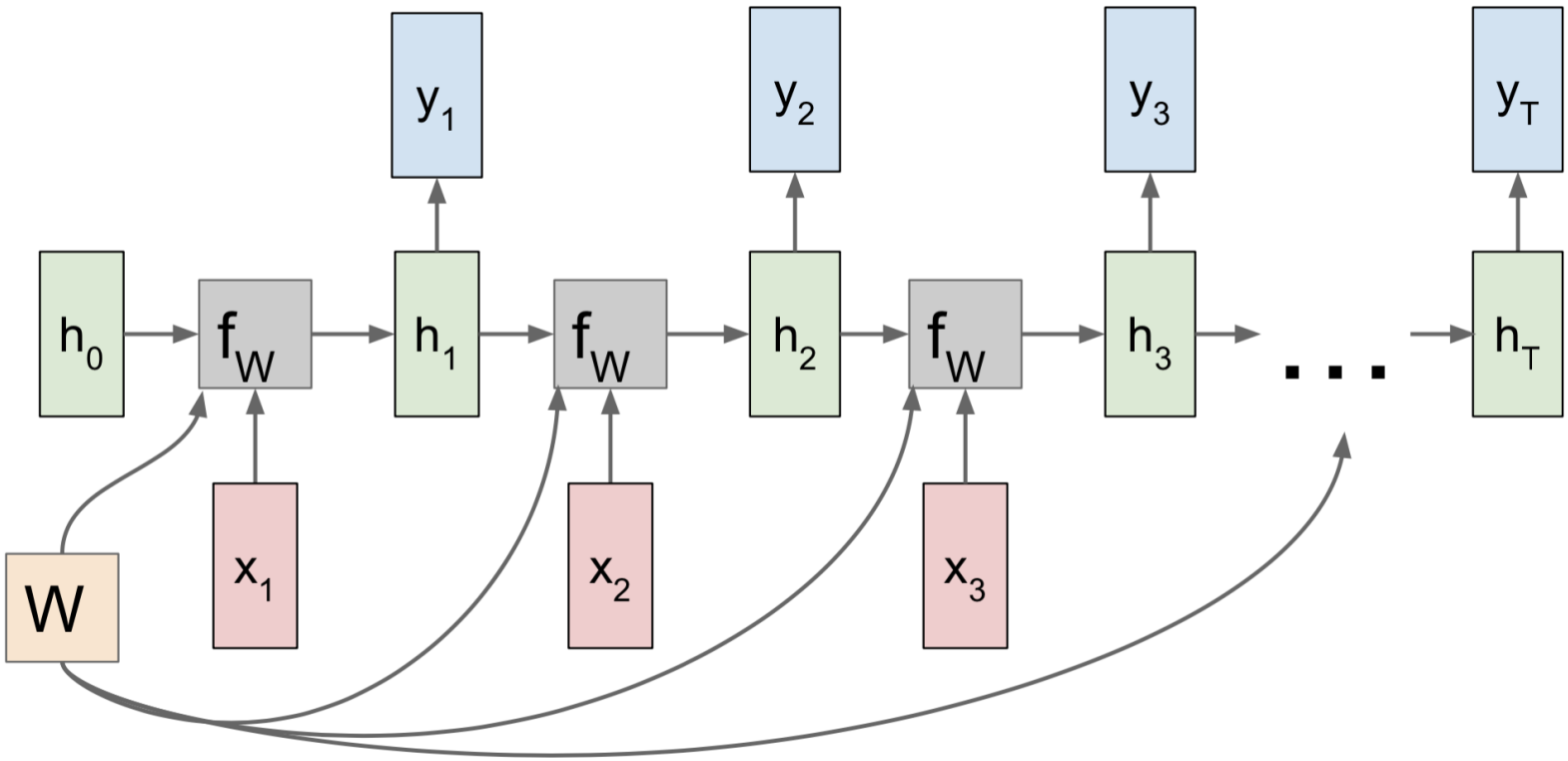
Many-to-one representation



The same function and the same set of parameters are used at every time step.

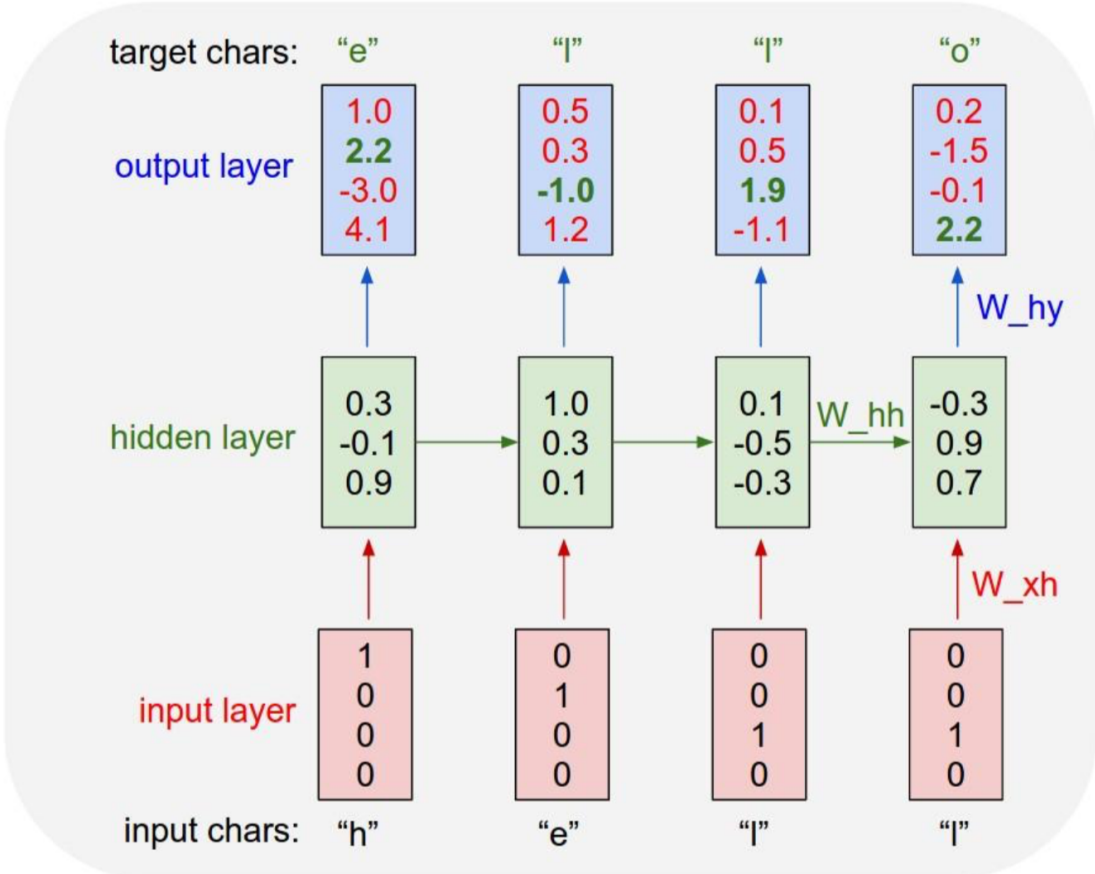
Recurrent neural networks: Representation

Many-to-many representation



Recurrent neural networks: Representation

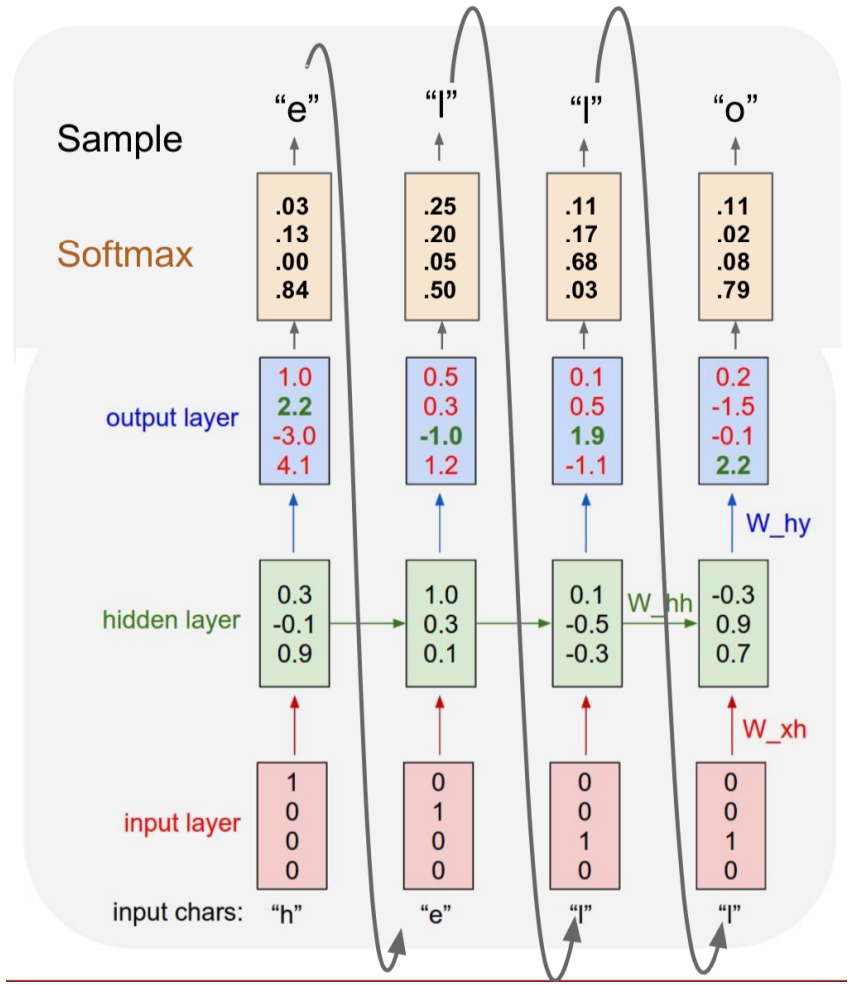
Example: Character-level language model
During training: learning sequence of characters



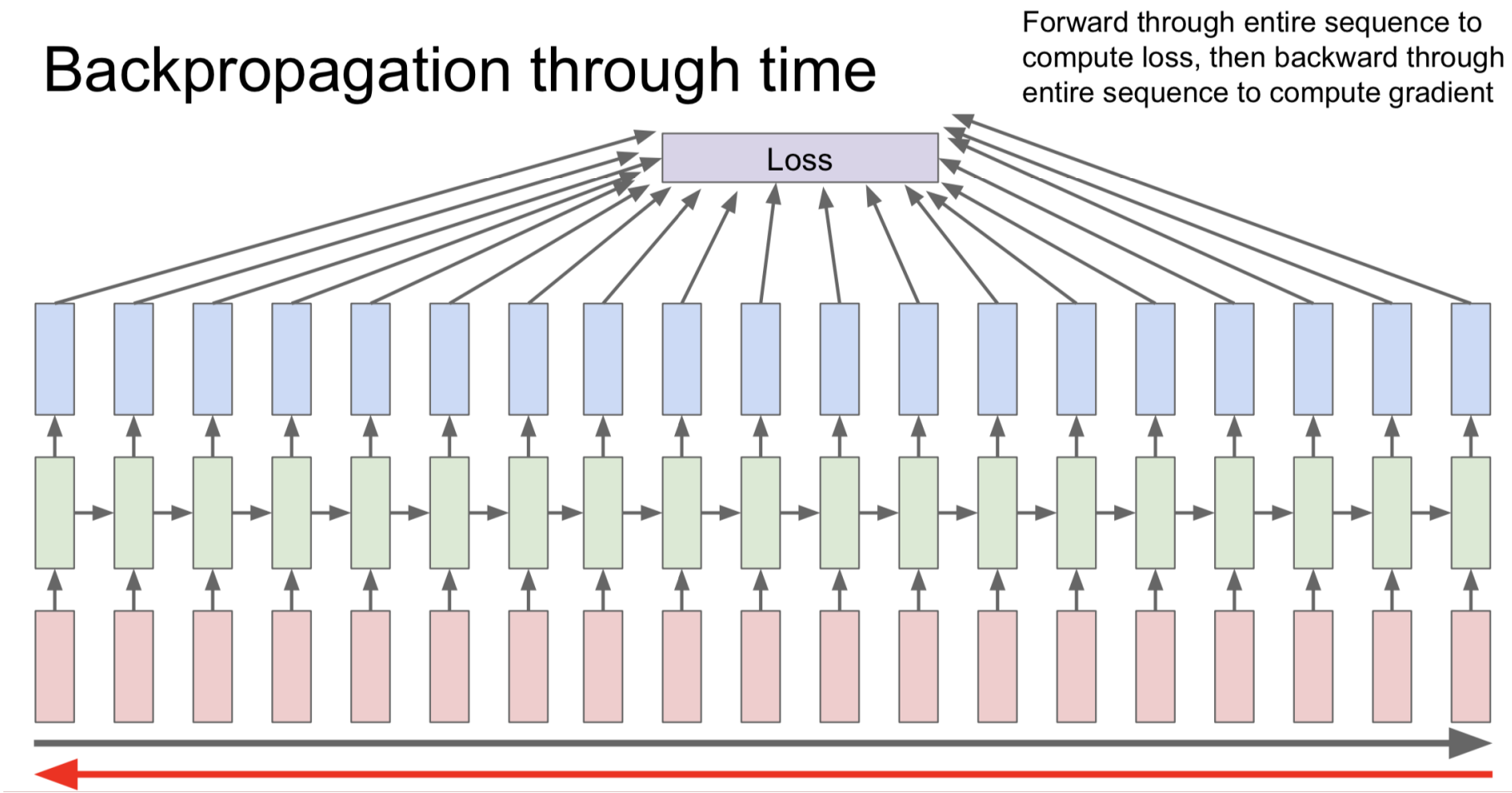
Recurrent neural networks: Representation

Example: Character-level language model

During testing: sample characters feed back to model one at a time

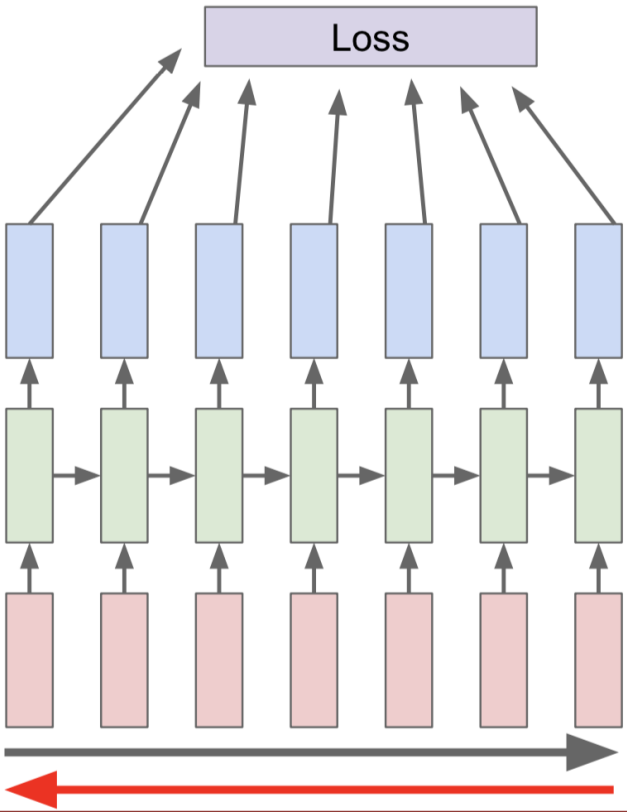


Recurrent neural networks: Learning



Recurrent neural networks: Learning

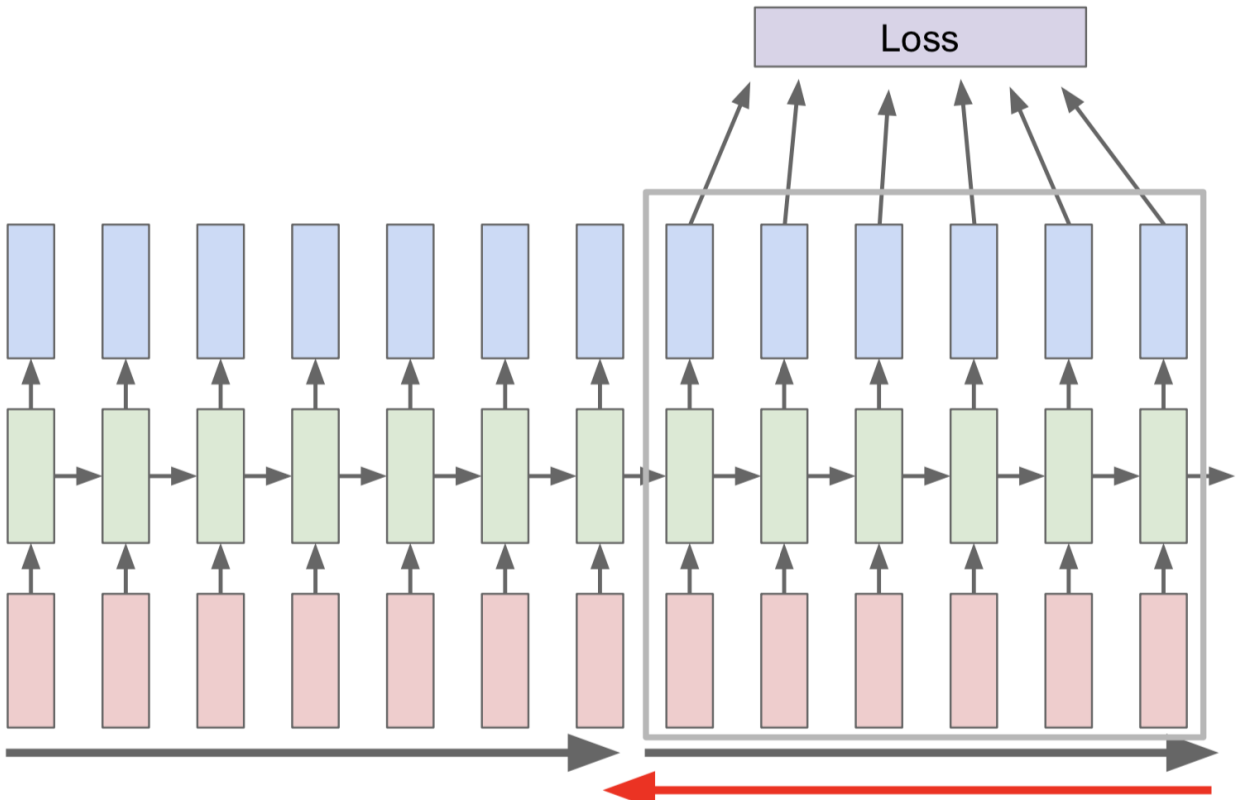
Truncated Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Recurrent neural networks: Learning

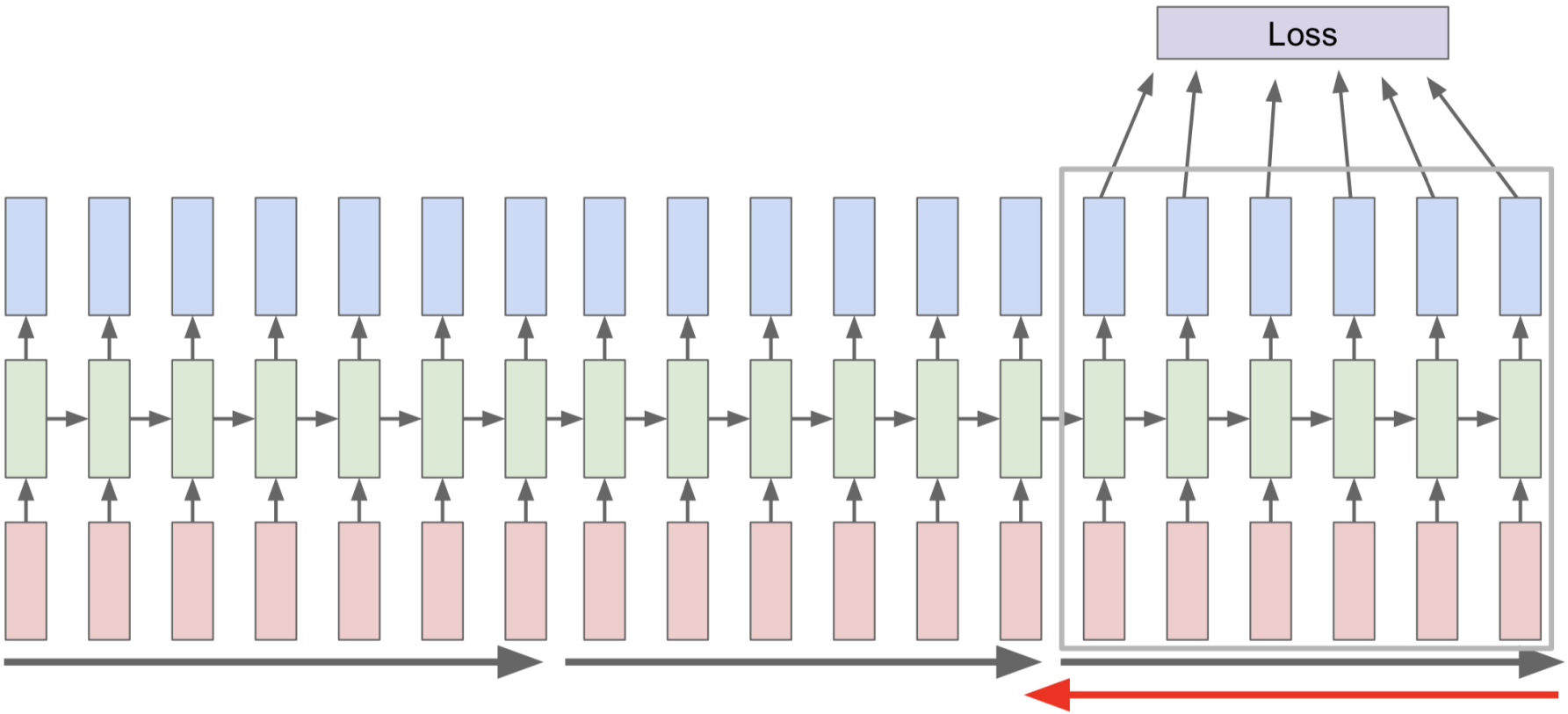
Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Recurrent neural networks: Learning

Truncated Backpropagation through time



Recurrent neural networks: Learning

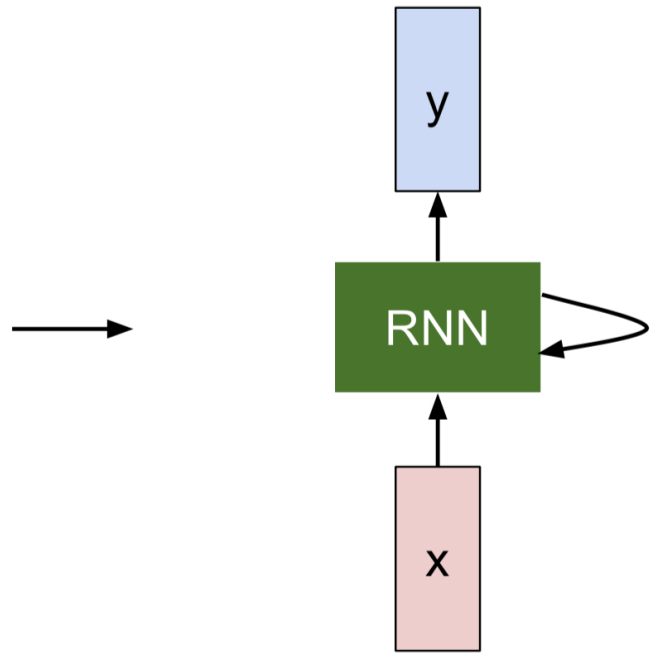
Example: Text generation

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



Recurrent neural networks: Learning

Example: Text generation

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

↓ train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

↓ train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Recurrent neural networks: Learning

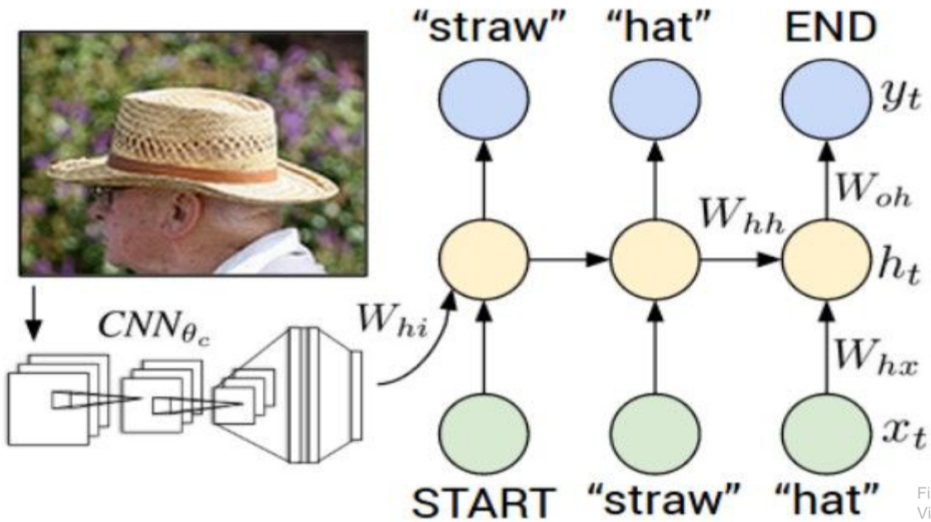
Example: Music generation

Music and Art Generation using Machine Learning | Curtis Hawthorne | TEDxMountainViewHighSchool

`https://www.youtube.com/watch?v=Q-Qq8ipUHEI`

Recurrent neural networks: Learning

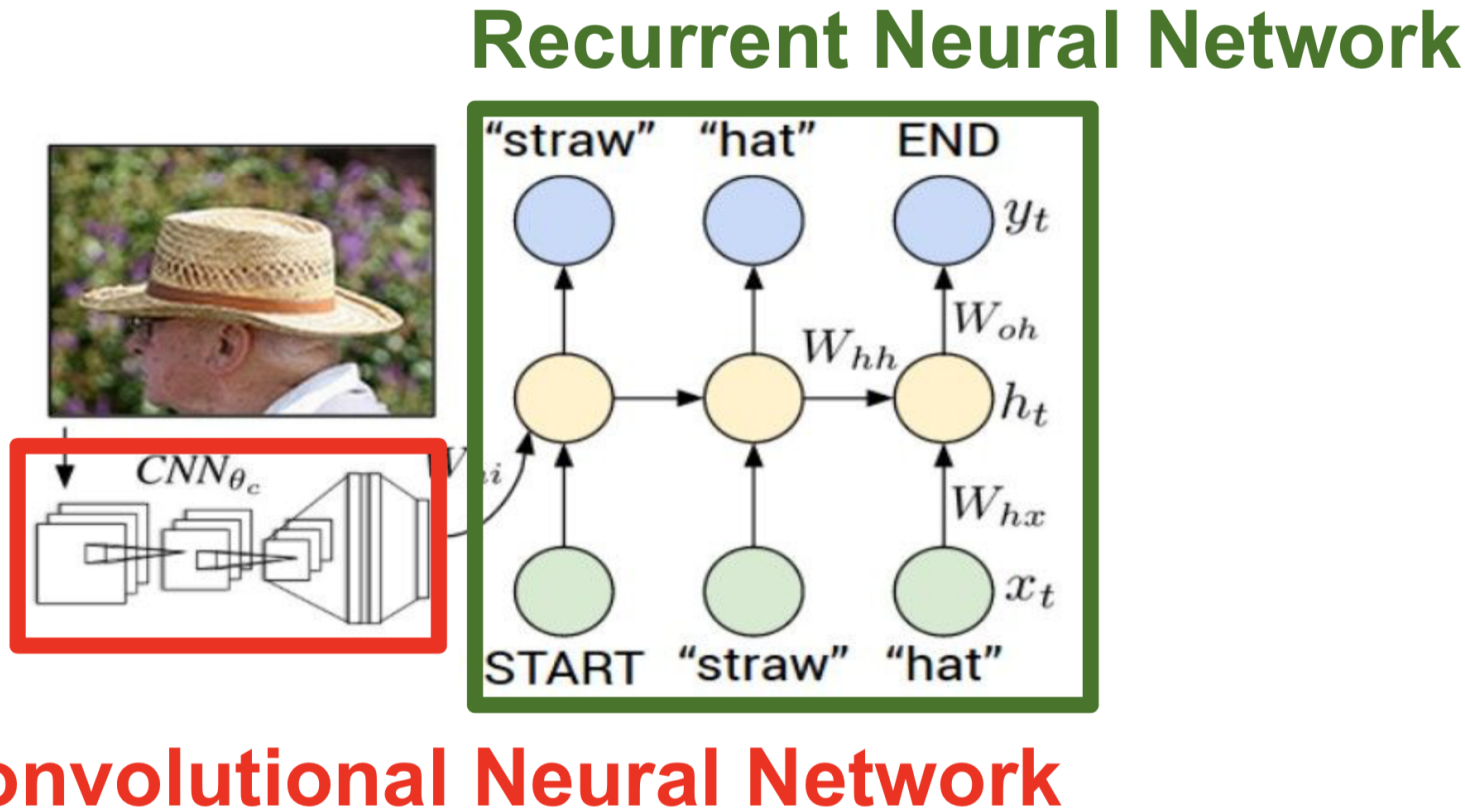
Example: Image captioning



Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

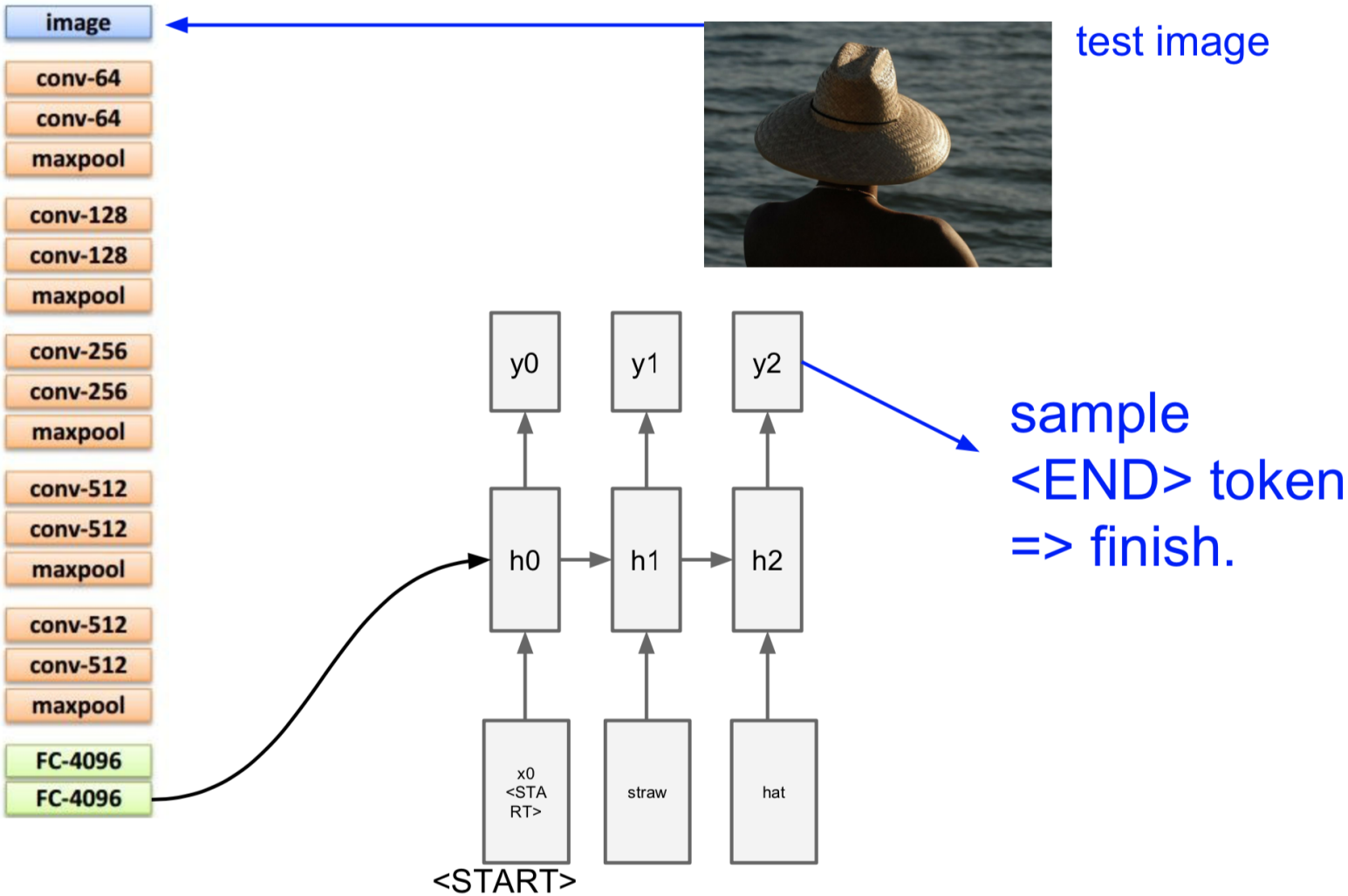
Recurrent neural networks: Learning

Example: Image captioning



Recurrent neural networks: Learning

Example: Image captioning



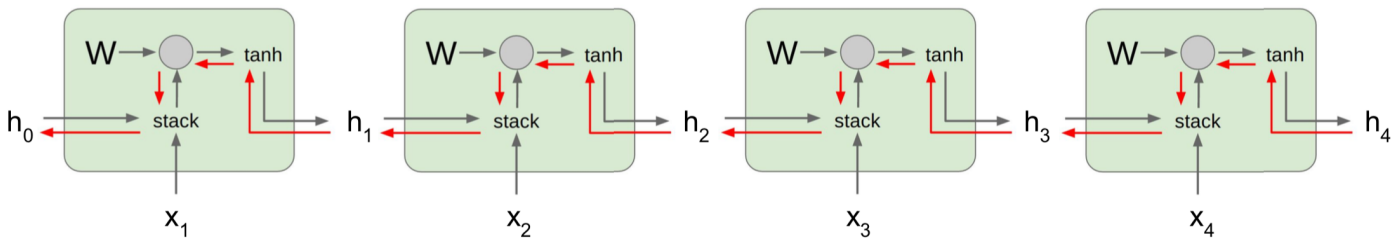
Recurrent neural networks: Learning

Example: Image captioning

Recurrent neural networks: Learning

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

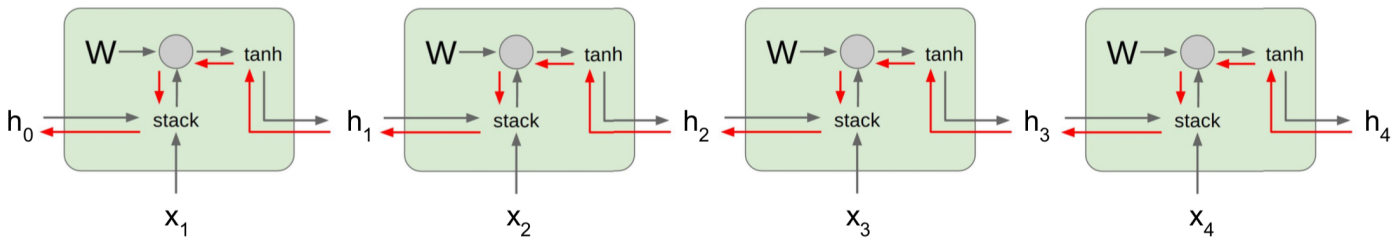
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1: **Vanishing gradients**

Recurrent neural networks: Learning

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients**

Largest singular value < 1: **Vanishing gradients** → Change RNN architecture

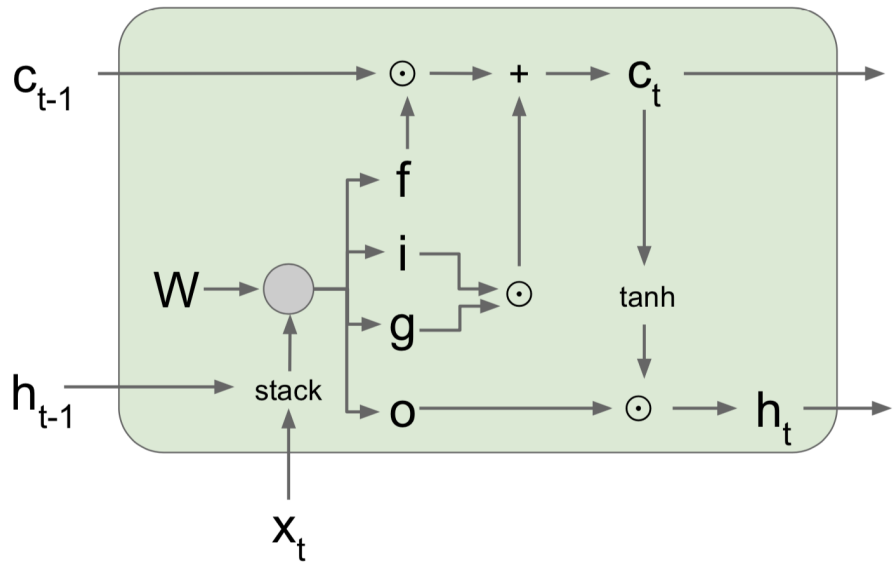
Overview

- Deep neural networks
  - Motivation & Challenges
  - Unsupervised pretraining: Deep belief networks & autoencoders
  - (Supervised) fine-tuning
  - Alternative optimization methods
  - Convolutional neural networks
  - Recurrent neural networks
  - Long short term memory neural networks

[The content for the following slides have been summarized from Li, Johnson, & Yeung, Stanford CSCE 231]

Long short term memory neural networks: Representation

Central Idea: A memory consists of an explicit memory and gating units which regulate the information flow into and out of the memory.



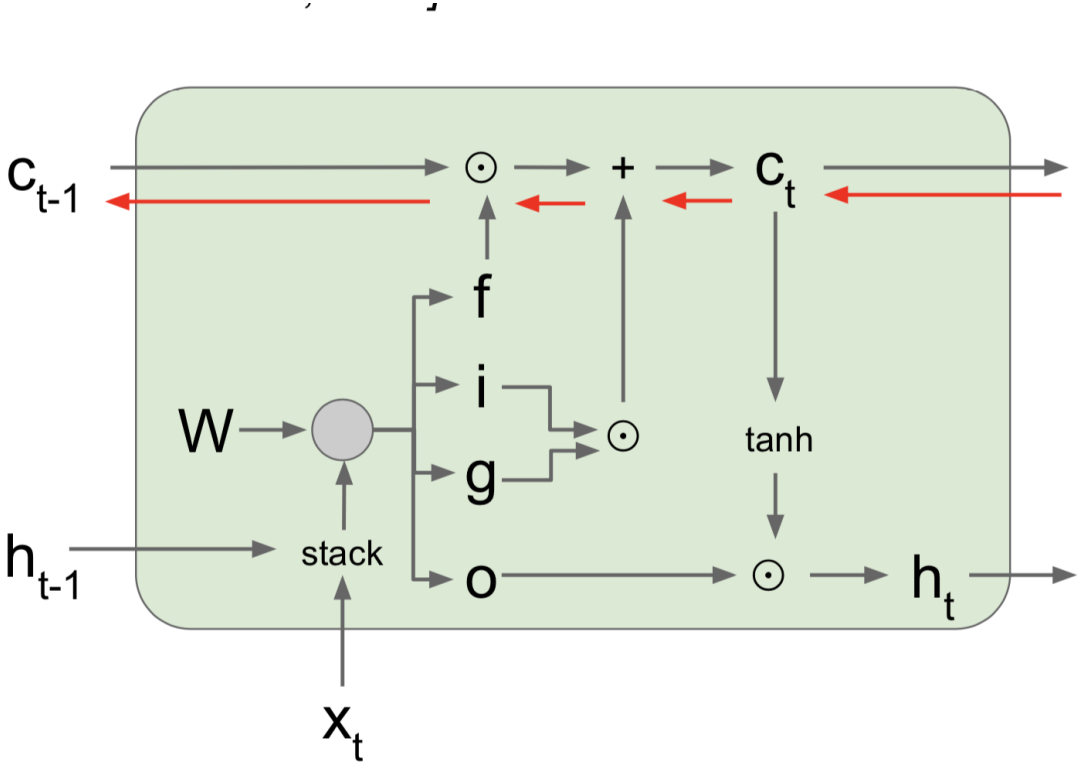
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

- f: Forget gate, Whether to erase cell
- i: Input gate, whether to write to cell
- g: How much to write to cell
- o: Output gate, How much to reveal cell

Long short term memory neural networks: Learning



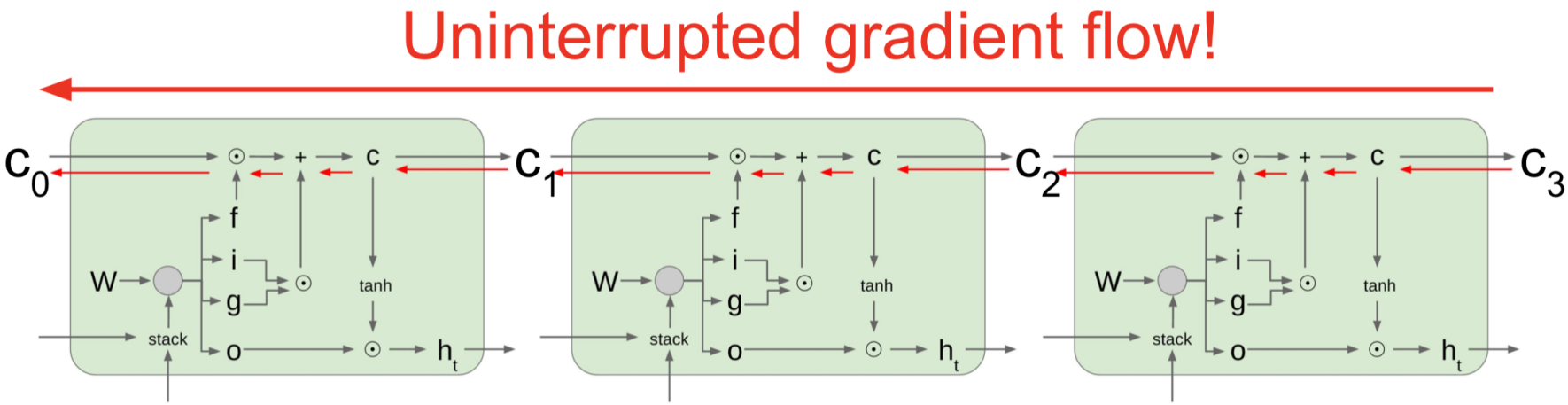
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long short term memory neural networks: Learning



What have we learnt so far

- DNNs allow hierarchical representations learned from raw data
- Challenges in terms of training $\rightarrow$ pretraining
  - deep belief networks
  - autoencoders
- Convolutional neural networks $\rightarrow$ image
  - convolution: local image properties
  - weight sharing: stationarity
  - max-pooling: robustness in the representation and reduced cost