COMP4222 Machine Learning with Structured Data

CNN and RNN

Instructor: Yangqiu Song

Slides credits: Dan Roth

Logistic Regression

- Logistic regression is designed as a **binary classifier** (output say {0,1} or {-1,1}) but actually **outputs the probability** that the input instance is in the "1" class.

- A logistic classifier has the form:

$$p(y = 1|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})}$$

where $\boldsymbol{x} = \left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(d)}\right)$ is a vector of features.

Loss

For training, we start with a collection of **input values** $\boldsymbol{x}_i$ and corresponding **output labels** $y_i \in \{-1,1\}$. Let p_i be the **predicted output** on input $\boldsymbol{x}_i$, so

$$p_i = \frac{1}{1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)}$$

Logistic regression minimize the loss: negative sum of the log accuracy (the total accuracy), i.e.,

$$f(\boldsymbol{w}) = -\sum_{i=1}^{N} \log p_i = \sum_{i=1}^{N} \log(1+\exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))$$

Cross Entropy Loss

- In binary classification, where the number of classes M equals 2 and $y_i \in \{0,1\}$, cross-entropy can be calculated as:

$$-(y\log(p) + (1-y)\log(1-p))$$

- If M>2 (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

- The output of softmax function

$$\langle x_1, x_2, \ldots, x_k \rangle \mapsto \left\langle \frac{e^{x_1}}{\sum_{j=1}^{k} e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^{k} e^{x_j}}, \ldots, \frac{e^{x_k}}{\sum_{j=1}^{k} e^{x_j}} \right\rangle$$

Example

0.1	0.3	0.3
0.2	0.4	0.3
0.7	0.3	0.4

"1"	"2"	"3"
1	0	0
0	1	0
0	0	1

Classification accuracy = 2/3
Cross-entropy loss = 4.14

0.3	0.1	0.1
0.4	0.7	0.2
0.3	0.2	0.7

"1"	"2"	"3"
1	0	0
0	1	0
0	0	1

Classification accuracy = 2/3
Cross-entropy loss = 1.92

why-you-should-use-cross-entropy-error

Training

For training, we start with a collection of **input values** $\boldsymbol{x}_i$ and corresponding **output labels** $y_i \in \{-1,1\}$. Let p_i be the **predicted output** on input $\boldsymbol{x}_i$, so

$$p_i = \frac{1}{1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)}$$

Logistic regression minimize the loss: negative sum of the log accuracy (the total accuracy), i.e.,

$$f(\boldsymbol{w}) = -\sum_{i=1}^{N} \log p_i = \sum_{i=1}^{N} \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i))$$

Gradient based methods

Our goal is to minimize $f(\boldsymbol{w})$

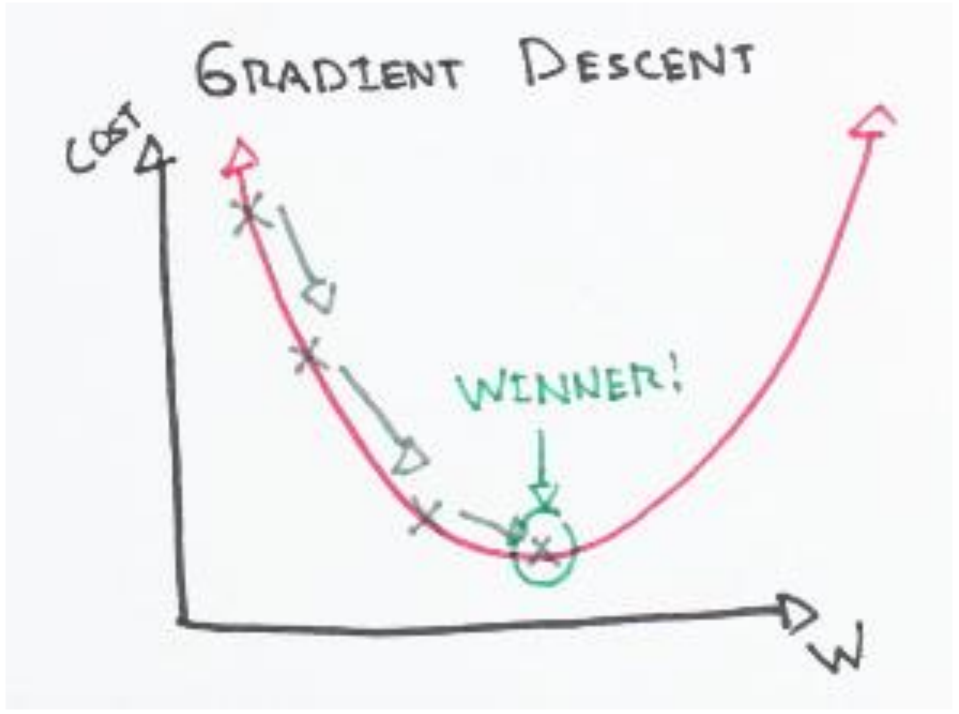
Gradient based method use

$$\boldsymbol{w}' = \boldsymbol{w} - \ \alpha\, \boldsymbol{d}$$

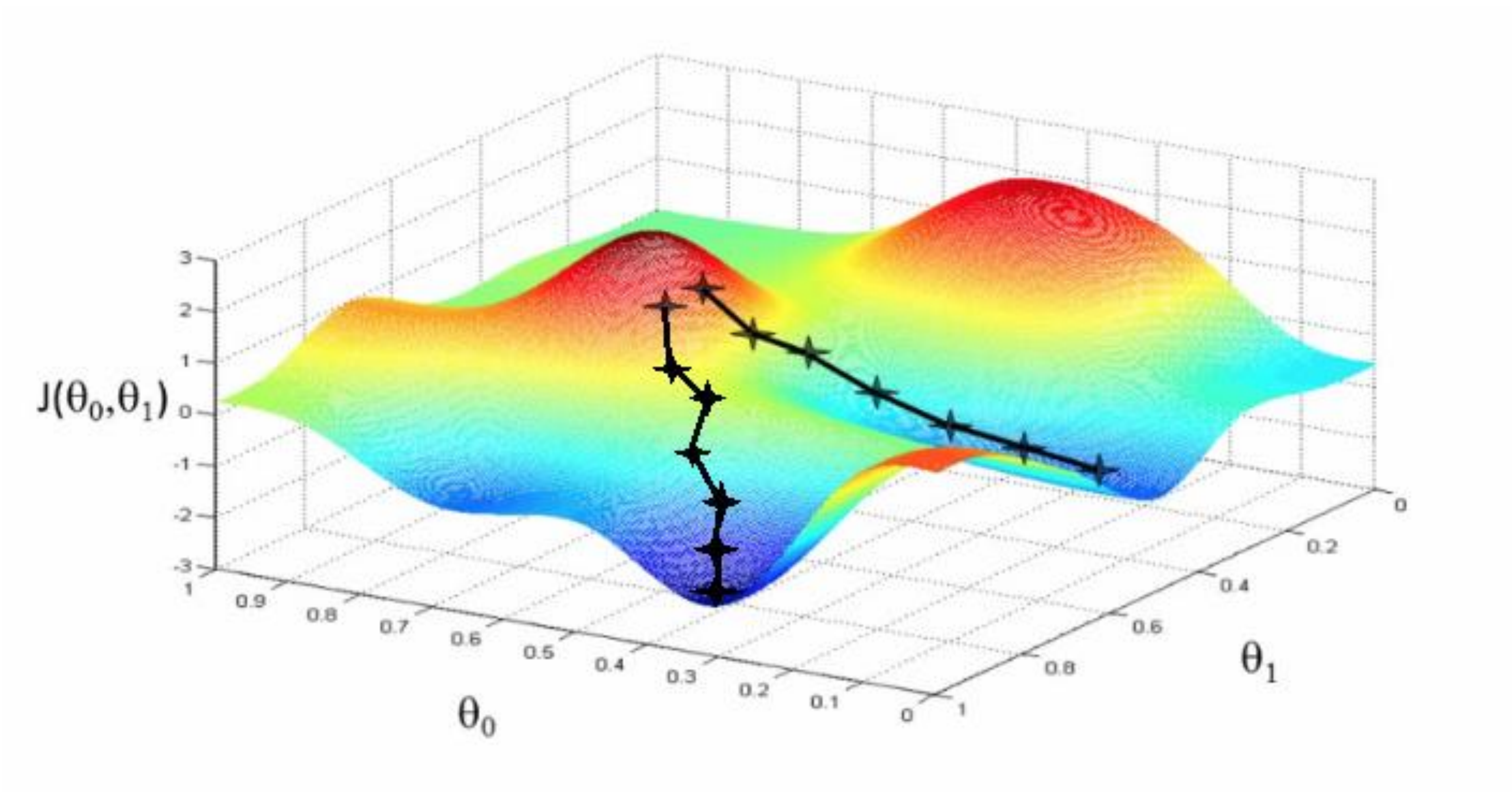
to update the weight vector where

$$\boldsymbol{d} \propto \nabla f(\boldsymbol{w})$$

$\nabla f(\boldsymbol{w}) = \left[\frac{df}{d\boldsymbol{w}^{(1)}}, \dots, \frac{df}{d\boldsymbol{w}^{(d)}}\right]^T$ is the gradient



Gradient Descent



Andrew Ng’s Machine Learning Course

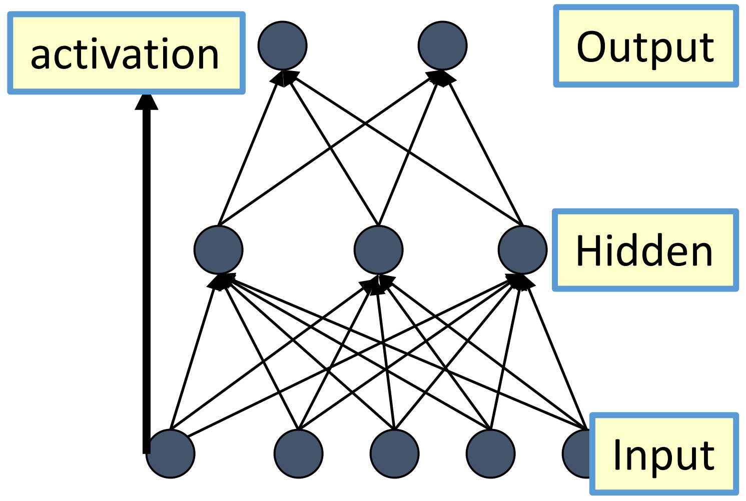
Neural Networks

- Neural Networks are **functions**: $\mathrm{NN}: X \rightarrow Y$
  - where $X = [0,1]^n$*, or* $\{0,1\}^n$ *and* $Y = [0,1]$*,* $\{0,1\}$ (or $\{-1,1\}$)

- NN can be used as an approximation of a target classifier
  - In their general form, even with a single hidden layer, NN can approximate any function
  - Algorithms exist that can learn a NN representation from labeled training data (e.g., Backpropagation).

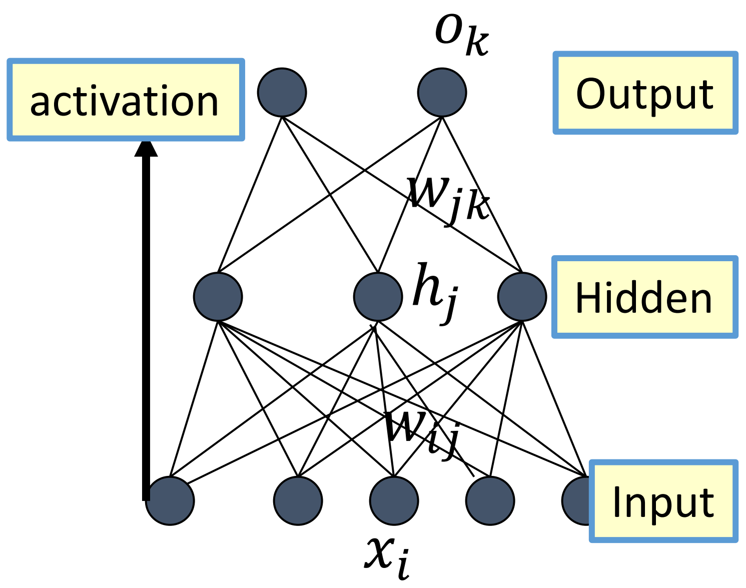
Multi-Layer Neural Networks

- Multi-layer network were designed to overcome the computational (**expressivity**) limitation of a single threshold element.
- The idea is to **stack** several layers of threshold elements, each layer using the output of the previous layer as input.



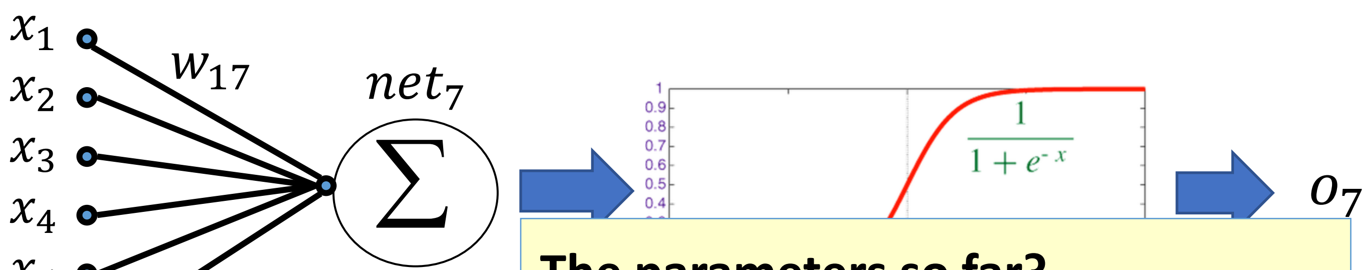
Basic Unit in Multi-Layer Neural Network

- **Linear Unit**: $o_j = \vec{w}.\vec{x}$ multiple layers of linear functions produce linear functions. We want to represent nonlinear functions.
  - Note: Here we use a slightly different notation for dot product $\vec{w}.\vec{x}$



Model Neuron (Logistic, slightly different notations)

- Neuron is modeled by a unit k connected by weighted links w_{jk} to other units j.
  - Note: we use a different kind of index of w to indicate different neurons



The parameters so far?
The set of connective weights: w_{jk}
The threshold/bias value: b_k

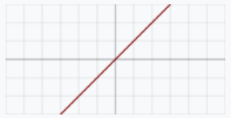
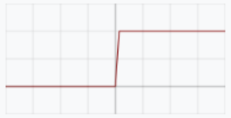
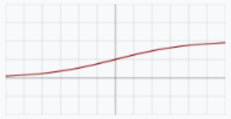
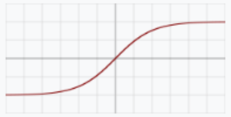
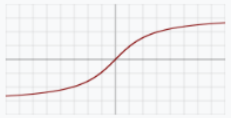
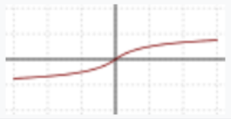
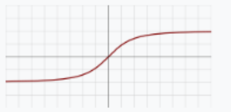
- Use a non-linear, differentiable output function such as the sigmoid or logistic function
- Net input to a unit is defined as:

$$\text{net}_k = \sum w_{jk}.x_j$$

- Output of a unit is defined as:

$$o_k = \frac{1}{1+\exp(-(\text{net}_k - b_k))}$$

Activation function

Name	Plot	Equation	Derivative (with respect to *x*)	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \ne 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$[1]	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
Softsign[9][10]		$f(x) = \frac{x}{1 + \lvert x \rvert}$	$f'(x) = \frac{1}{(1 + \lvert x \rvert)^2}$	$(-1, 1)$
Inverse square root unit (ISRU)[11]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$

- https://en.wikipedia.org/wiki/Activation_function

Learning with a Multi-Layer Perceptron

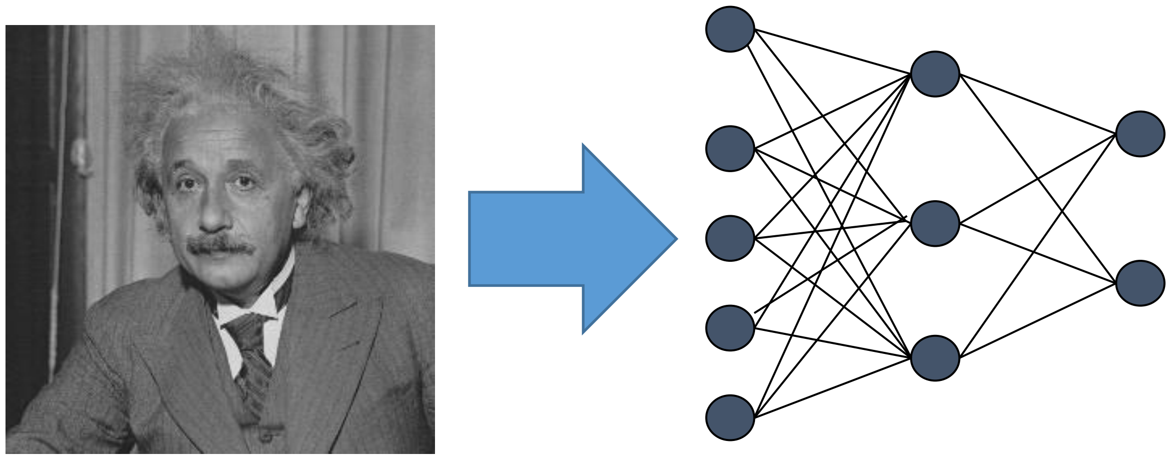
- Solution: Define an error function (e.g., sum of squares) that is a differentiable function of the output, i.e. this error function is also a differentiable function of the weights.

- We can then evaluate the derivatives of the error with respect to the weights, and use these derivatives to find weight values that minimize this error function, using gradient descent (or other optimization methods).

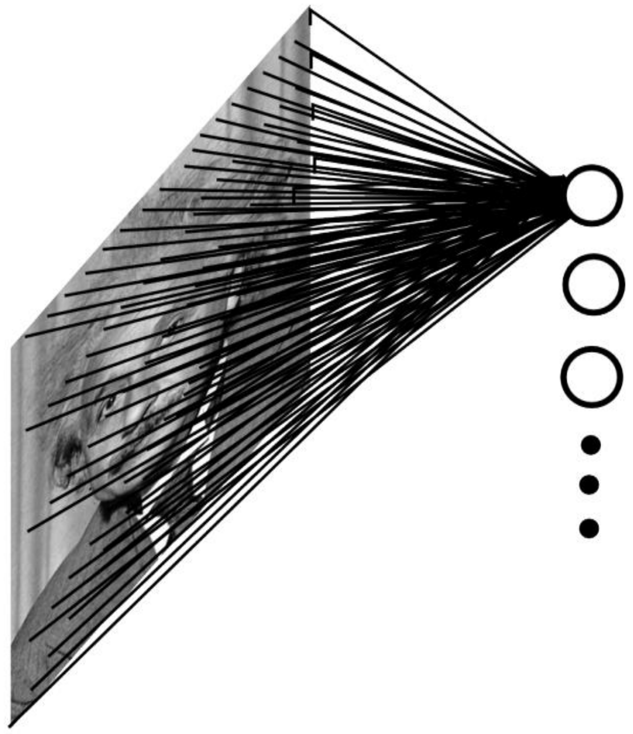
- This results in an algorithm called back-propagation.

Receptive Fields

- Consider a task with image inputs
  - Receptive fields should give expressive features from the raw input to the system
  - How would you design the receptive fields for this problem?

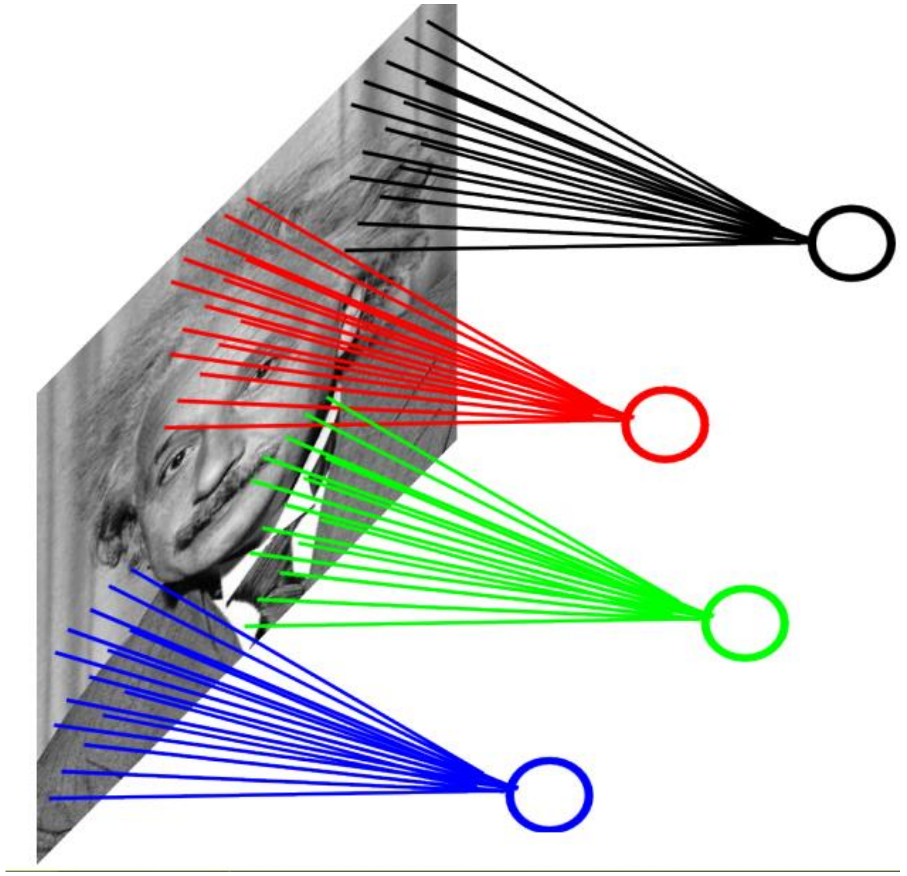


- A **fully connected layer**:
  - Example:
    - 100x100 images
    - 1000 units in the input
  - Problems:
    - 10^7 edges!
    - Spatial correlations lost!
    - Variables sized inputs.



Slide Credit: Marc'Aurelio Ranzato

- Consider a task with image inputs:
- A **locally connected layer**:
  - Example:
    - 100x100 images
    - 1000 units in the input
    - Filter size: 10x10
  - Local correlations preserved!
  - Problems:
    - 10^5 edges
    - This parameterization is good when input image is registered (e.g., face recognition).
    - Variable sized inputs, again.

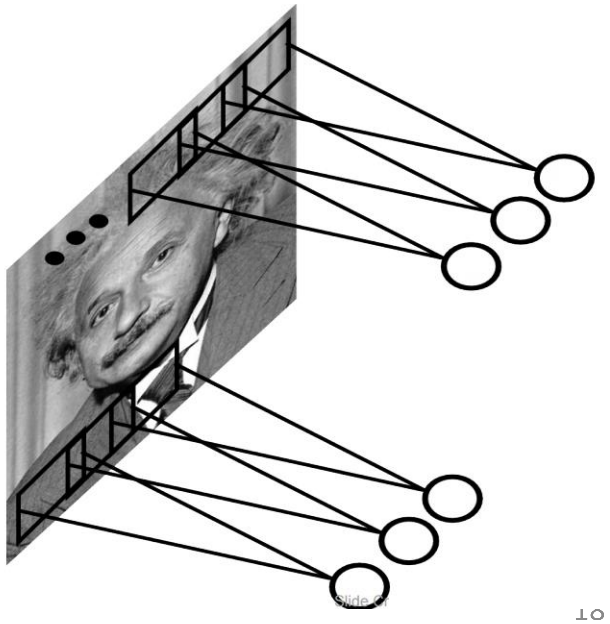


Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer

- **A solution:**
  - **Filters** to capture different patterns in the input space.
    - **Share** parameters across different locations (assuming input is stationary)
    - **Convolutions** with learned filters
  - Filters will be **learned** during training.
  - The issue of variable-sized inputs will be resolved with a **pooling** layer.

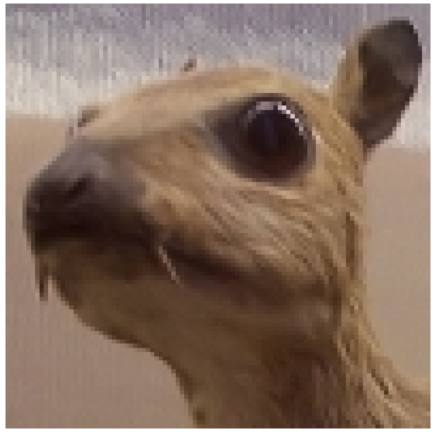
So what is a convolution?



Convolution Operator (2)

- Convolution in two dimension:
  - The same idea: flip one matrix and slide it on the other matrix
  - Example: edge detection kernel:

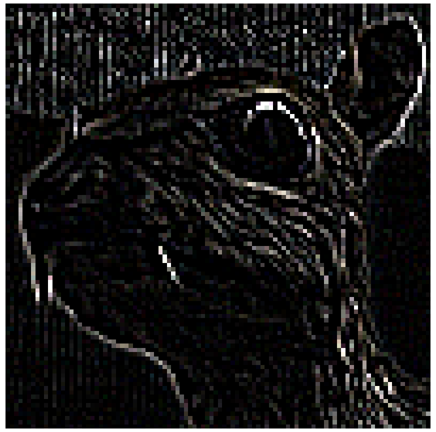
Input image



Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



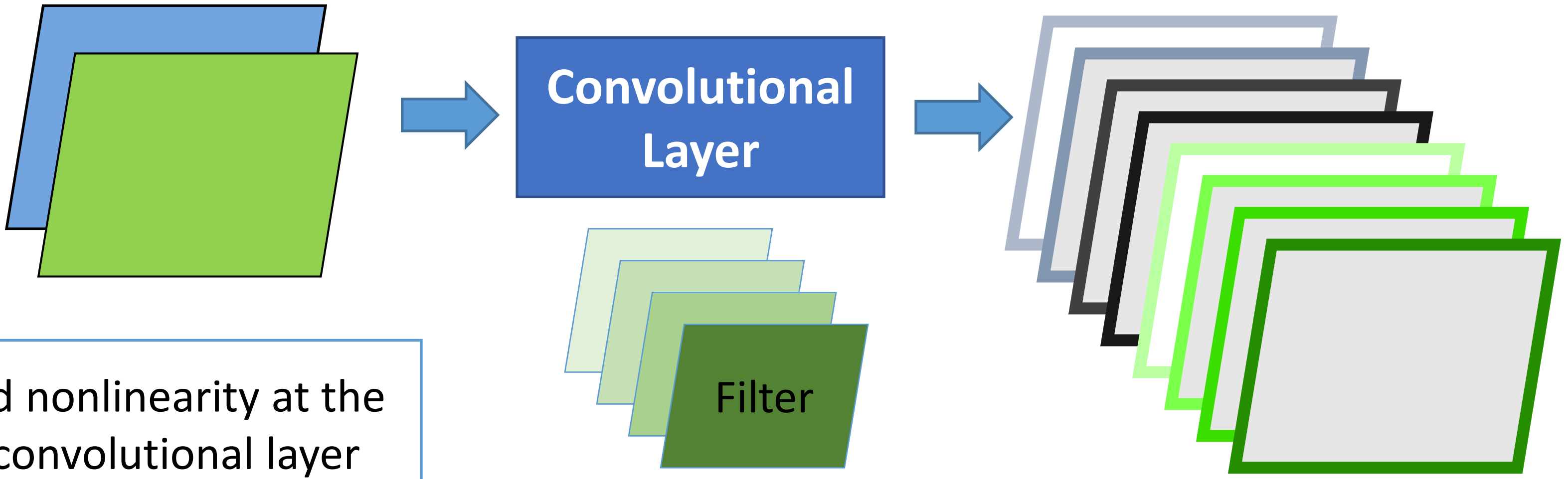
Try other kernels: http://setosa.io/ev/image-kernels/

Demo of CNN

- https://setosa.io/ev/image-kernels/

Convolutional Layer

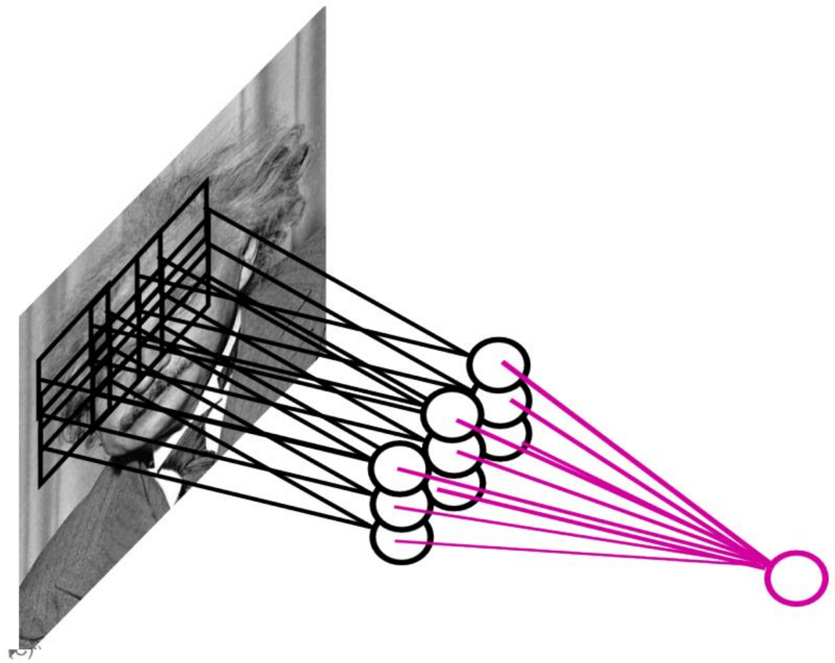
- The convolution of the **input (vector/matrix)** with weights **(vector/matrix)** results in a **response vector/matrix**.
- We can have **multiple filters** in each convolutional layer, each producing an output.
- If it is an intermediate layer, it can have **multiple inputs**!



One can add nonlinearity at the output of convolutional layer

Pooling Layer

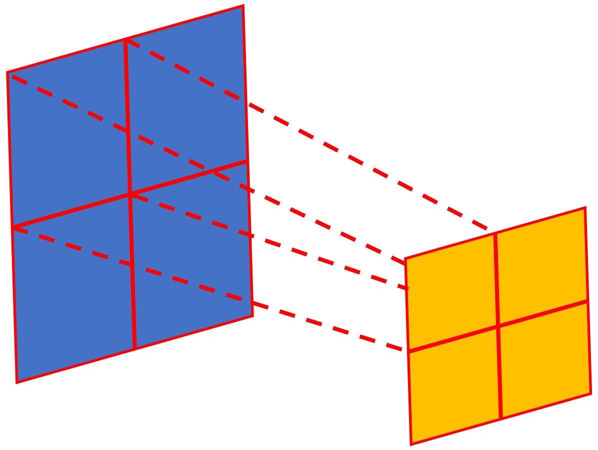
- How to handle variable sized inputs?
  - A layer which reduces inputs of different size, to a fixed size.
  - **Pooling**



Slide Credit: Marc'Aurelio Ranzato

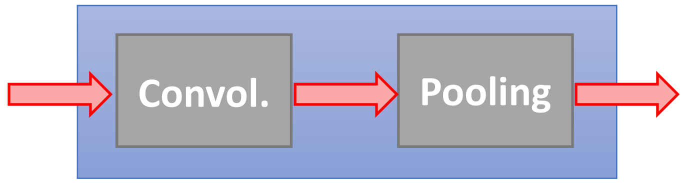
Pooling Layer

- How to handle variable sized inputs?
  - A layer which reduces inputs of different size, to a fixed size.
  - **Pooling**
  - Different variations
    - Max pooling
      $$h_i[n] = \max_{i \in N(n)} \tilde{h}\,[i]$$
    - Average pooling
      $$h_i[n] = \frac{1}{n} \sum_{i \in N(n)} \tilde{h}\,[i]$$
    - L2-pooling
      $$h_i[n] = \frac{1}{n} \sqrt{\sum_{i \in N(n)} \tilde{h}^2\,[i]}$$
    - etc

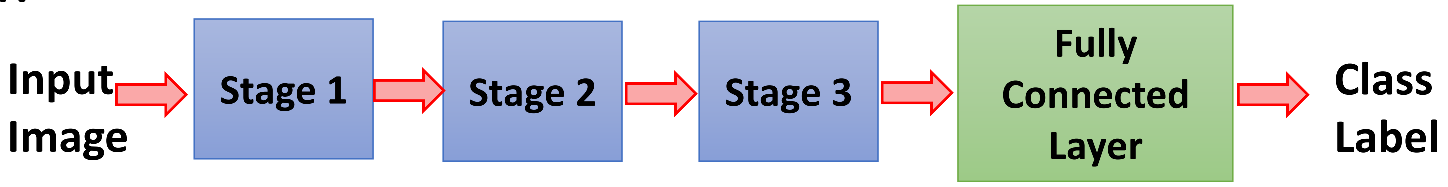


Convolutional Nets

- One stage structure:



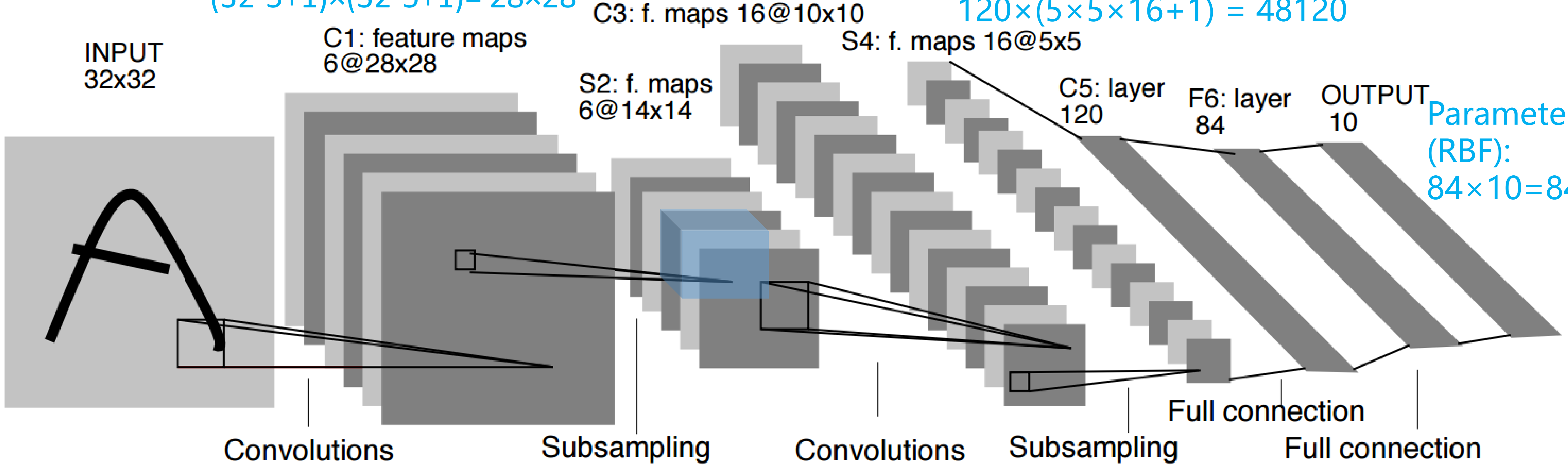
- Whole system:



An example system (LeNet)

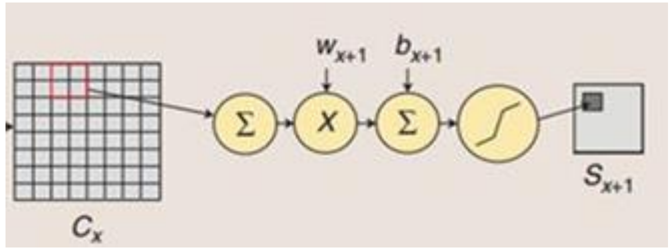
(32-5+1)×(32-5+1)= 28×28

Parameters:
120×(5×5×16+1) = 48120



Parameters
(RBF):
84×10=840

Parameters: (5*5+1)*6=156
(Conv+bias)*channels

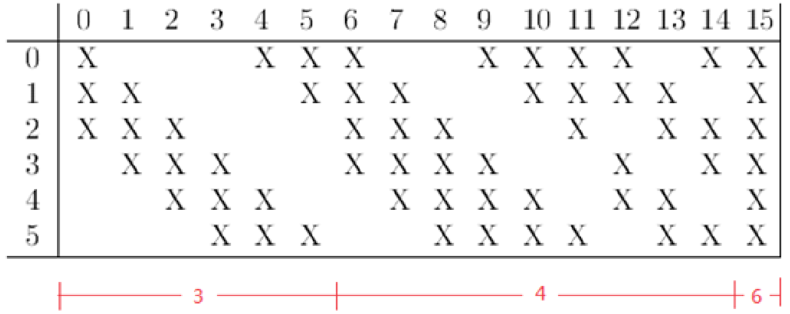


Parameters: (1+1)*6=12
(Weight+bias)*channels

Parameters:
(5×5×3+1)×6+
(5×5×4+1)×9+
5×5×6+1 =
1516

Parameters:
16×2 = 32

Parameters:
(120+1)×84
=10164

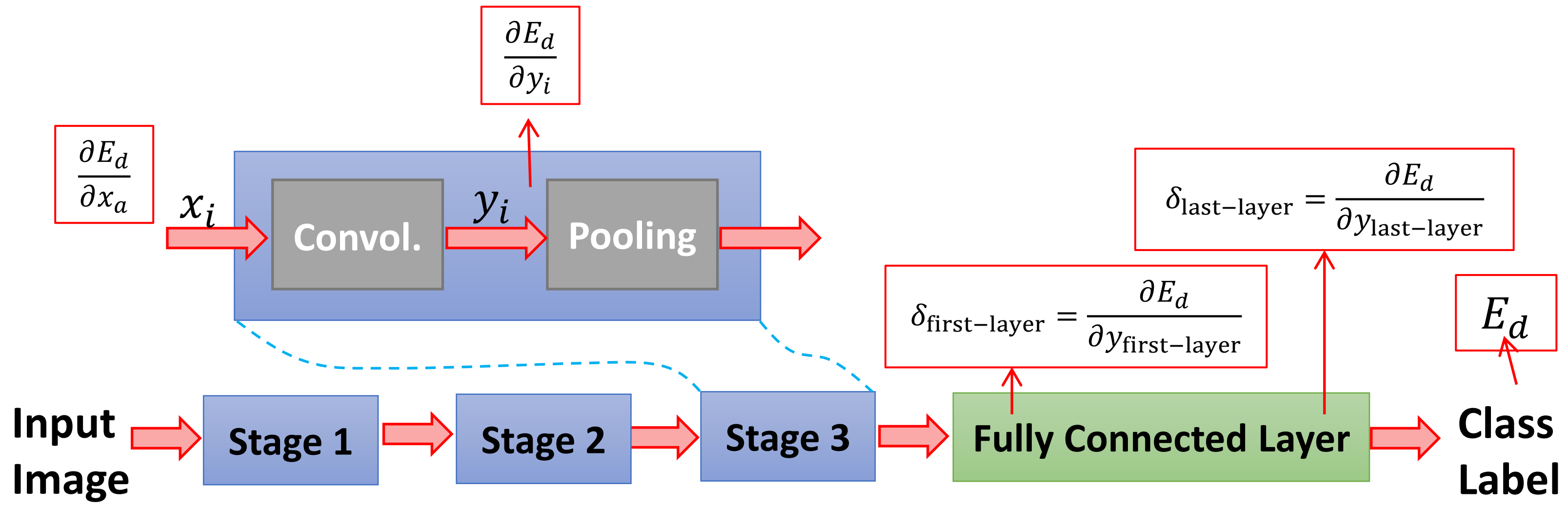


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

3 4 6

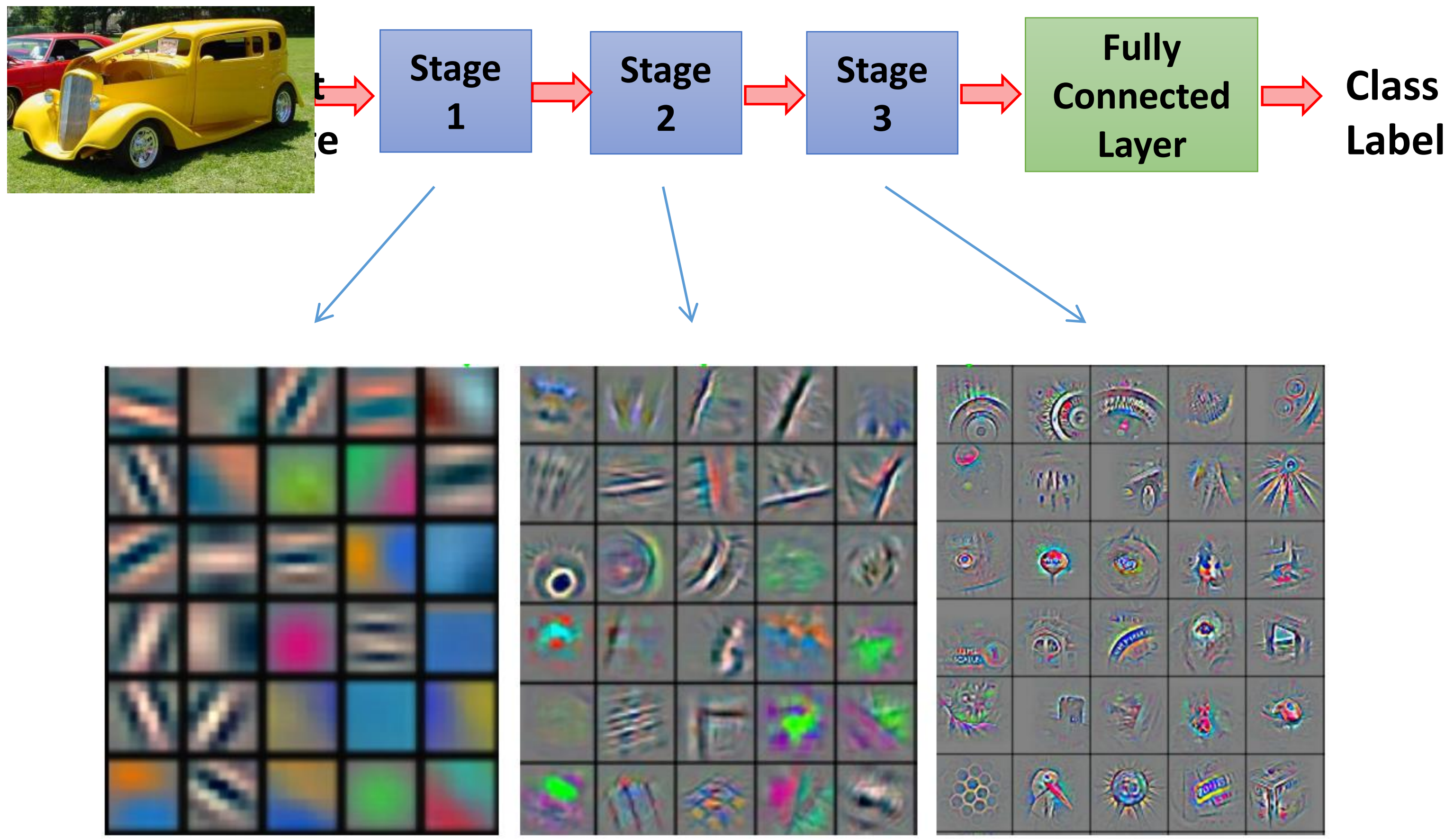
Training a ConvNet

- The same procedure from Back-propagation applies here.
  - Remember in backprop we started from the error terms in the last stage, and passed them back to the previous layers, one by one.



$\frac{\partial E_d}{\partial y_i}$
$\frac{\partial E_d}{\partial x_a}$
x_i
Convol.
y_i
Pooling
$\delta_{\text{last-layer}} = \frac{\partial E_d}{\partial y_{\text{last-layer}}}$
$\delta_{\text{first-layer}} = \frac{\partial E_d}{\partial y_{\text{first-layer}}}$
E_d
Input Image
Stage 1
Stage 2
Stage 3
Fully Connected Layer
Class Label

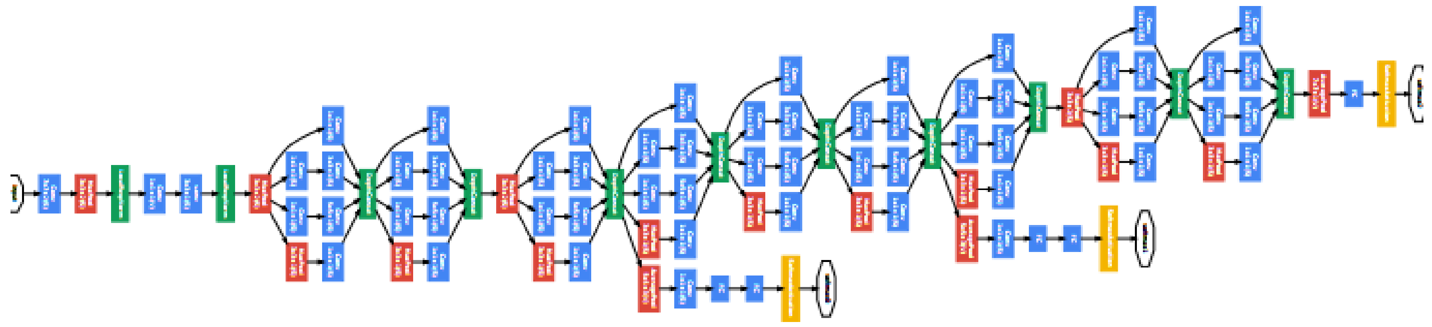
Convolutional Nets



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

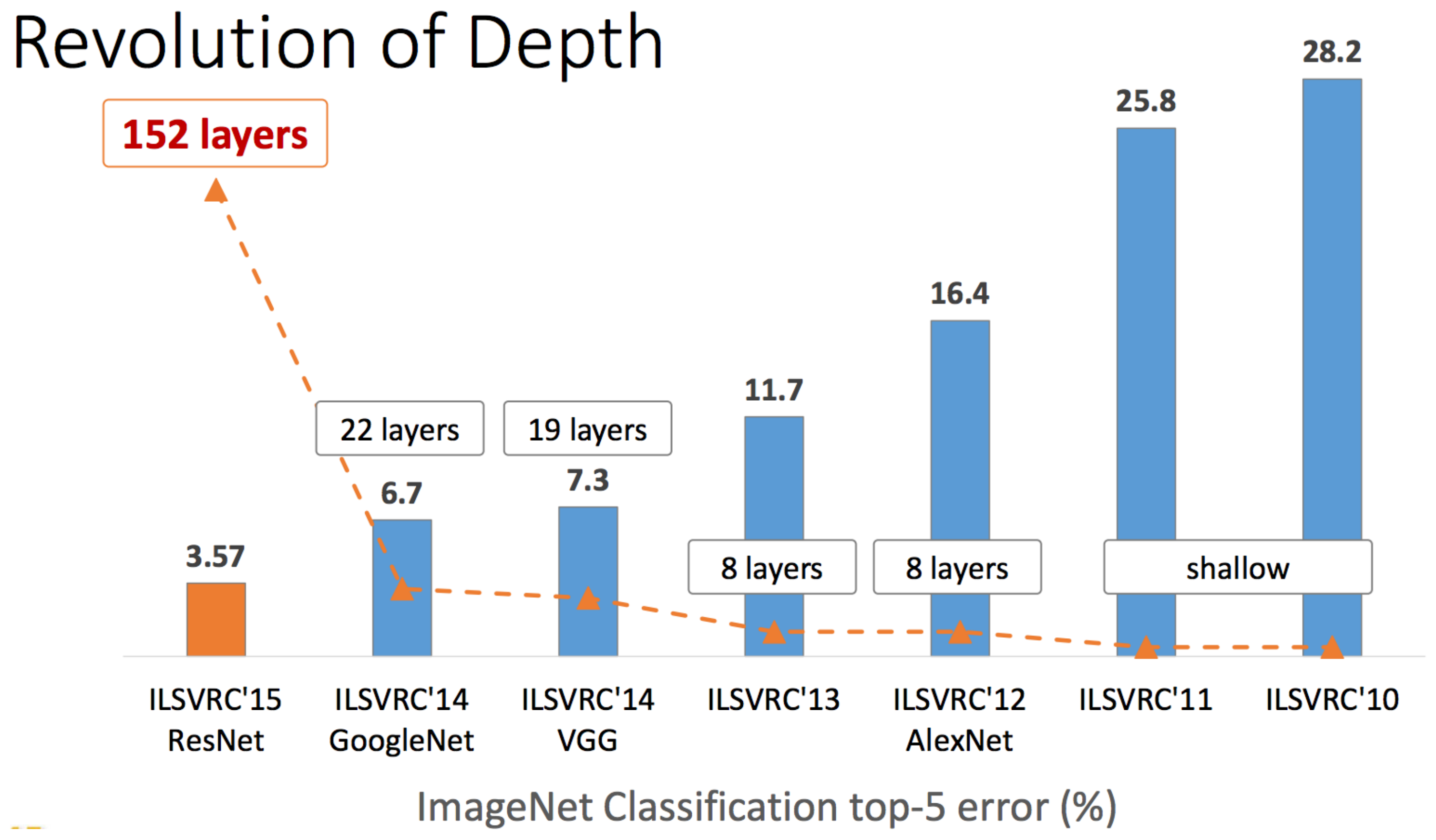
ConvNet roots

- **Fukushima, 1980s** designed network with same basic structure but did not train by backpropagation.
- The first successful applications of **Convolutional Networks** by Yann LeCun in 1990's (LeNet)
  - Was used to read zip codes, digits, etc.
- Many variants nowadays, but the core idea is the same
  - Example: a system developed in Google (GoogLeNet)
    - Compute different filters
    - Compose one big vector from all of them
    - Layer this iteratively



See more: http://arxiv.org/pdf/1409.4842v1.pdf

Depth matters



Slide from [Kaiming He 2015]

Practical Tips

- Before large scale experiments, test on a small subset of the data and check the error should go to zero.
  - Overfitting on small training
- Visualize features (feature maps need to be uncorrelated) and have high variance
- Bad training: many hidden units ignore the input and/or exhibit strong correlations.

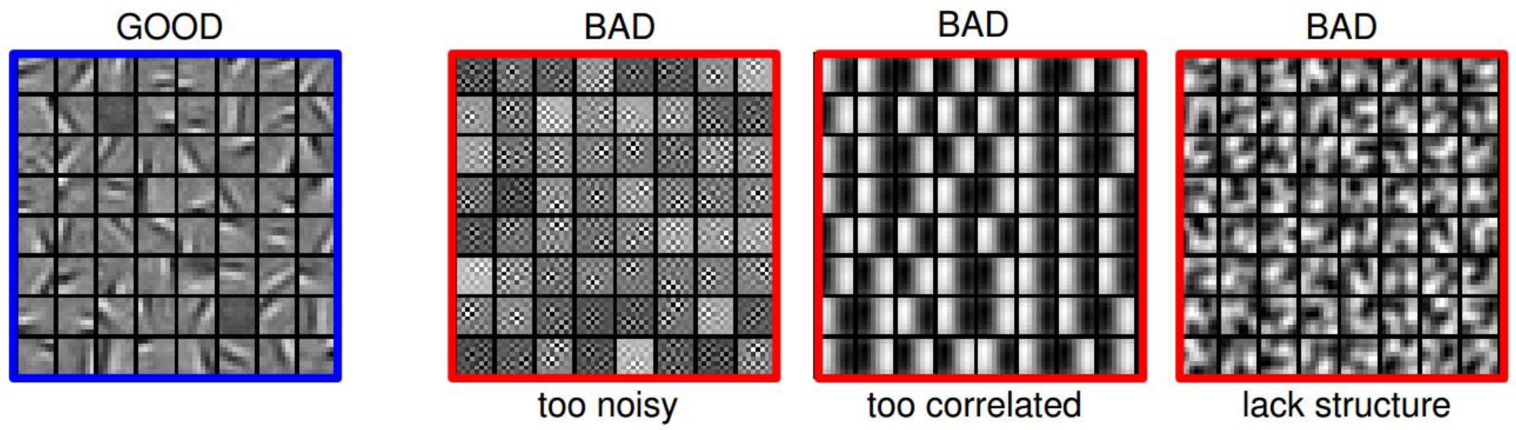


Figure Credit: Marc'Aurelio Ranzato

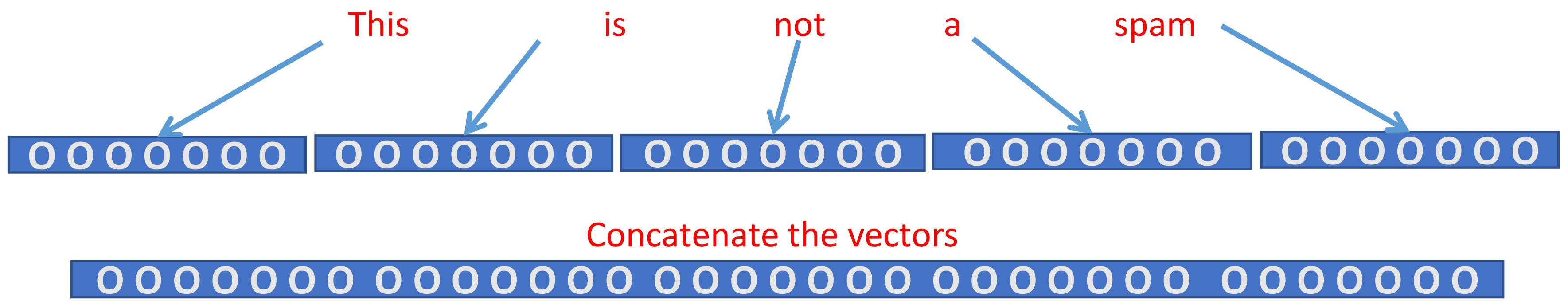
Debugging

- Training diverges:
  - Learning rate may be too large → decrease learning rate
  - BackProp is buggy → numerical gradient checking
- Loss is minimized but accuracy is low
  - Check loss function: Is it appropriate for the task you want to solve? Does it have degenerate solutions?
- NN is underperforming / under-fitting
  - Compute number of parameters → if too small, make network larger
  - Layer normalization, batch normalization, and other tricks (included in tutorials)
- NN is too slow
  - Compute number of parameters → Use distributed framework, use GPU, make network smaller

Many of these points apply to many machine learning models, no just neural networks.

CNN for text (sequence) inputs

- Let's study another variant of CNN for language
  - Example: sentence classification (say spam or not spam)
- First step: represent each word with a vector in $\mathbb{R}^d$

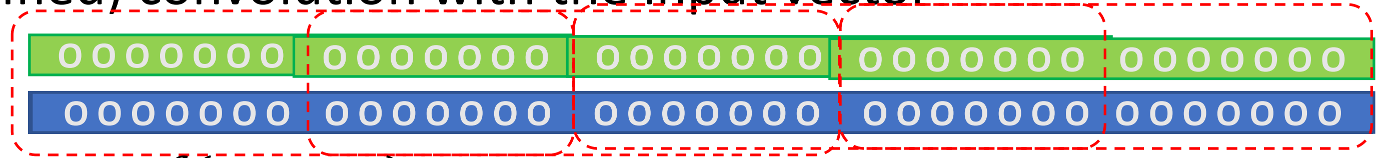


- Now we can assume that the input to the system is a vector $\mathbb{R}^{dl}$
  - Where the input sentence has length l ($l = 5$ in our example)
  - Each word vector's length d ($d = 7$ in our example)

Convolutional Layer on vectors

- Think about a single convolutional layer
  - A bunch of **vector** filters
    - Each defined in $\mathbb{R}^{dh}$
      - Where h is the number of the words the filter covers
      - Size of the word vector d

  - Find its (modified) convolution with the input vector



$c_1 = f(w.x_{1:hd})$ $c_2 = f(w.x_{(d+1):(d+hd)})$ $c_3 = f(w.x_{(2d+1):(2d+hd)})$ $c_4 = f(w.x_{(3d+1):(3d+hd)})$

  - Result of the convolution with the filter

$$c = [c_1, \ldots., c_{n-h+1}]$$

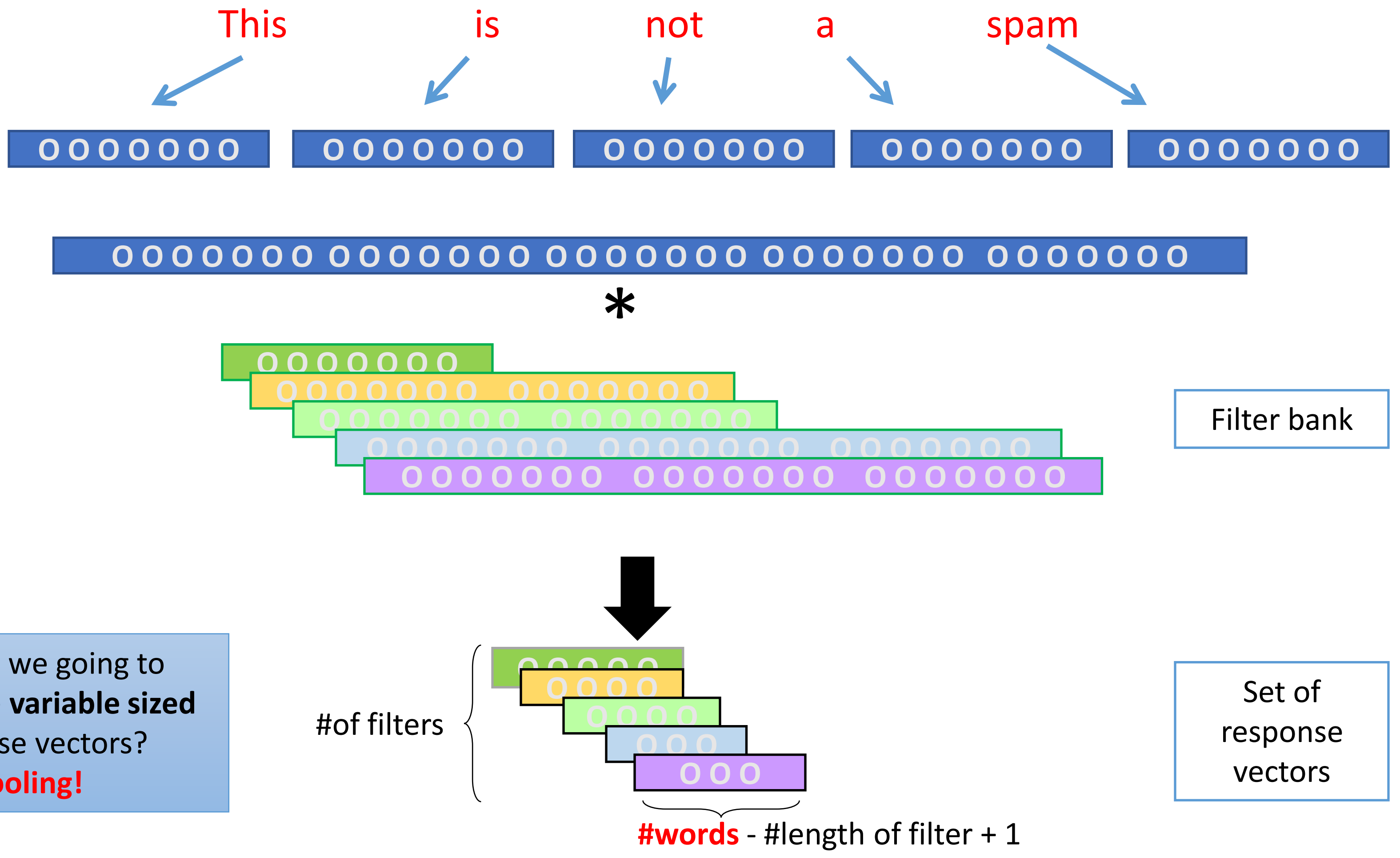
  - Convolution with a filter that spans 2 words, is operating on all of the bi-grams (vectors of two consecutive word, concatenated): “this is”, “is not”, “not a”, “a spam”.
  - Regardless of whether it is grammatical (not appealing linguistically)

Convolutional Layer on vectors

Get word vectors for each words

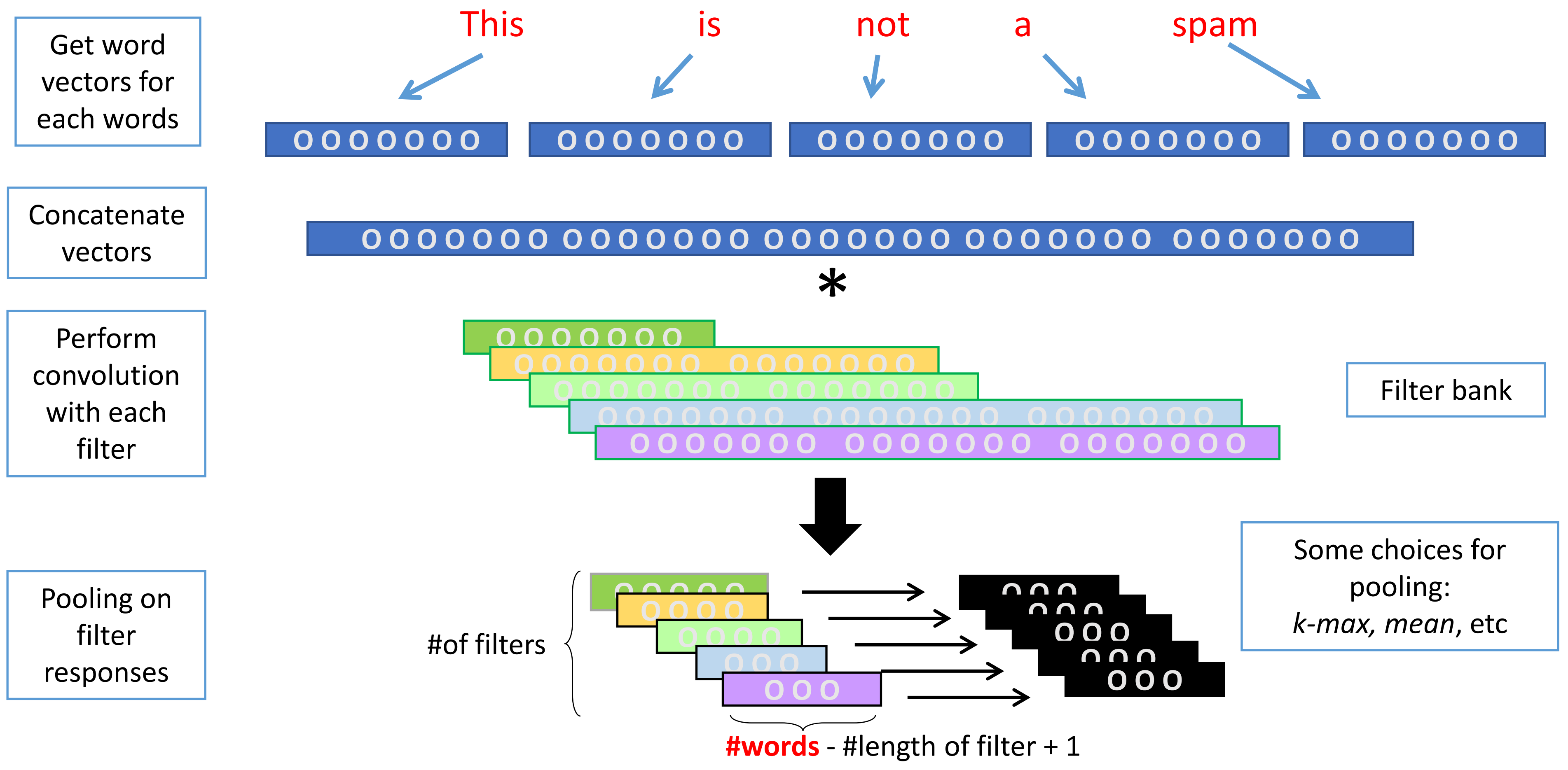
Concatenate vectors

Perform convolution with each filter



How are we going to handle the **variable sized** response vectors?
Pooling!

Convolutional Layer on vectors



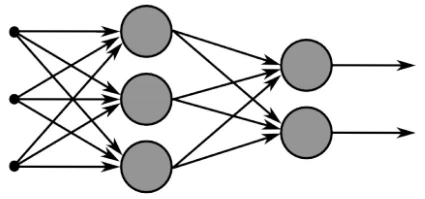
Get word vectors for each words
This
is
not
a
spam
Concatenate vectors
*
Perform convolution with each filter
Filter bank
Pooling on filter responses
#of filters
#words - #length of filter + 1
Some choices for pooling: *k-max*, *mean*, etc

- Now we can pass the fixed-sized vector to a logistic unit (softmax), or give it to multi-layer network (last session)

Recurrent Neural Networks

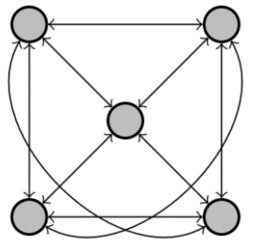
- General ideas
  - Can we use more historical words?
  - Can parameters be shared?

Recurrent Neural Networks



- Multi-layer feed-forward NN:
  - Just computes a fixed sequence of

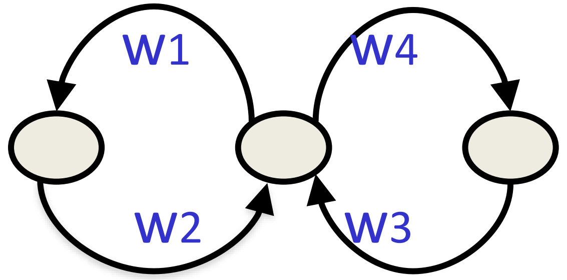
  non-linear learned transformations to convert an input patter into an output pattern

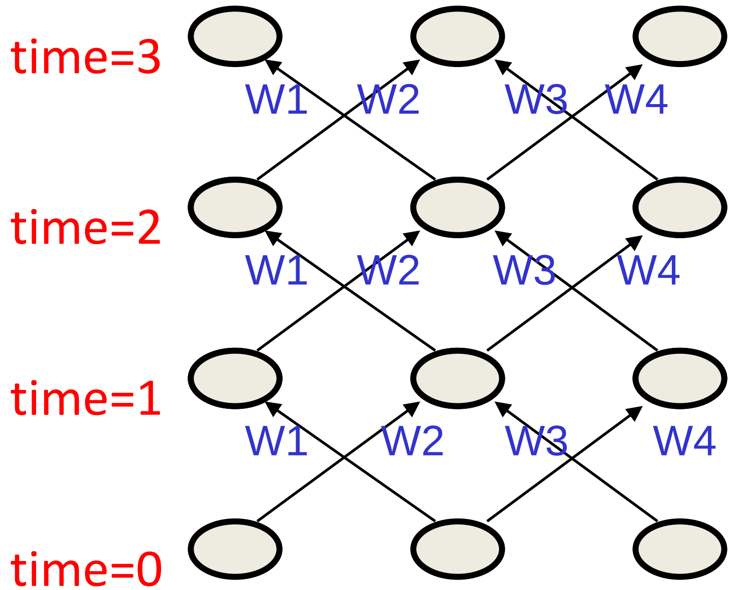


- Recurrent Neural Network:
  - Has cycles.
  - Cycle can act as a memory;
  - The hidden state of a recurrent net can carry along information about a “potentially” unbounded number of previous inputs.
  - They can model sequential data in a much more natural way.

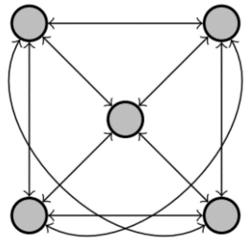
Equivalence between RNN and Feed-forward NN

- Assume that there is a time delay of 1 in using each connection.
- The recurrent net is just a layered net that keeps reusing the same weights.





Recurrent Neural Networks



- Training a general RNN’s can be hard
  - Here we will focus on a **special family of RNN’s**
- Prediction on chain-like input:
  - Language model

$X, h =$	This	is	a	sample	sentence	.
$Y =$	is	a	sample	sentence	.	<EOS>

  - POS tagging words of a sentence

$X =$	This	is	a	sample	sentence	.
$Y =$	DT	VBZ	DT	NN	NN	.

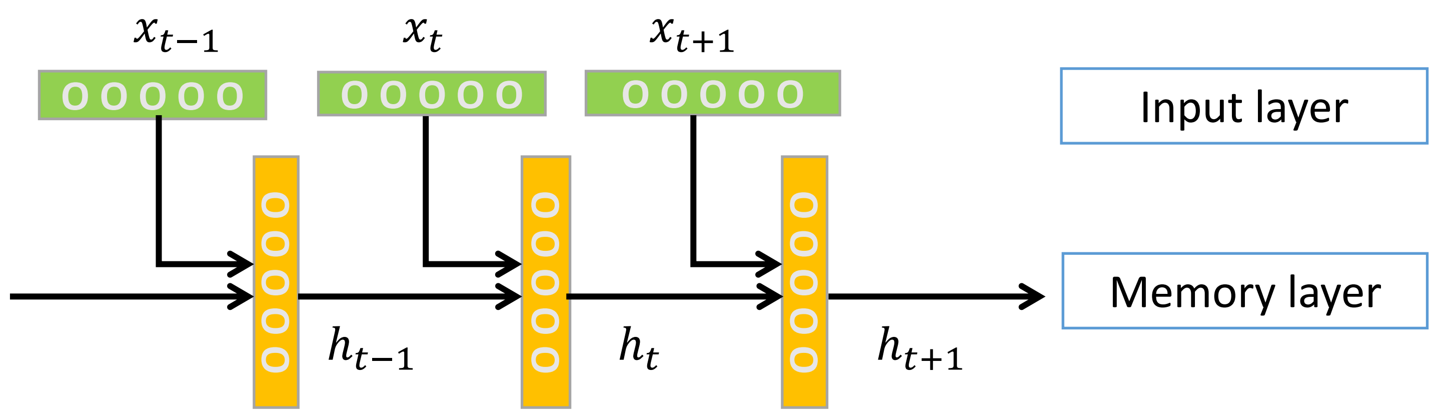
  - Sentiment classification

$X =$	This	is	a	sample	sentence	.

$Y =$ Positive/Negative

Recurrent Neural Networks

- A chain RNN:
  - Has a chain-like structure
  - Each input is replaced with its vector representation x_t
  - Hidden (memory) unit h_t contain information about previous inputs and previous hidden units h_{t-1}, h_{t-2}, etc
    - Computed from the past memory and current word. It summarizes the sentence up to that time.

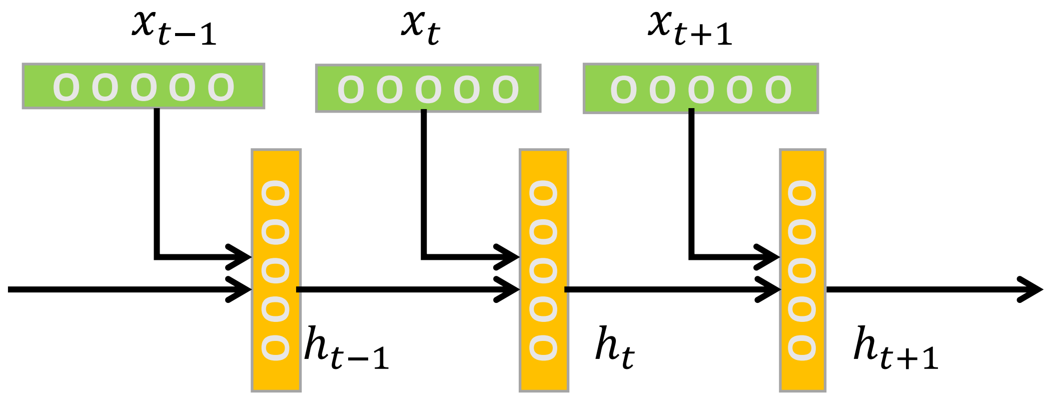


Recurrent Neural Networks

- A popular way of formalizing it:

$$h_t = f(W_h h_{t-1} + W_i x_t)$$

  - Where f is a nonlinear, differentiable function.
- Outputs?
  - Many options; depending on problem and computational resource



Recurrent Neural Networks

- Prediction for x_t, with h_t

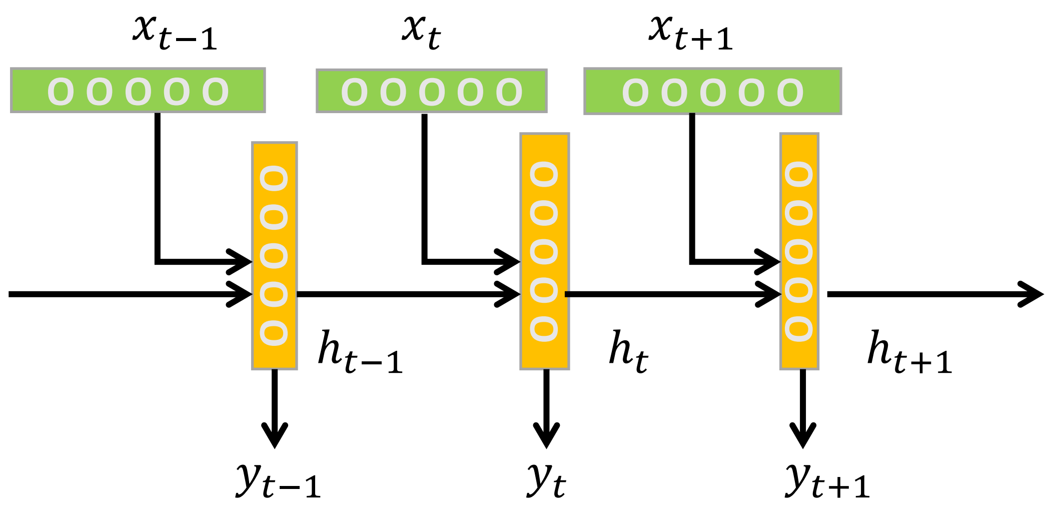
$$y_t = \text{softmax}(W_o h_t)$$

- Prediction for x_t, with $h_t, \dots, h_{t-\tau}$

$$y_t = \text{softmax}\left(\sum_{i=0}^{\tau} \alpha^i W_o^{t-i} h_{t-i}\right)$$

- Prediction for the whole chain

$$y_T = \text{softmax}(W_o h_T)$$

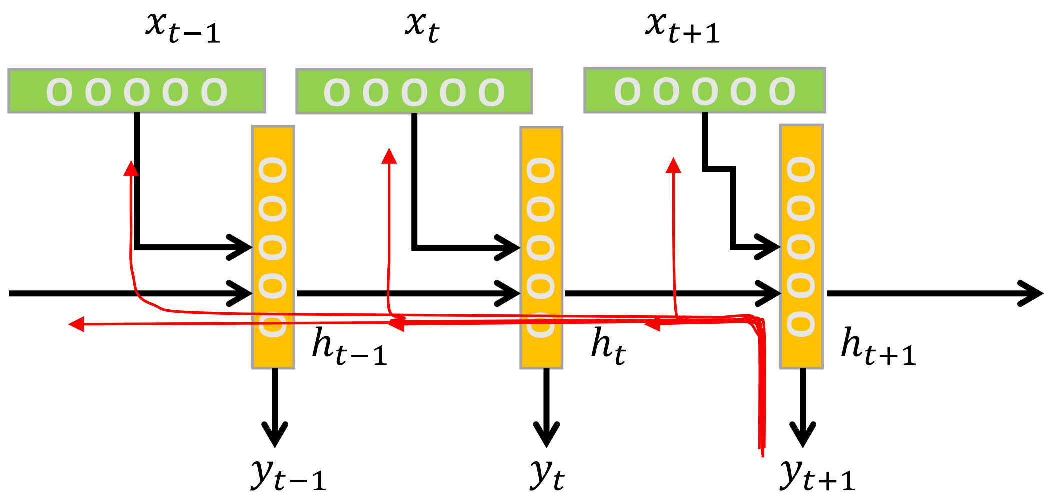


Training RNNs

- How to train such model?
  - Generalize the same ideas from back-propagation
- Total output error: $E(\vec{y}, \vec{t}) = \sum_{t=1}^{T} E_t(y_t, t_t)$

Reminder:
$y_t = \text{softmax}(W_o h_t)$
$h_t = f(W_h h_{t-1} + W_i x_t)$

Parameters?
W_o, W_i, W_h + vectors for input

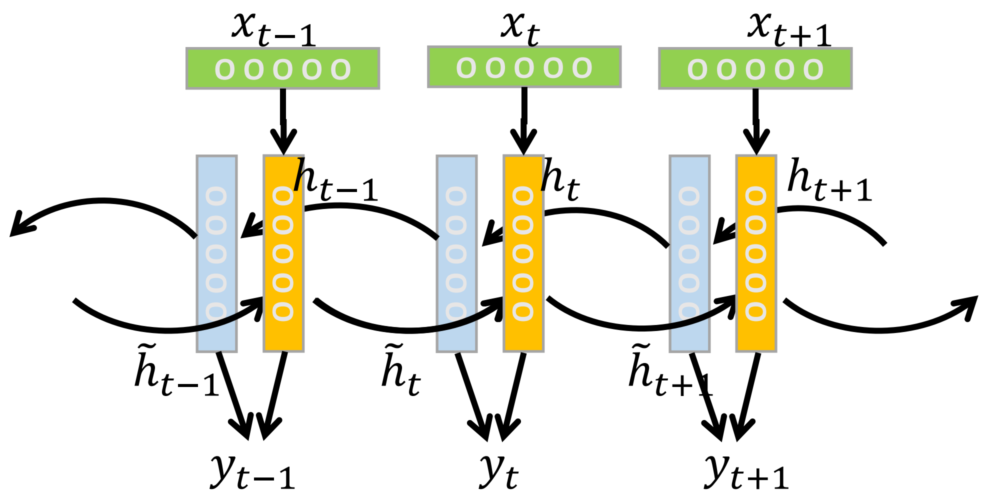


Backpropagation for RNN

This sometimes is called "Backpropagation Through Time", since the gradients are propagated back through time.

Bi-directional RNN

- One of the issues with RNN:
- Hidden variables capture only one side context
- A bi-directional structure



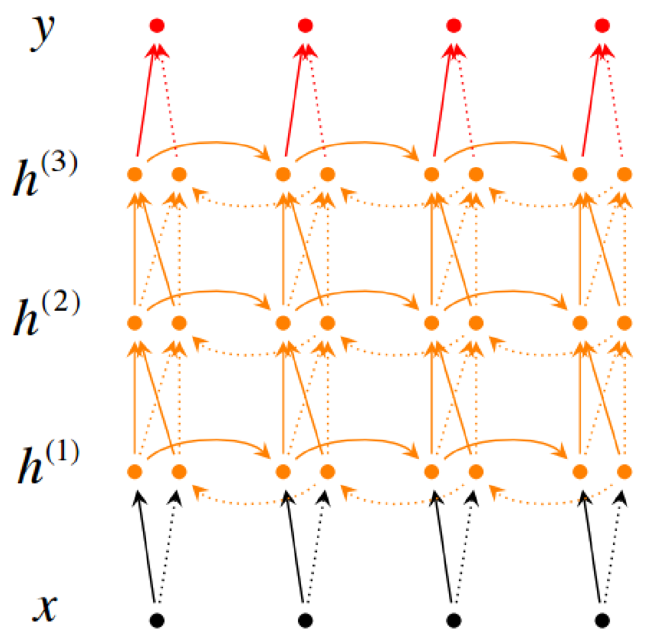
$$h_t = f(W_h h_{t-1} + W_i x_t)$$

$$\tilde{h}_t = f(\widetilde{W}_h \tilde{h}_{t+1} + \widetilde{W}_i x_t)$$

$$y_t = \text{softmax}\big(W_o h_t + \widetilde{W}_o \tilde{h}_t\big)$$

Stack of bi-directional networks

- Use the same idea and make your model further complicated:



Notes

- We will come back to these concepts when introducing more applications
- We will provide examples in tutorials for you to play with