Yale

Deep Learning Basics, CNNs, RNNs

CPSC 471 / 571: Trustworthy Deep Learning

Rex Ying

Readings

- Readings are updated on the website (syllabus page)

- **Lecture 1 readings**: AI Sustainability
- **Lecture 2 readings**:
  Trustworthy Machine Learning Book – **Chapter 1 Establishing Trust**
  - *Chapter 1.1 Defining Trust*

  Trustworthy Machine Learning by Kush R. Varshney

Outline of Today's Lecture

1. **Basics of deep learning**

2. **Deep learning for images**

3. **Deep learning for sequences**

Outline of Today's Lecture

1. **Basics of deep learning**

2. Deep learning for images

3. Deep learning for sequences

Basics of Deep Learning

Machine Learning as Optimization (1)

- **Supervised learning:** we are given input $\boldsymbol{x}$, and the goal is to predict label $\boldsymbol{y}$
- **Input $\boldsymbol{x}$ can be:**
  - Vectors of real numbers
  - Sequences (natural language)
  - Matrices (images)
  - Graphs (potentially with node and edge features)
- **We formulate the task as an optimization problem**

How Much Information is the Machine Given during Learning?

- "Pure" Reinforcement Learning (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

Machine Learning as Optimization (2)

- **Formulate the task as an optimization problem:**

$$\min_{\Theta} \boxed{\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))}$$

Objective function

- Θ: a set of **parameters** we optimize
  - Could contain one or more scalars, vectors, matrices …
  - E.g. $\Theta = \{Z\}$ in the shallow encoder (the embedding lookup)
- $\mathcal{L}$: **loss function**. Example: L2 loss

$$\mathcal{L}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = \|y - f(\boldsymbol{x})\|_2$$

  - Other common loss functions:
    - L1 loss, huber loss, max margin (hinge loss), cross entropy …
    - See https://pytorch.org/docs/stable/nn.html#loss-functions

Loss Function Example: Cross Entropy (1)

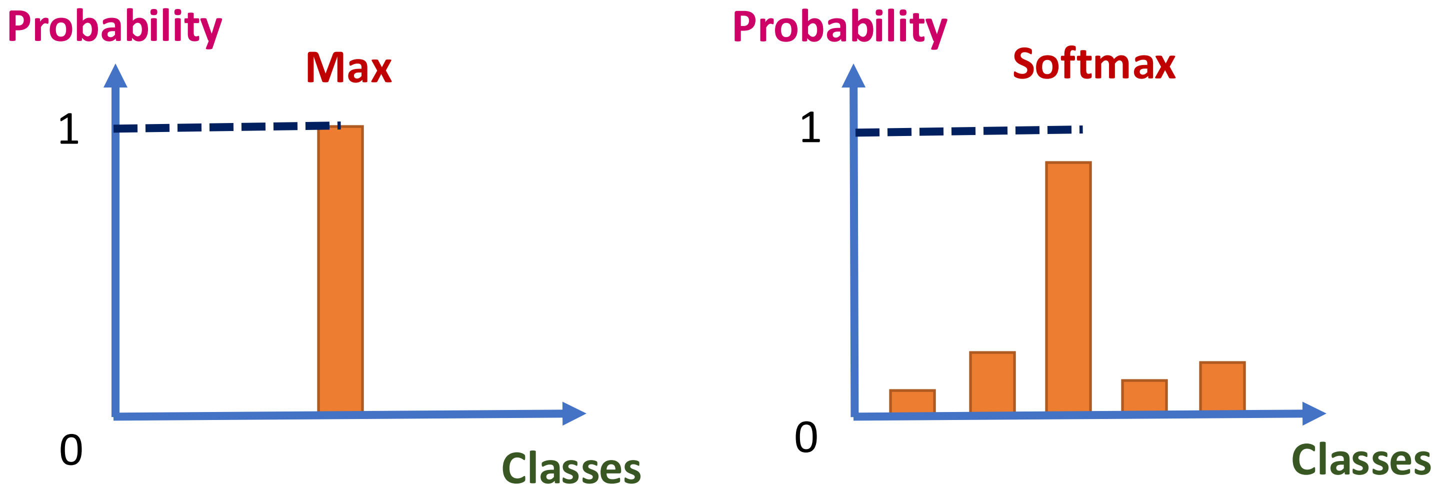
- One common loss for **classification**: cross entropy (CE). Supposed that:
- $f(\boldsymbol{x})$ is the output of a model
  - E.g. $f(\boldsymbol{x}) = [0.1, 0.1, 0.6, 0.2, 0]$
- Label $\boldsymbol{y}$ is a categorical vector (**one-hot** encoding)
  - E.g. $y = [0, 0, 1, 0, 0]^T$ y is of class "3"

- $\text{Softmax}(f(\boldsymbol{x}))_i = \frac{e^{f(\boldsymbol{x})_i}}{\sum_{j=1}^{C} e^{f(\boldsymbol{x})_j}}$ ⟶ $f(\boldsymbol{x})_i$ denotes i-th coordinate of the vector $f(\boldsymbol{x})$
  - Where C is the number of classes. ($C = 5$ in this example)
  - E.g. $f(\boldsymbol{x}) = [0.1767, 0.1767, 0.2914, 0.1953, 0.1599]^T$

Softmax

- Softmax is a **differentiable** (or soft) version of the max function

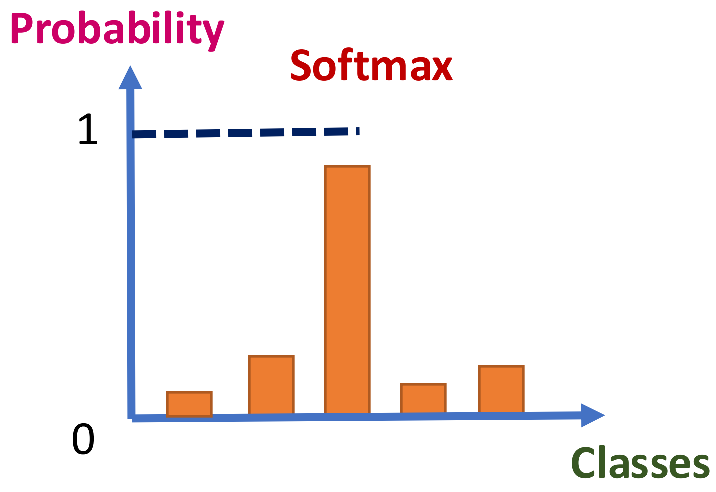
- $\text{Softmax}(f(\boldsymbol{x}))_i = \dfrac{e^{f(\boldsymbol{x})_i}}{\sum_{j=1}^{C} e^{f(\boldsymbol{x})_j}}$



Question: if we want to explain such a deep learning model, should we care about class or should we care about logit?

Loss Function Example: Cross Entropy (2)

- $\mathrm{CE}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = -\sum_{i=1}^{C}\left(\boldsymbol{y}_i \log f(\boldsymbol{x})_i\right)$
  - $\boldsymbol{y}_i, f(\boldsymbol{x})_i$ are the **actual** and **predicted** value of the i-th class.
  - **Intuition:** the lower the loss, the closer the prediction is to one-hot
- In classification, $\boldsymbol{y}$ is **one-hot**, whereas $f(\boldsymbol{x})$ is the output of a softmax
  - The summation in CE only has **1 non-zero term**
- Total loss over all training examples
  - $\mathcal{L} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{T}} \mathrm{CE}\big(\boldsymbol{y}, f(\boldsymbol{x})\big)$
  - $\mathcal{T}$: training set containing all pairs of data and labels $(\boldsymbol{x}, \boldsymbol{y})$



How is the probability useful in model trustworthiness?

Machine Learning as Optimization (1)

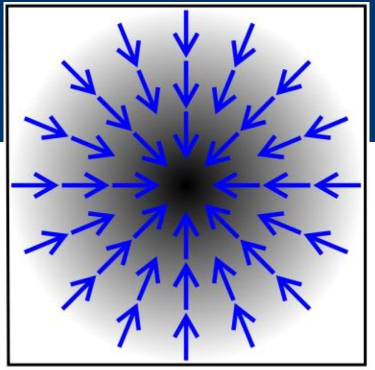
- **How to optimize the objective function?**
- Non-gradient approaches
  - Bayesian optimization, Gaussian processes, Simulated annealing, Evolutionary algorithms

In deep learning, we use gradient-based optimization for scalability

- Therefore, we require the loss function $\mathcal{L}$ to be **differentiable**
  - There are ways to tackle optimization for non-differentiable functions:
    - Straight-through estimator (Gumbel Softmax)
    - Reinforce algorithm, or more generally, reinforcement learning (algorithms to solve MDPs)

What are the pros and cons in terms of trustworthy AI for gradient-based methods?

Machine Learning as Optimization (2)



https://en.wikipedia.org/wiki/Gradient

- **How to optimize the objective function?**
- **Gradient vector:** Direction and rate of fastest increase

$$\nabla_{\Theta}\mathcal{L} = \left(\frac{\partial\mathcal{L}}{\partial\Theta_1}, \frac{\partial\mathcal{L}}{\partial\Theta_2}, \ldots\right) \longleftarrow \textbf{Partial derivative}$$

  - Θ_1, Θ_2 … : components of Θ
- Recall **directional derivative** of a multi-variable function (e.g. $\mathcal{L}$) along a given vector represents the instantaneous rate of change of the function along the vector.
- Gradient is the directional derivative in the **direction of largest increase**

Gradient Descent

- **Iterative algorithm:** repeatedly update weights in the (opposite) direction of gradients until convergence

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}$$

- **Training:** Optimize Θ iteratively
  - **Iteration**: 1 step of gradient descent
- **Learning rate (LR) η:**
  - Hyperparameter that controls the size of gradient step
  - Can vary over the course of training (LR scheduling)
- **Ideal termination condition: 0 gradient**
  - In practice, we stop training if it no longer improves performance on **validation set** (part of dataset we hold out from training)

Stochastic Gradient Descent (SGD)

- **Problem with gradient descent:**
  - Exact gradient requires computing $\nabla_{\Theta}\mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$, where $\boldsymbol{x}$ is the **entire** dataset!
    - This means summing gradient contributions over all the points in the dataset
    - Modern datasets often contain billions of data points
    - Extremely expensive for every gradient descent step

- **Solution: Stochastic gradient descent (SGD)**
  - At every step, pick a different **minibatch** $\boldsymbol{\mathcal{B}}$ containing a subset of the dataset, use it as input $\boldsymbol{x}$

Minibatch SGD

- **Concepts:**
  - **Batch size**: the number of data points in a minibatch
    - E.g. number of nodes for node classification task
  - **Iteration**: 1 step of SGD on a minibatch
  - **Epoch**: one full pass over the dataset (# iterations is equal to ratio of dataset size and batch size)
- SGD is unbiased estimator of full gradient:
  - But there is no guarantee on the rate of convergence
  - In practice often requires tuning of learning rate
- Common optimizer that improves over SGD:
  - Adam, Adagrad, Adadelta, RMSprop …

How is the gradient useful in model trustworthiness?

Neural Network Function (1)

- **Objective:** $\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$
- In deep learning, the function f can be very complex
- To start simple, consider linear function

$$f(\boldsymbol{x}) = W \cdot \boldsymbol{x}, \qquad \Theta = \{W\}$$

- If f returns a scalar, then W is a learnable **vector**

$$\nabla_W f = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3} \dots\right)$$

- If f returns a vector, then W is the **weight matrix**

$$\nabla_W f = W^T$$

Neural Network Function (2)

Derivative of f w.r.t. X	Scalar	Vector	Matrix
Scalar	Scalar	Vector	Matrix
Vector	Vector	Matrix	Tensors
Matrix	Matrix	Tensors	Tensors

Jacobian matrix of f

Back-propagation

- **How about a more complex function:**

$$f(\boldsymbol{x}) = a = W_2(\underbrace{W_1\boldsymbol{x}}_{\boldsymbol{z}}), \qquad \Theta = \{W_1, W_2\}$$

- Recall **chain rule**:
- E.g. $\nabla_{\boldsymbol{x}} f = \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial \boldsymbol{x}}$

We define:

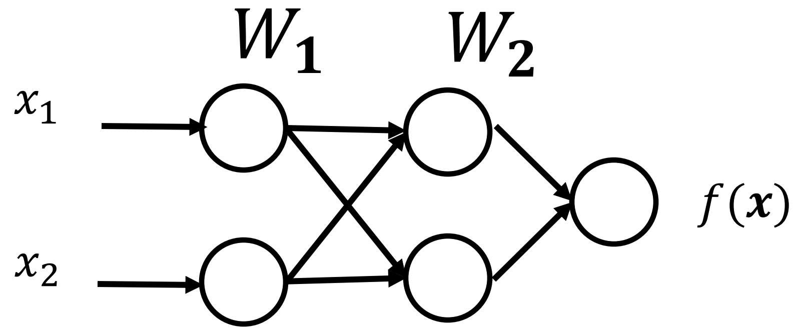
$\boldsymbol{z} = W_1\boldsymbol{x}$

$a = f(\boldsymbol{x}) = W_2\boldsymbol{z}$

- **Back-propagation**: Use of **chain rule** to propagate gradients of intermediate steps, and finally obtain gradient of $\mathcal{L}$ w.r.t. Θ

Back-propagation Example (1)

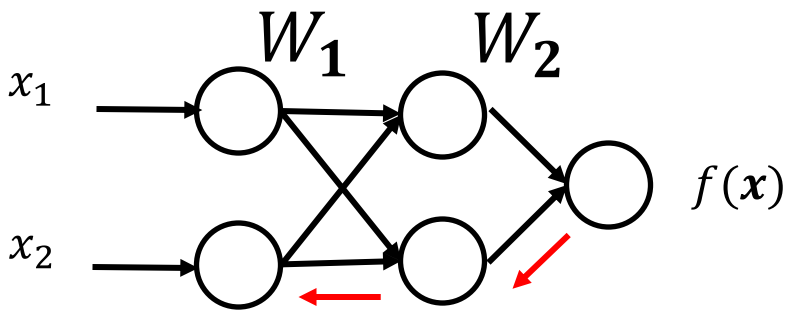
- **Example:** Simple 2-layer linear network, **regression** task
- $f(\boldsymbol{x}) = a = W_2\boldsymbol{z} = W_2(\underbrace{W_1\boldsymbol{x}}_{\boldsymbol{z}})$
- $\mathcal{L} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{B}} \left\|\left(\boldsymbol{y} - f(\boldsymbol{x})\right)\right\|_{\mathbf{2}}$ sums the L2 loss in a minibatch $\mathcal{B}$
- **Hidden layer:** intermediate representation for input $\boldsymbol{x}$
  - Here we use $\boldsymbol{z} = W_1\boldsymbol{x}$ to denote the hidden layer



Back-propagation Example (2)

- **Forward propagation:**
  Compute loss starting from input
  - $\boldsymbol{x} \longrightarrow \boldsymbol{z} \longrightarrow a \longrightarrow \mathcal{L}$

    Multiply W_1 Multiply W_2 Loss



- **Back-propagation to compute gradient of**
  $\Theta = \{W_1, W_2\}$

Remember:

$f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$

$\boldsymbol{z} = W_1\boldsymbol{x}$

$a = W_2\boldsymbol{z}$

- Start from loss, compute the gradient

$$\frac{\partial \mathcal{L}}{\partial W_2} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial W_2}, \qquad \frac{\partial \mathcal{L}}{\partial W_1} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial \boldsymbol{z}} \cdot \frac{\partial \boldsymbol{z}}{\partial W_1}$$

Compute backwards **Compute backwards**

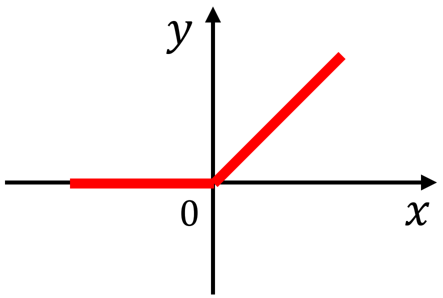
How about $\frac{\partial \mathcal{L}}{\partial x}$?

Non-linearity

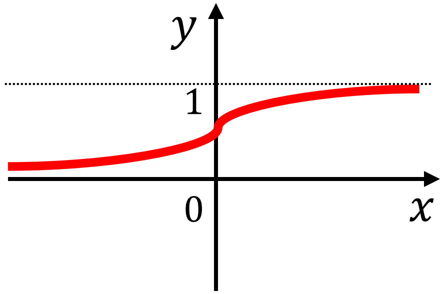
- Note that in $f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$, W_2W_1 is another matrix (or vector, if we do binary classification and output only 1 logit)
- Hence $f(\boldsymbol{x})$ is still linear w.r.t. $\boldsymbol{x}$ no matter how many weight matrices we compose
- **Introduce non-linearity:**
  - **Rectified linear unit (ReLU)**

    $ReLU(x) = \max(x, 0)$
  - **Sigmoid**

    $\sigma(x) = \frac{1}{1+e^{-x}}$



ReLU



Sigmoid

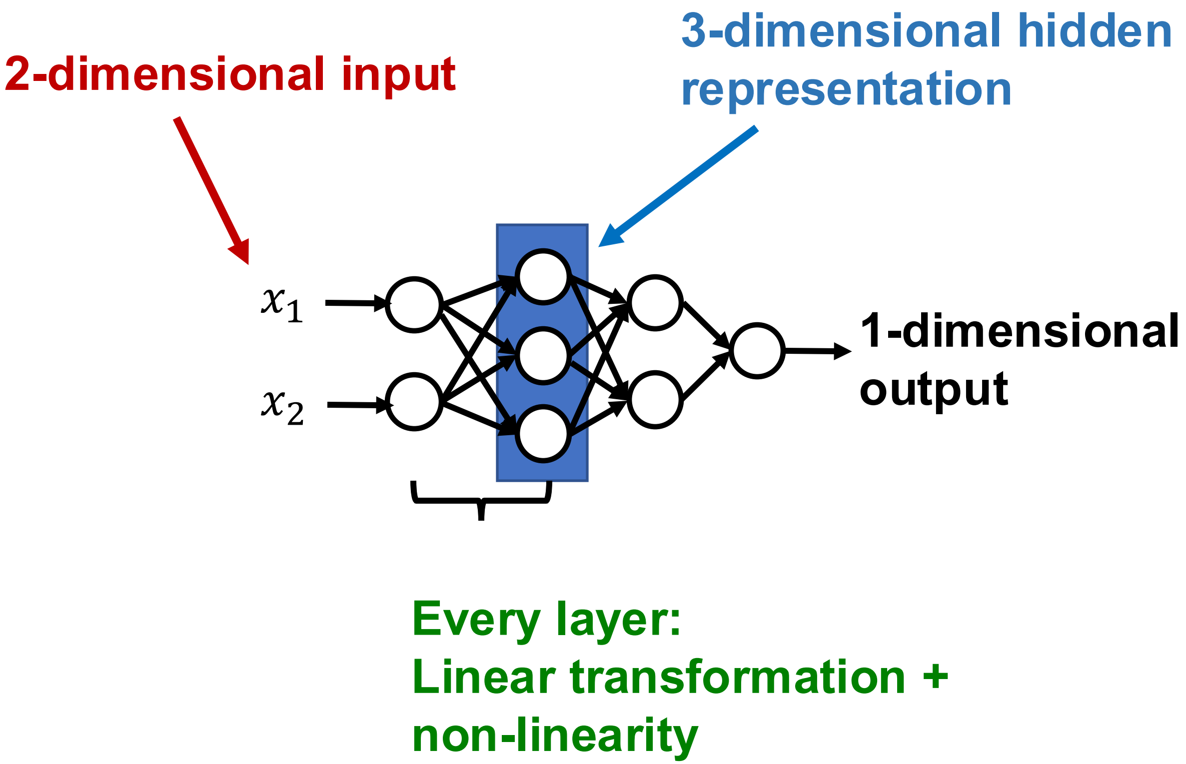
What's the implication of non-linearity in model trustworthiness

Multi-layer Perceptron (MLP)

- **Each layer of MLP combines linear transformation and non-linearity:**

$$\boldsymbol{x}^{(l+1)} = \sigma(W_l \boldsymbol{x}^{(l)} + b^l)$$

  - where W_l is weight matrix that transforms hidden representation at layer l to layer $l+1$
  - b^l is bias at layer l, and is added to the linear transformation of $\boldsymbol{x}$
  - σ is non-linearity function (e.g., sigmoid)
- Suppose $\boldsymbol{x}$ is 2-dimensional, with entries x_1 and x_2



Why Going Deep?

- Many classical ML models are linear (1-layer)
- Model neural networks are typically characterized by **deep architectures** (varies from 3 to thousands of layers depending on use cases)
- Why more than 1 non-linearity?

Universal approximation theorem: Let $C(X, Y)$ denote the set of continuous functions from X to Y. Let $\sigma \in C(\mathbb{R}, \mathbb{R})$. Note that $(\sigma \circ x)_i = \sigma(x_i)$, so $\sigma \circ x$ denotes σ applied to each component of x.

Then σ is not polynomial if and only if for every $n \in \mathbb{N}$, $m \in \mathbb{N}$, compact $K \subseteq \mathbb{R}^n$, $f \in C(K, \mathbb{R}^m), \varepsilon > 0$ there exist $k \in \mathbb{N}$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$, $C \in \mathbb{R}^{m \times k}$ such that

$$\sup_{x \in K} \|f(x) - g(x)\| \boxed{< \varepsilon}$$

Arbitrarily close approximation

where

$$g(x) = \underbrace{\boxed{C \cdot (\sigma \circ (\underbrace{\boxed{A \cdot x + b}}_{\text{Layer 1}}))}}_{\text{Layer 2}}$$

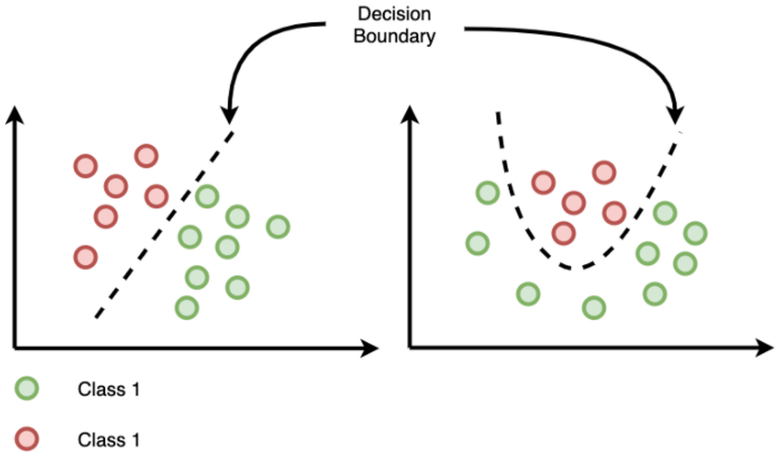
Layer 1

Layer 2

Caveat: the width (number of neurons) at hidden layer might need to be arbitrarily wide

Problem with Deep NNs

- The model has arbitrarily complex decision boundary

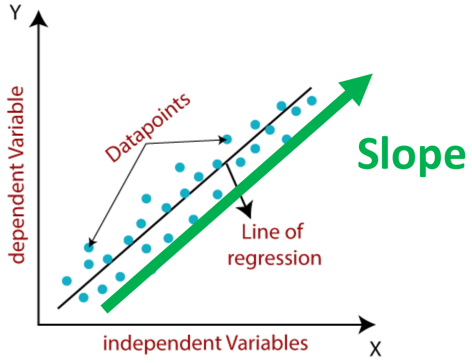


Way more complex with model architectures!

Vulnerable to attacks!

Linear boundary   polynomial boundary   3-layer MLP decision boundary

- The model is no longer interpretable
  - We can no longer use linear weights to understand the importance of each features



Summary

- **Objective function:**

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$$

- f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a minibatch of input $\boldsymbol{x}$
- **Forward propagation:** compute $\mathcal{L}$ given $\boldsymbol{x}$
- **Back-propagation:** obtain gradient $\nabla_{\Theta}\mathcal{L}$ using a chain rule
- Use **stochastic gradient descent (SGD)** to optimize for Θ over many iterations

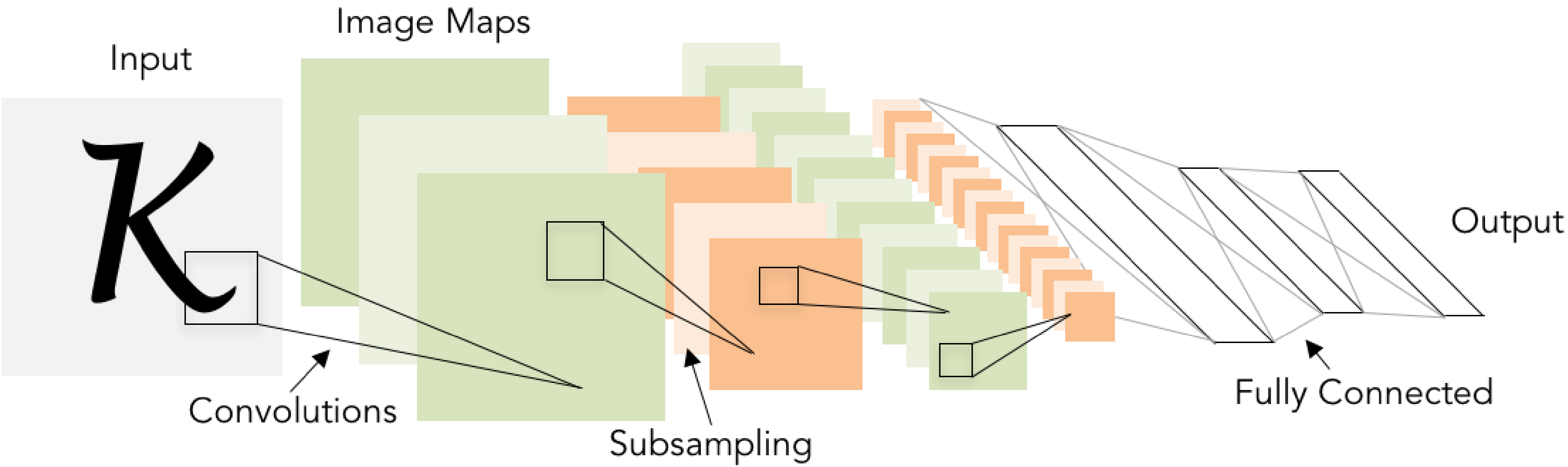
Outline of Today's Lecture

1. Basics of deep learning

2. **Deep learning for images**
   (Credit: Fei-Fei Li, Danfei Xu, Ranjay Krishna)

3. Deep learning for sequences

ConvNets



Gradient-based learning applied to document recognition*[LeCun, Bottou, Bengio, Haffner 1998]*

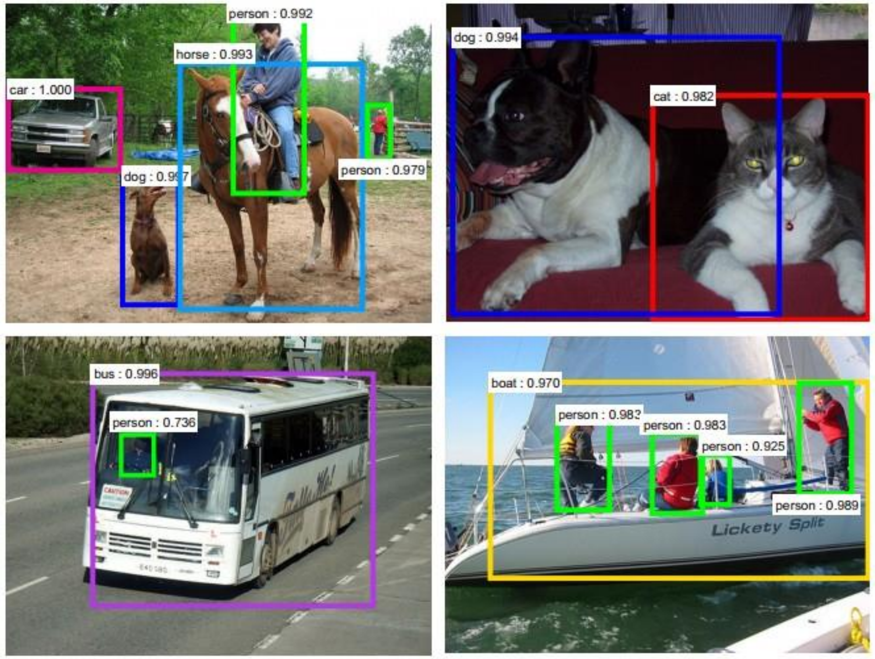
ConvNets are Everywhere

Classification

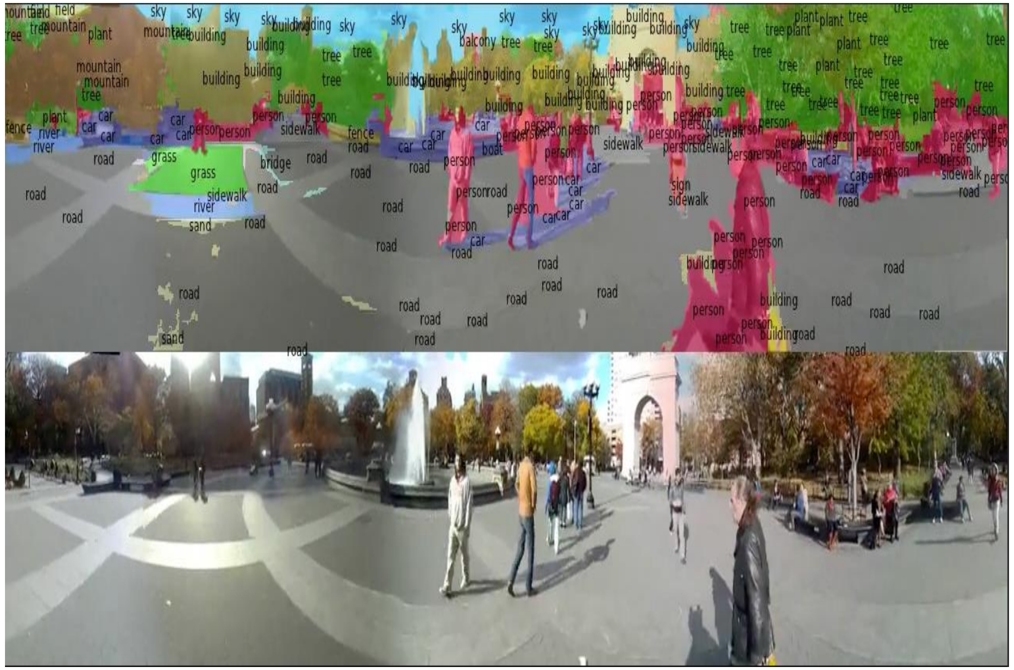
Retrieval

ConvNets are Everywhere

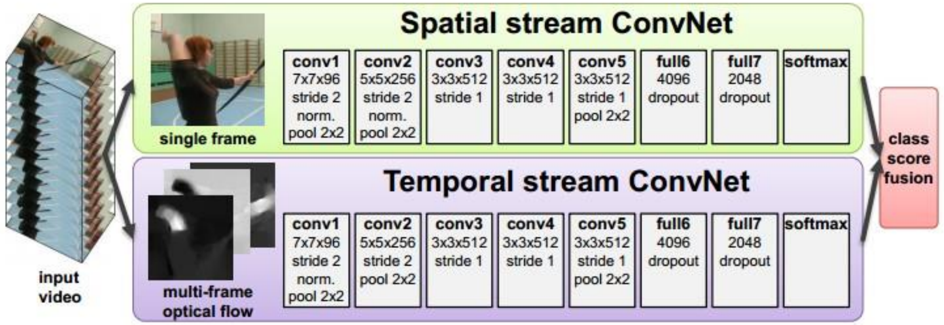
Detection



Segmentation



ConvNets Applications



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

[Dieleman et al. 2014]

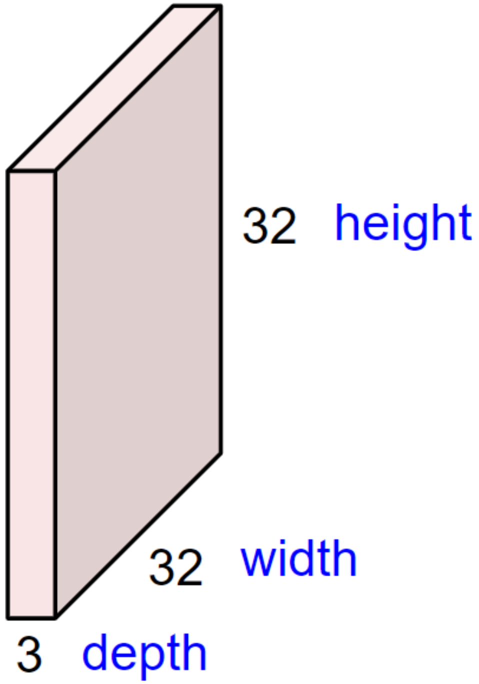
From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

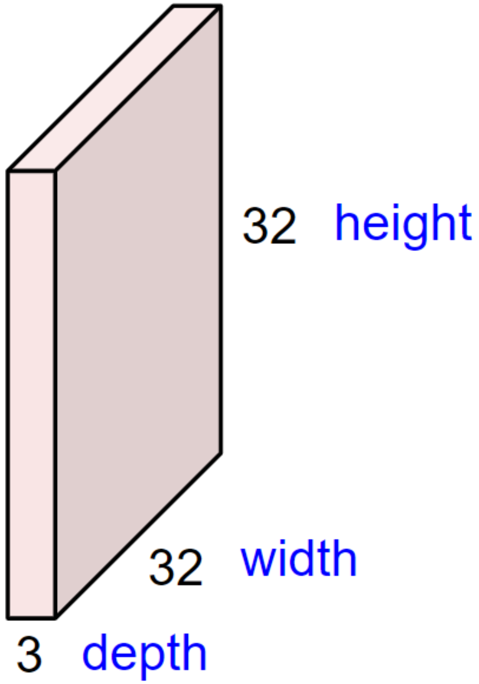
Convolution Layer

32x32x3 image -> preserve spatial structure

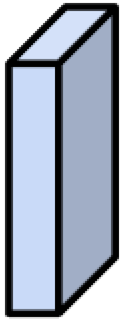


Convolution Layer

32x32x3 image -> preserve spatial structure



5x5x3 filter

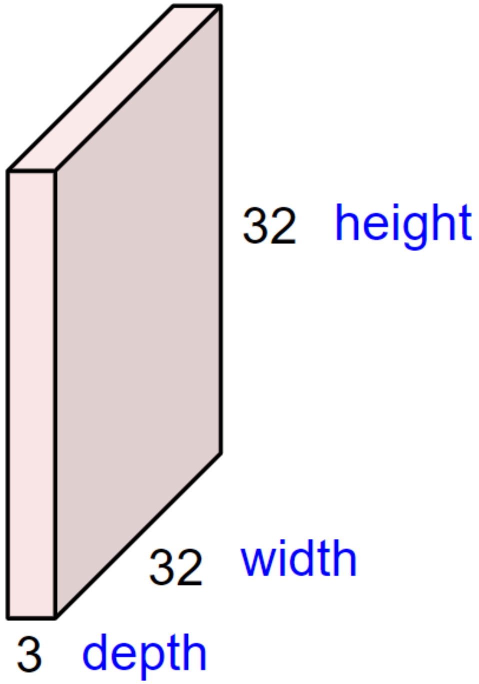


Convolve the filter with the imagei.e. “slide over the image spatially, computing dot products”

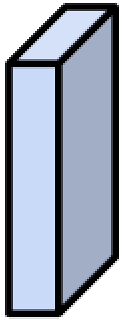
Convolution Layer

32x32x**3** image -> preserve spatial structure

Filters always extend the full depth of the input volume

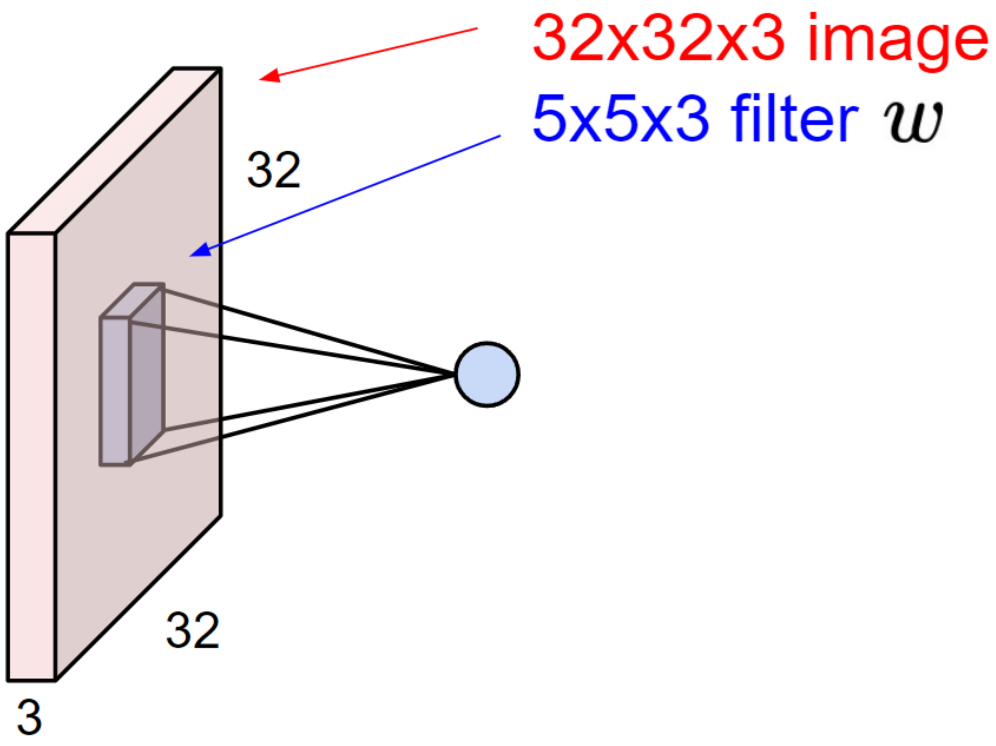


5x5x**3** filter



Convolve the filter with the imagei.e. “slide over the image spatially, computing dot products”

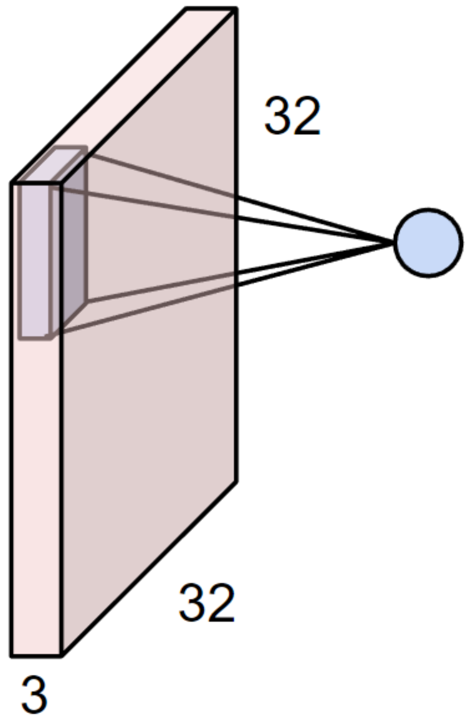
Convolution Layer



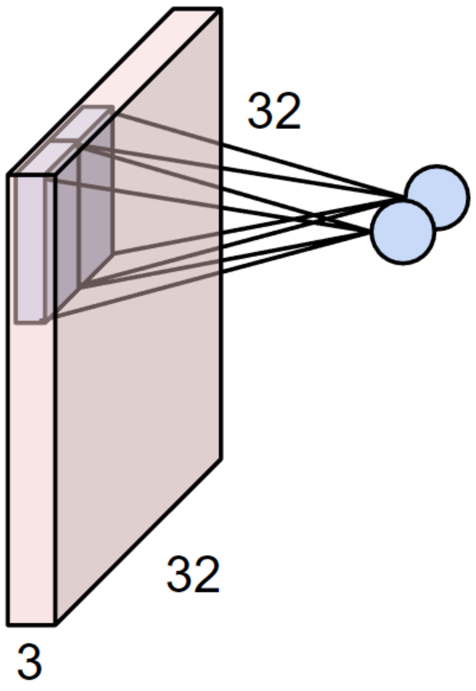
1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

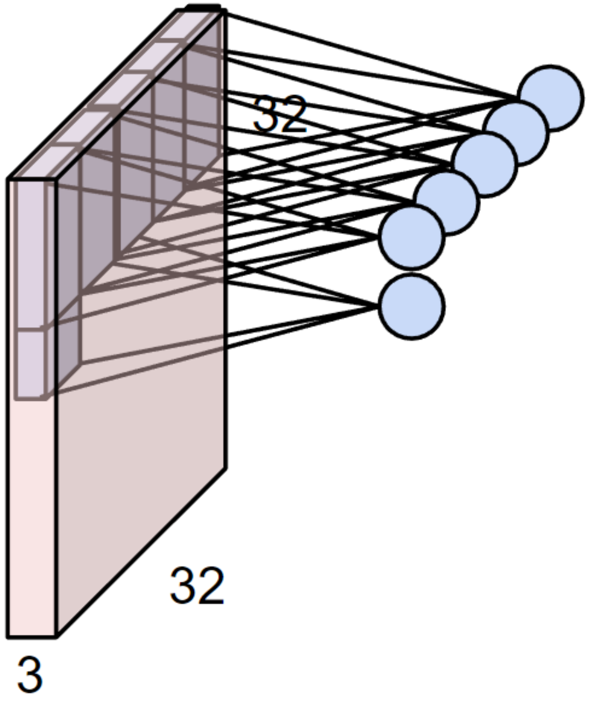
Convolution Layer



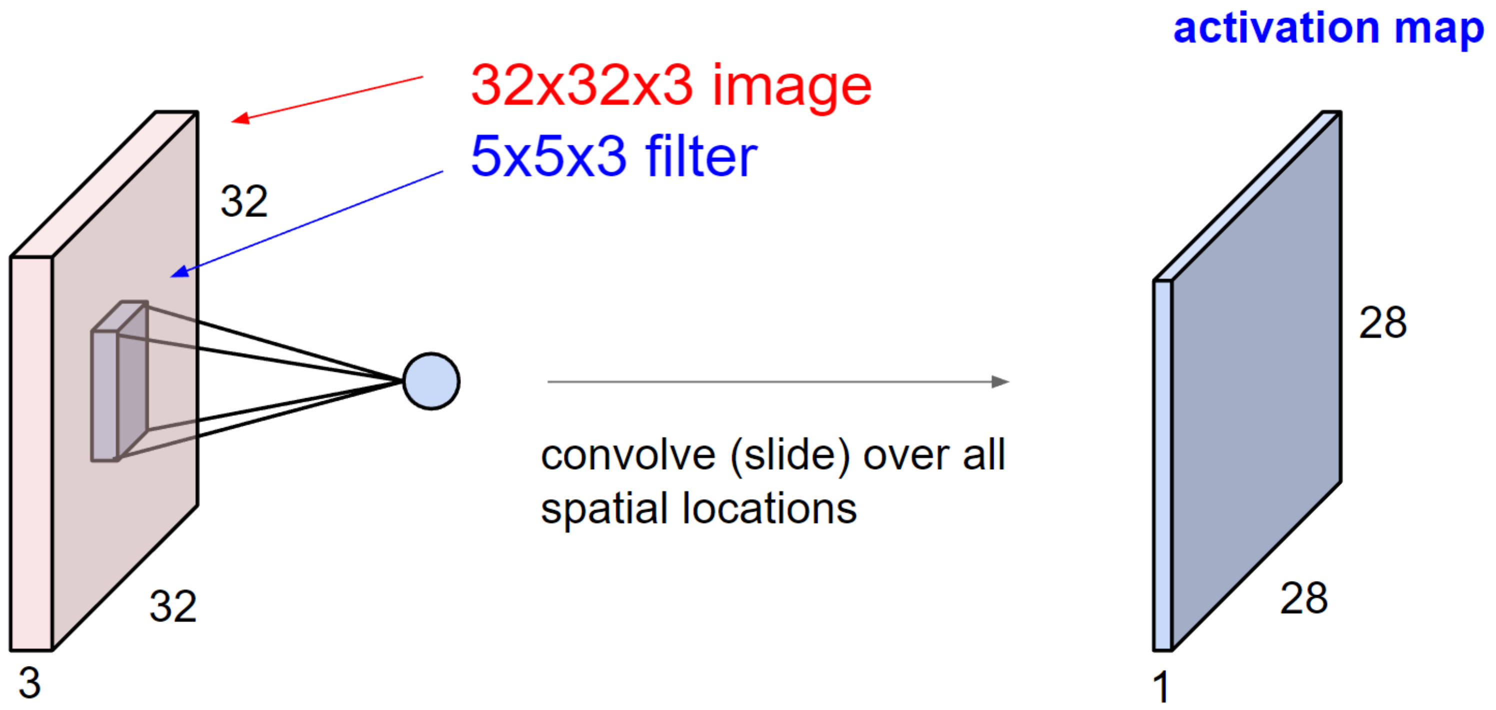
Convolution Layer



Convolution Layer

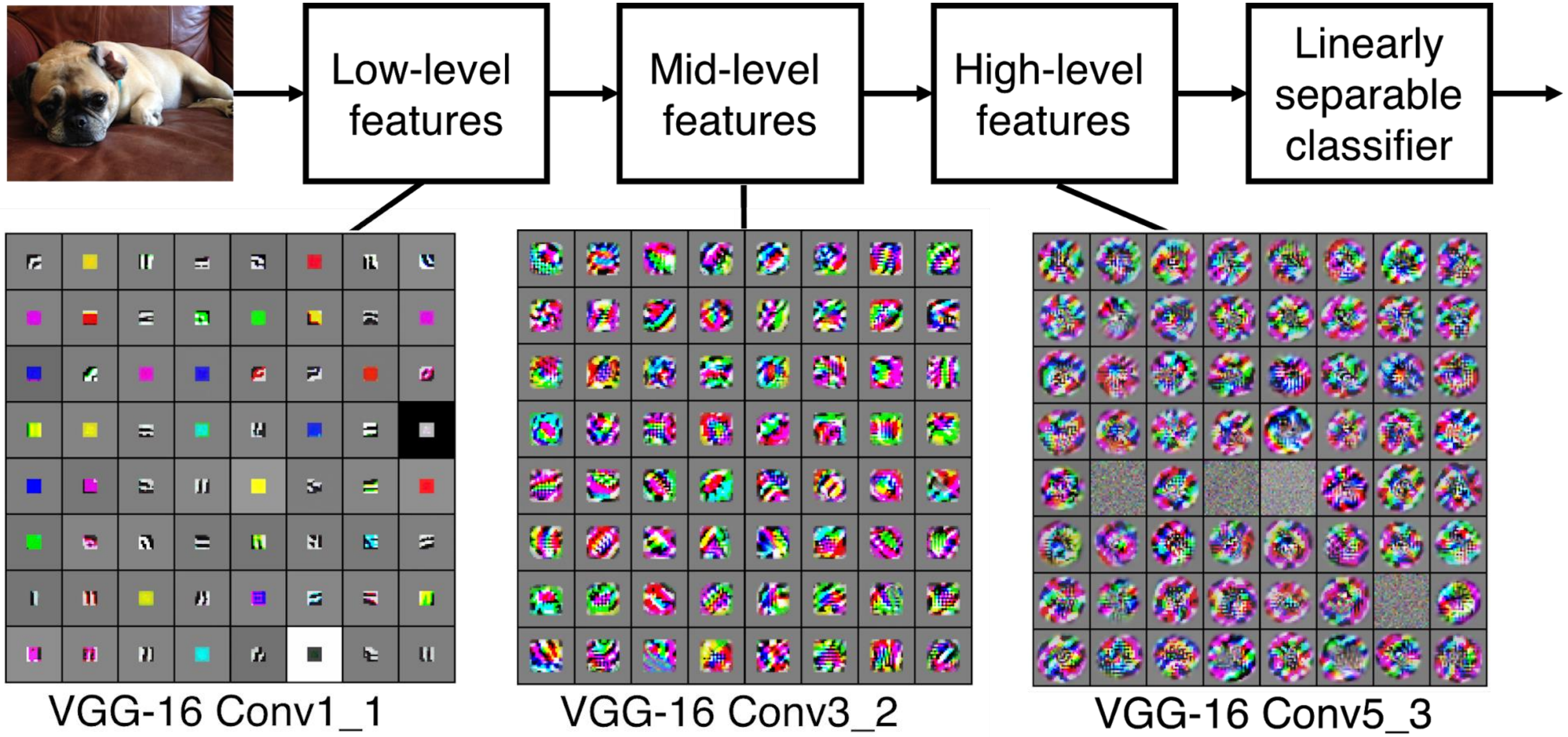


Convolution Layer



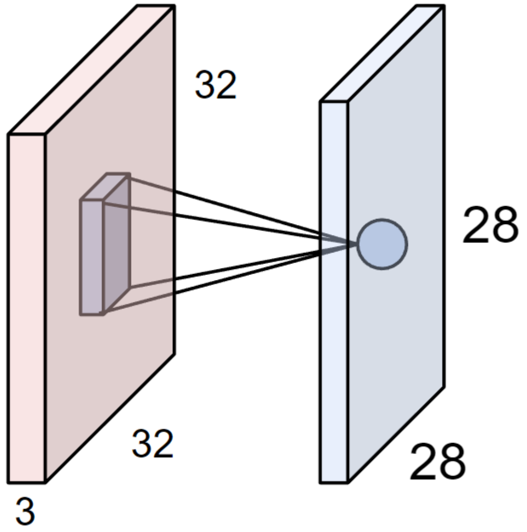
Interpretation

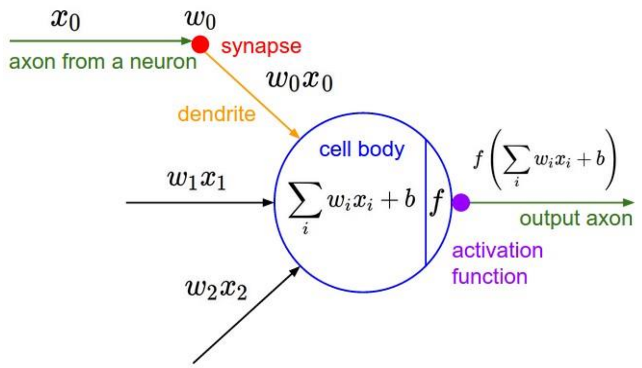
[Zeiler and Fergus 2013]
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



Which level do we usually look at for explaining ConvNets?

Receptive Field



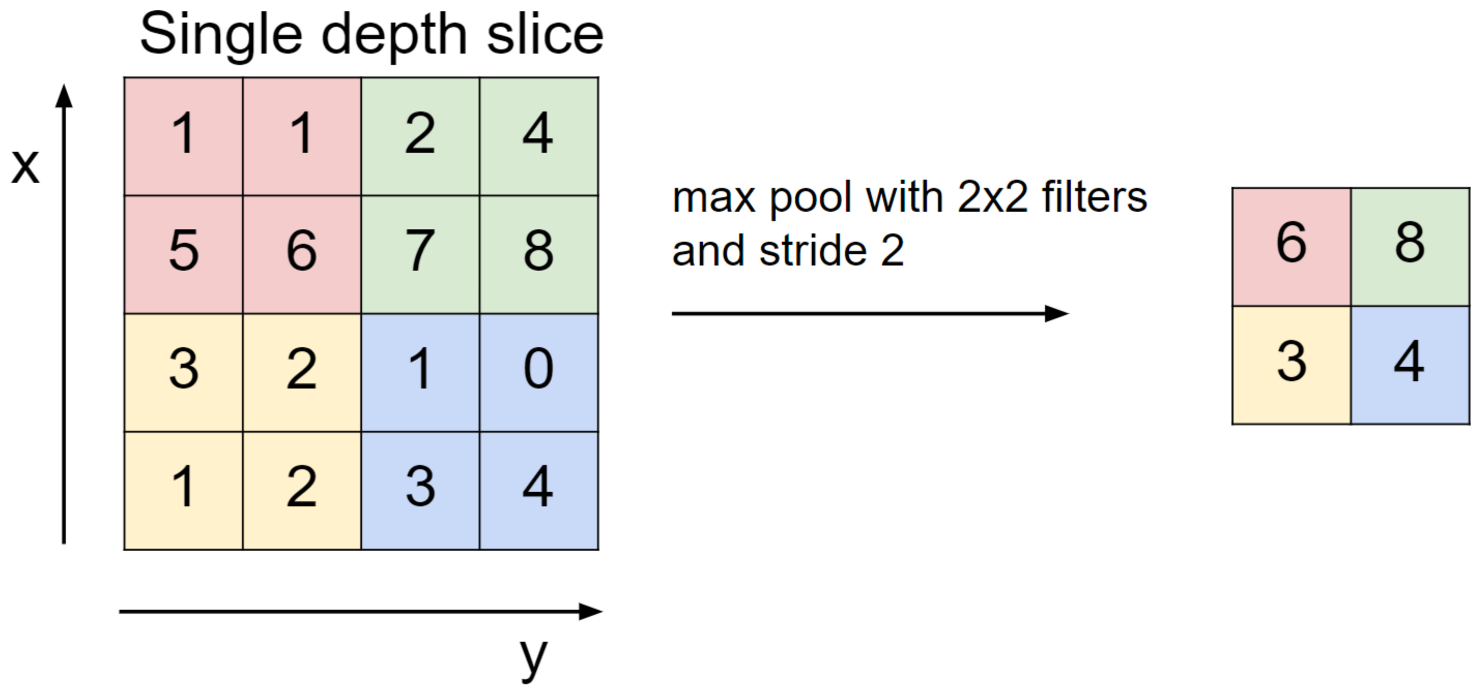


An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

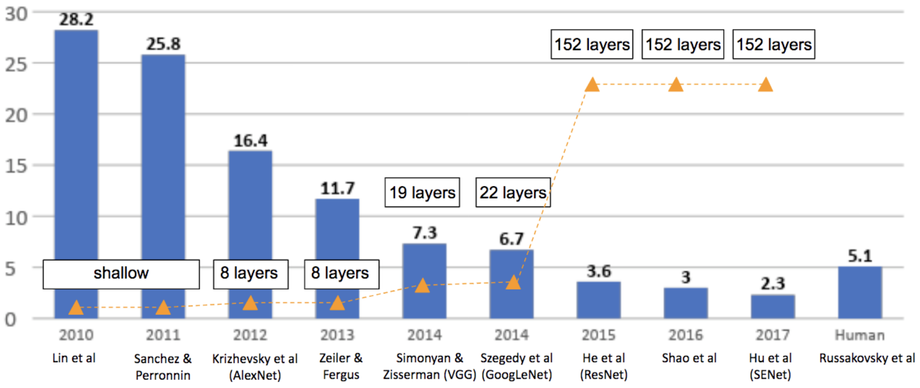
"5x5 filter" -> "5x5 receptive field for each neuron"

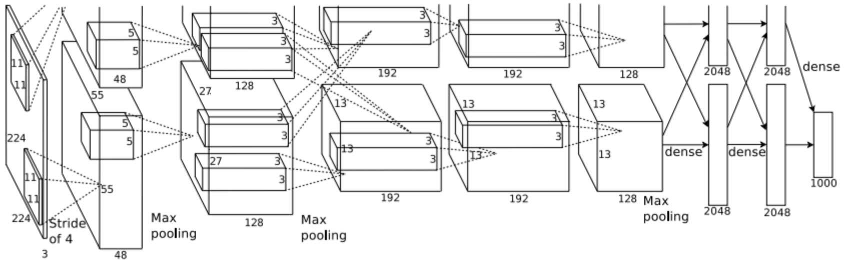
Max Pooling


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Variants of ConvNets

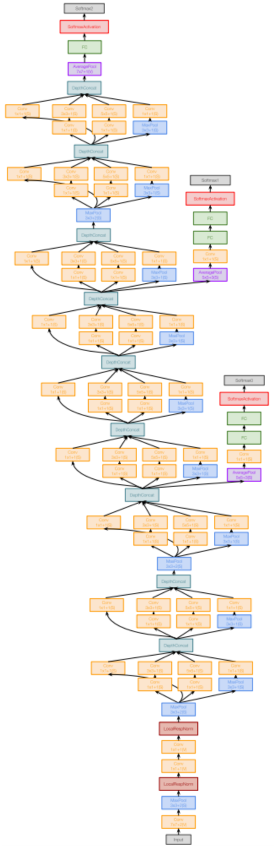
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



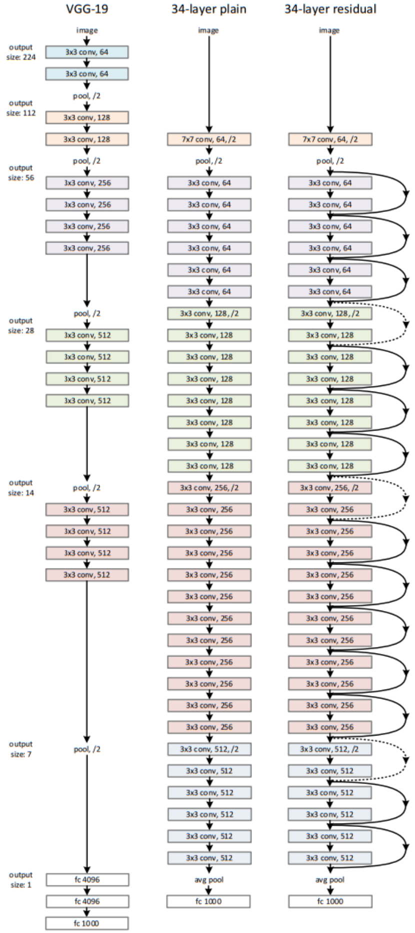


AlexNet

VGG16 VGG19



GoogLeNet



ResNet

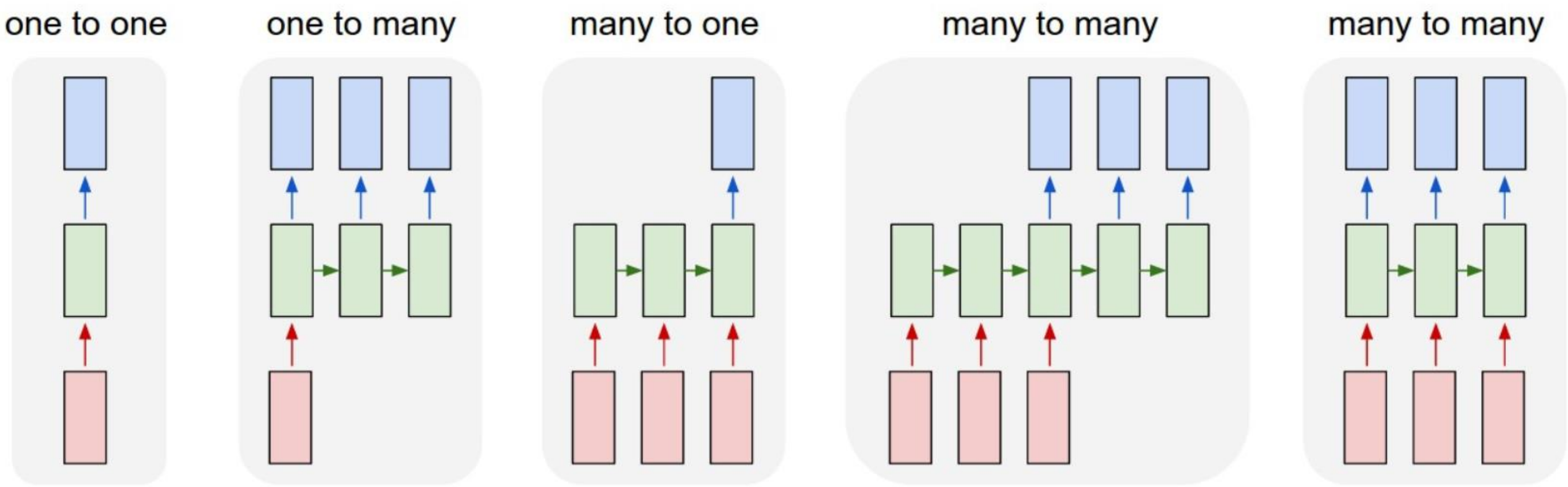
Outline of Today's Lecture

1. Basics of deep learning

2. Deep learning for images

3. **Deep learning for sequences**

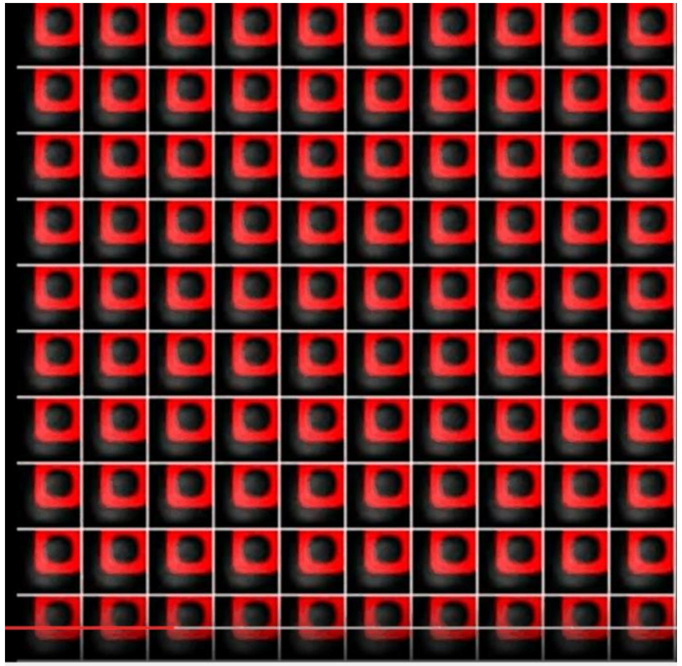
Processing Sequences

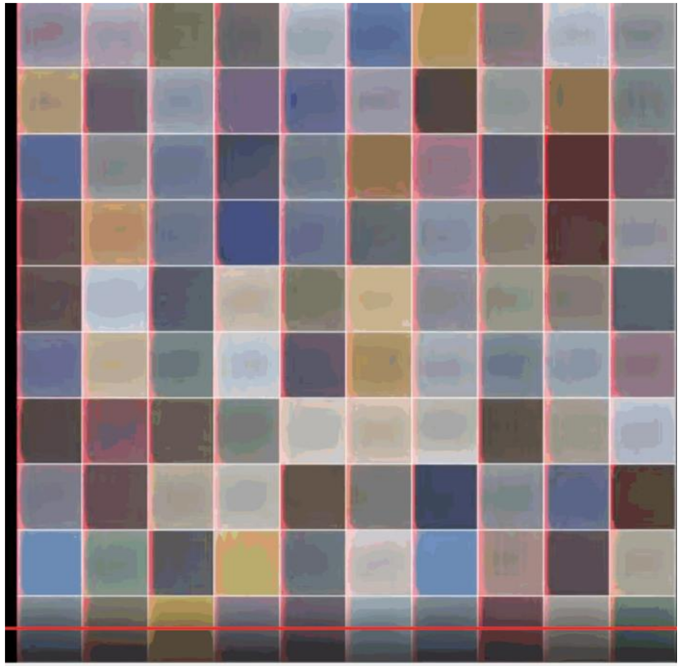


Sequential Processing of Non-Sequence Data

Classify images by taking a series of “glimpses”

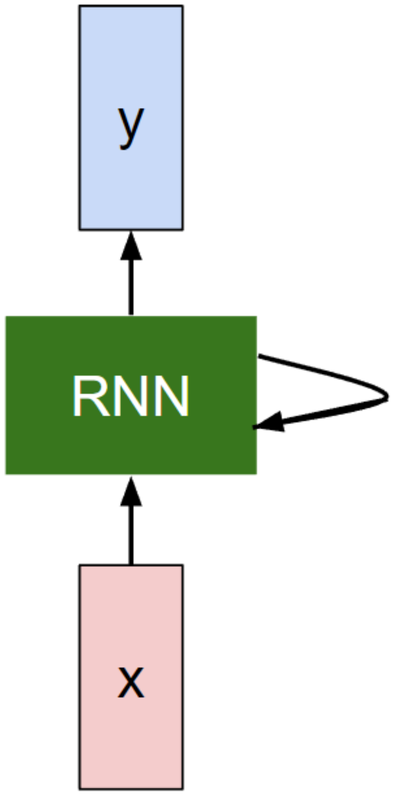
Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.





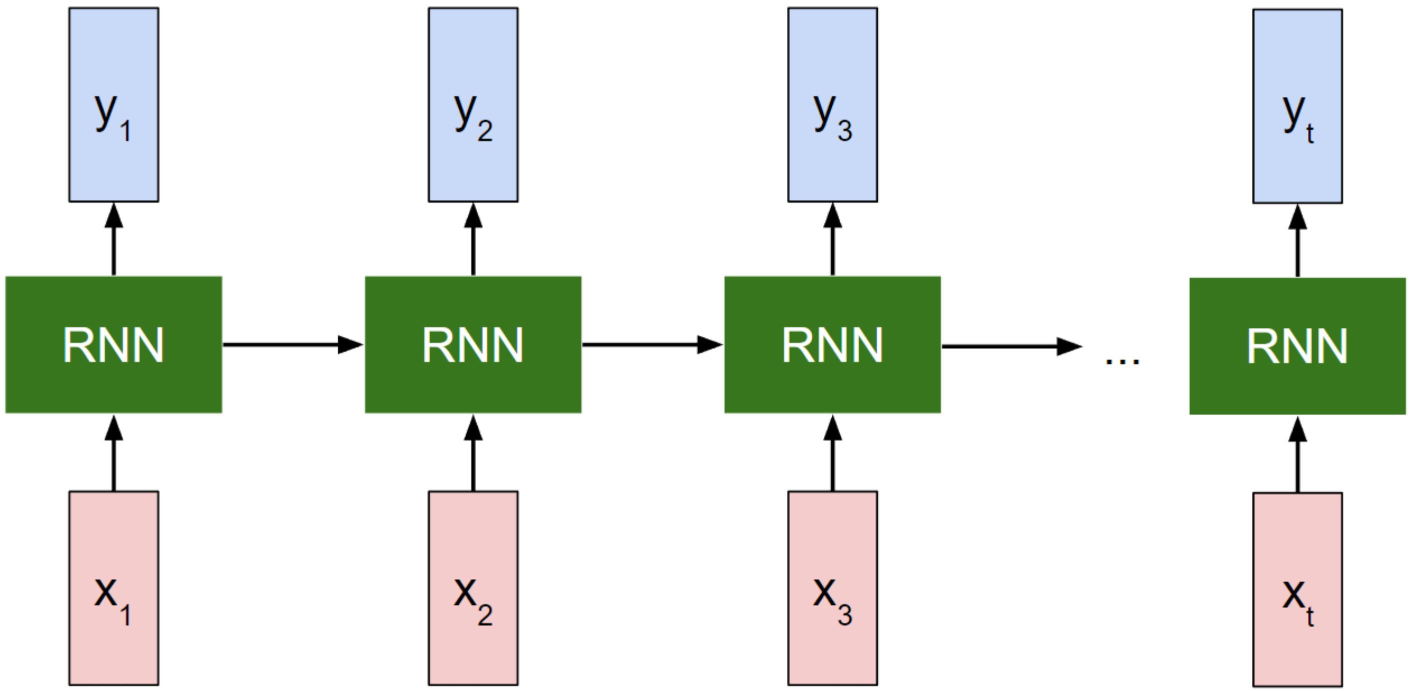
Gregor et al, "DRAW: A Recurrent Neural Network For Image Generation", ICML 2015

Recurrent Neural Network (RNN)



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Unrolling RNN



RNN Hidden State Update

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

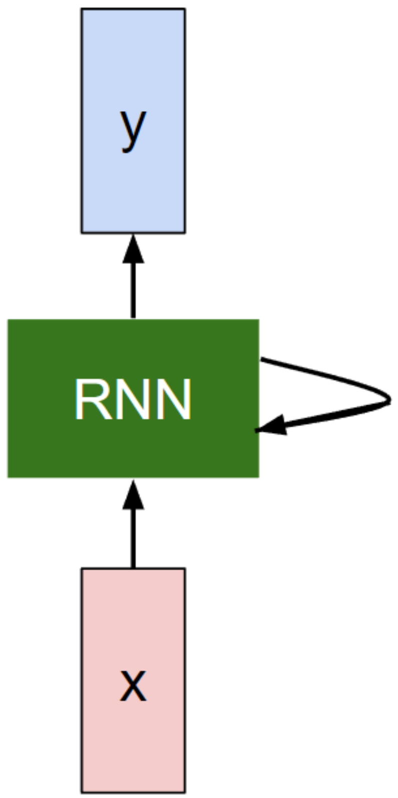
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

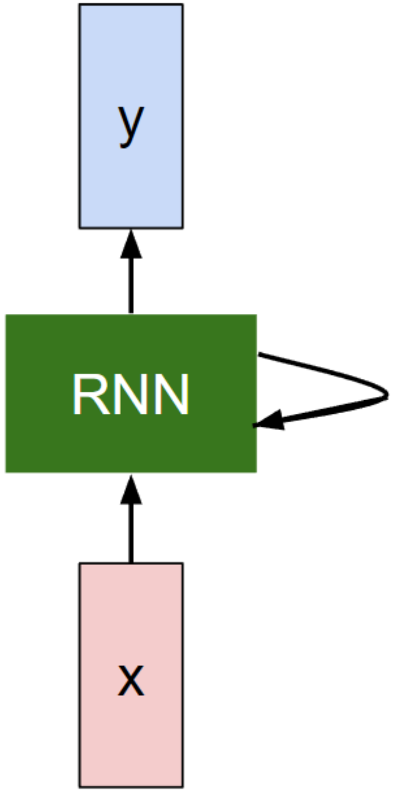
input vector at some time step



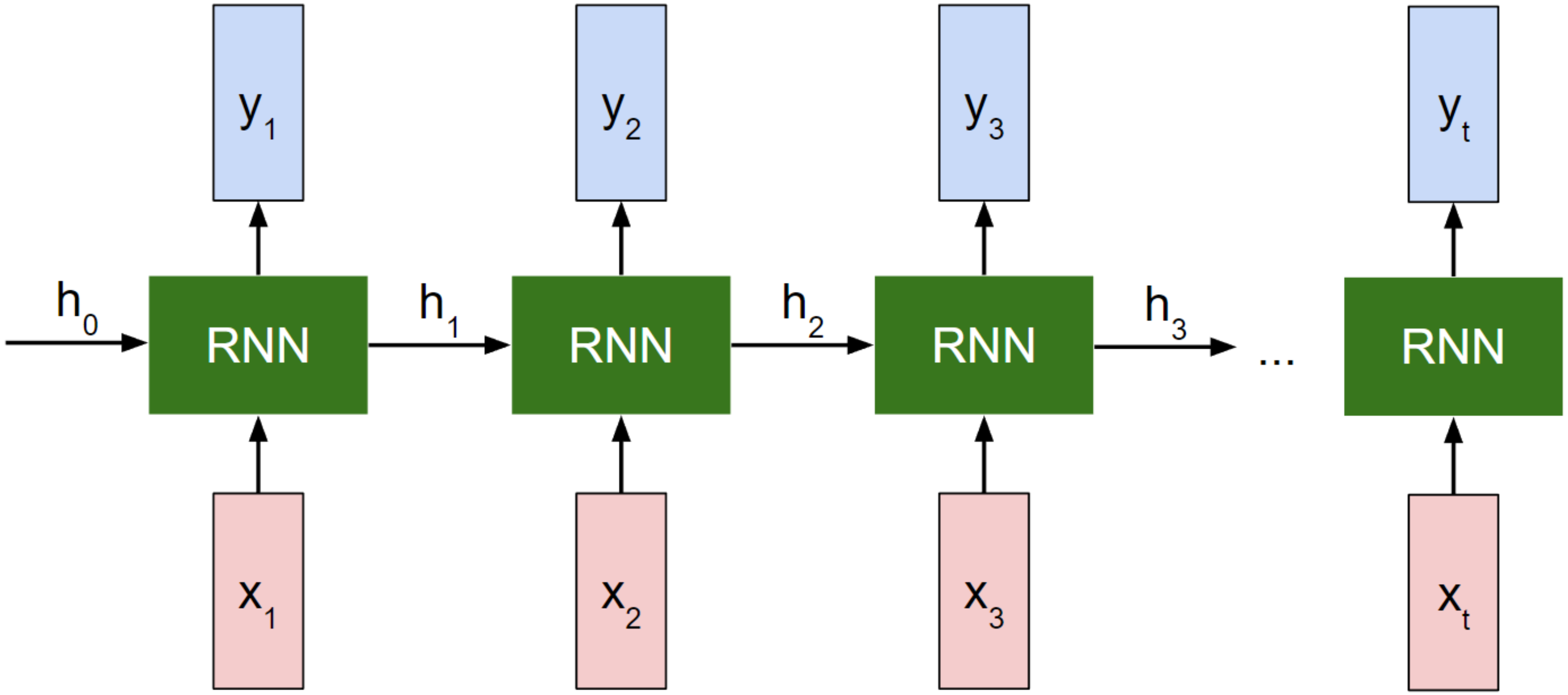
RNN Prediction Output

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

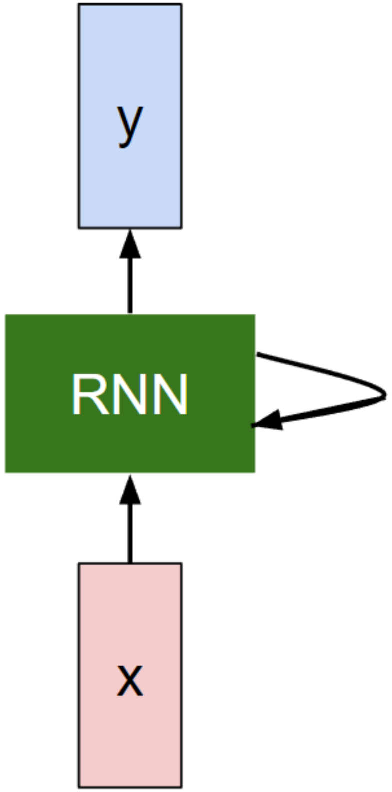
$$\underbrace{\boxed{y_t}}_{\text{output}} = \underbrace{\boxed{f_{W_{hy}}}}_{\text{another function with parameters } W_o}(\underbrace{\boxed{h_t}}_{\text{new state}})$$



RNN Architecture



Simple RNN Architecture Example



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

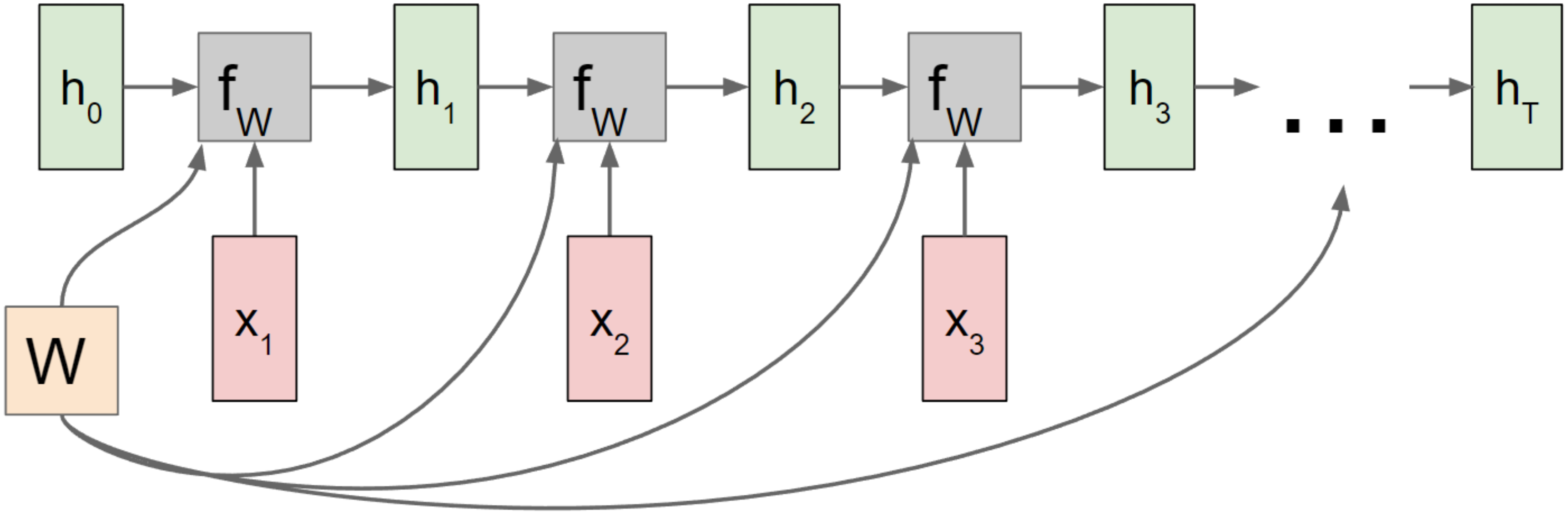
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$ Hidden State

$$y_t = W_{hy}h_t$$ Output State

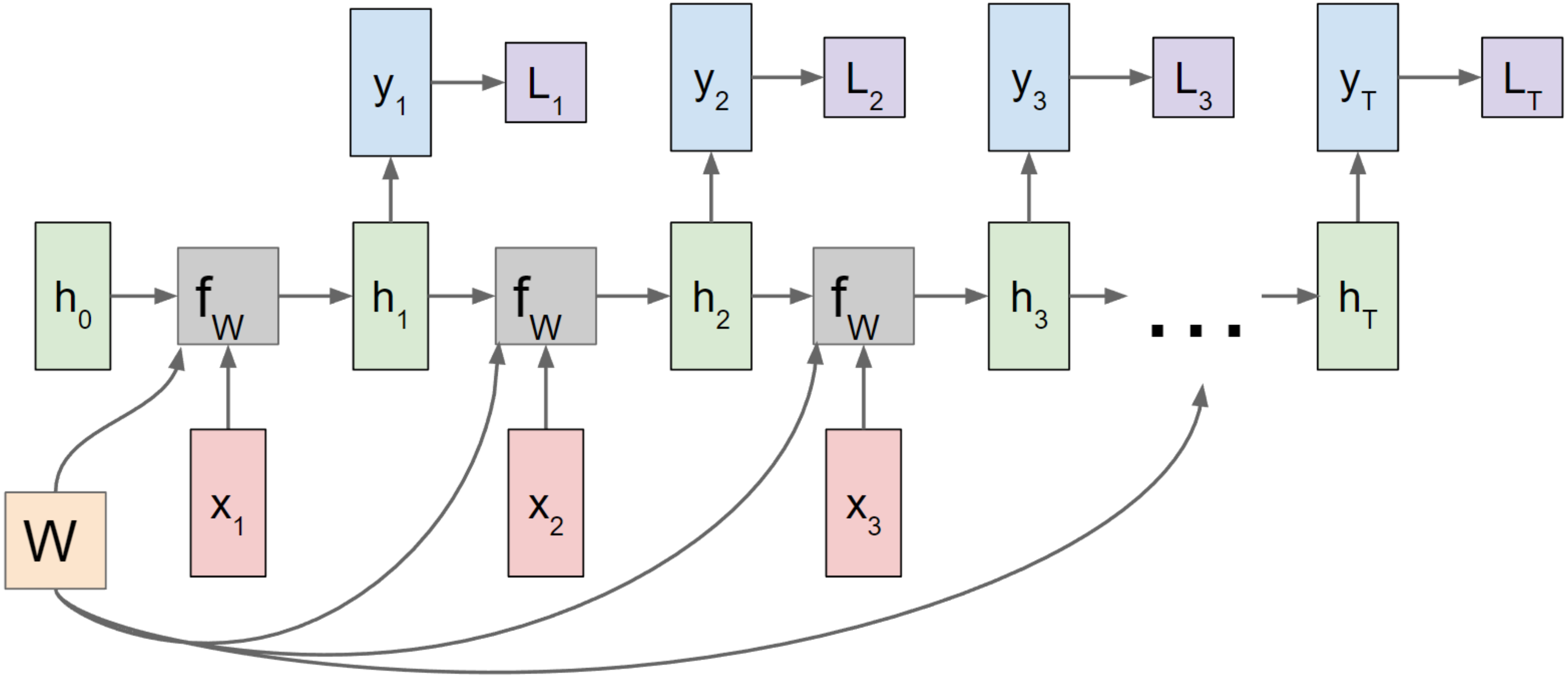
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN Computation Graph

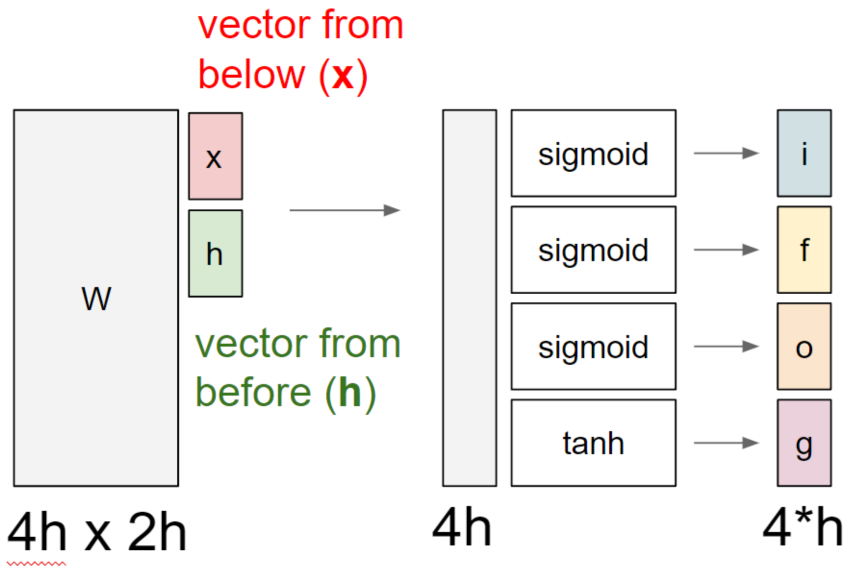
Re-use the same weight
matrix at every time-step



RNN Computation Graph: Many-to-many



Long Short Term Memory (LSTM)



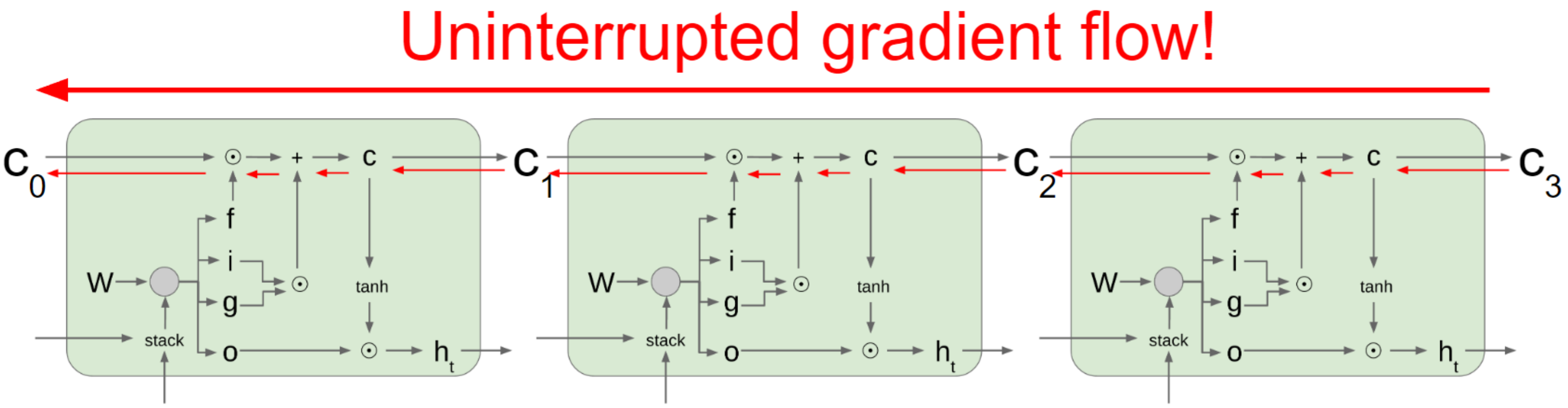
i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate, How much to write to cell

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

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

$$h_t = \boxed{o} \odot \boxed{\tanh}(c_t)$$

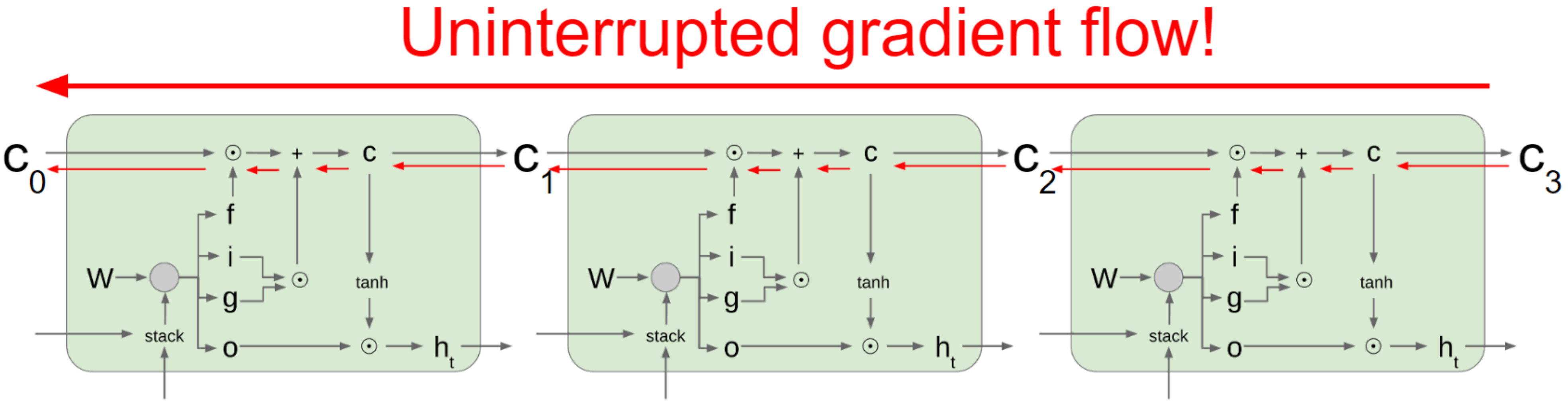
Gradient Flow (1)



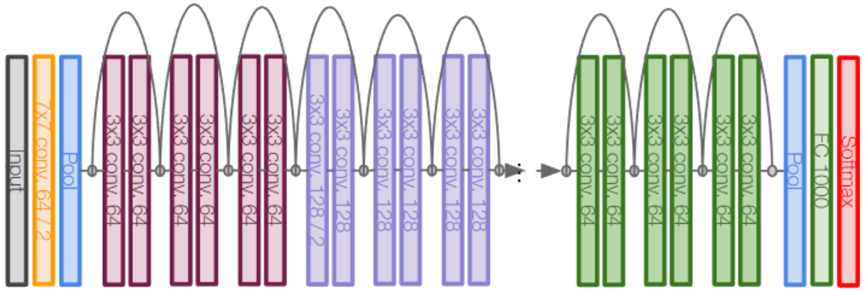
Notice that the gradient contains the **f** gate's vector of activations

- Allows better **control of gradients values**, using suitable parameter updates of the forget gate.
- Gradients are added through the **f, i, g,** and **o** gates

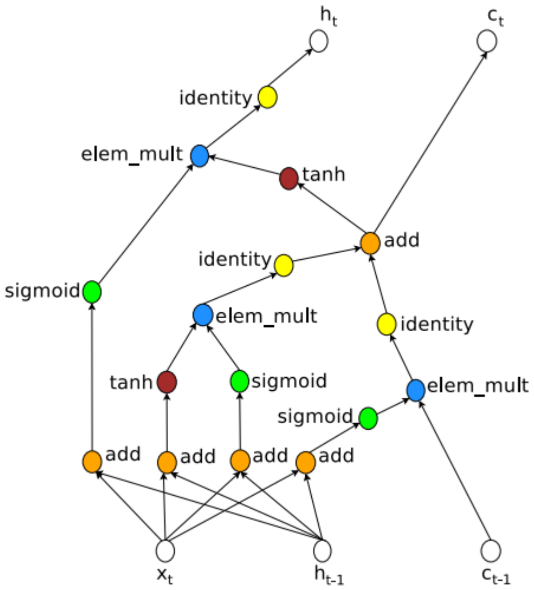
Gradient Flow (2)



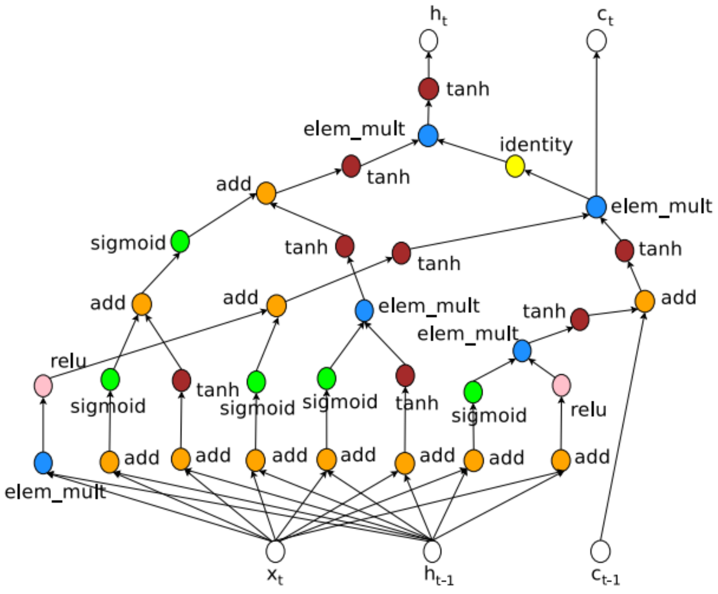
Similar to ResNet!



Architecture Search



LSTM cell



Optimized Cell

Zoph et Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017

Outline of Today's Lecture

1. Basics of deep learning

2. Deep learning for images

3. Deep learning for sequences

Summary

- We covered common architectures such as **MLP, CNN, RNN**
- Modern ML models are characterized by **deep layers and large learnable parameter spaces**
- The nature of stochasticity and non-linearity poses a much larger challenge in all aspects of trustworthy AI