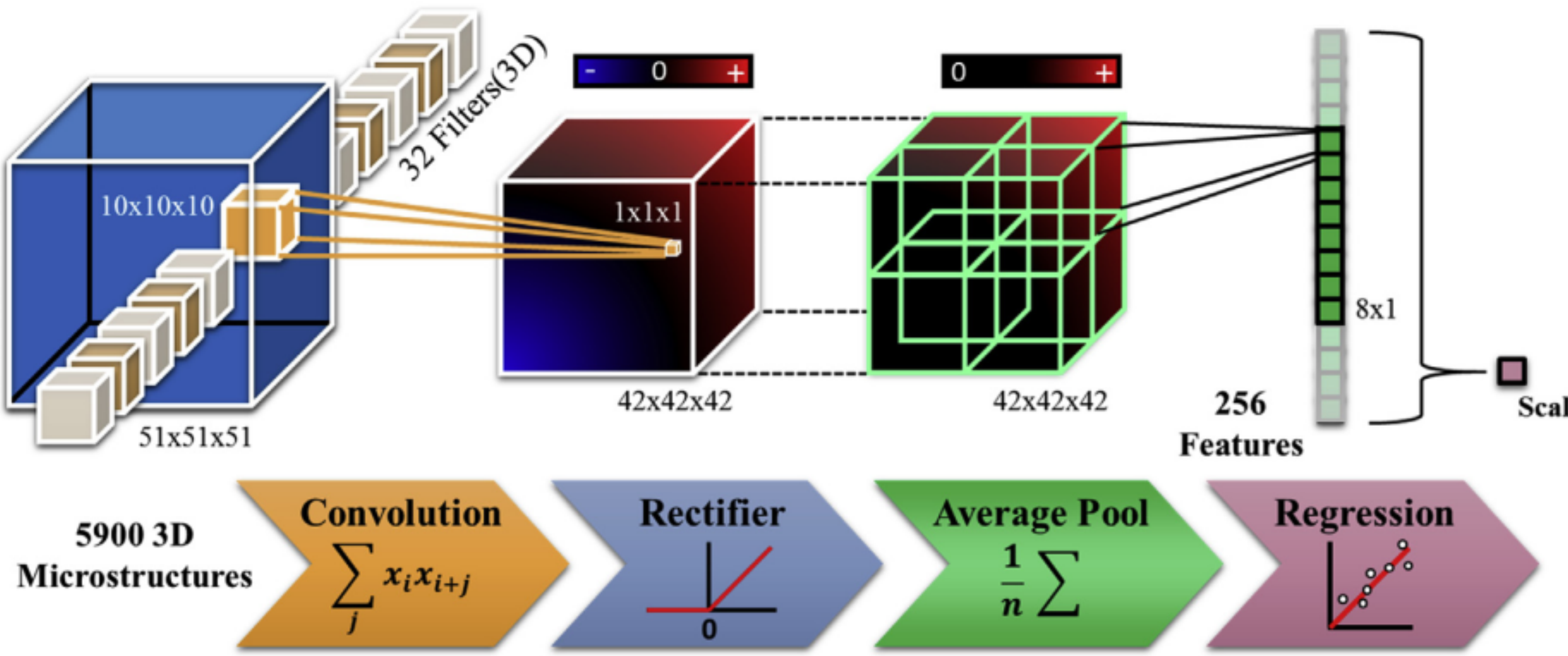
Introduction to Deep Learning

Apaar Shanker

Georgia Institute of Technology



Before we begin

- It is encouraged that analysis be carried out in either python or matlab
- [Python Download Link (https://www.anaconda.com/distribution/)](https://www.anaconda.com/distribution/)
- Several tools dveloped by the group are open source and hosted online
  - [pymks (https://pymks.org)](https://pymks.org)
  - [matlab tools (https://github.com/ahmetcecen/MATLAB-Spatial-Correlation-Toolbox](https://github.com/ahmetcecen/MATLAB-Spatial-Correlation-Toolbox)

Overview

* Description of Neural Network Model

* Training a Neural Network Model: Backpropogation

* Applications of Neural Net models in materials domain

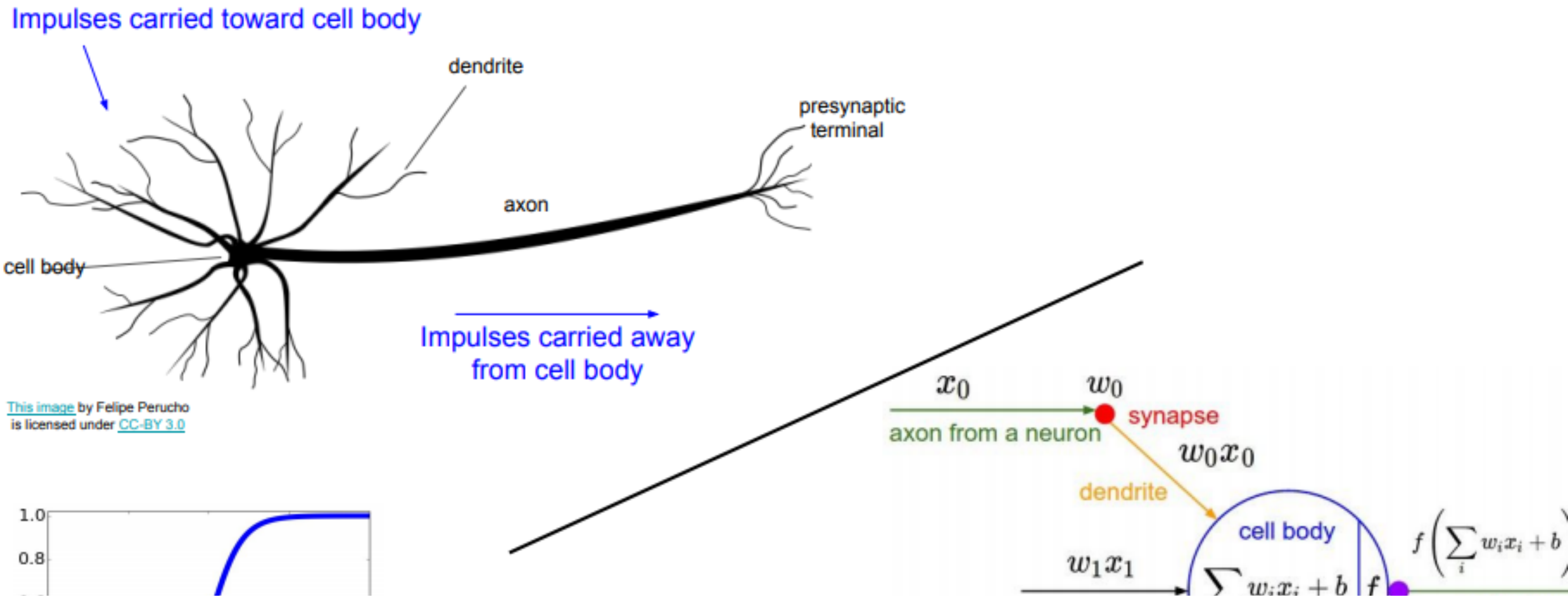
* Popular Libraries : How to NN?

* Convolutional Neural Networks

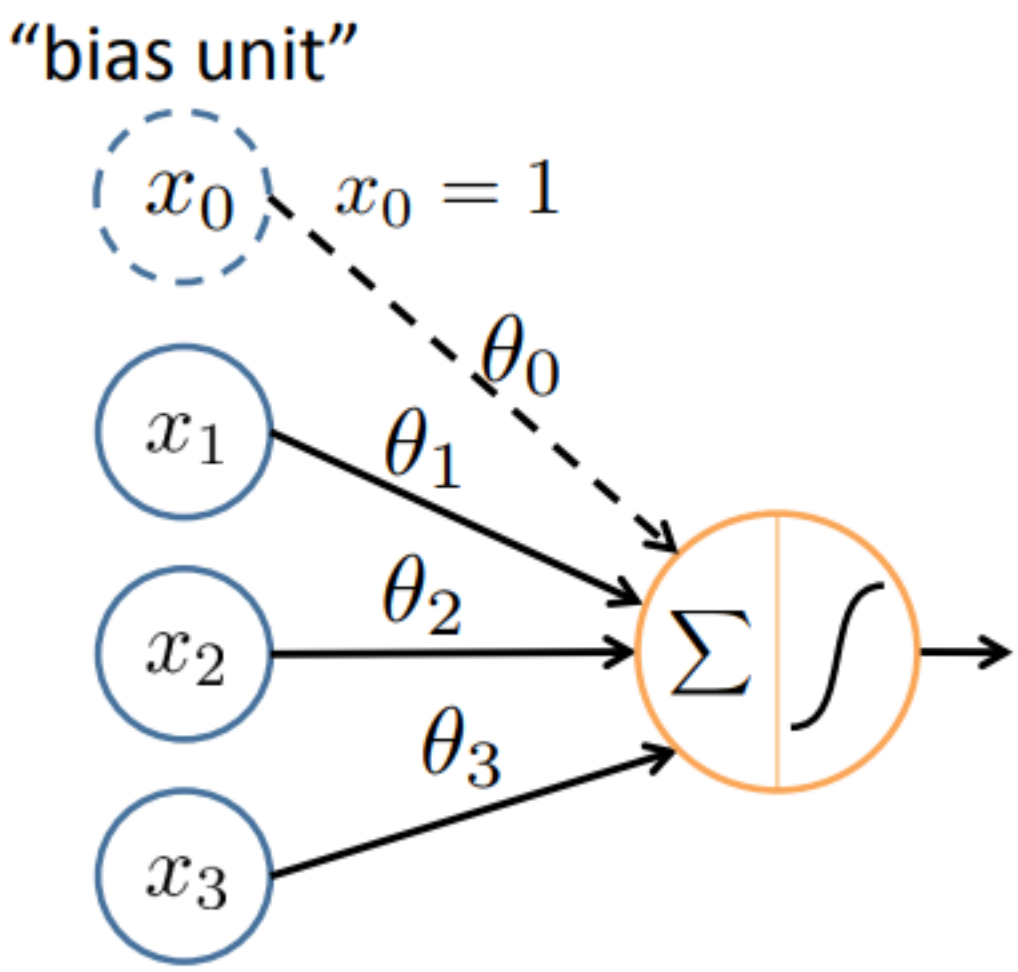
* Analogy between CNNs and MKS Localization

* PDE-NETs and learning Differential Equations using Conv-Net filters

The inevitable brain analogy and the Perceptron



Zooming in on the Perceptron



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\intercal \mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\intercal \mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

let's first talk about linear regression

- $f = Wx$
    - $x = \{x_1, x_2, \cdots, x_m\}$ is a set of features in $\mathbb{R}^m$
    - $W = \{w_1, w_2, \cdots, w_m\}$ is a set of parameters in $\mathbb{R}^m$
    - f is the scalar output

Given a set of N input data points and corresponding target (or property) values, W can be computed using techniqques like **ordinary least square**.

$x : (m \times 1), W_1 : (m \times 1), f : (1)$

A simple linear transformation

- f
  $= Wx$

The neural network model

A simple linear model

- $f = Wx$

A 2-layer Neural Network

- $f = W_2 max(0, W_1 x)$

$max(0, x)$ is known as the ReLU (Regularized Linear Unit) function

$x : (n \times 1), W_1 : (m1 \times n), W_2 : (m2 \times m1), f : (m2 \times 1)$

The neural network model

A simple linear trnasformation of input feature vector

- $f = Wx$

A 2-layer Neural Network

- $f = W_2 max(0, W_1 x)$

or A 3-layer Neural Network

- $f = W_3 max(0, W_2 max(0, W_1 x))$

The neural network model

A simple linear model

- $f = Wx$

A 2-layer Neural Network

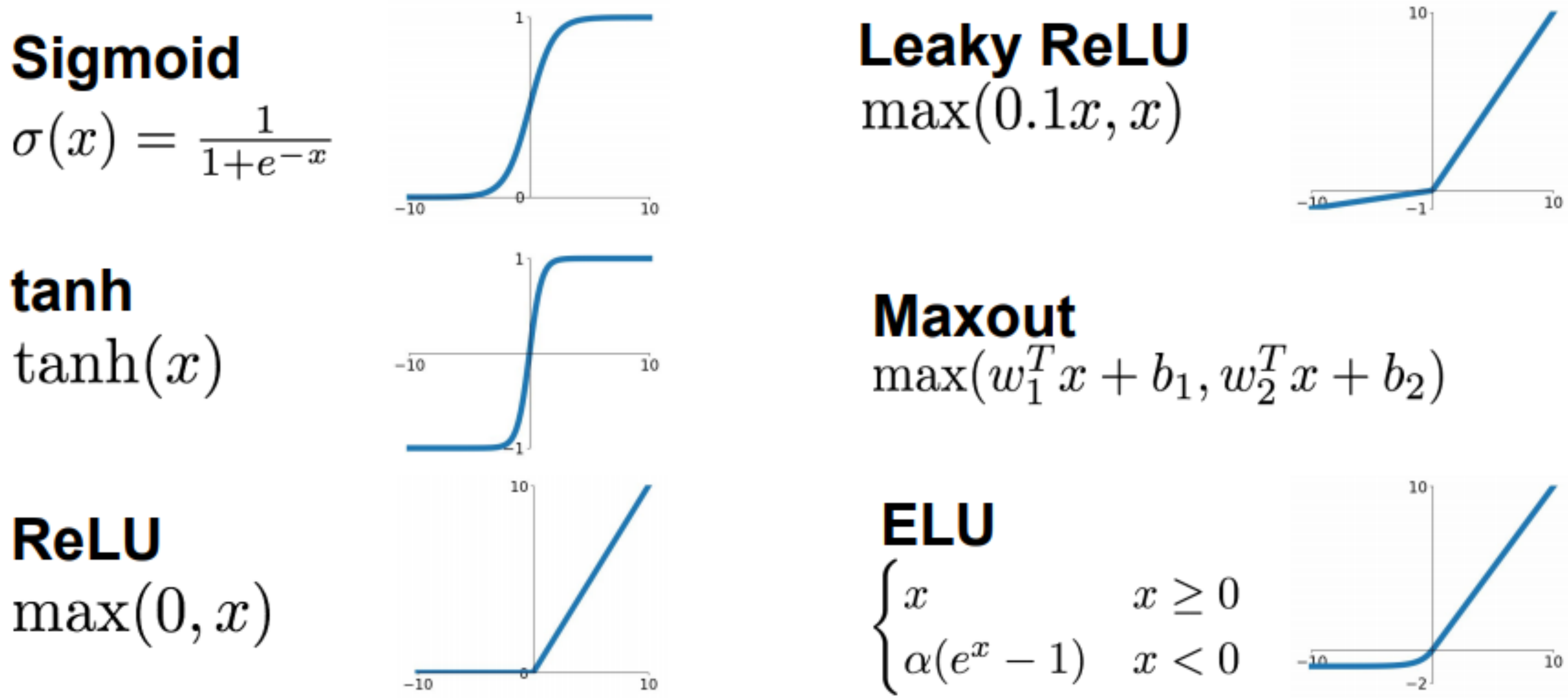
- $f = W_2 max(0, W_1 x)$

or if you fancy, 3-layer network with both ReLU and Sigmoid activation

- $f = W_3 max(0, W_2 \sigma(W_1 x))$ where $\sigma(h) = \frac{1}{1 - \exp -h}$

$x : (n \times 1), W_1 : (m1 \times n), W_2 : (m2 \times m1), W_3 : (m3 \times m2), f : (m3 \times 1)$

Commonly Used Activation Functions



- Sigmoid and tanh functions is most commonly used in MultiLayer Perceptron models, wherea ReLU is the standard for conv-nets described later.
- Please note that the derivatives of all these functions are really easy to compute, for eg:
  $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

The Neural Network as a Computational Graph

s = [3, 3, 2, 1]

Training the model: Optimizing the loss function

Consider the linear regression model:

- $y = w^T x$

Training the model: Optimizing the loss function

Consider the linear model:

- $y = w^T x$

We can define a function $\mathcal{L}$:

$$\begin{aligned}\mathcal{L}(w) &= \sum^{N}(\hat{y}_i - y_i)^2 \\ &= \sum^{N}(\hat{y}_i - w^T x_i)^2\end{aligned}$$

Training the model: Optimizing the loss function

Consider the linear model:

- $y = w^T x$

We can define a function $\mathcal{L}$:

$$\begin{aligned}\mathcal{L}(w) &= \sum^{N}(\hat{y}_i - y_i)^2 \\ &= \sum^{N}(\hat{y}_i - w^T x_i)^2\end{aligned}$$

such that the problem of guessing the weights reduces to the problem of minimizing the function $\mathcal{L}$ also known as the loss function.

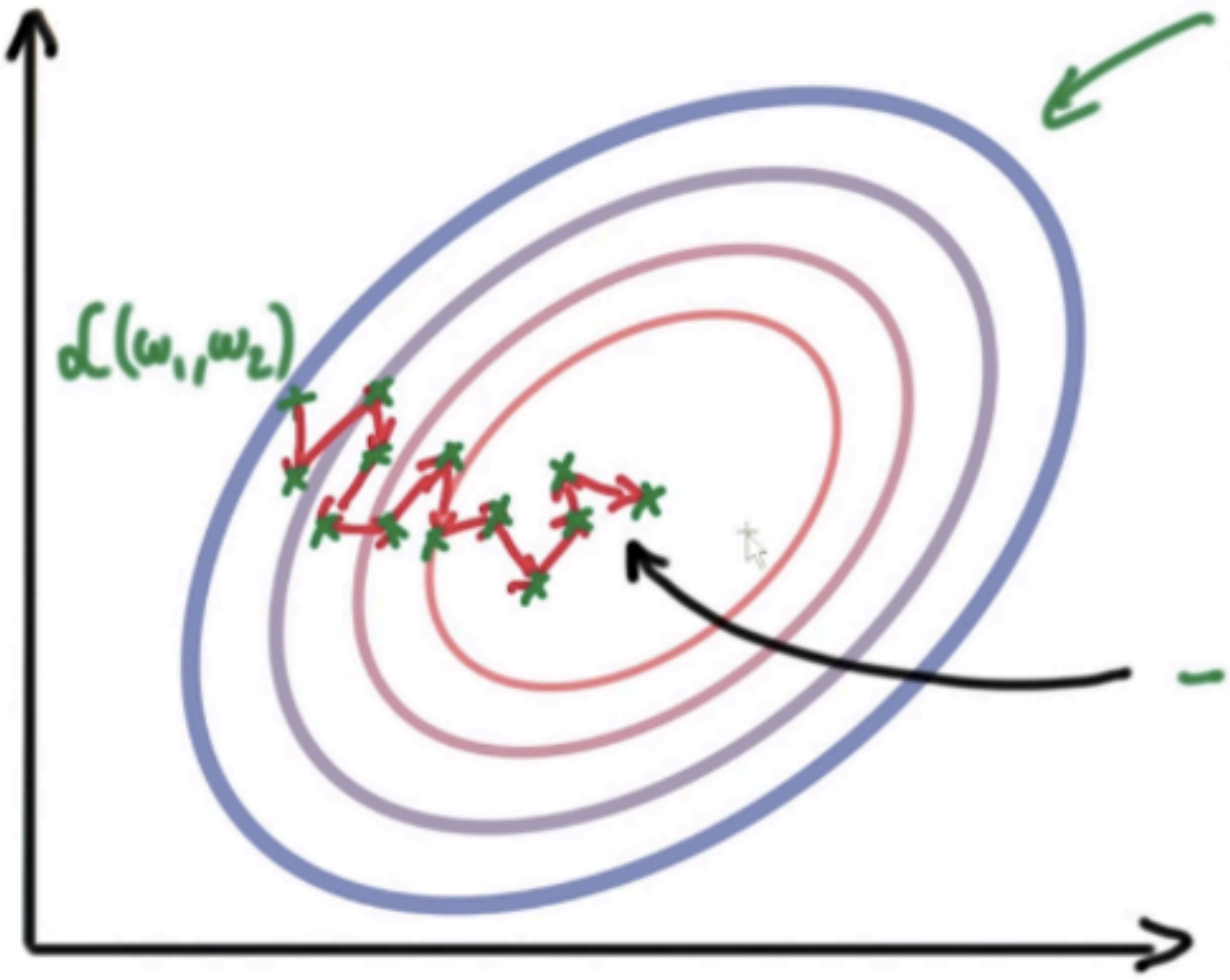
In this case, the function is clearly convex, i.e. a parabola in w space, so we have an analytical solution to the problem as:

- $\hat{w} =$ where $X : \{x_i\}$ and $\hat{Y} : \{\hat{y}_i\}$

Training the model: Gradient Descent

Training the model: Gradient Descent

- The gradient at any point in the loss function denoted as $\nabla_w \mathcal{L}$
- It is a vector that gives the direction of maximal positive change in the loss function.
- As such, loss function can be minimized by moving in the direction opposite to the gradient.
- This gives us an update rule
  - $w_i^{t+1} = w_i^t - \lambda \dfrac{\partial \mathcal{L}(w)}{\partial w_i}$
  - λ is reffered to as the learning rate and controls the speed of descent.



$\mathcal{L}(w_1, w_2)$

Training the model: Stochastic Gradient Descent

- Recal:
  - $\mathcal{L} = \frac{1}{N} \sum^{N} (\hat{y}_i - f(x_i))^2$
- For large datasets, it is expensive to compute loss for the entire dataset in each update step.
- An alternative is to compute gradient over batches of training data.
- **Stochastic refers to the fact that the "mini-batch" loss function is a "stochastic" approximation of the actual loss**
- This gives us a modified update rule
  - $w_i^{t+1} = w_i^t - \lambda \frac{\partial l_j(w)}{\partial}$

Training the model: Backpropogation

- Recal the form of the 3-layer Neural Network Model:
  - $f = W_3 max(0, W_2 max(0, W_1 x))$

Training the model: Backpropogation

- Recal the form of the 3-layer Neural Network Model:
    - $f = W_3 max(0, W_2 max(0, W_1 x))$
- We again define the loss function as:
    - $\mathcal{L} = \frac{1}{N} \sum^N (\hat{y}_i - f(x_i))^2$

Training the model: Backpropogation

Training the model: Backpropogation

What if we use chain rule?

Recall, chain rule:

$$\frac{d(f \cdot g)(x)}{dx} = \frac{f(g(x))}{d(g(x))}\frac{d(g(x))}{dx}$$

- A simplified illustration of backpropogation using the univariate logistic least squares model

Computing the loss:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

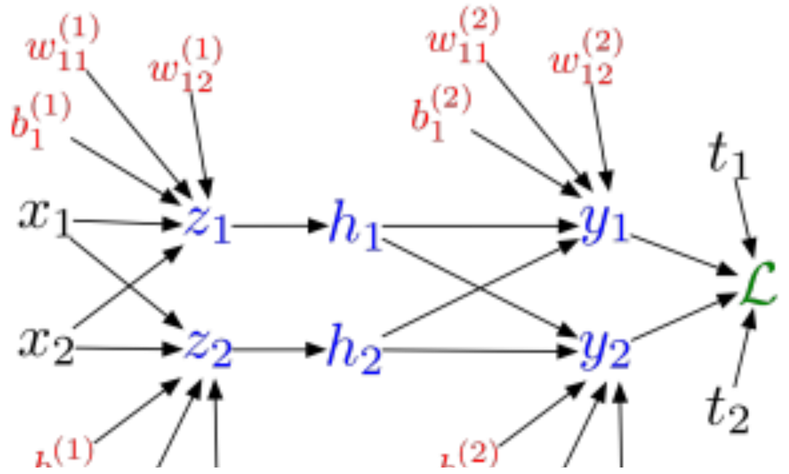
Computing the derivatives:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} \cdot \sigma'(z)$$
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} \cdot x$$
$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/slides/lec6.pdf (http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/slides/lec6.pdf)

Training the model: Backpropogation

Multilayer Perceptron (multiple outputs):



Backward pass:

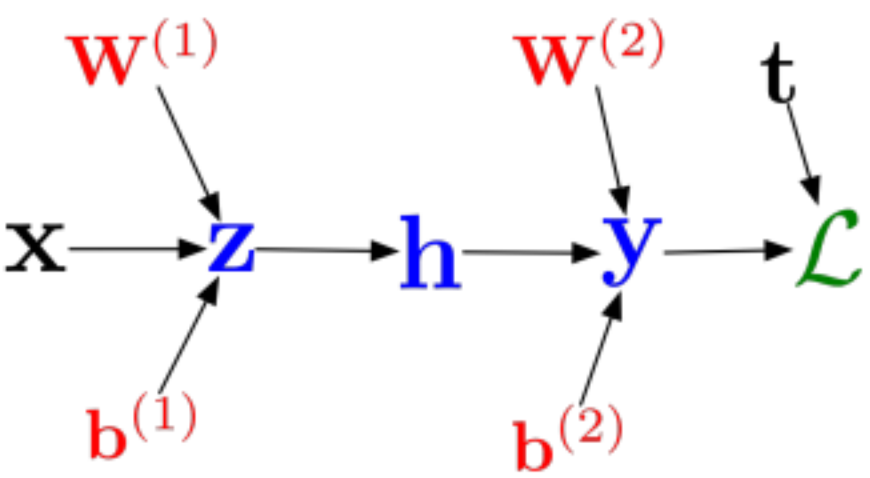
$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}}\,(y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$

Training the model: Backpropogation

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{\mathbf{y}} = \overline{\mathcal{L}}\,(\mathbf{y} - \mathbf{t})$$

$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$

$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$

$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$

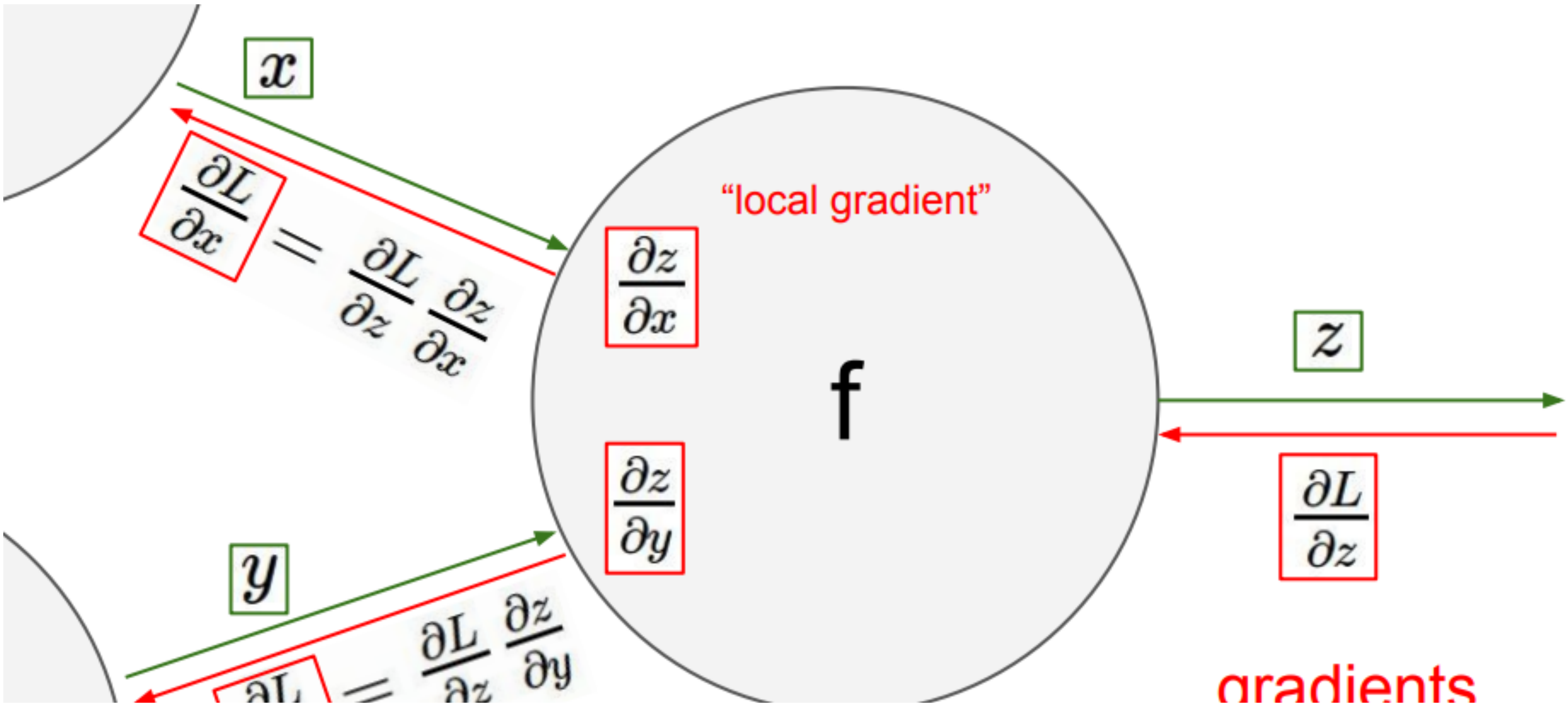
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \cdot \sigma'(\mathbf{z})$$

$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$

$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Training the model: Backpropogation

- In the message passing notation:



Back to the equation

A 3-layer feed-forward Neural Network

- $f = W_3 max(0, W_2 max(0, W_1 x))$

To Summarize:

- A multilayered perceptron is a just a set of linear followed by non-linear transforms performe on a input vector.
- A feed-forward fully connected neural network with a single hidden layer using practically an nonlinear activation function can approximate any continuous function of any number of real variables on any compact set to any desired degree of accuracy.
- Number of Parameters in the model = $\sum_{i=1}^{N} (L_{n-1} + 1) * L_n$
- **How to guess the values of these parameters?**
- [https://papers.nips.cc/paper/874-how-to-choose-an-activation-function.pdf (https://papers.nips.cc/paper/874-how-to-choose-an-activation-function.pdf)](https://papers.nips.cc/paper/874-how-to-choose-an-activation-function.pdf)

Resources for implementing Neural Networks

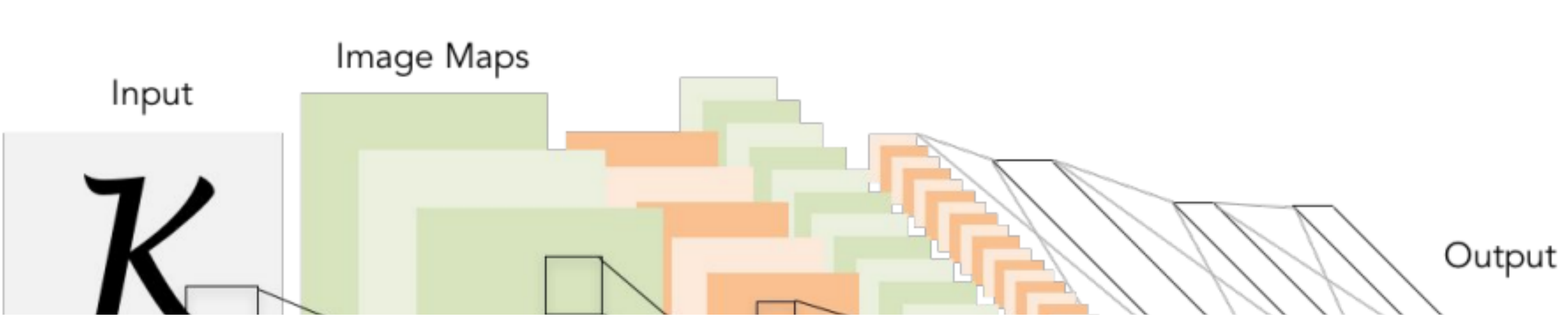
- Pytorch - http://pytorch.org/ (http://pytorch.org/)
- Tensorflow - http://tensorflow.org/ (http://tensorflow.org/)
- Theano - http://deeplearning.net/software/theano/ (http://deeplearning.net/software/thean

- Keras - https://keras.io/ (https://keras.io/)

  A useful learning resource - https://playground.tensorflow.org/ (https://playground.tensorflow.org/)

  Background http://cs231n.github.io/ (http://cs231n.github.io/)

Convolutional Neural Networks



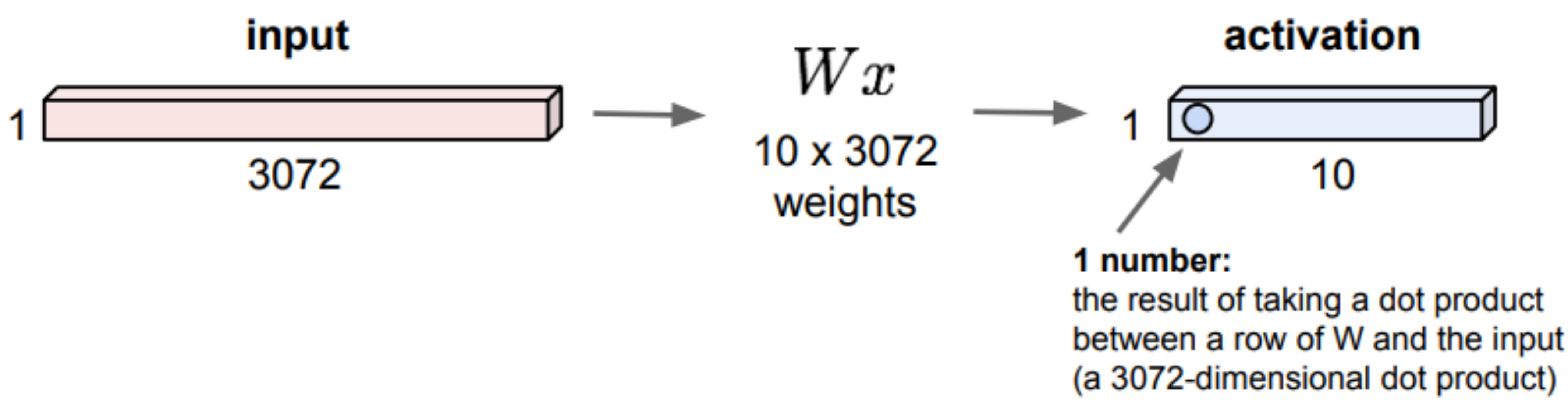
Convolutional Neural Networks

- Image data are high dimensional and have local embedded structures.
- CNNs were conceptualized to overcome the limitations of Fully Connected neural networks i processing image data

Convolutional Neural Networks

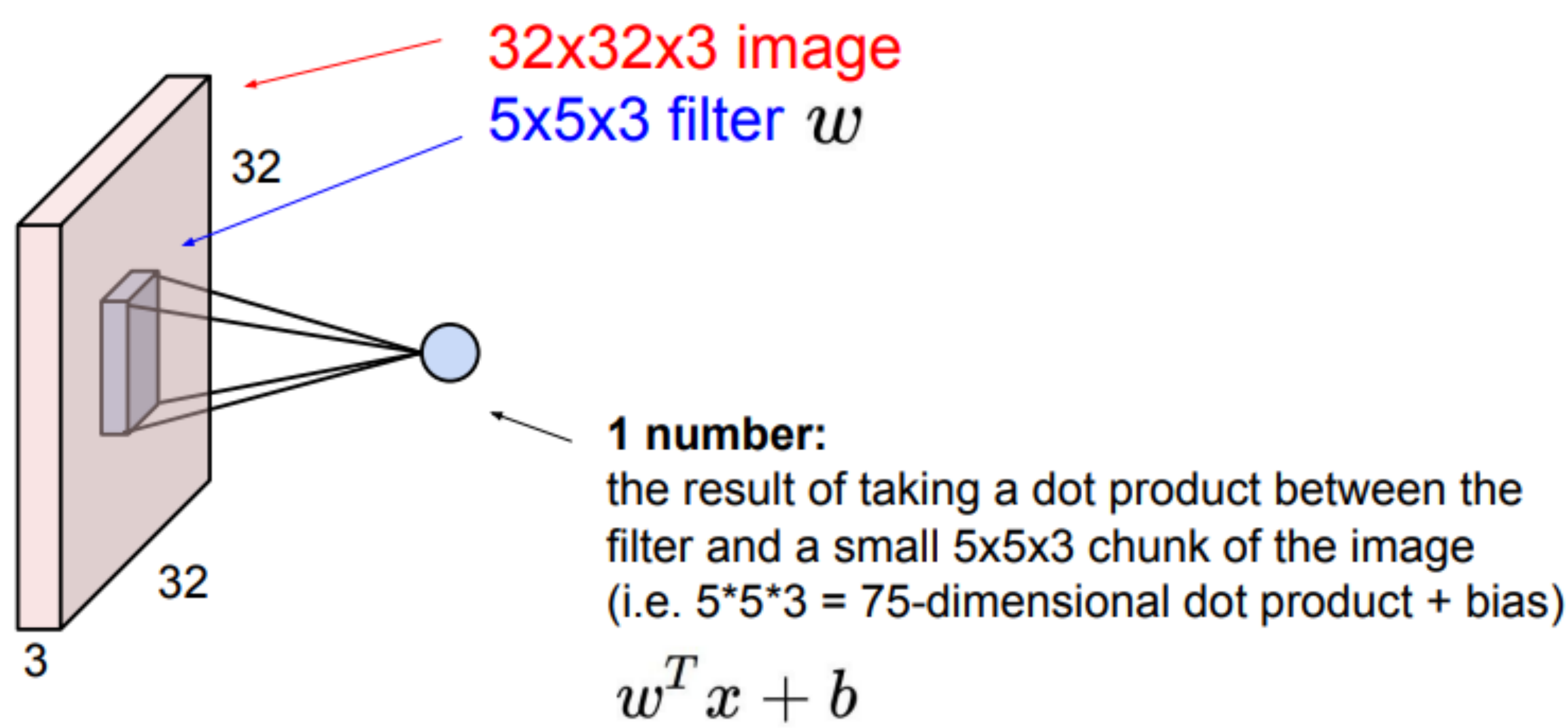
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Convolutional Neural Networks

Convolution Layer



Recall convolution:

$$f[x, y] * g[x, y] = \sum_{n_1=-\inf}^{\inf} \sum_{n_2=-\inf}^{\inf} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

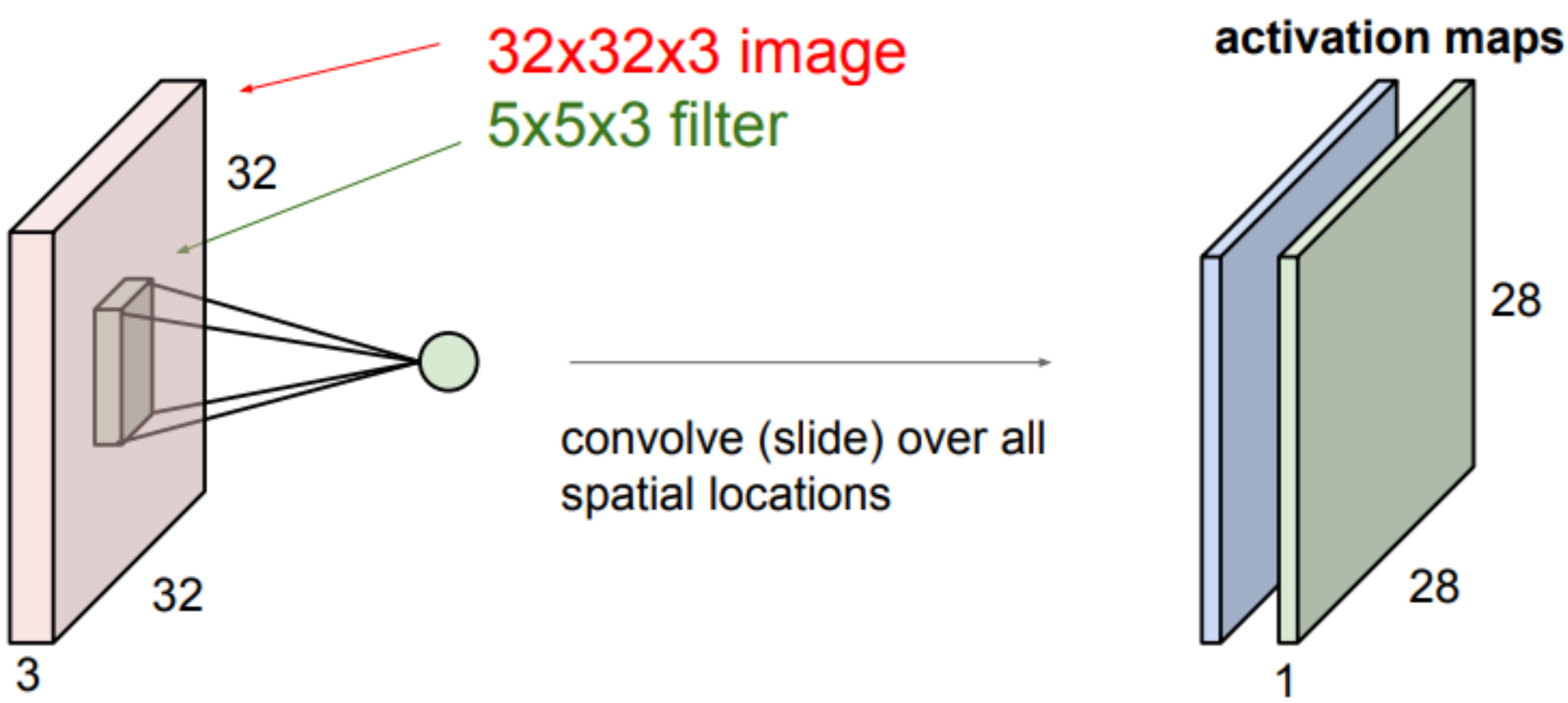
Convolutional Neural Networks

Convolution Layer

Convolutional Neural Networks

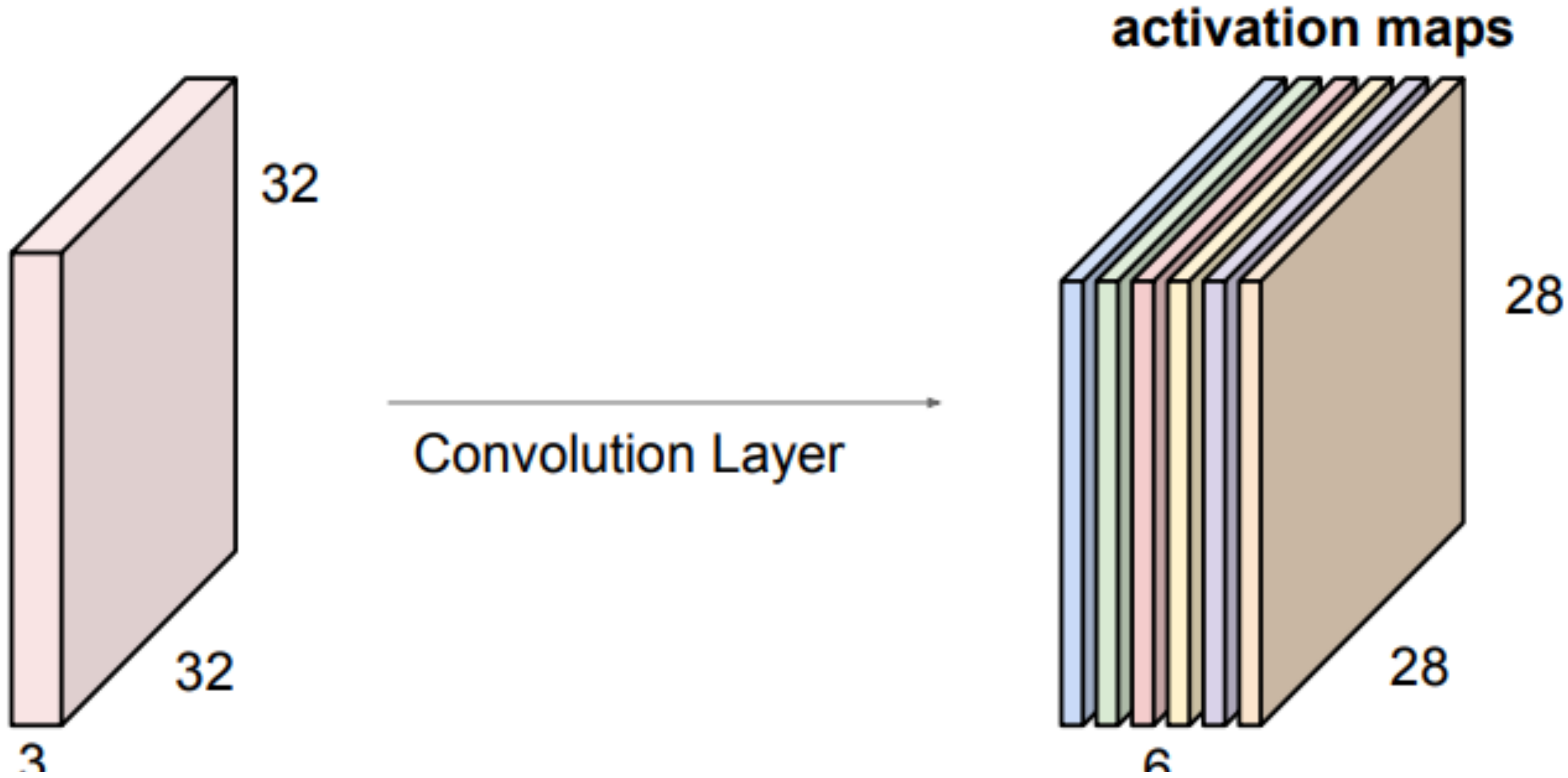
Convolution Layer

consider a second, green filter



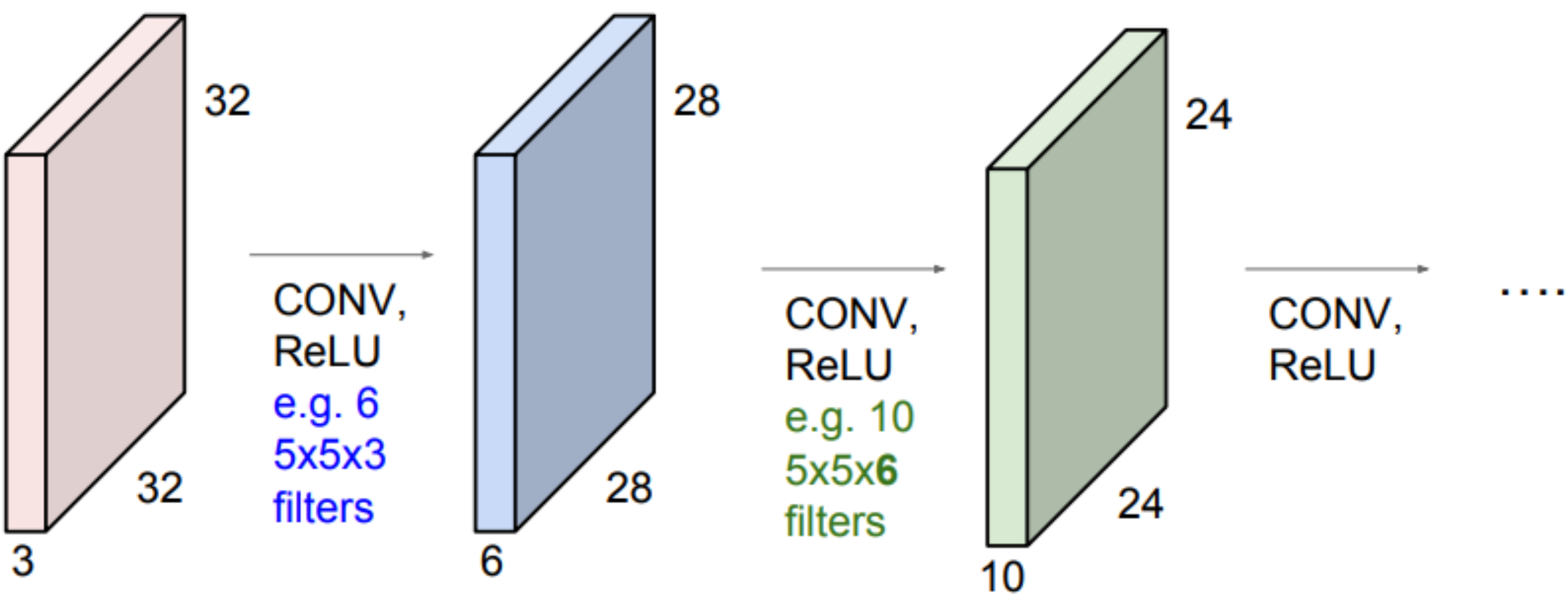
Convolutional Neural Networks

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



Convolutional Neural Networks

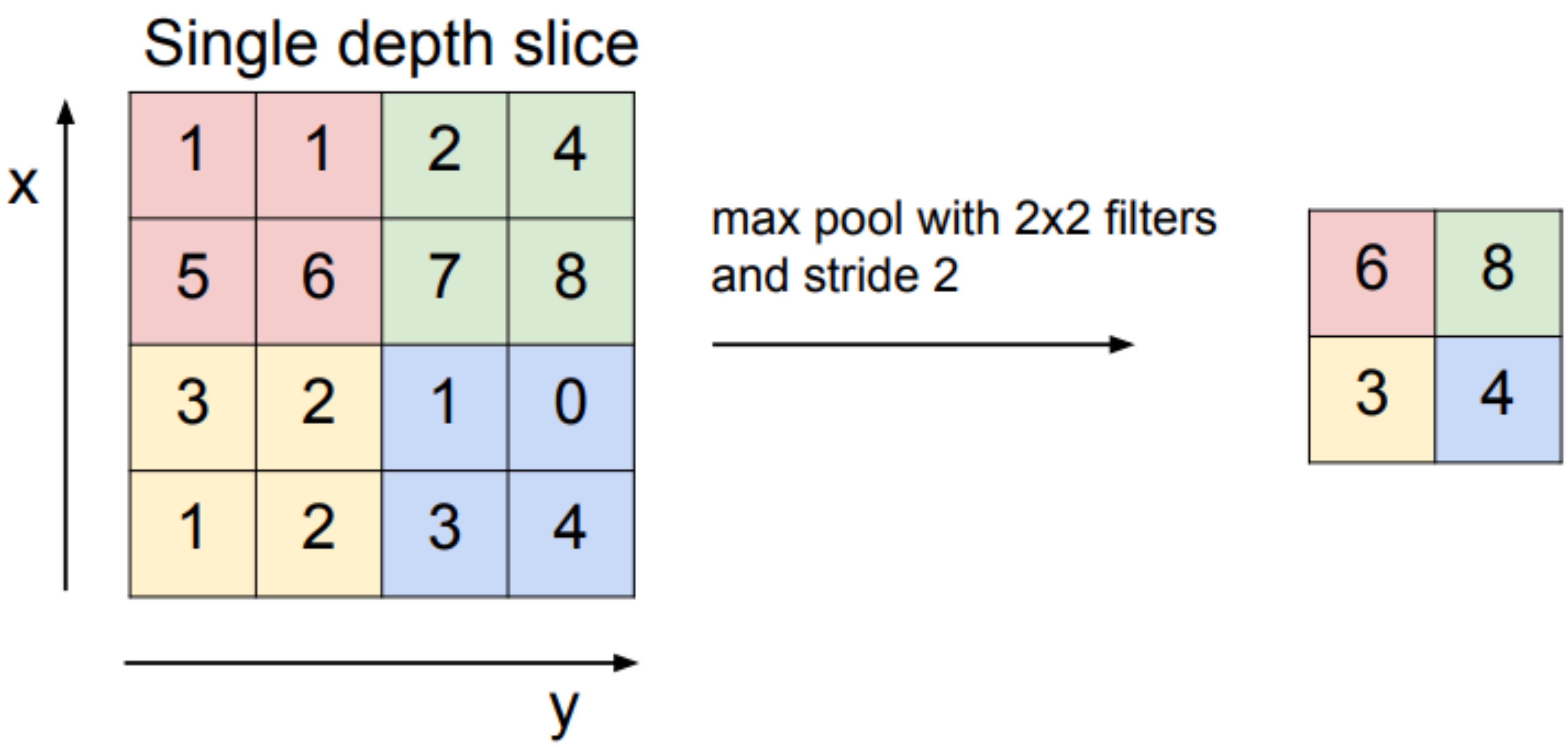
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolutional Neural Networks

Inorder to reduce number of parameters and prevent overfitting.

MAX POOLING



Convolutional Neural Networks

Typical off the shelf CNN / Deep Learning Model

Convolutional Neural Networks

VGG-Net : A Production CNN

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16

Common names

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Convolutional Neural Networks

Why you should care?

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners