Advanced Data Visualization

CS 6965

Fall 2019

Prof. Bei Wang Phillips

University of Utah

Lecture 15

Deep Learning and Vis
Part I

HD

The goal of this lecture

- Not a complete overview of neural networks or deep learning
- But rather a high level view of the technique and its connection to visualization

Deep learning tutorial

- http://neuralnetworksanddeeplearning.com/
- http://deeplearning.stanford.edu/tutorial/
- http://www.deeplearningbook.org/
- And many more…

TensorFlow

- TensorFlow programming environment:
  - https://www.tensorflow.org/get_started/get_started_for_beginners
  - https://www.tensorflow.org/get_started/premade_estimators

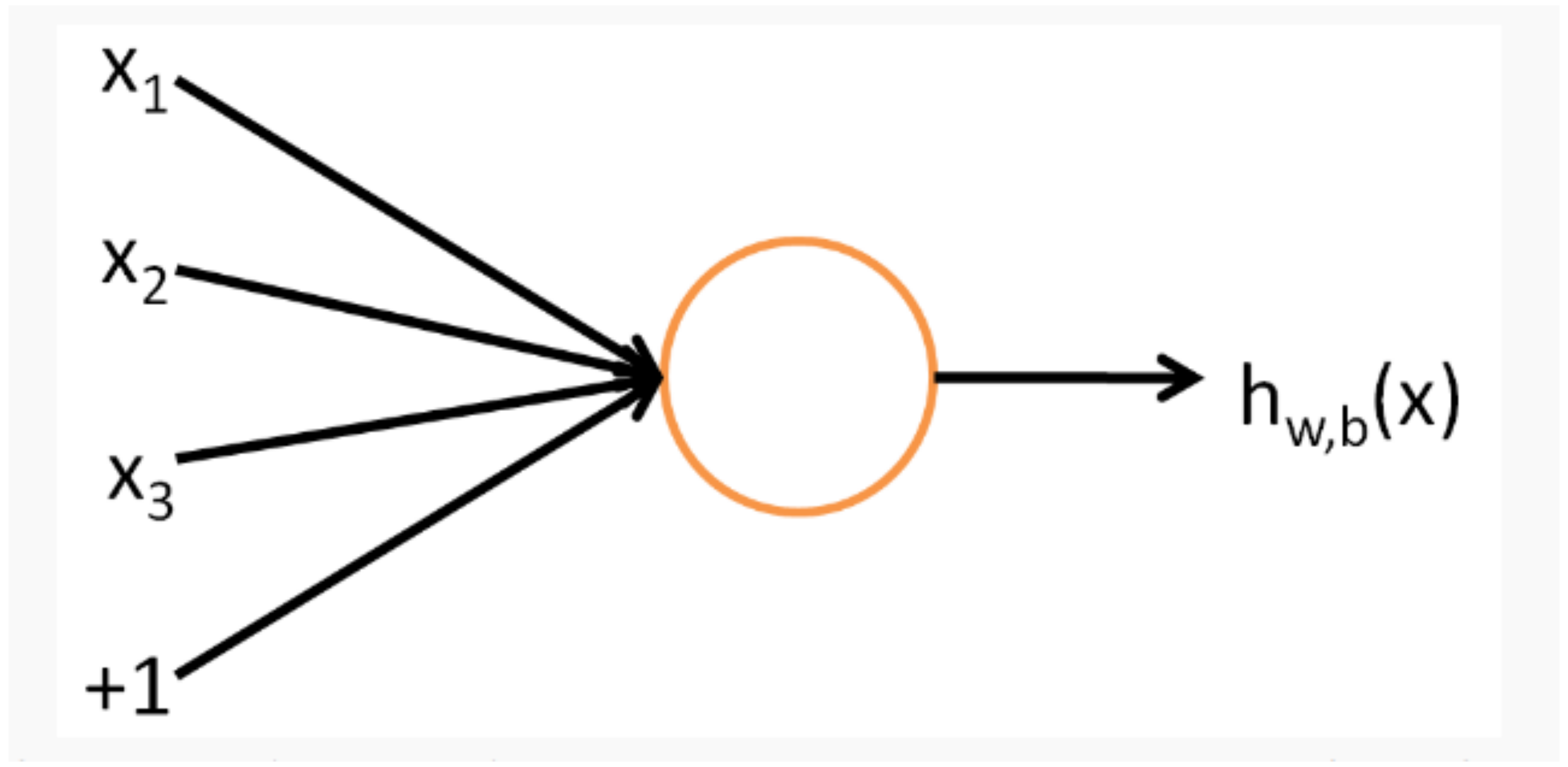
Multi-Layer Neural Network in a nutshell

A review based on materials from UFLDL Tutorial and Michael Nielsen

http://neuralnetworksanddeeplearning.com/

http://ufldl.stanford.edu/tutorial/

A single neuron



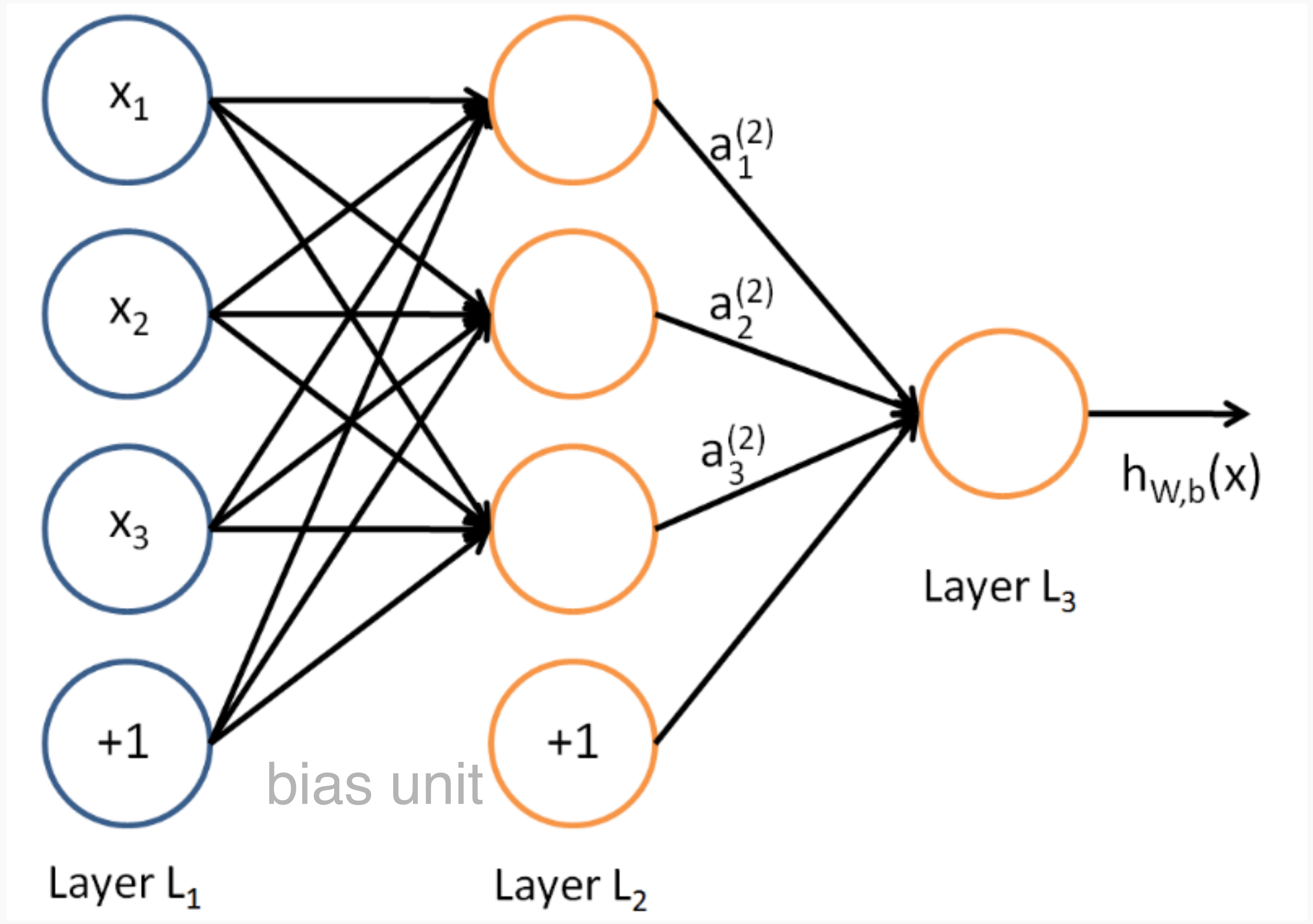
This "neuron" is a computational unit that takes as input x_1, x_2, x_3 (and a +1 intercept term), and outputs $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, where $f : \Re \mapsto \Re$ is called the **activation function**. In these notes, we will choose $f(\cdot)$ to be the sigmoid function:

$$f(z) = \frac{1}{1 + \exp(-z)}.$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

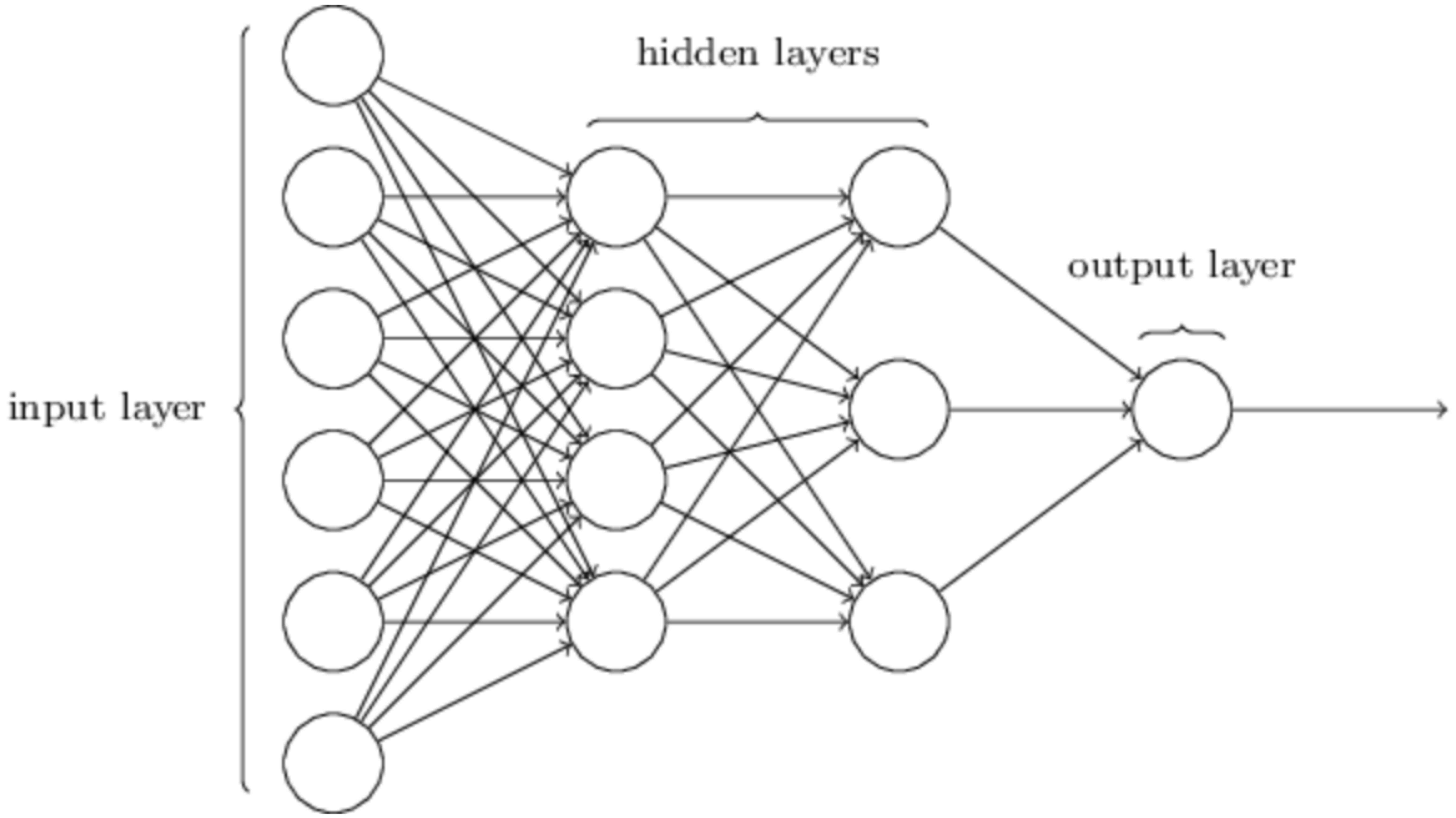
A Neural Network

A neural network is put together by hooking together many of our simple "neurons," so that the output of a neuron can be the input of another. For example, here is a small neural network:



http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

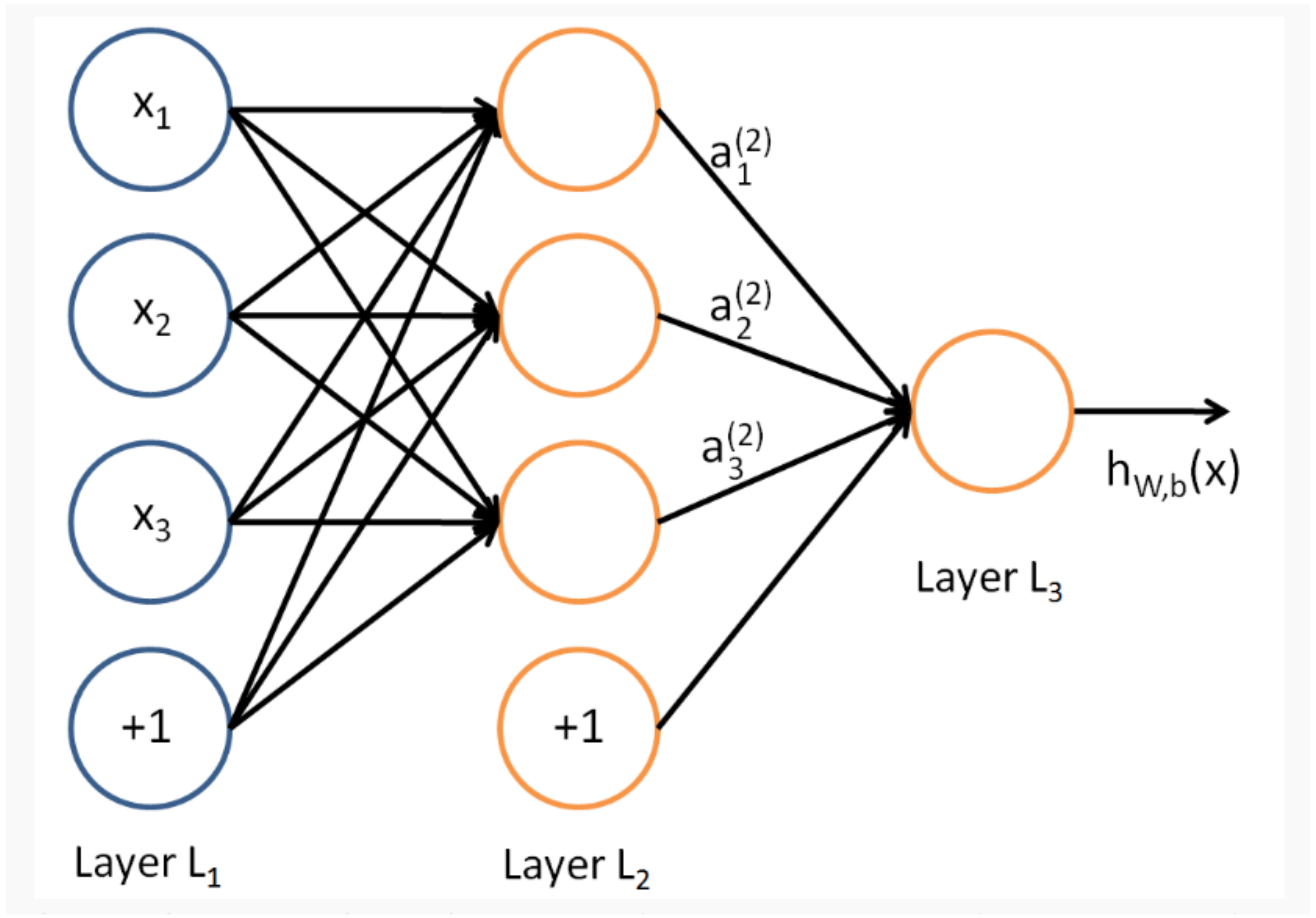
A Neural Network



http://neuralnetworksanddeeplearning.com/chap1.html

Forward propagation

- Multiplying input with weights and add bias before applying activation function at each node



$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) = a^{(3)} &= f(z^{(3)})
\end{aligned}
$$

$$
\begin{aligned}
z^{(l+1)} &= W^{(l)}a^{(l)} + b^{(l)} \\
a^{(l+1)} &= f(z^{(l+1)})
\end{aligned}
$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Learning with gradient descent

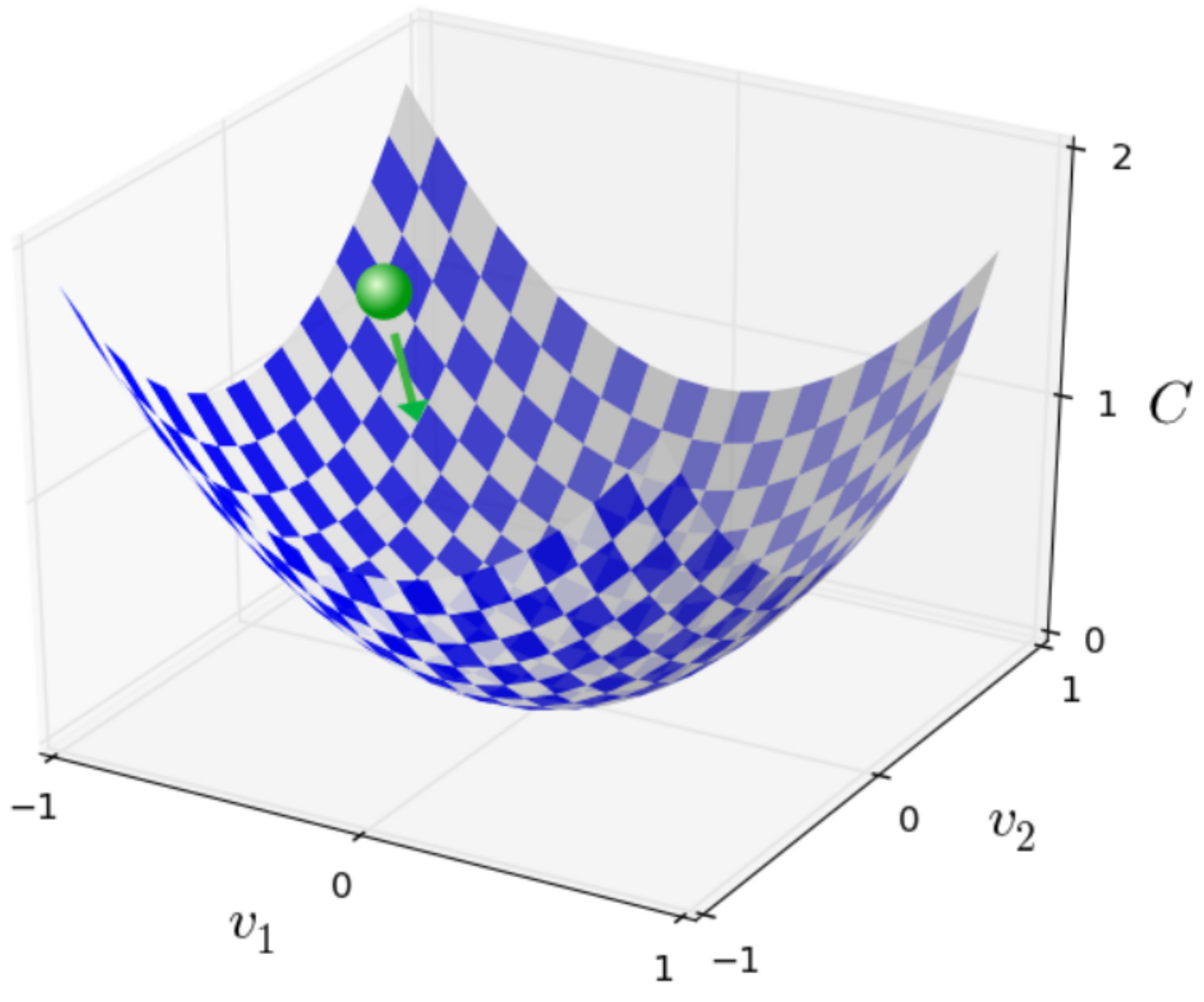
$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

- Cost function
  - x: input
  - y(x): approximate
  - w: collection of all weights
  - b: all the biases
  - n: total number of training inputs
  - a: the vector of outputs from the network when x is input

http://neuralnetworksanddeeplearning.com/chap1.html

Learning with gradient descent

Summing up, the way the gradient descent algorithm works is to repeatedly compute the gradient ∇C, and then to move in the *opposite* direction, "falling down" the slope of the valley. We can visualize it like this:



http://neuralnetworksanddeeplearning.com/chap1.html

Back propagation Algorithm

Cost function with a single training example:

$$J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2.$$

Cost function with m training examples:

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{i=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

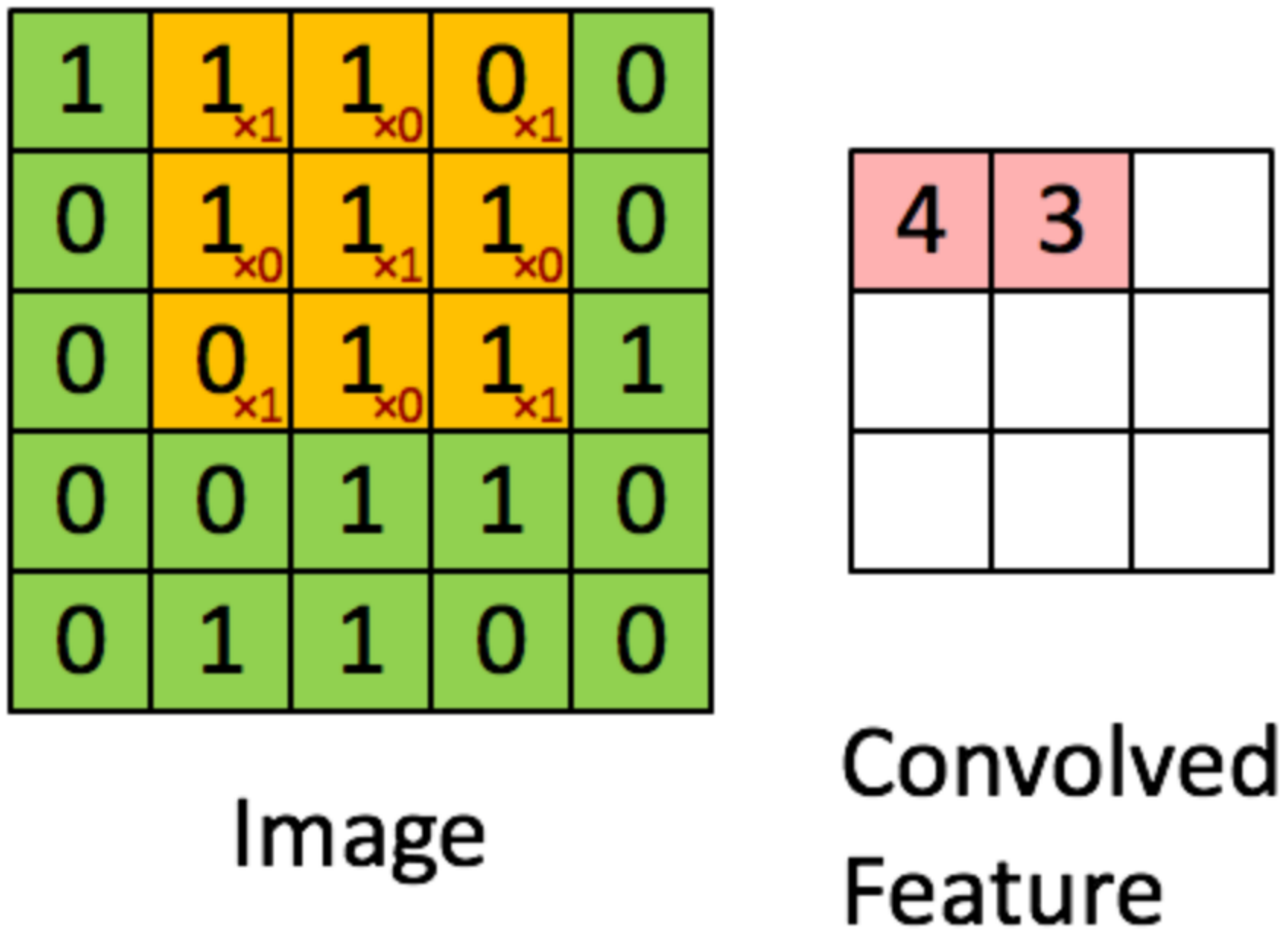
One iteration of gradient descent updates the parameters W, b as follows:

$$\begin{aligned} W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b) \end{aligned}$$

Back propagation algorithm: gives an efficient way to compute these partial derivatives.

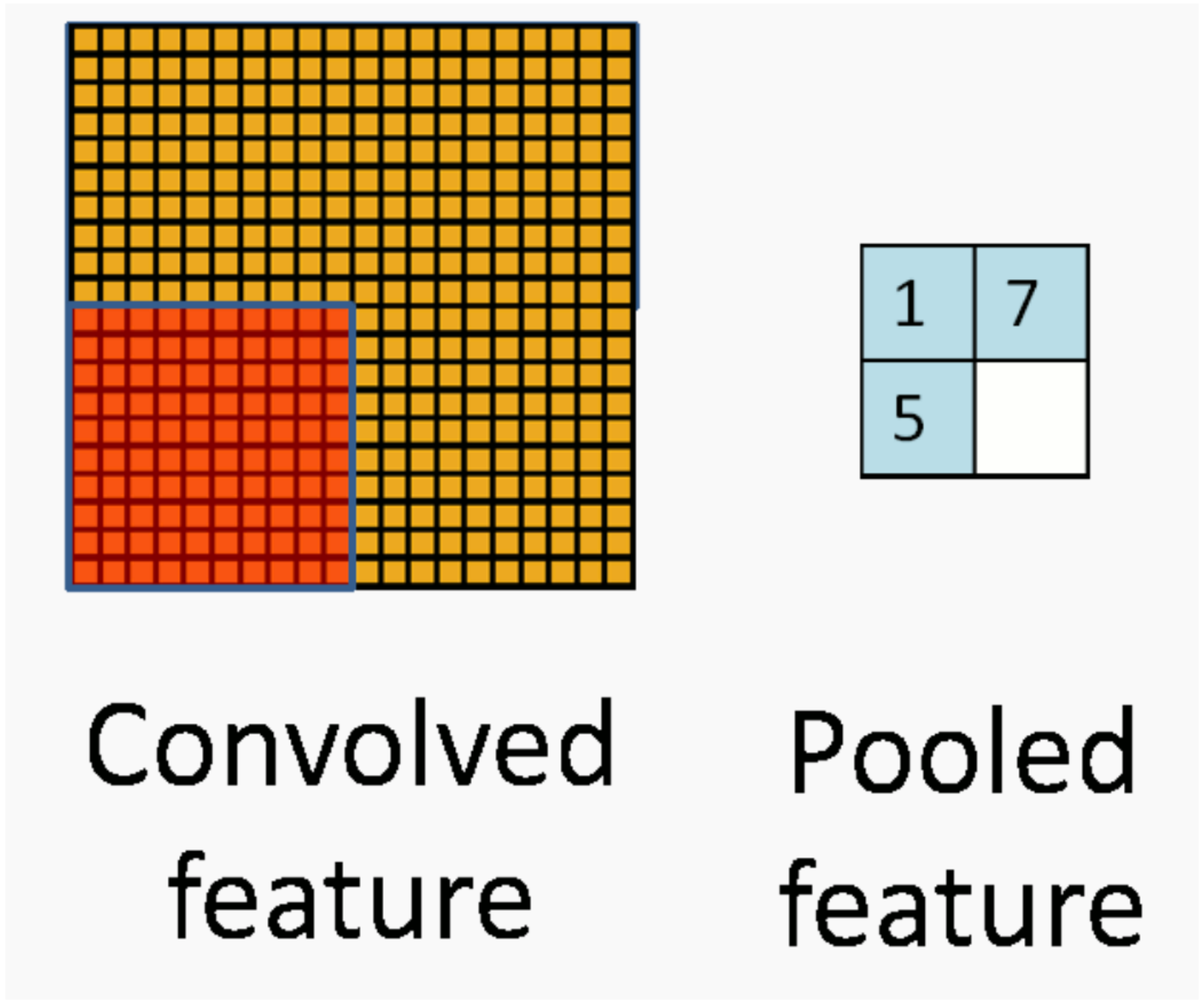
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Feature convolution



http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

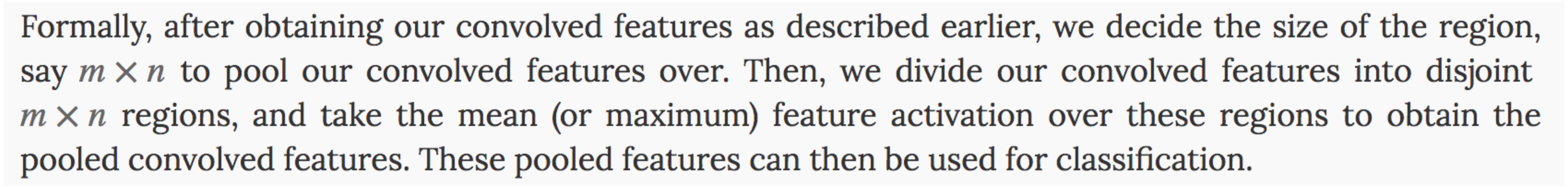
Pooling



Aggregate statistics of convolved features at various locations

http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Pooling



Formally, after obtaining our convolved features as described earlier, we decide the size of the region, say $m \times n$ to pool our convolved features over. Then, we divide our convolved features into disjoint $m \times n$ regions, and take the mean (or maximum) feature activation over these regions to obtain the pooled convolved features. These pooled features can then be used for classification.

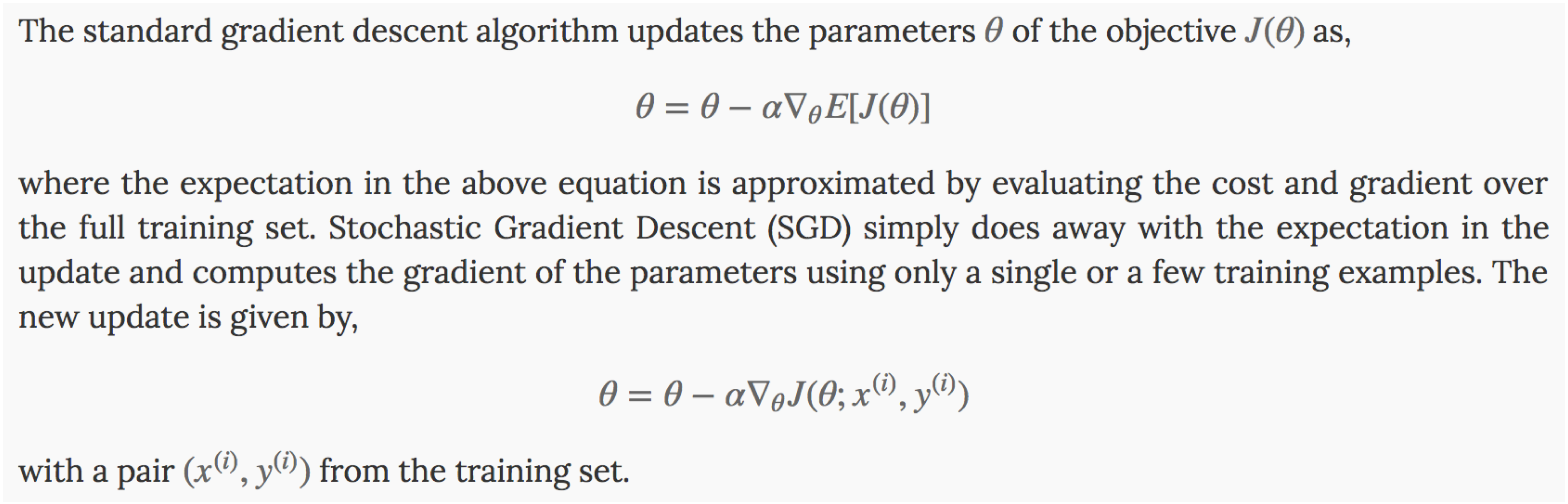
Aggregate statistics of convolved features at various locations

http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Convolutional Neural Network

- A CNN consists of an input and an output layer, as well as multiple hidden layers.
- The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers

Stochastic Gradient Descent



The standard gradient descent algorithm updates the parameters θ of the objective $J(\theta)$ as,

$$\theta = \theta - \alpha \nabla_\theta E[J(\theta)]$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set. Stochastic Gradient Descent (SGD) simply does away with the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by,

$$\theta = \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)})$$

with a pair $(x^{(i)}, y^{(i)})$ from the training set.

http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

Visualization for Deep Learning

Topics

- Directly visualizing the activations and parameters in intuitive aggregates
- Visualizing weights as features
- Visualizing gradient aggregates during training
- Improving interpretability of networks
- Localizing “responsibility” in the network for particular outputs
- Sensitivity/stability of network behavior
- Visualizing loss function geometry and the trajectory of the gradient descent process
- Visual representation of the input-output mapping of the network
- Visualizing alternative structures and their performance
- Monitoring/debugging the training process, i.e to detect saddle points or local optima, saturation units
- Visualizing distributed training methods across a cluster
- Using animation in network visualization
- Interactive visualizations for exploration or parameter tuning
- Software architectures for effective visualization
- Visualization and interaction user interfaces

https://icmlviz.github.io/icmlviz2016/

Visualizing the inner workings of neurons

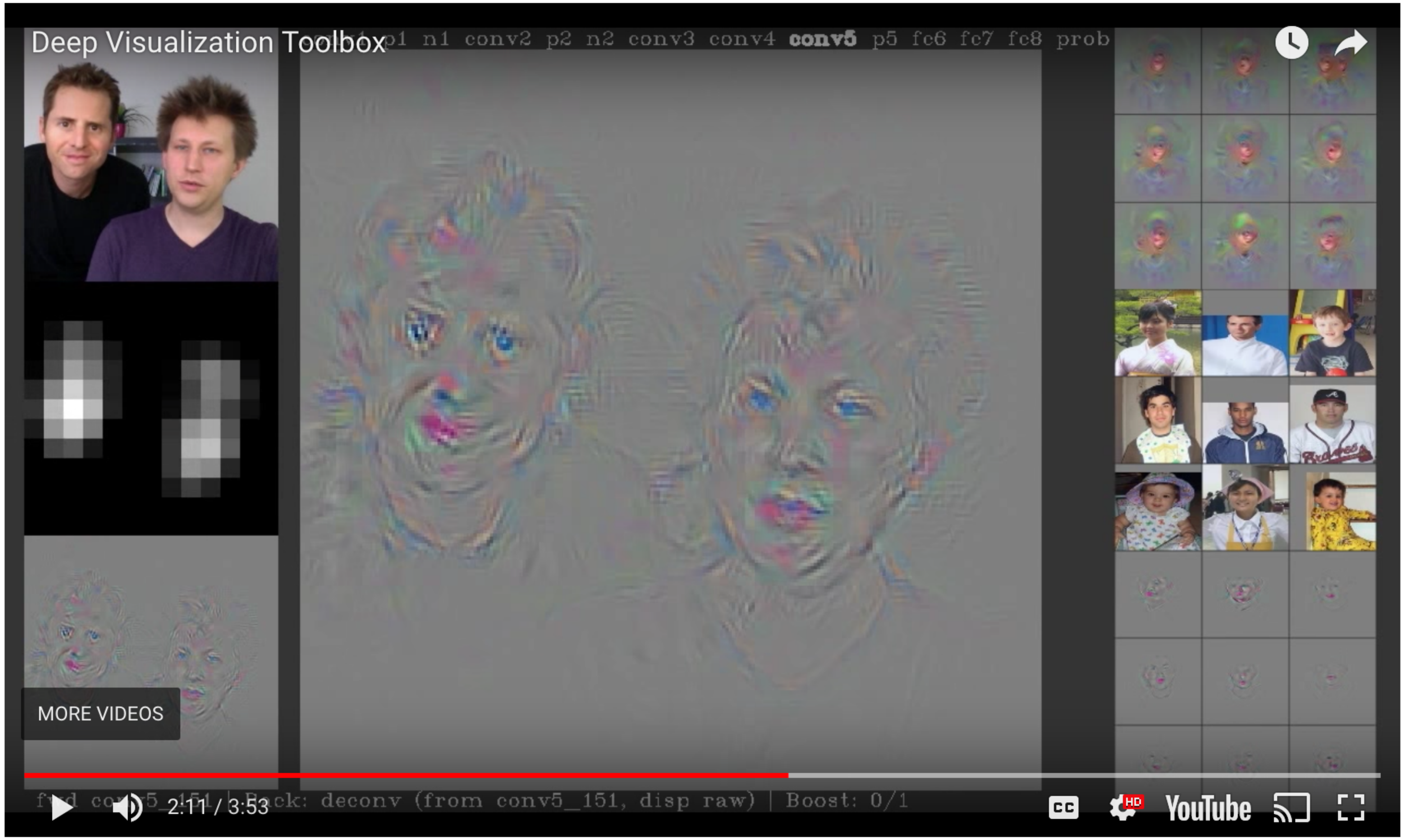
Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
4 neurons
2 neurons
OUTPUT
Test loss 0.509
Training loss 0.503
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

http://playground.tensorflow.org/

Deep Vis


Deep Visualization Toolbox
p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
MORE VIDEOS
fwd conv5_151 | Back: deconv (from conv5_151, disp raw) | Boost: 0/1
2:11 / 3:53
YouTube

http://yosinski.com/deepvis#toolbox

Multifaceted Feature Vis

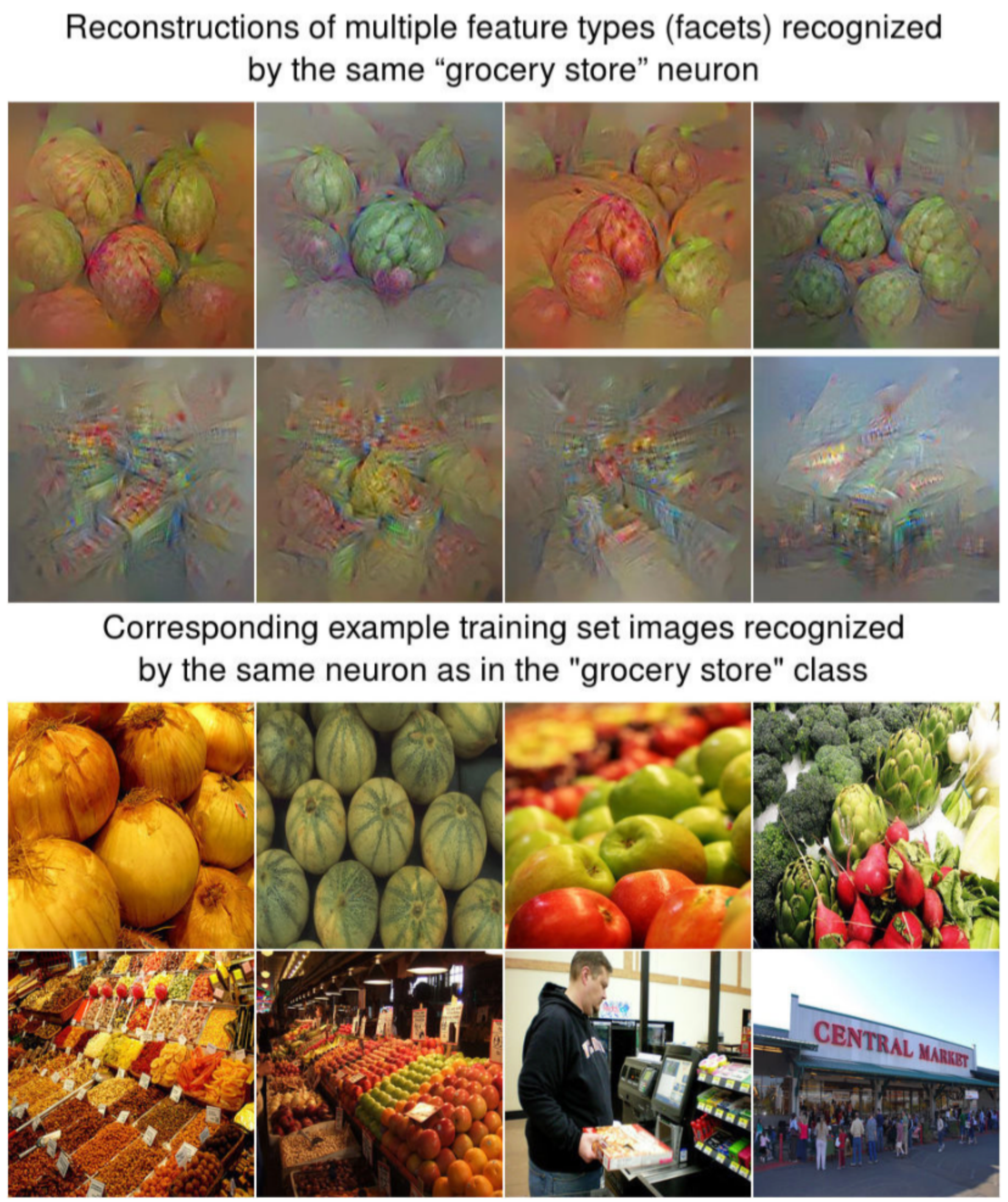
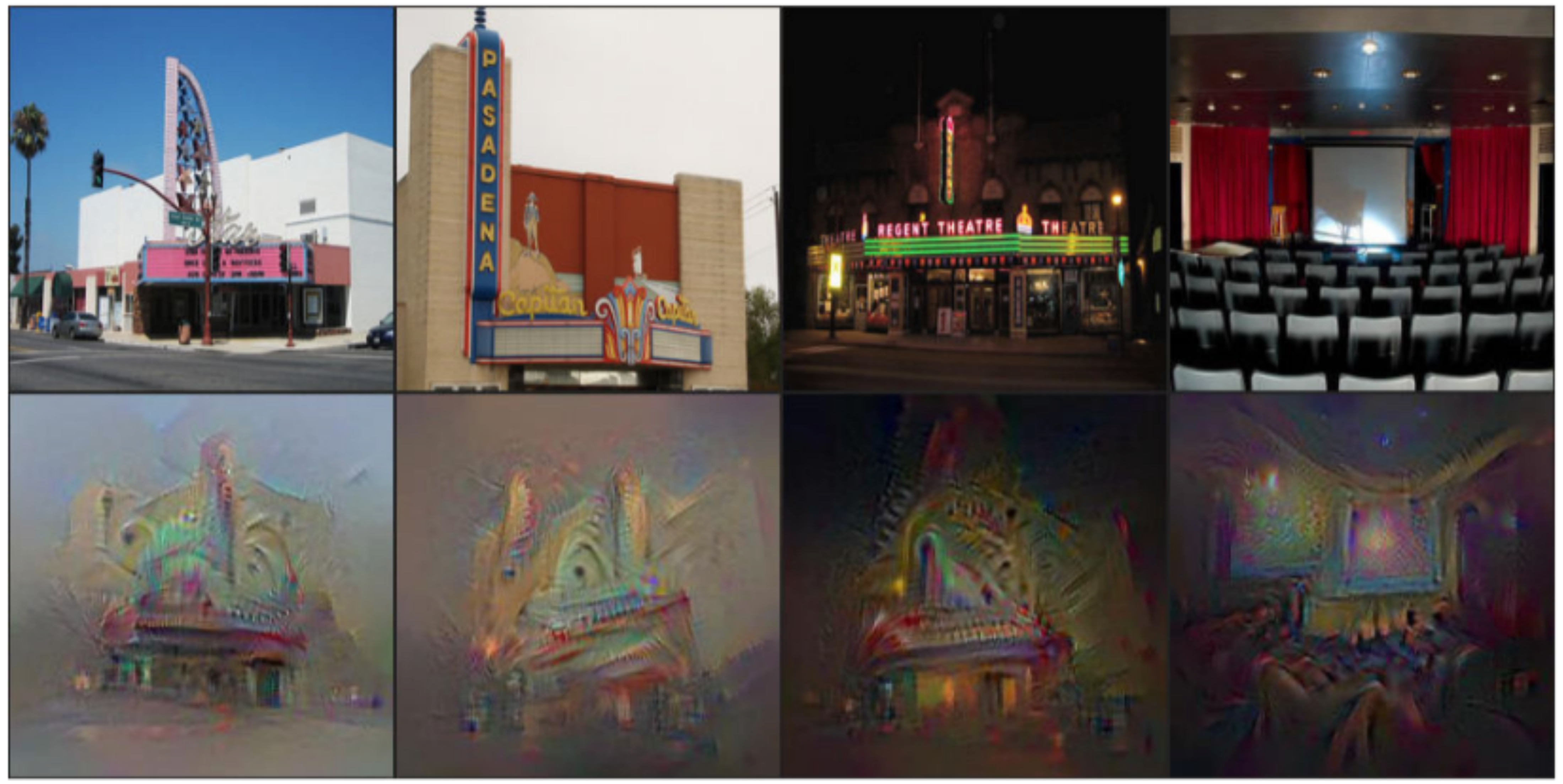

Reconstructions of multiple feature types (facets) recognized by the same “grocery store” neuron
Corresponding example training set images recognized by the same neuron as in the "grocery store" class
CENTRAL MARKET

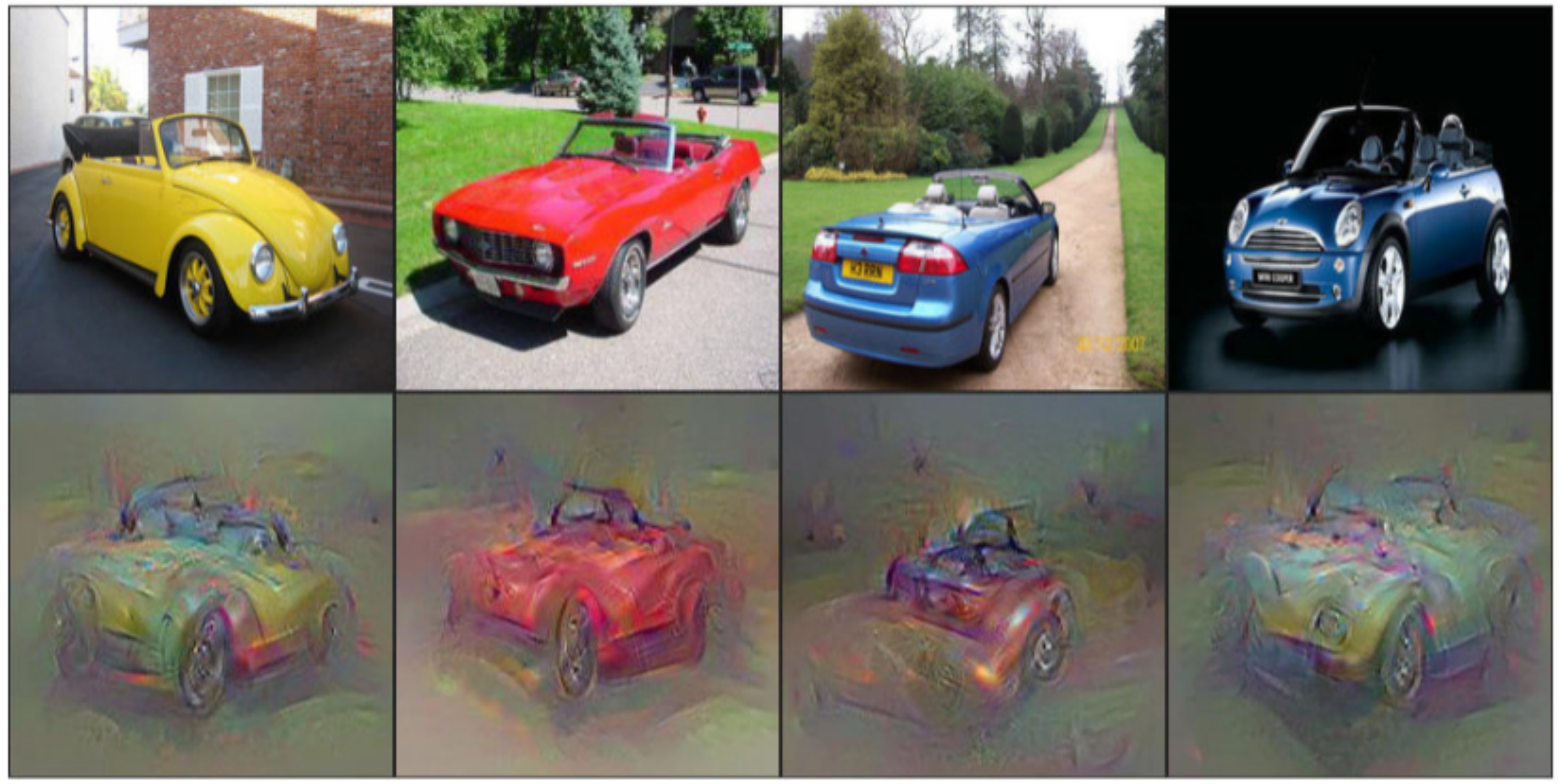
Figure 1. **Top:** Visualizations of 8 types of images (feature facets) that activate the same “grocery store” class neuron. **Bottom:** Example training set images that activate the same neuron, and resemble the corresponding synthetic image in the top panel.

Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks

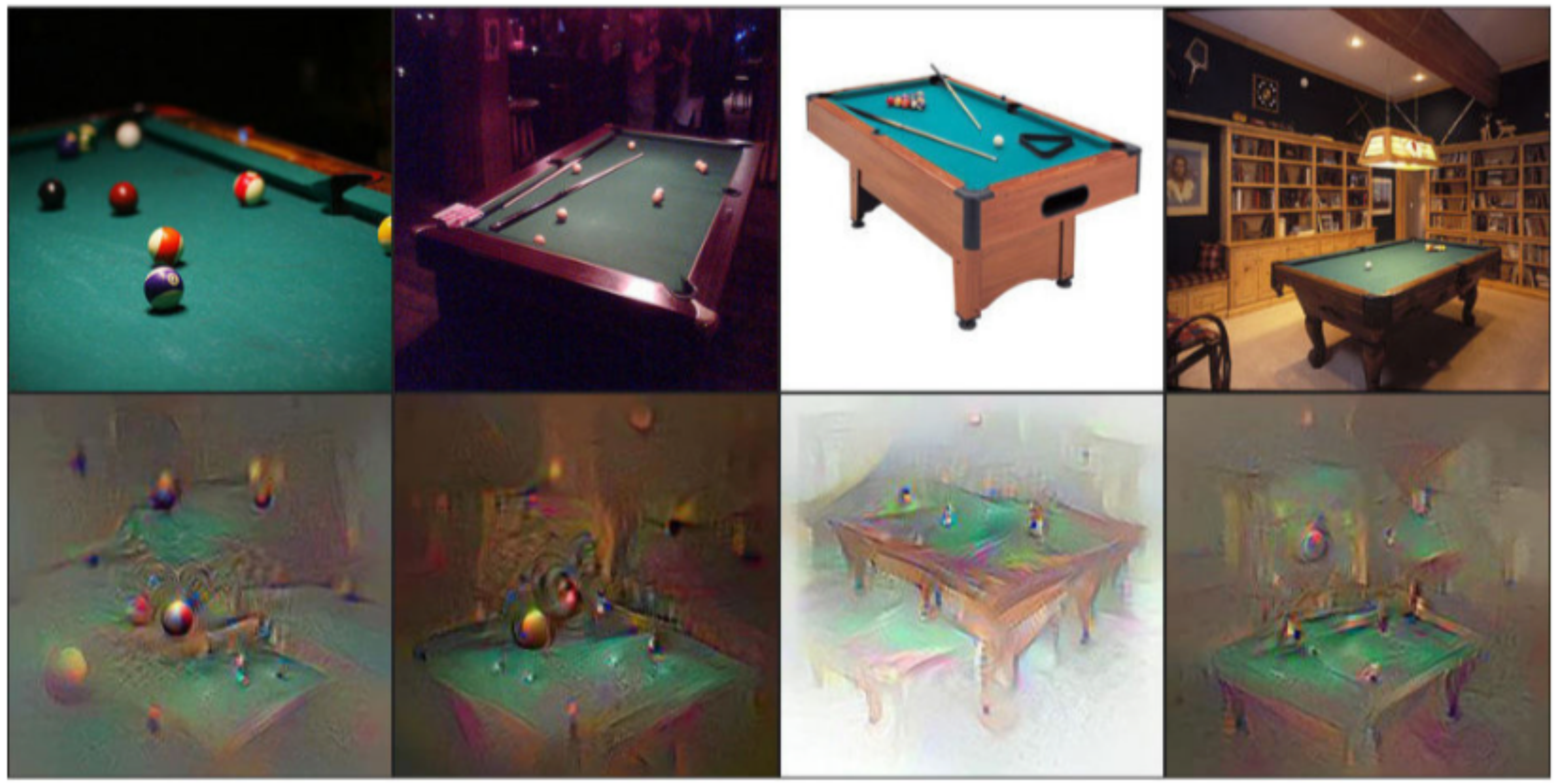
NguyenYosinskiClune2016



(a) *Movie theater*: outside (day & night) and inside views.



(b) *Convertible*: with different colors and both front & rear views.



(c) *Pool table*: Up close & from afar, with different backgrounds.

Figure 4. Multifaceted visualization of fc8 units uncovers interesting facets. We show 4 different facets for each neuron. In each pair of images, the bottom is the facet visualization that represents a cluster of images from the training set, and the top is the closest image to the visualization from the same cluster.

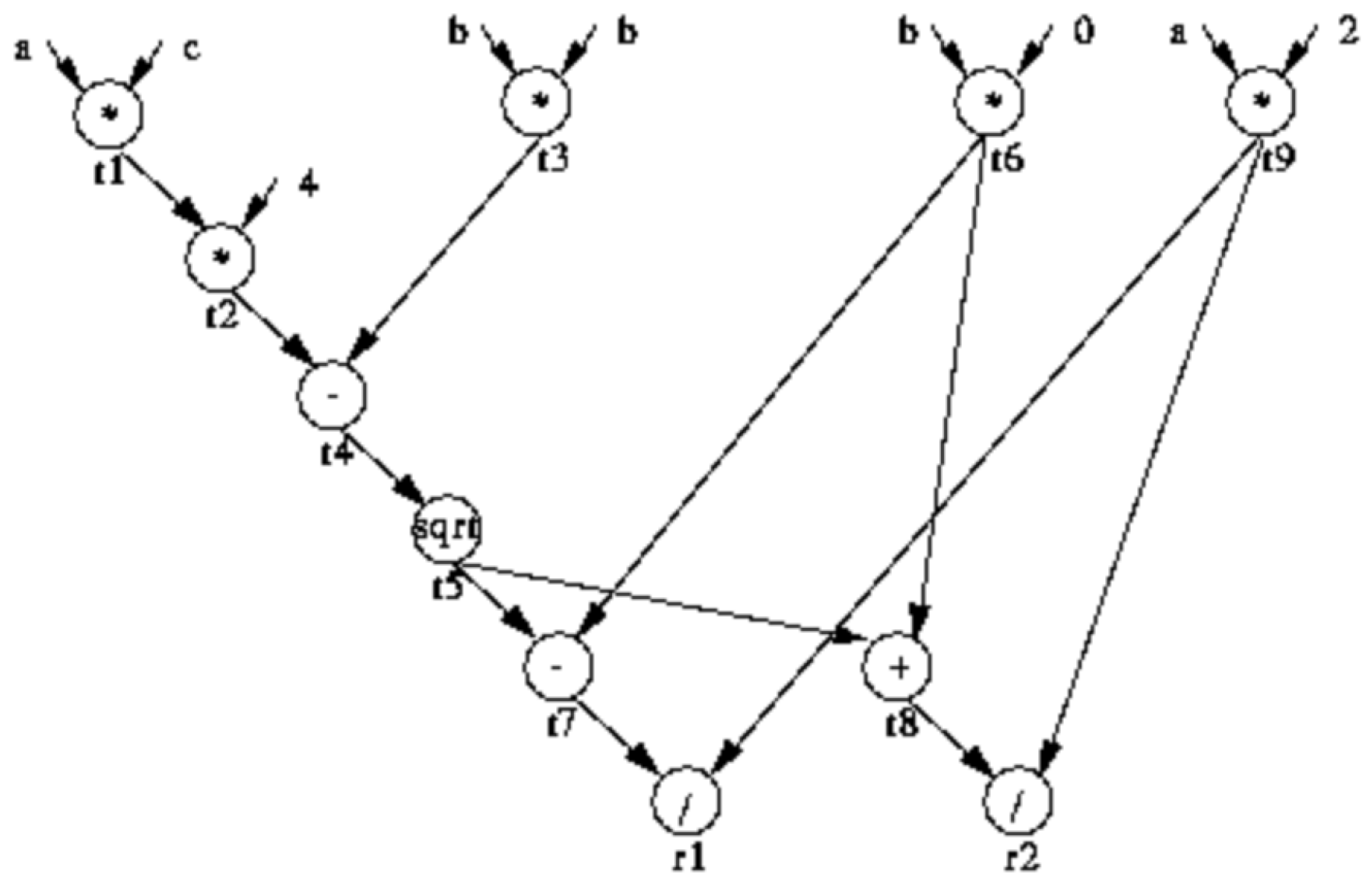
NguyenYosinskiClune2016

Visualizing the Data flow of DL algorithms

Data flow graph

- A data flow graph (DFG) is a graph which represents a data dependancies between a number of operations.

```
quad( a, b, c)
t1 = a*c;
t2 = 4*t1;
t3 = b*b;
t4 = t3 - t2;
t5 = sqrt( t4);
t6 = -b;
t7 = t6 - t5;
t8 = t7 + t5;
t9 = 2*a;
r1 = t7/t9;
r2 = t8/t9;
```



http://bears.ece.ucsb.edu/research-info/DP/dfg.html

Dataflow graph in TensorFlow

- A TensorFlow model is a data flow graph that represents a computation.
- Nodes in the graph represent various operations: addition, matrix multiplication, summary variable operations for storing model parameters, etc.
- Edges in TensorFlow:
  - Data dependency edges represent tensors, or multidimensional arrays, that are input and output data of the operations.
  - Reference edges, or outputs of variable operations, represent pointers to the variable rather than its value
  - Control dependency edges do not represent any data but indicate that their source operations must execute before their tail operations can start.

Simplifying data flow graph

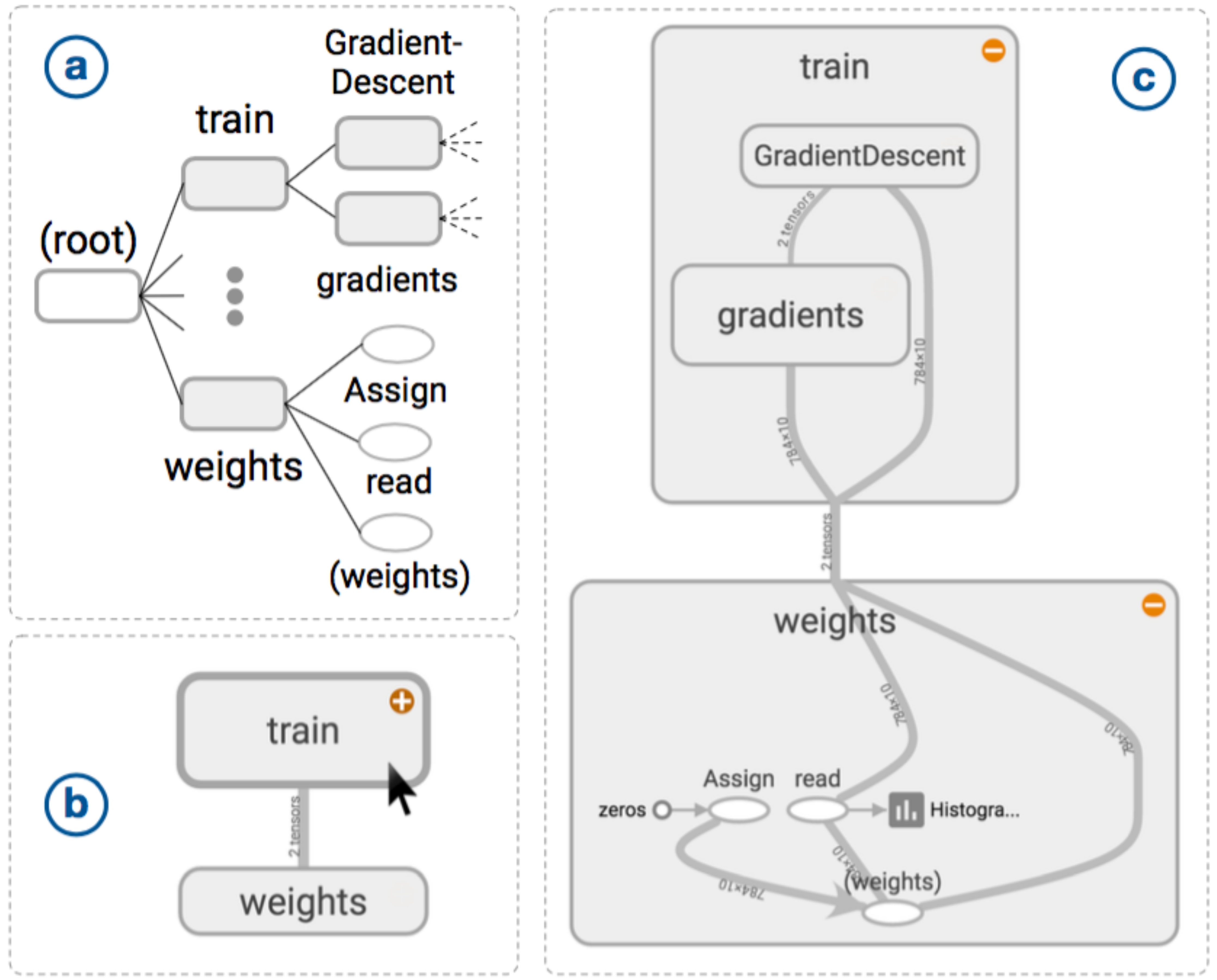
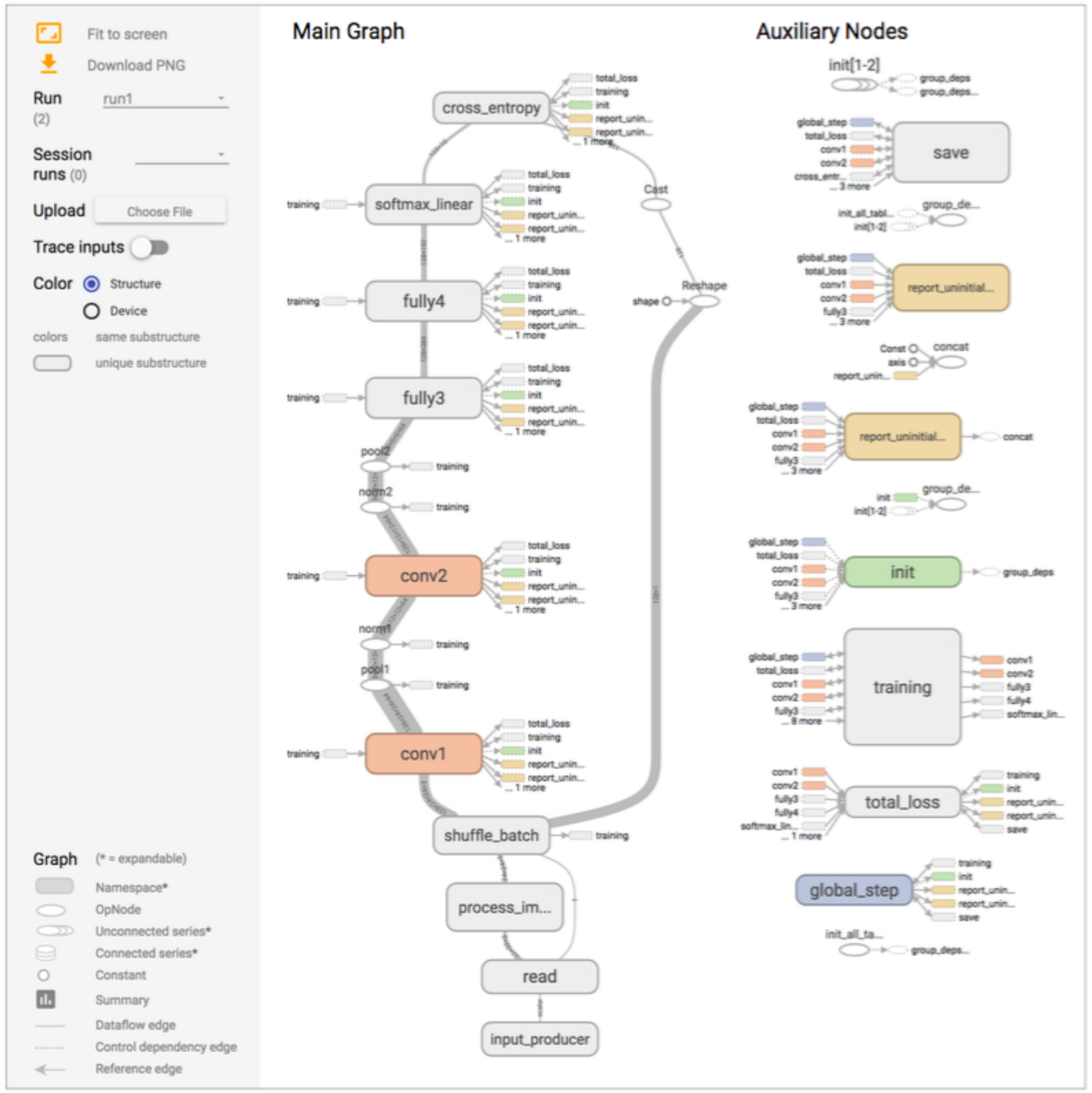


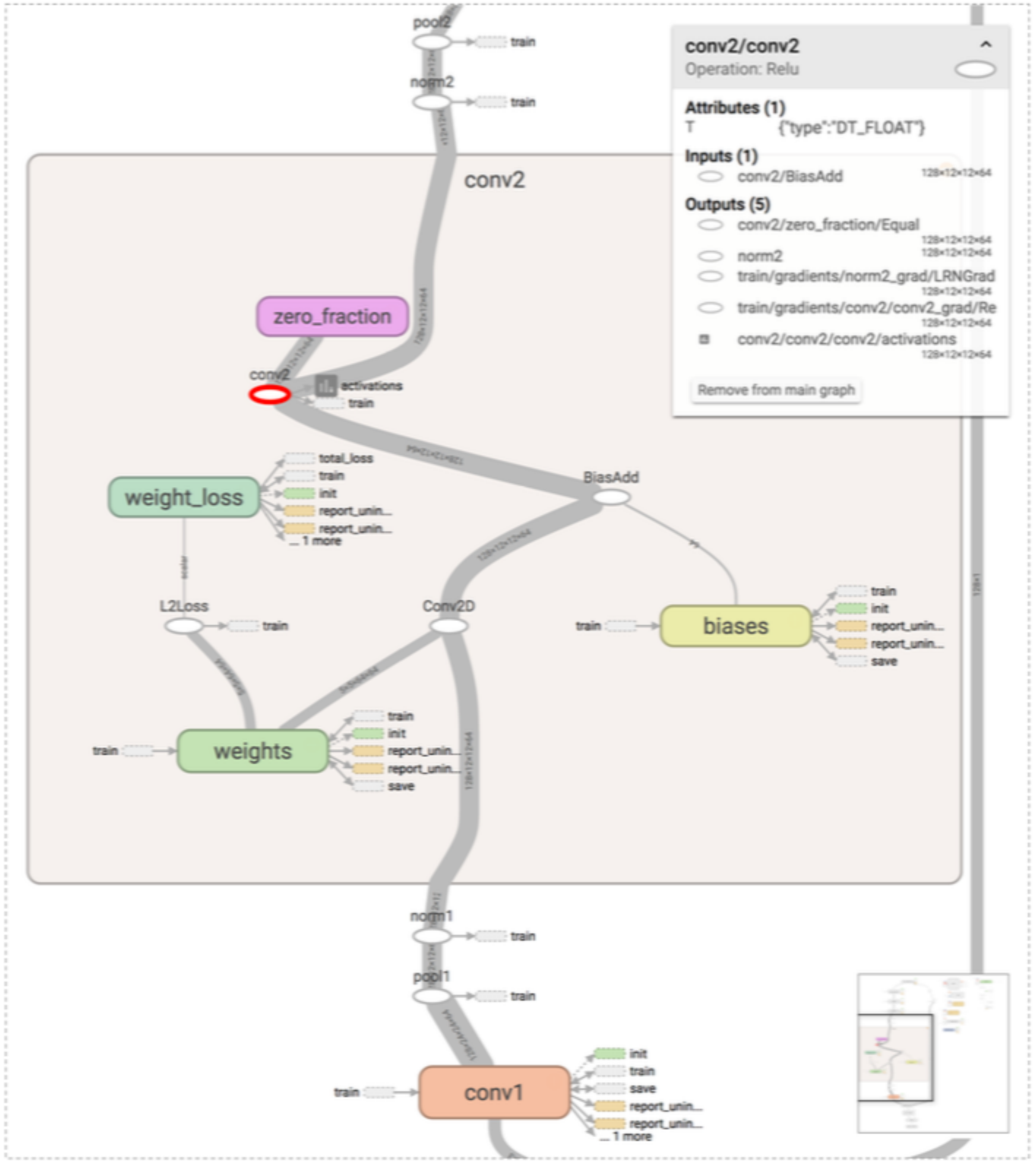
Fig. 5. Build a hierarchical clustered graph. (a) A hierarchy showing only `train` and `weights` namespaces from **`tf_mnist_simple`** in Figure 4. (b) A high-level diagram showing dependency between `train` and `weights`. Hovering over the `train` namespace shows a button for expansion. (c) A diagram with `train` and `weights` expanded.

- Given a low-level directed data flow graph of a model as input, produce an interactive visualization that shows the high-level structure of the model.
- Enables user to explore its nested structure on demand.

WongsuphasawatSmilkovWexler2018



(a)



(b)

Fig. 1. The TensorFlow Graph Visualizer shows a convolutional network for classifying images (**tf_cifar**) . (a) An overview displays a dataflow between groups of operations, with *auxiliary nodes* extracted to the side. (b) Expanding a group shows its nested structure.

WongsuphasawatSmilkovWexler2018

Techniques employed

- Overview: a clustered graph by grouping nodes based on their hierarchical namespaces
- Exploration: edge bundling that supports expansion of clusters
- Declutter: heuristics to extract non-critical nodes
- Detect and highlight repeated structures
- Overlay the graph with additional quantitative information to help developers inspect their models.

Learn more on deep learning

- Stanford deep learning tutorial:
  - http://deeplearning.stanford.edu/tutorial/
  - http://neuralnetworksanddeeplearning.com/

Further Reading

- Workshop on Visualization for Deep Learning
  - http://icmlviz.github.io/
  - https://icmlviz.github.io/icmlviz2016/

Thanks!

Any questions?

You can find me at: beiwang@sci.utah.edu

CREDITS

Special thanks to all people who made and share these awesome resources for free:

- Presentation template designed by Slidesmash
- Photographs by unsplash.com and pexels.com
- Vector Icons by Matthew Skiles

Presentation Design

This presentation uses the following typographies and colors:

Free Fonts used:

http://www.1001fonts.com/oswald-font.html

https://www.fontsquirrel.com/fonts/open-sans

Colors used

🟦 🟦 🟨 🟩 ⬛