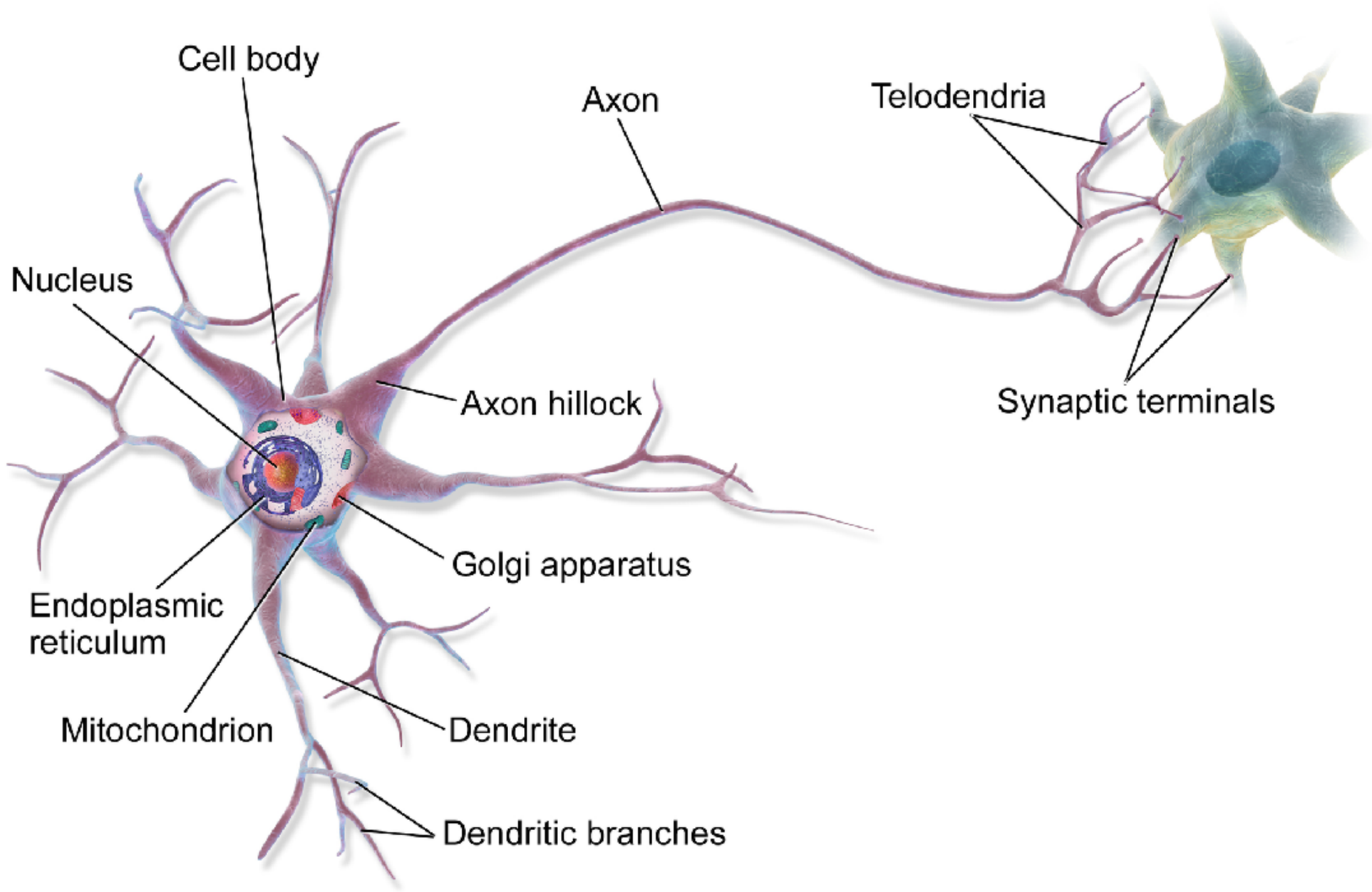
Lecture 11:

Deep Learning Internals

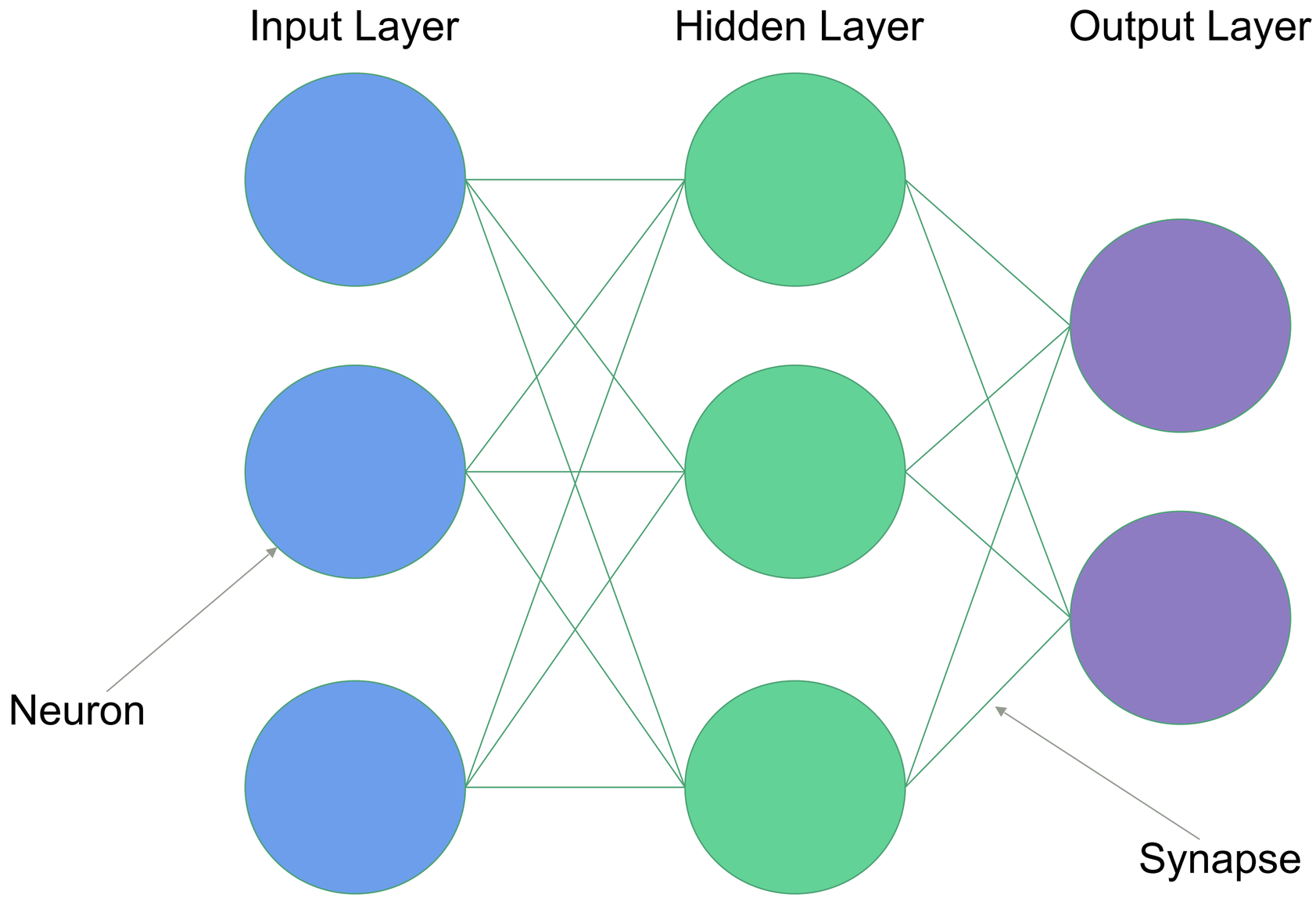
Modern Parallel Computing
John Owens
EEC 289Q, UC Davis, Winter 2018

Deck credit: J. Seng

Biological Inspiration



Neural Network Architecture

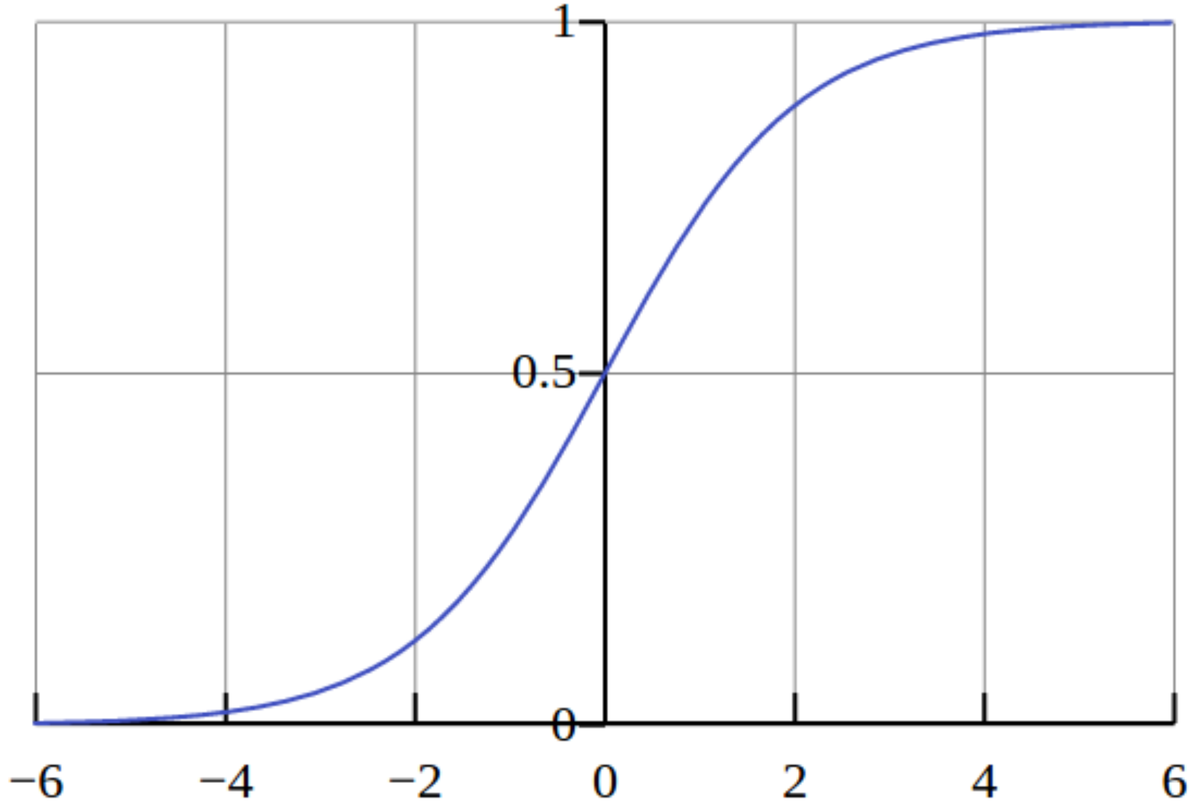


Activation Functions

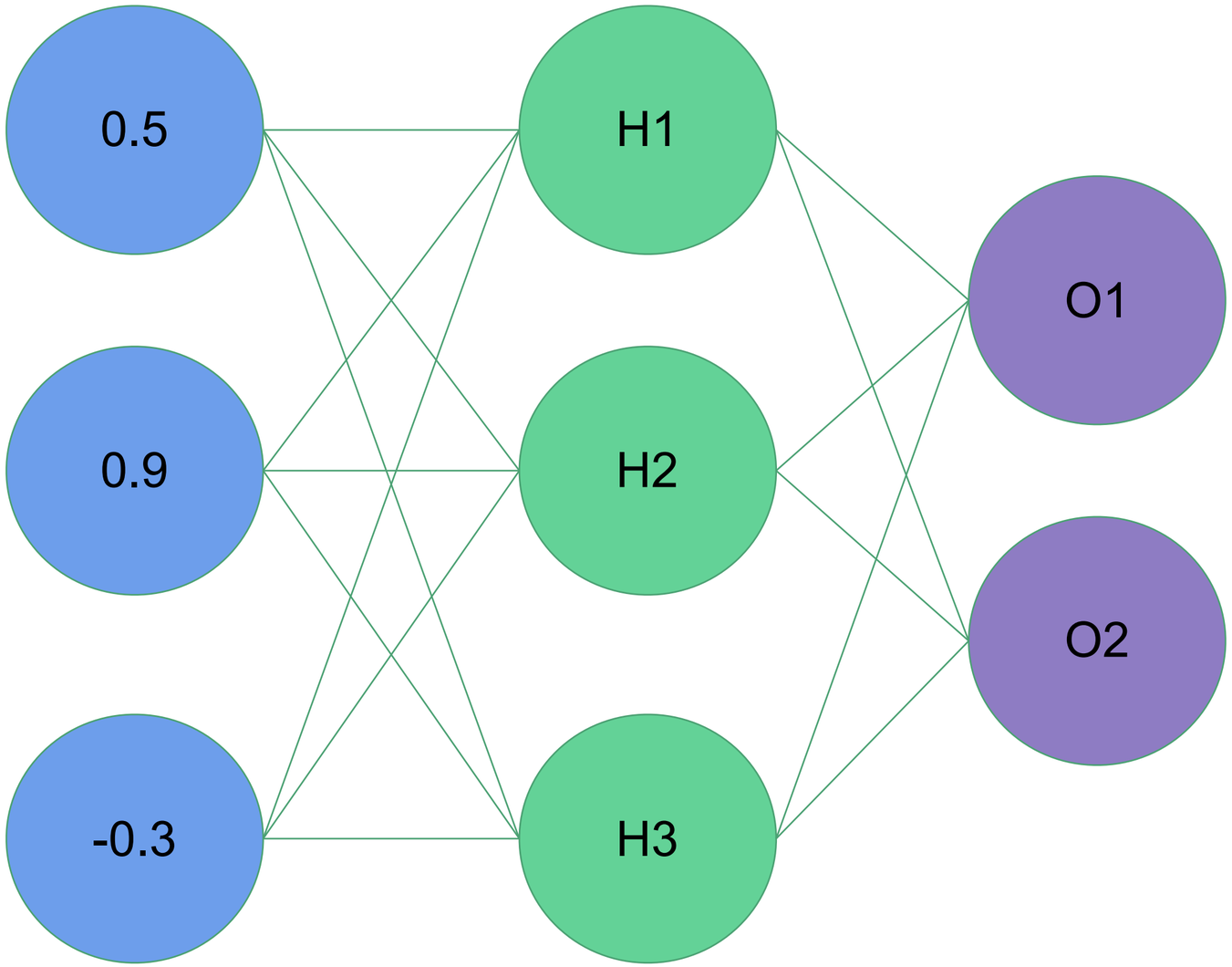
my reference says outputs

- Activation Functions are applied to the inputs at each neuron
  - A common activation function is the Sigmoid
    - Small changes in input -> small changes in output (this is desirable)

$$S(t) = \frac{1}{1 + e^{-t}}$$



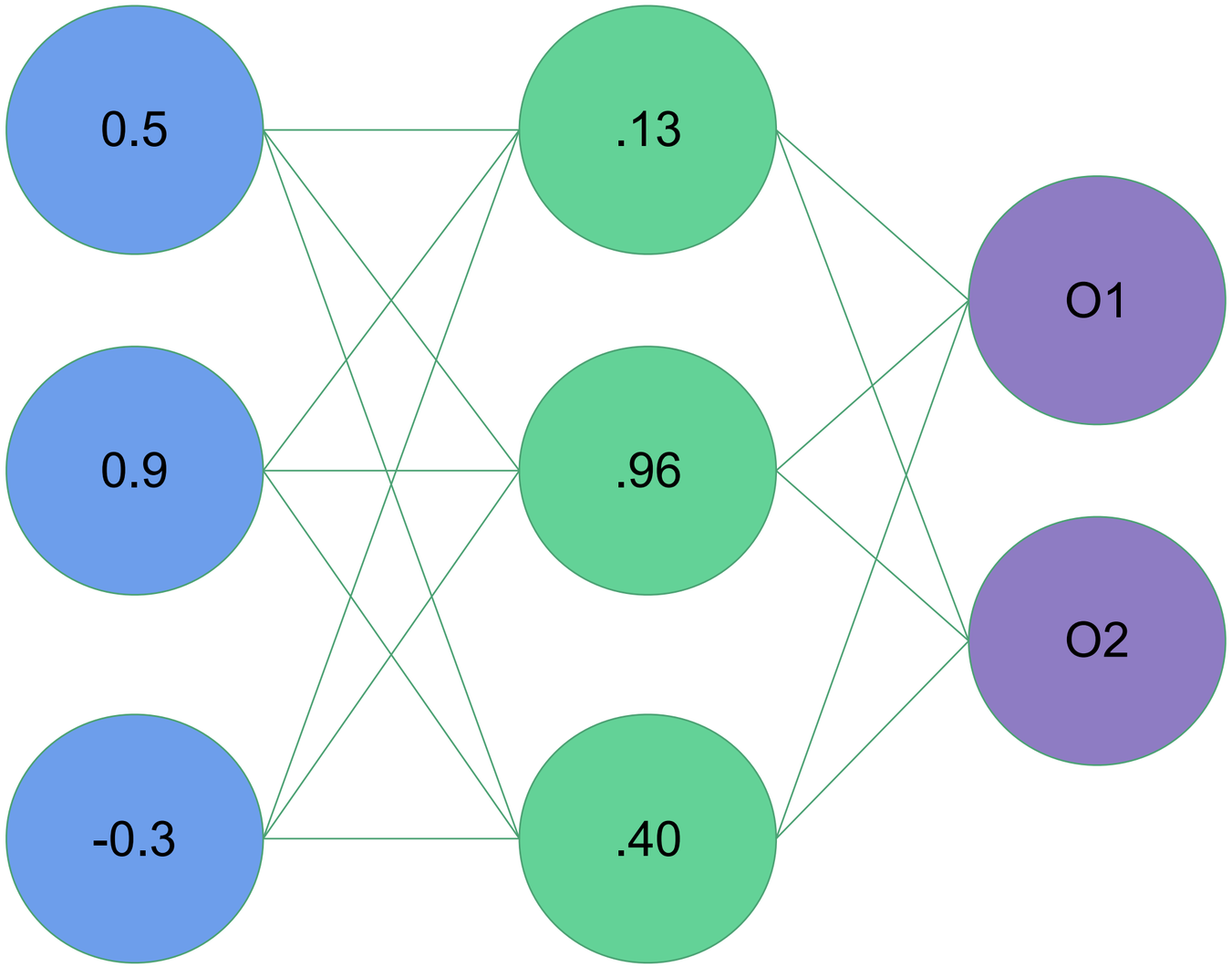
Inference



H1 Weights = (1.0, -2.0, 2.0)
H2 Weights = (2.0, 1.0, -4.0)
H3 Weights = (1.0, -1.0, 0.0)

O1 Weights = (-3.0, 1.0, -3.0)
O2 Weights = (0.0, 1.0, 2.0)

Inference

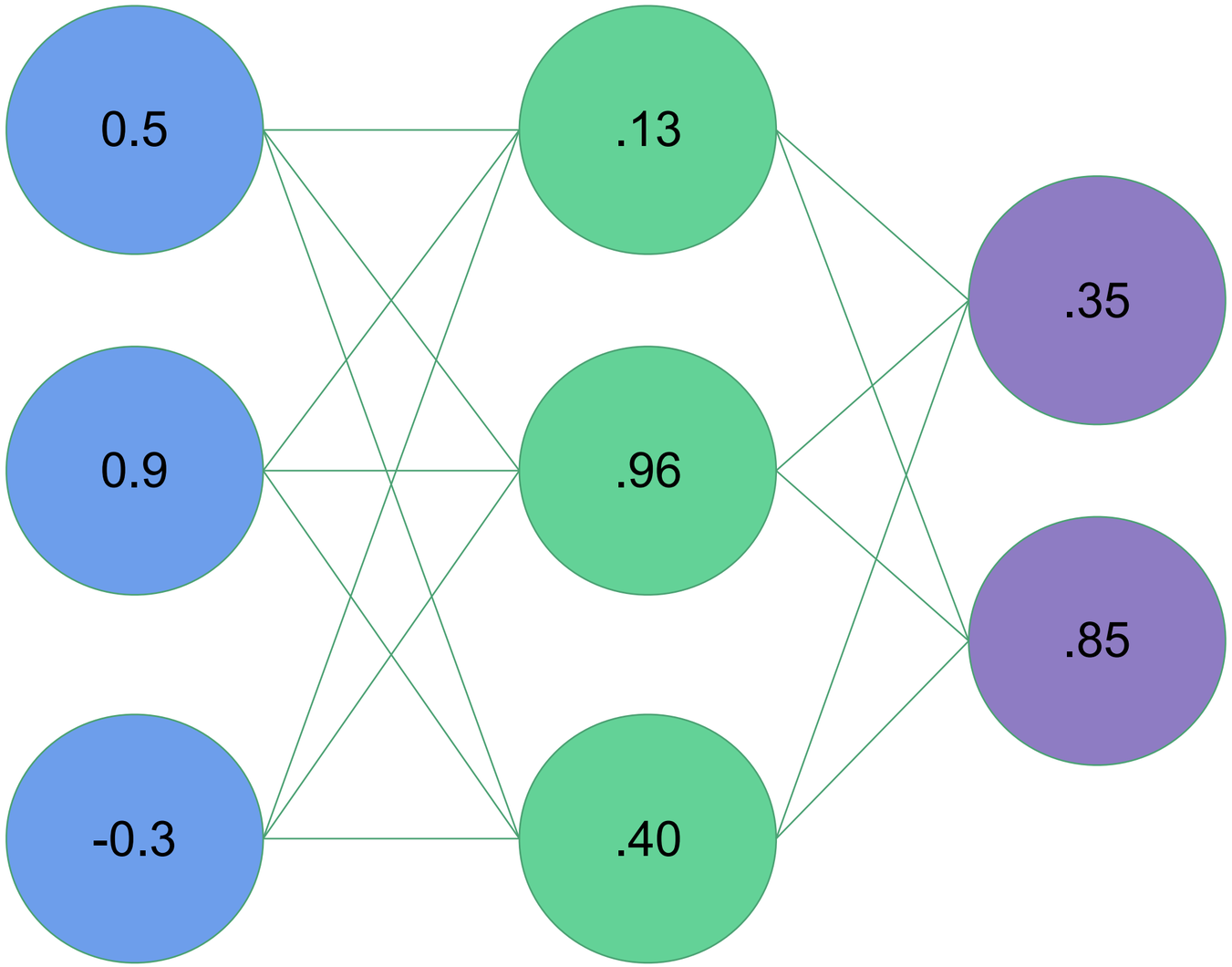


H1 Weights = (1.0, -2.0, 2.0)
H2 Weights = (2.0, 1.0, -4.0)
H3 Weights = (1.0, -1.0, 0.0)

O1 Weights = (-3.0, 1.0, -3.0)
O2 Weights = (0.0, 1.0, 2.0)

H1 = S(0.5 * 1.0 + 0.9 * -2.0 + -0.3 * 2.0) = S(-1.9) = .13
H2 = S(0.5 * 2.0 + 0.9 * 1.0 + -0.3 * -4.0) = S(3.1) = .96
H3 = S(0.5 * 1.0 + 0.9 * -1.0 + -0.3 * 0.0) = S(-0.4) = .40

Inference



H1 Weights = (1.0, -2.0, 2.0)
H2 Weights = (2.0, 1.0, -4.0)
H3 Weights = (1.0, -1.0, 0.0)

O1 Weights = (-3.0, 1.0, -3.0)
O2 Weights = (0.0, 1.0, 2.0)

O1 = S(.13 * -3.0 + .96 * 1.0 + .40 * -3.0) = S(-.63) = .35
O2 = S(.13 * 0.0 + .96 * 1.0 + .40 * 2.0) = S(1.76) = .85

Matrix Formulation

H1 Weights = (1.0, -2.0, 2.0)
H2 Weights = (2.0, 1.0, -4.0)
H3 Weights = (1.0, -1.0, 0.0)

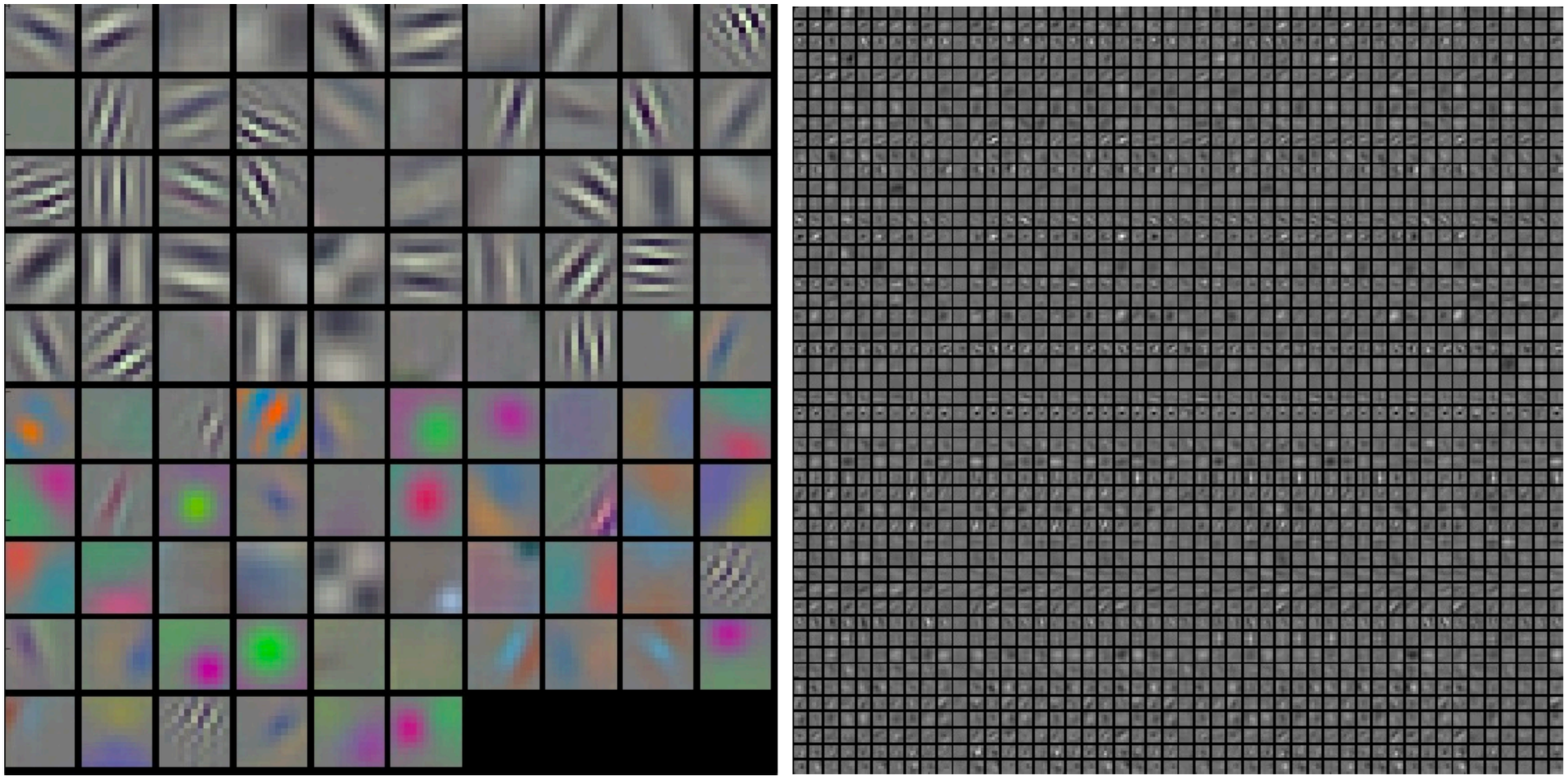
	Hidden Layer Weights				Inputs			Hidden Layer Outputs		
S(	1.0	-2.0	2.0	*	0.5	) = S(	-1.9	3.1	-0.4	) =
	2.0	1.0	-4.0		0.9					.13 .96 0.4
	1.0	-1.0	0.0		-0.3					

This is straightforward to parallelize: it's a dense-matrix, dense-vector product

Training Neural Networks

- Procedure for training Neural Networks
  - Perform inference on the training set
  - Calculate the error between the predictions and actual labels of the training set
  - Determine the contribution of each Neuron to the error
  - Modify the weights of the Neural Network to minimize the error
- Error contributions are calculated using Backpropagation
- Error minimization is achieved with Gradient Descent

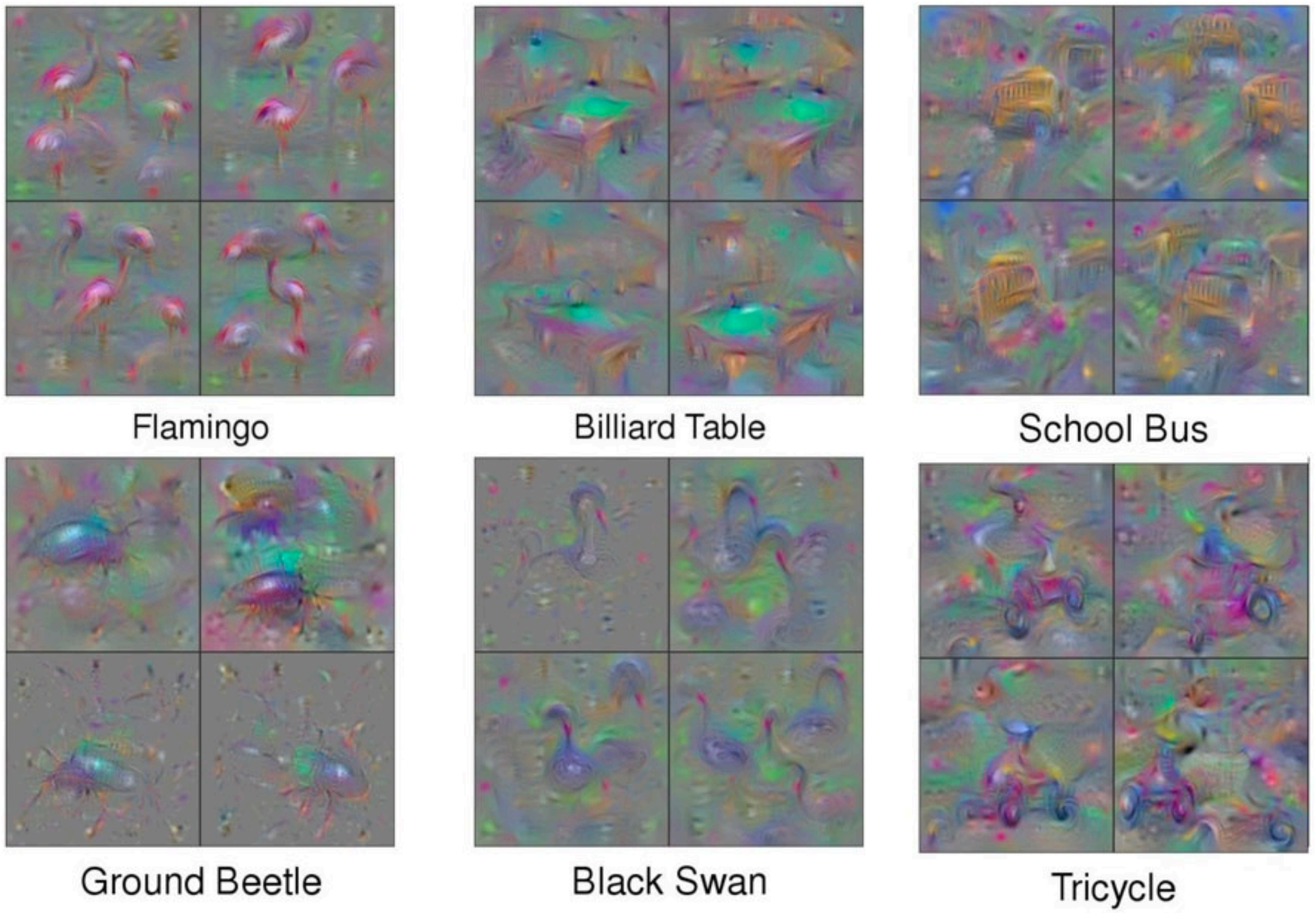
Visualizing what ConvNets learn



Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained AlexNet. Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

http://cs231n.github.io/understanding-cnn/

Understanding Neural Networks Through Deep Visualization

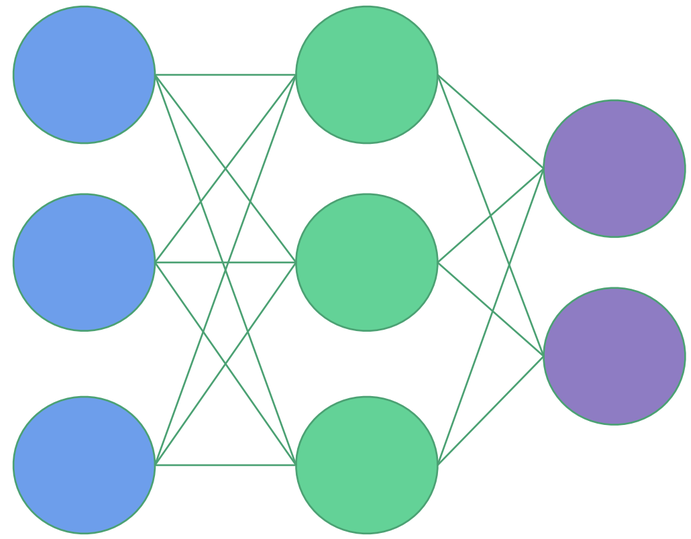


These images are synthetically generated to maximally activate individual neurons in a Deep Neural Network (DNN). They show what each neuron "wants to see", and thus what each neuron has learned to look for. The neurons selected for these images are the output neurons that a DNN uses to classify images as flamingos or school buses. Below we show that similar images can be made for all of the hidden neurons in a DNN. Our paper describes that the key to producing these images with optimization is a good natural image prior.

Yosinski, Jason, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. "Understanding neural networks through deep visualization." *arXiv preprint arXiv:1506.06579* (2015).

How good is our network?

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



– C: cost (objective, loss) function
– w: weights in network
– b: biases in network
– n: all training inputs
– x: inputs
– a: vector of outputs when x is input

– Why quadratic? Smooth function, unlike "number of correct classifications"

also http://neuralnetworksanddeeplearning.com/chap1.html

Find the global minimum (gradient descent)

- Problem: Which weights should be updated and by how much?
  - Insight: Use the derivative of the error with respect to weight to assign “blame”

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

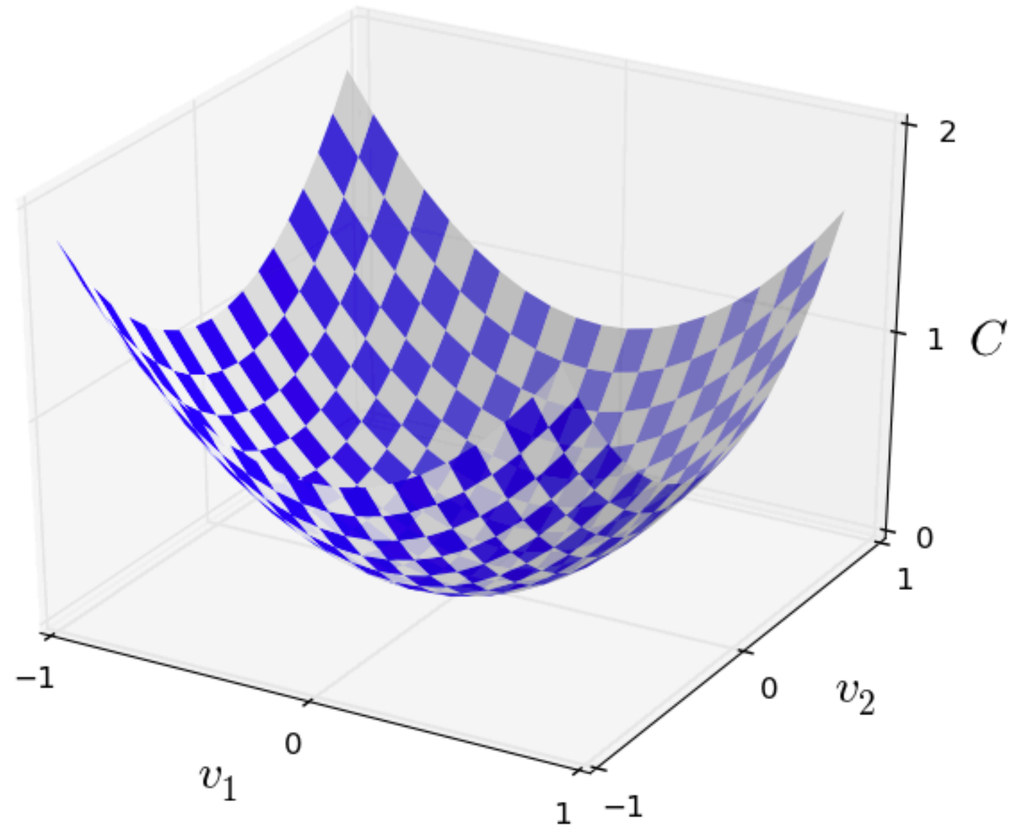
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

$$\Delta v = (\Delta v_1, \Delta v2)^T$$

$$\nabla C = (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2})^T$$

$$\text{Choose } v \rightarrow v' = v - \eta \nabla C$$

$$\text{Now } \Delta C \approx \nabla C \cdot \Delta v = -\eta \nabla C \cdot \nabla C$$

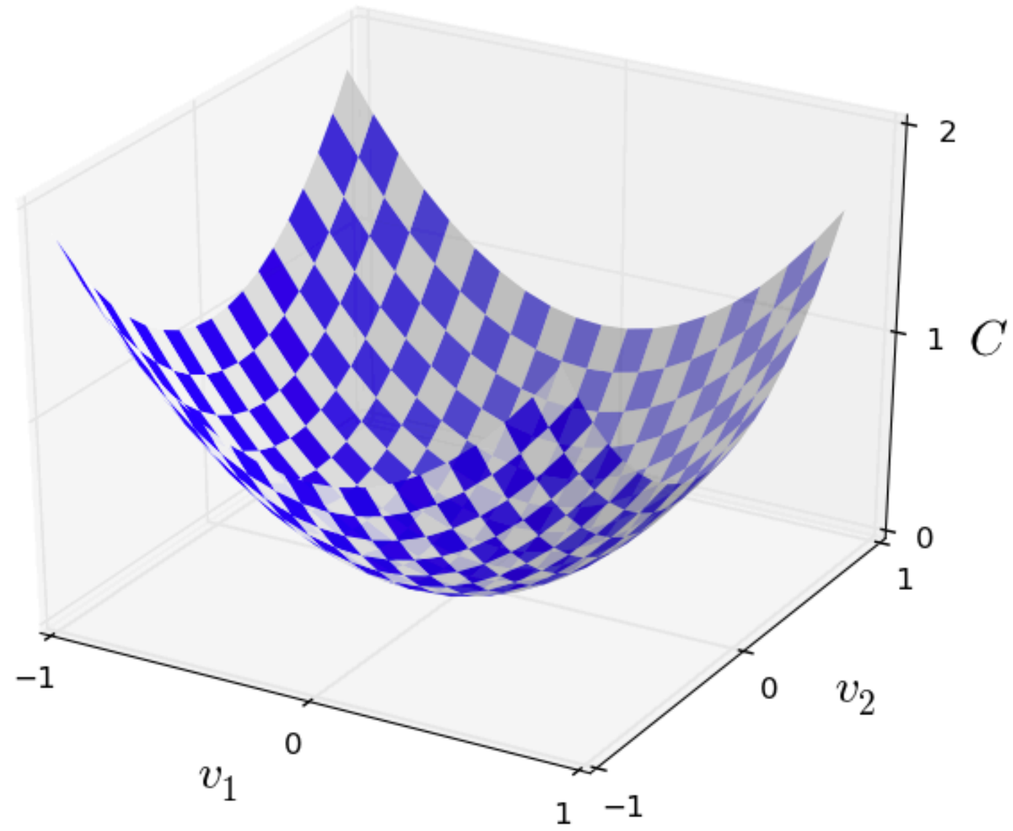


also http://neuralnetworksanddeeplearning.com/chap1.html

What's missing

– How do we compute the partial derivative of the cost function?

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$



http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation

– How do we compute the partial derivative of the cost function?

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

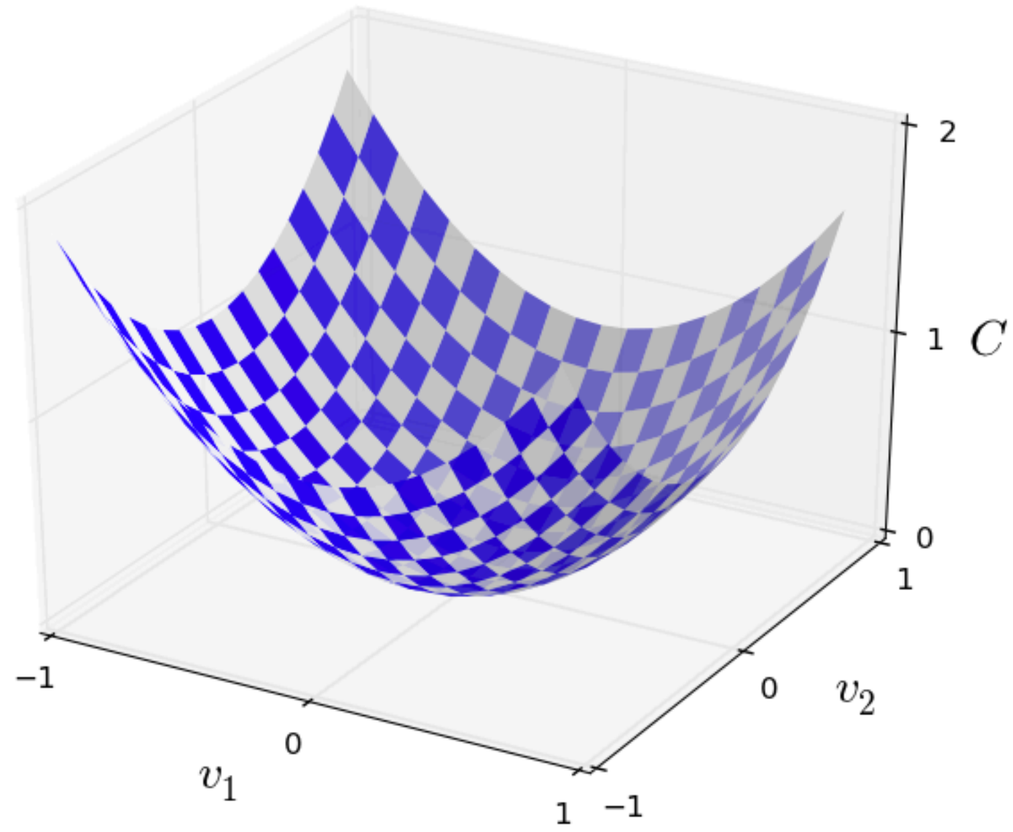
Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(BP2)}$$

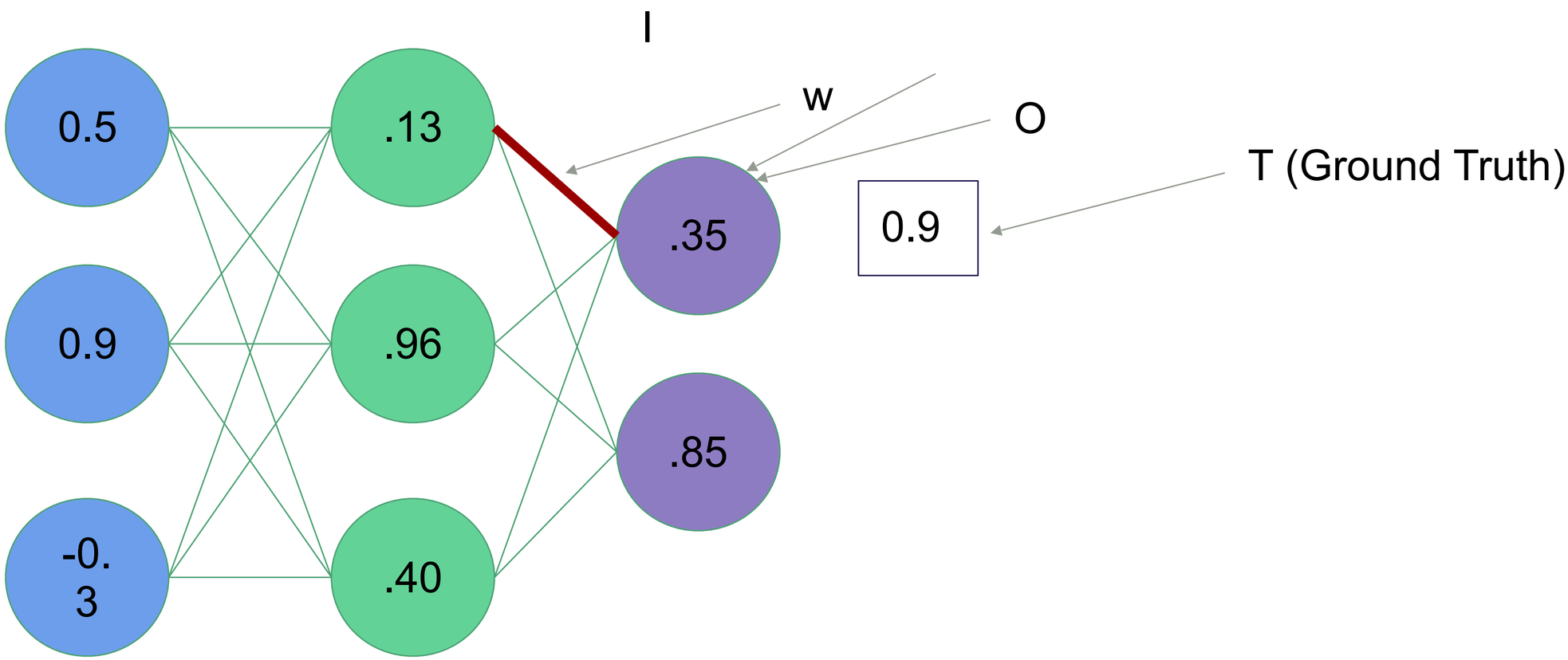
$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad \text{(BP4)}$$



http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation Example



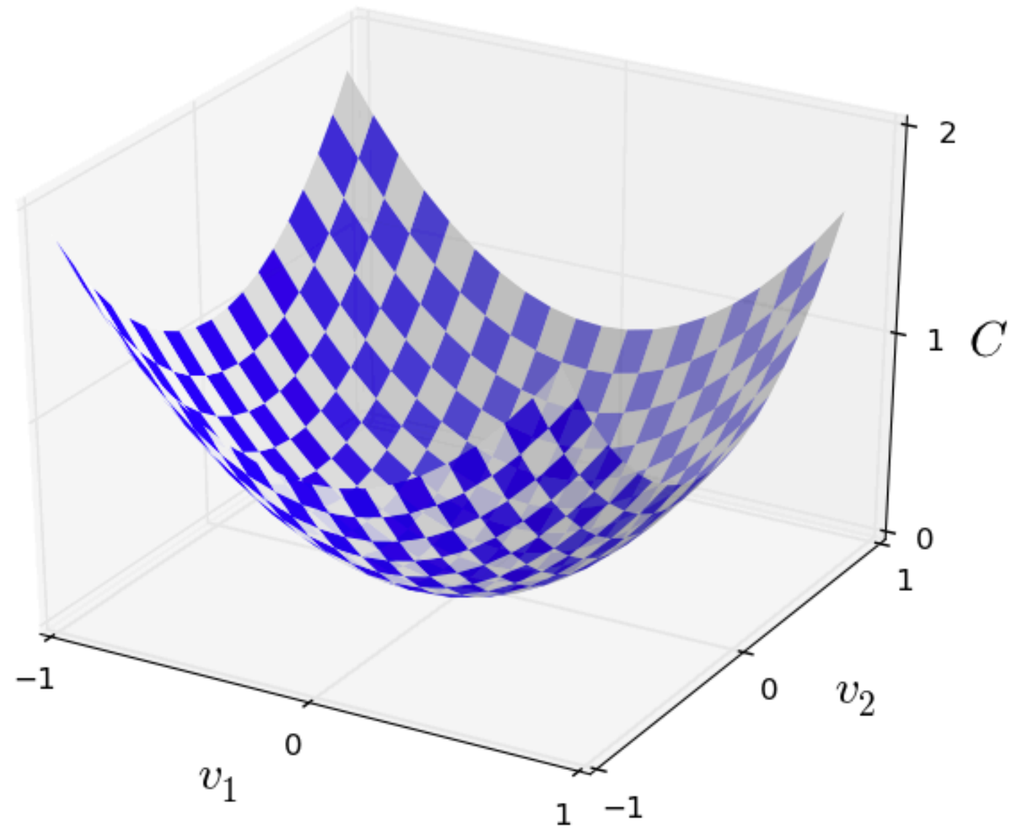
$$\frac{\partial E}{\partial w} = I \cdot (O - T) \cdot O \cdot (1 - O)$$

$$\frac{\partial E}{\partial w} = .13 \cdot (.35 - .9) \cdot .35 \cdot (1 - .35)$$

Stochastic gradient descent

- Gradient Descent minimizes the neural network's error
  - At each time step the error of the network is calculated on the training data
  - Then the weights are modified to reduce the error
- Gradient Descent terminates when
  - The error is sufficiently small
  - The max number of time steps has been exceeded
- Instead of training on all inputs, train on repeated input subsets ("mini-batches") of size m, repeat over all inputs, that's an "epoch"

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$



also http://neuralnetworksanddeeplearning.com/chap1.html

What this means computationally

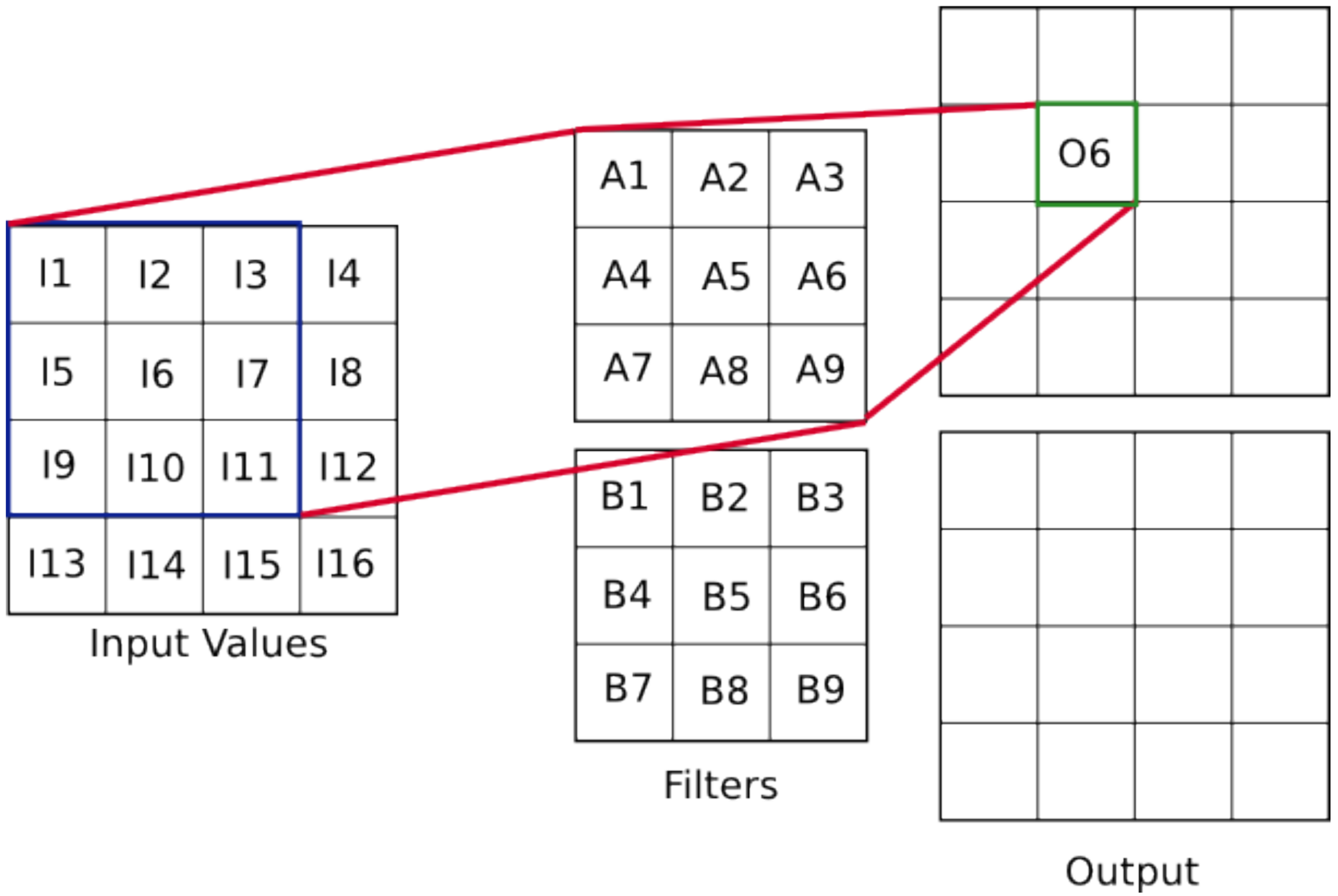
- Forward pass, per stage: dense matrix times dense vector
- Backward (backpropagation) pass, per state: (transpose of) dense matrix times dense vector

- If we train more than one instance at a time (mini-batch):
  - Dense matrix times dense matrix, both directions

More advanced stuff

- Different connectivity of networks (next two slides)
- Different topologies
- Mixed precision
- Special-purpose hardware
- Sparsity

Convolutional Neural Networks



$$
\begin{aligned}
O_6 = A_1 \cdot I_1 + A_2 \cdot I_2 + A_3 \cdot I_3 \\
+ A_4 \cdot I_5 + A_5 \cdot I_6 + A_6 \cdot I_7 \\
+ A_7 \cdot I_9 + A_8 \cdot I_{10} + A_9 \cdot I_{11}
\end{aligned}
$$

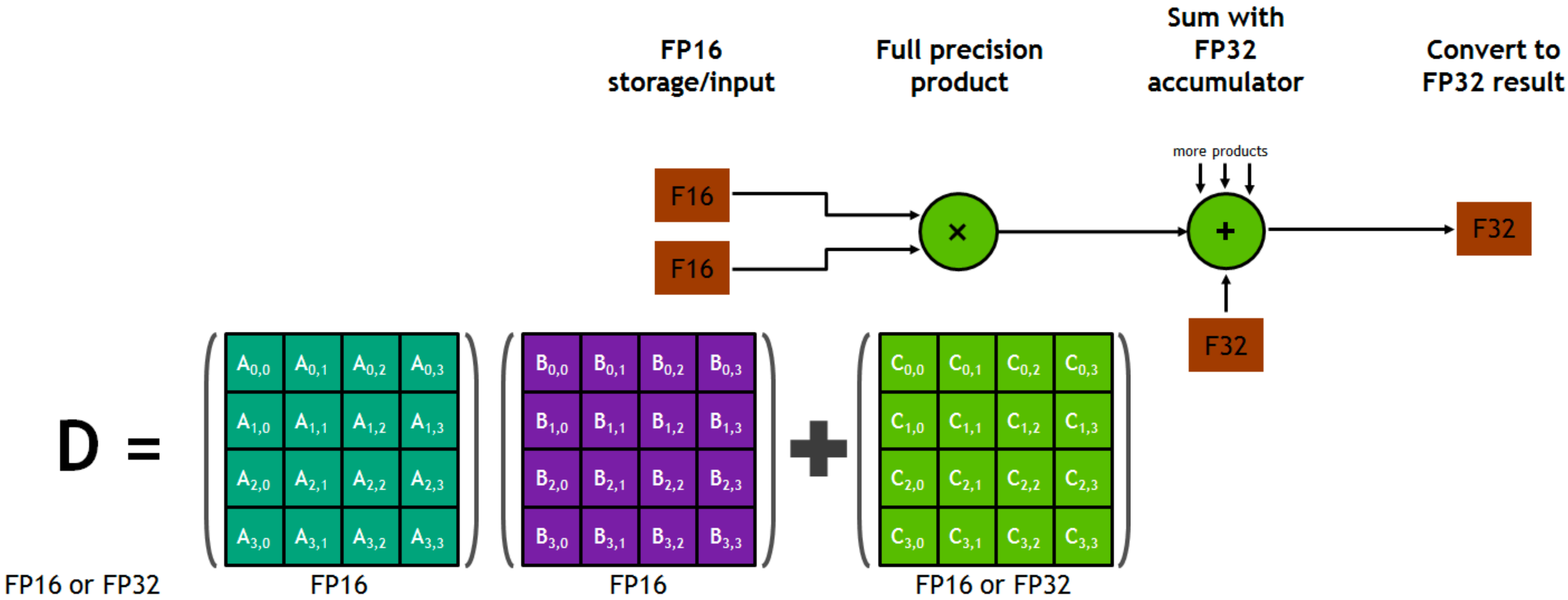
Max-Pooling

1	0	2	2
1	3	0	1
3	1	4	1
2	0	2	1

2x2 Max Pooling

3	2
3	4

Tesla V100 Tensor Cores



- D = A * B + C, where each is a 4x4 matrix
- 8 tensor cores/SM
- Each tensor core does 64 FLOPS/clock
- 8x throughput/SM compared to Pascal