A BEGINNER'S GUIDE TO DEEP LEARNING

Irene Chen

@irenetrampoline

PyCon 2016

"A beginner's guide to deep learning"

“A **beginner**’s guide to **deep learning**”

Convolutional nets

Backpropagation

Image recognition

Restricted Boltzmann machines

DeepMind’s AlphaGo beating professional Go player Lee Sedol

Nvidia and its latest GPU architecture

Toyota’s $1 billion AI investment

Facebook is building AI that builds AI

Geoff Hinton

Yann LeCun

Andrew Ng

Yoshua Bengio

deep learning

All News Videos Books Images

About 13,100,000 results (0.48 seconds)

The standard type of RBM has binary-valued (Boolean/Bernoulli) hidden and visible units, and consists of a matrix of weights $W = (w_{i,j})$ (size *m*×*n*) associated with the connection between hidden unit h_j and visible unit v_i, as well as bias weights (offsets) a_i for the visible units and b_j for the hidden units. Given these, the *energy* of a configuration (pair of boolean vectors) (v,h) is defined as

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j$$

or, in matrix notation,

$$E(v,h) = -a^{\mathrm{T}} v - b^{\mathrm{T}} h - v^{\mathrm{T}} W h$$

This energy function is analogous to that of a Hopfield network. As in general Boltzmann machines, probability distributions over hidden and/or visible vectors are defined in terms of the energy function:[9]

$$P(v,h) = \frac{1}{Z} e^{-E(v,h)}$$

where Z is a partition function defined as the sum of $e^{-E(v,h)}$ over all possible configurations (in other words, just a normalizing constant to ensure the probability distribution sums to 1). Similarly, the (marginal) probability of a visible (input) vector of booleans is the sum over all possible hidden layer configurations:[9]

$$P(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

```
import theano
from theano import tensor as T
from theano.sandbox.rng_mrg import MRG_RandomStreams as RandomStreams
import numpy as np
from load import mnist

srng = RandomStreams()

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def rectify(X):
    return T.maximum(X, 0.)

def softmax(X):
    e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
    return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')

def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        acc = theano.shared(p.get_value() * 0.)
        acc_new = rho * acc + (1 - rho) * g ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        g = g / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((p, p - lr * g))
    return updates

def dropout(X, p=0.):
    if p > 0:
        retain_prob = 1 - p
        X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.confi
        X /= retain_prob
    return X
```

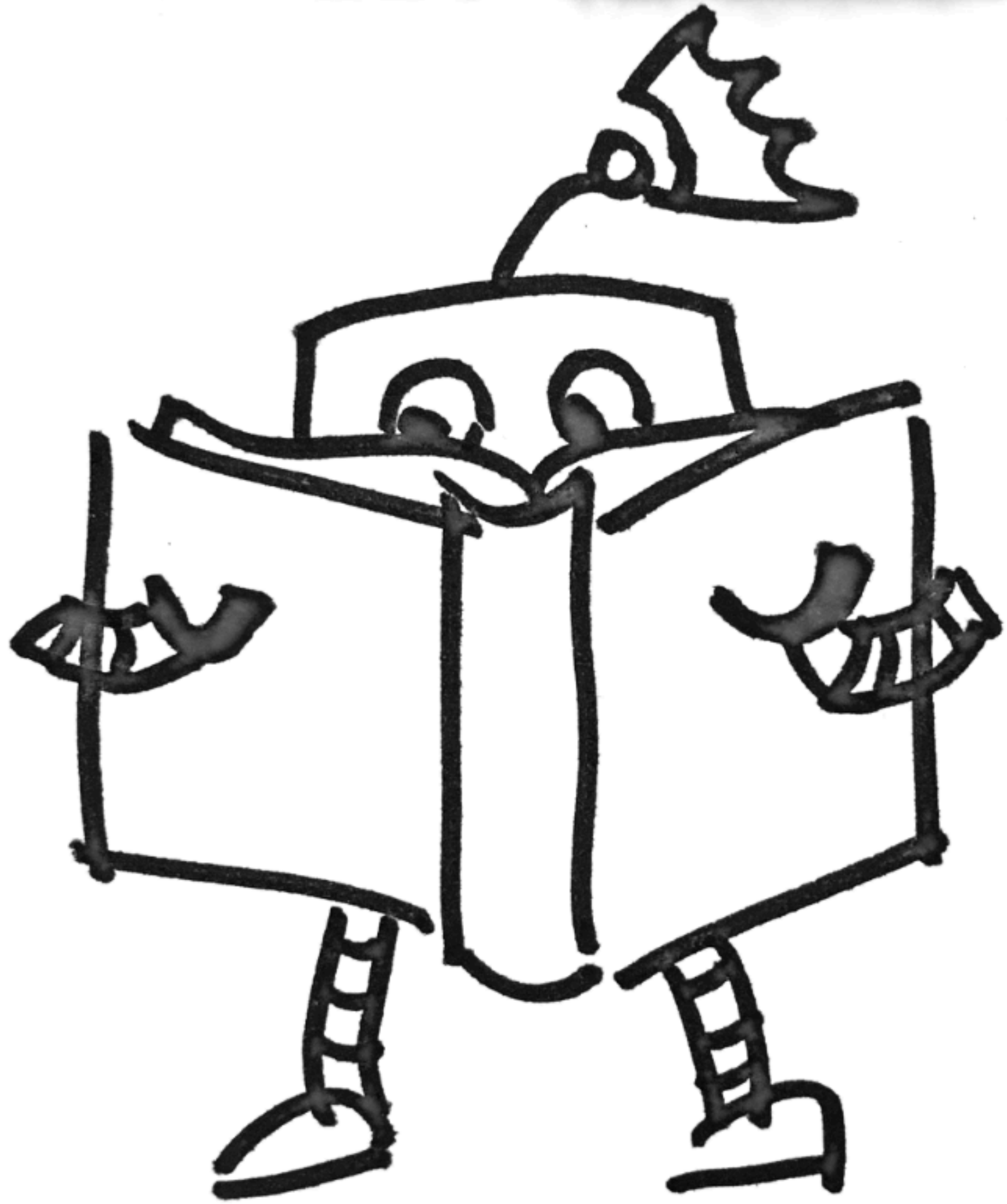
Too much math

Too much code

Today

- Why now?
- Neural Networks in 7 minutes
- Deep nets in Caffe

WHY NOW?



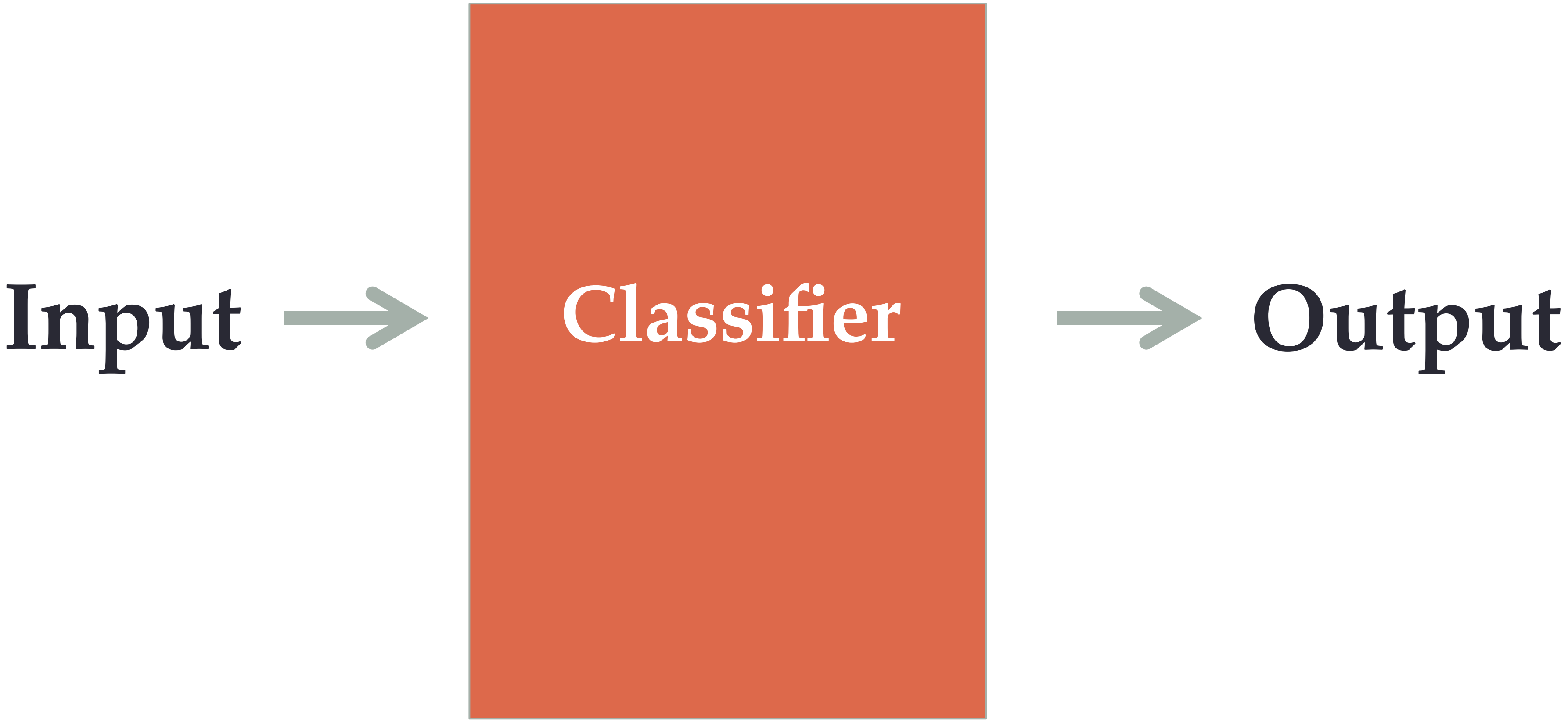

ATLAS

Engine (neural network)

Engine (neural network)

Fuel (data)

Classifier


Input
Classifier
Output

Classifier

Ripe?

pe?

Ripe?

Trained Classifier

Logistic regression

Naïve Bayes

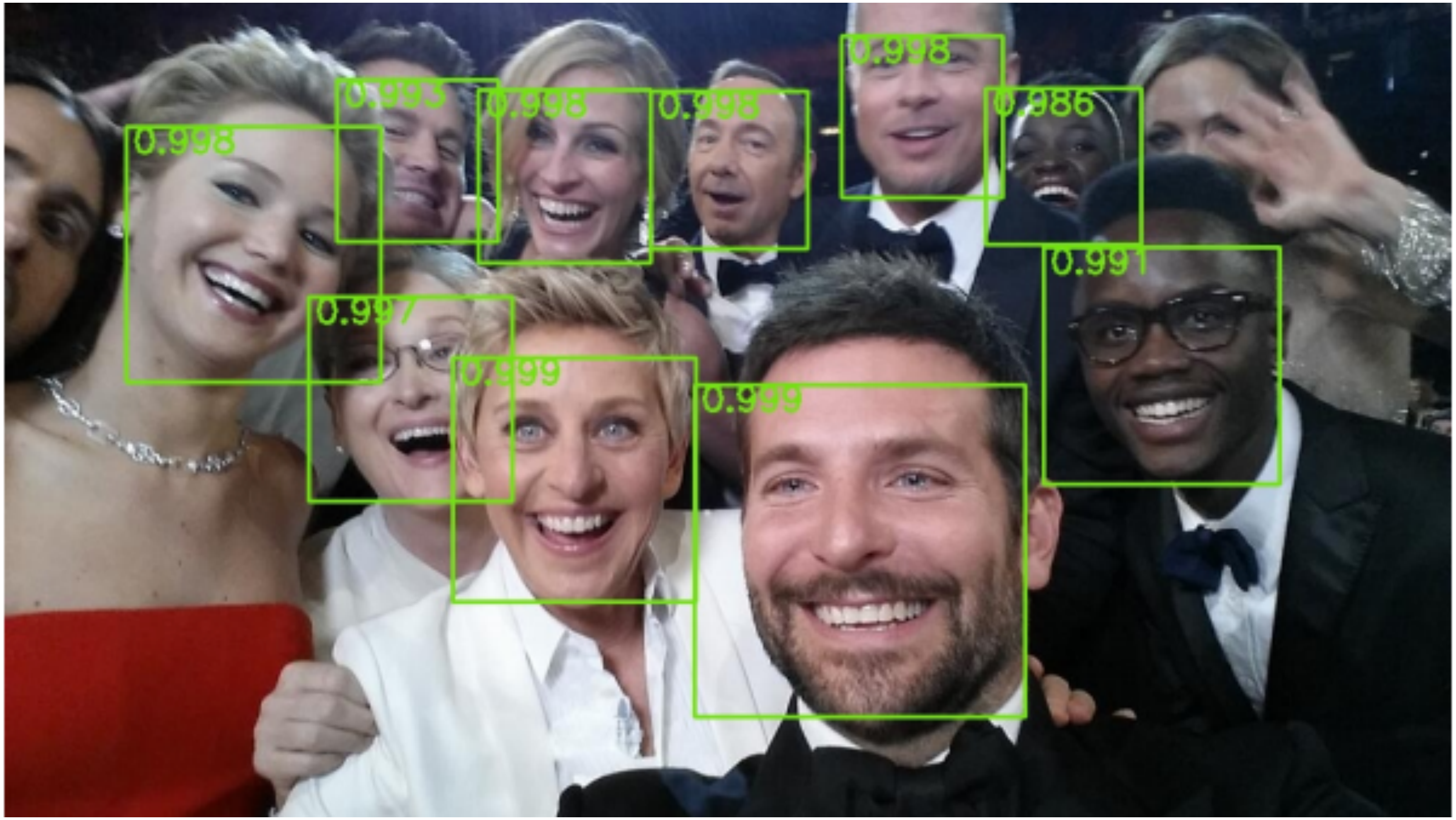
Support vector machine

K-nearest neighbors

Random forests

Ripe?

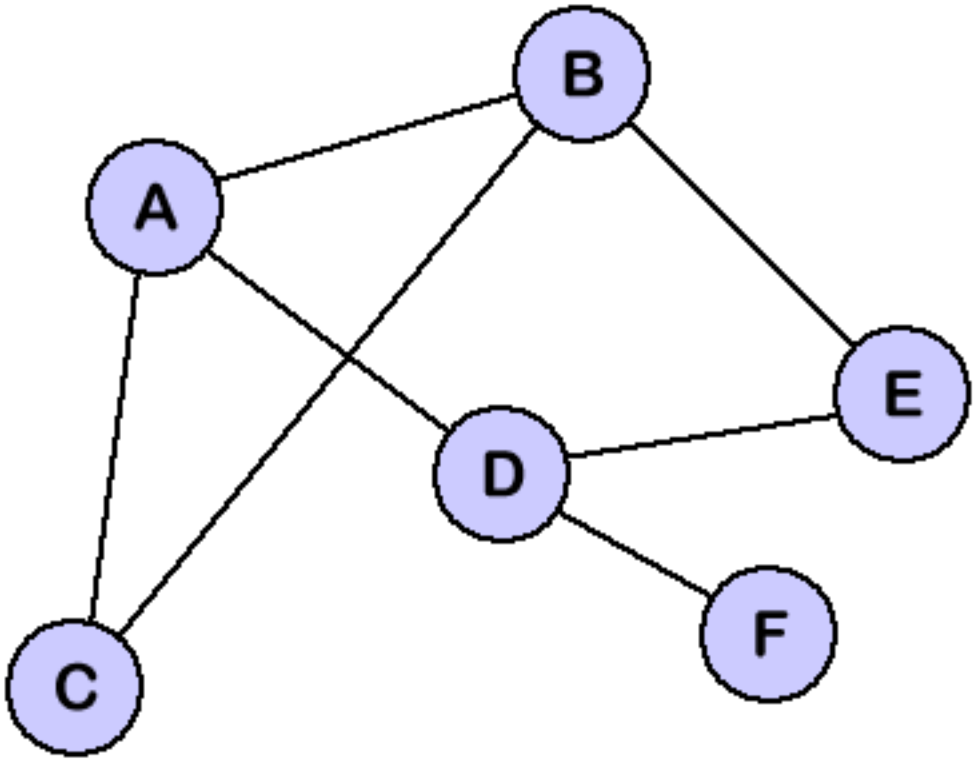
Trained Classifier

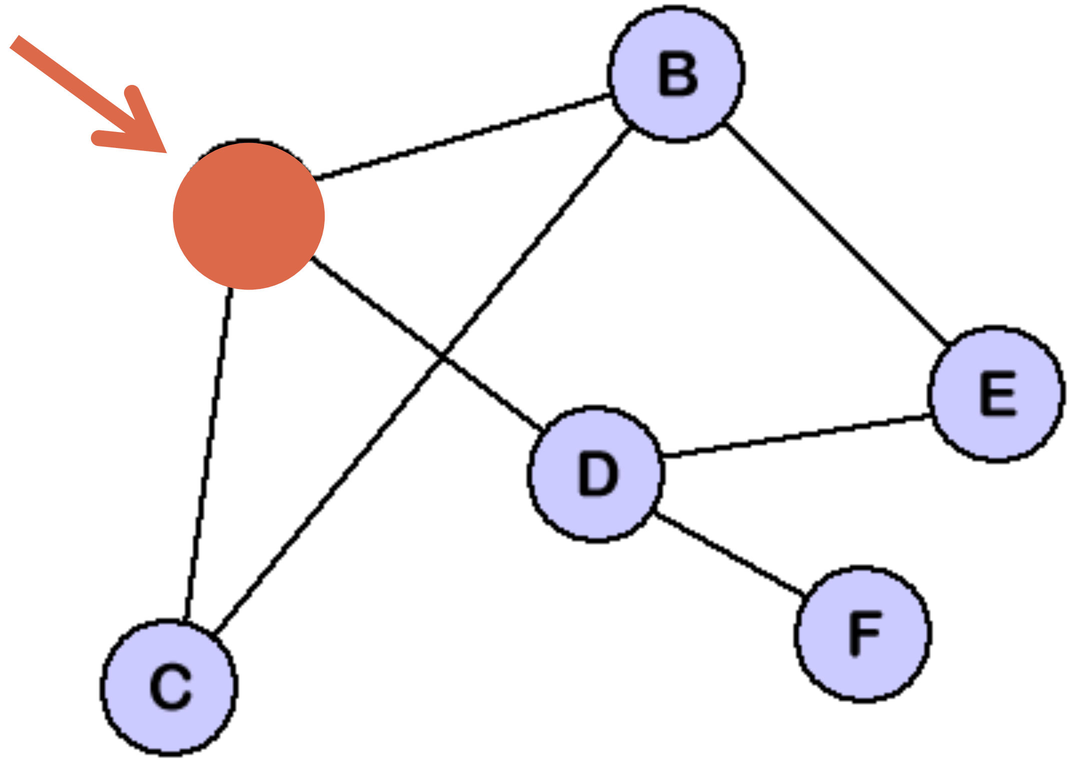

0.993
0.998
0.998
0.998
0.998
0.986
0.991
0.997
0.999
0.999

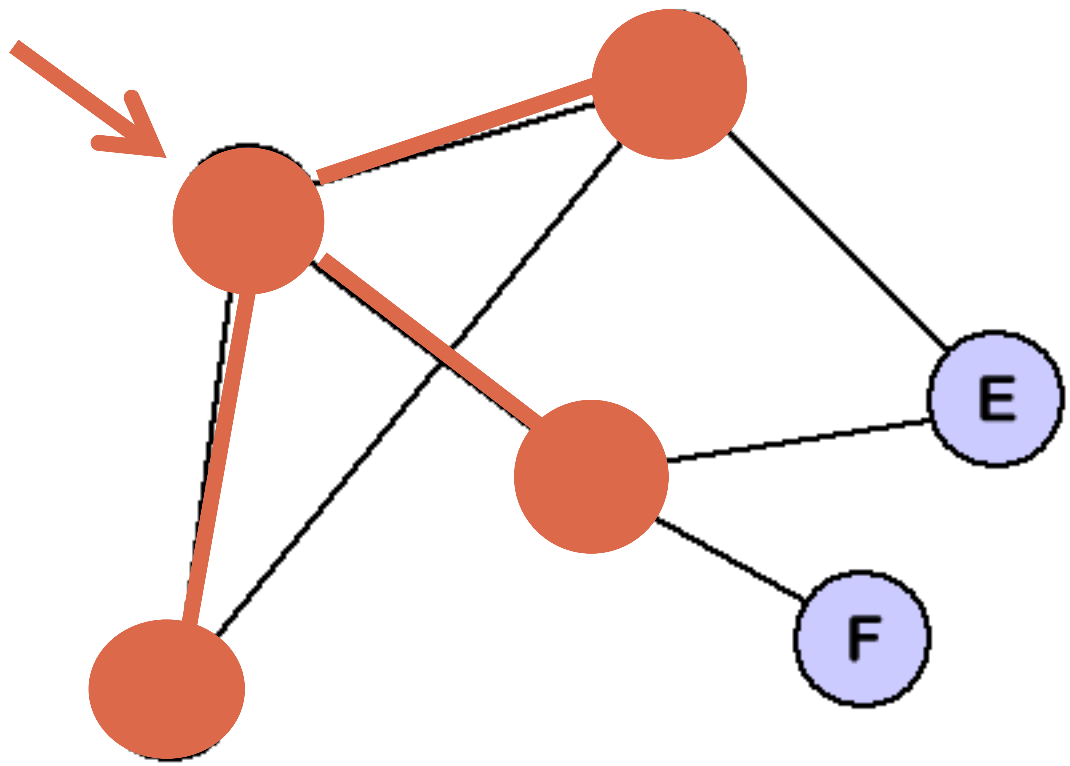
Lesson 1: Why now? Big data, big processing power, robust neural networks

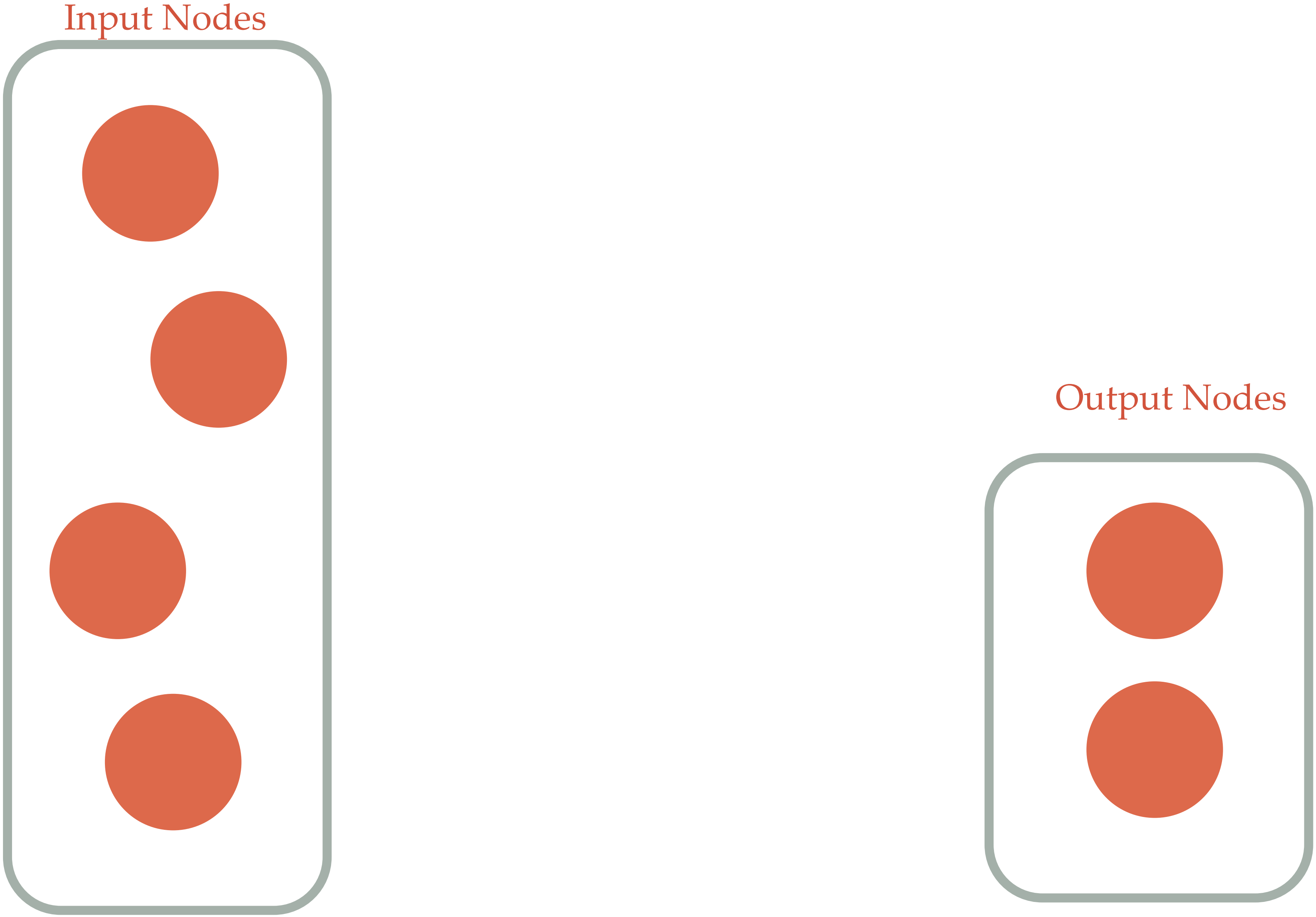
NEURAL NETWORKS IN 7

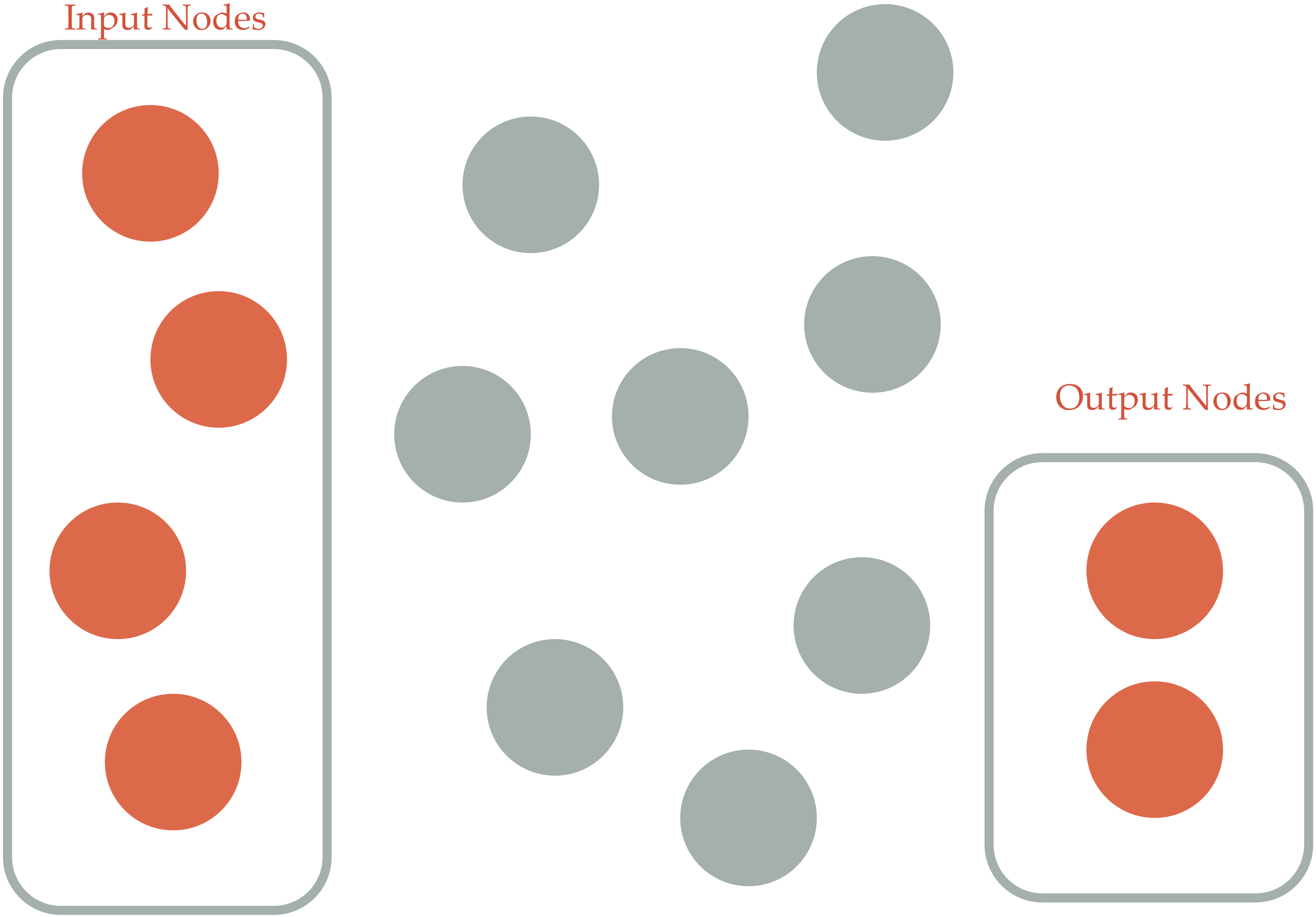
Photo: Rebecca-Lee (Flickr)

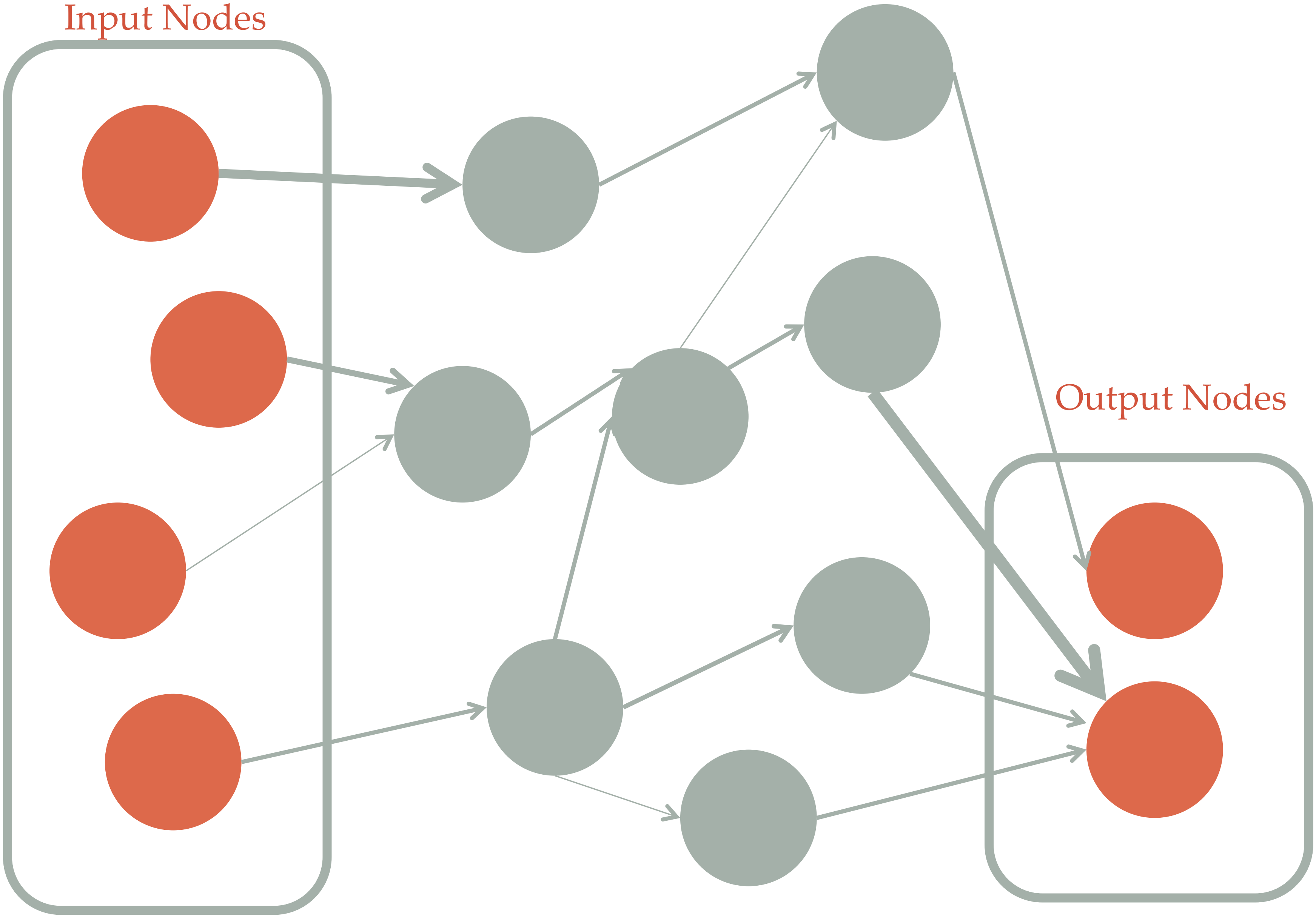

A
B
C
D
E
F

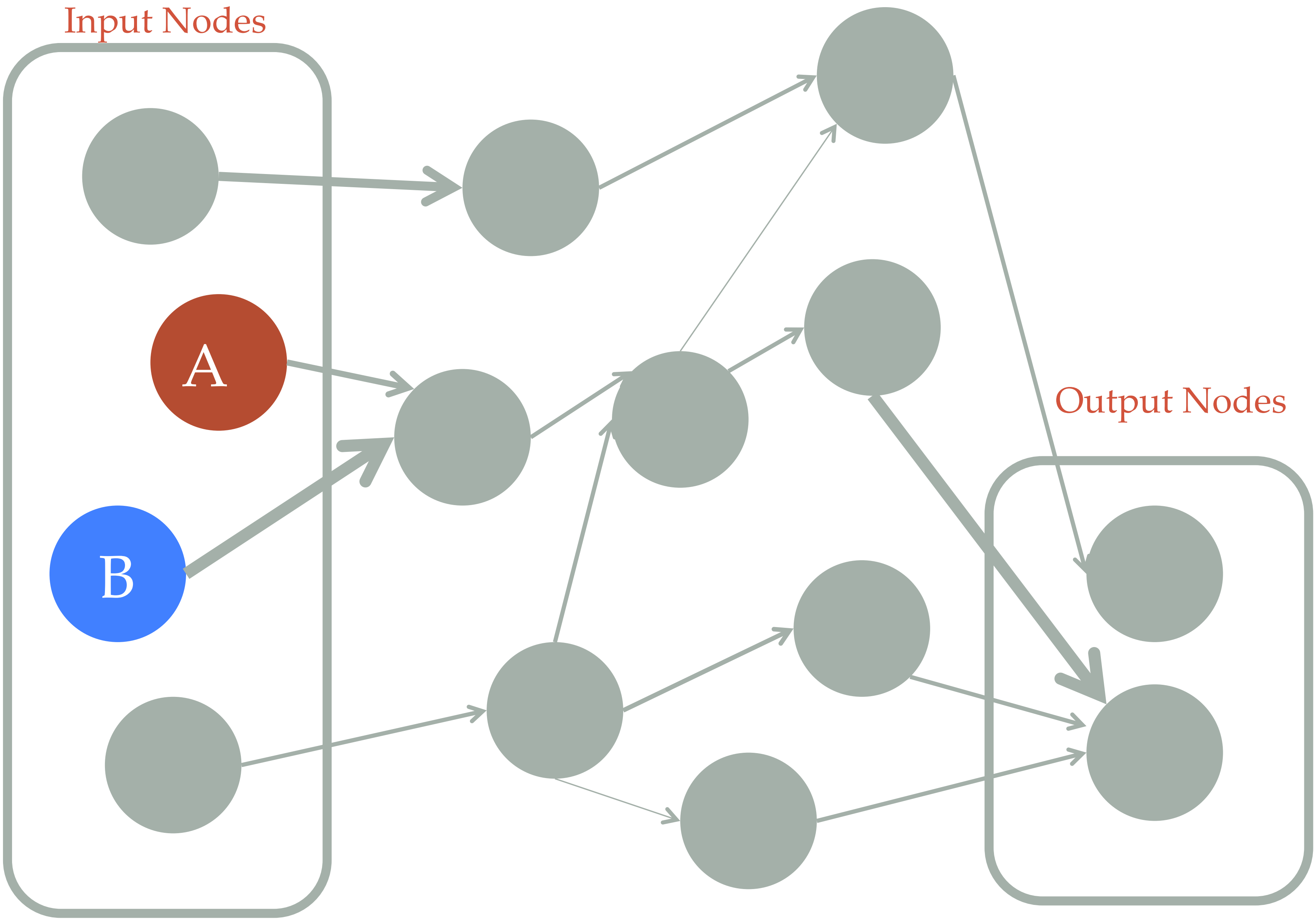

B
E
D
F
C

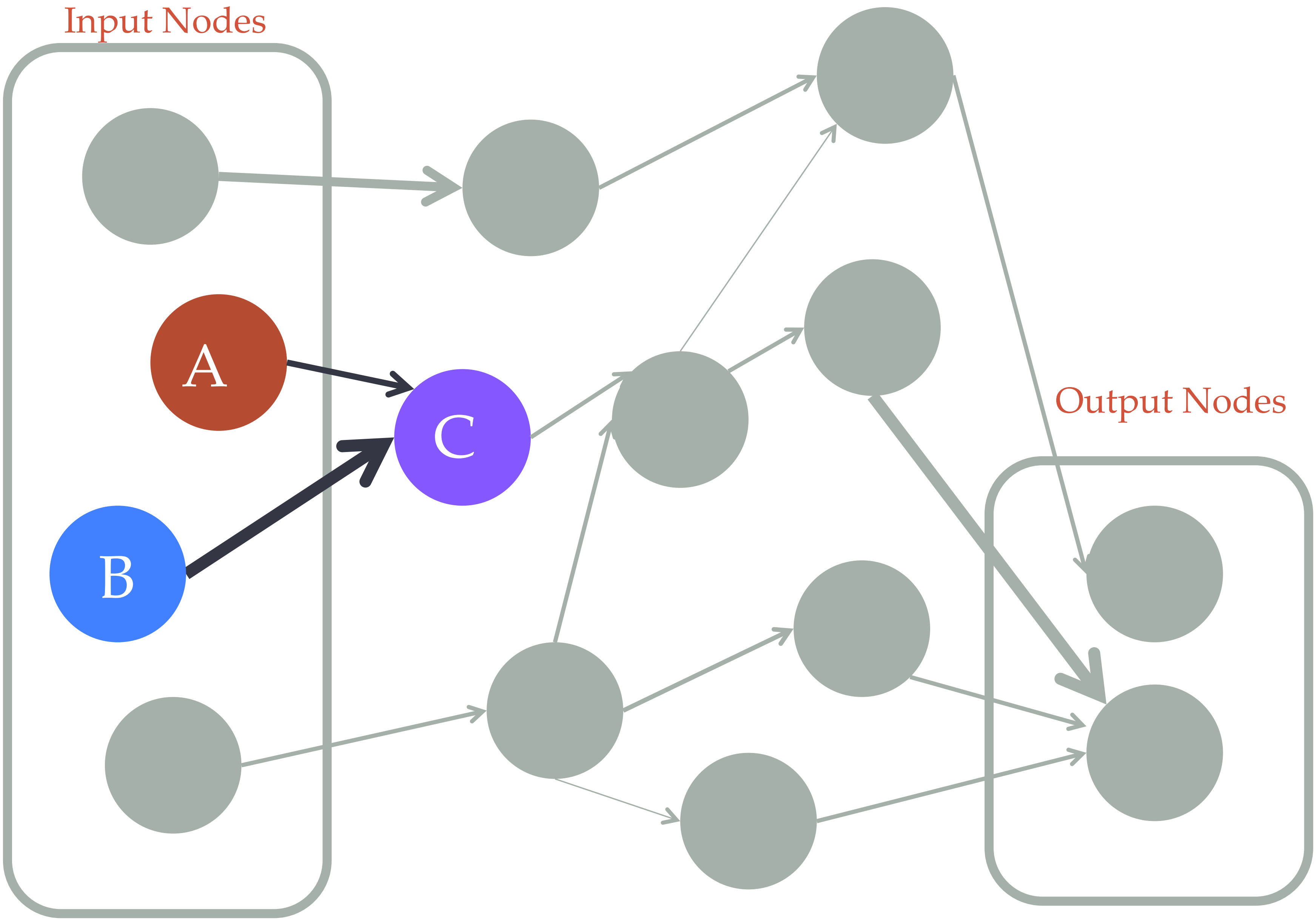

E
F


Input Nodes
Output Nodes


Input Nodes
Output Nodes


Input Nodes
Output Nodes

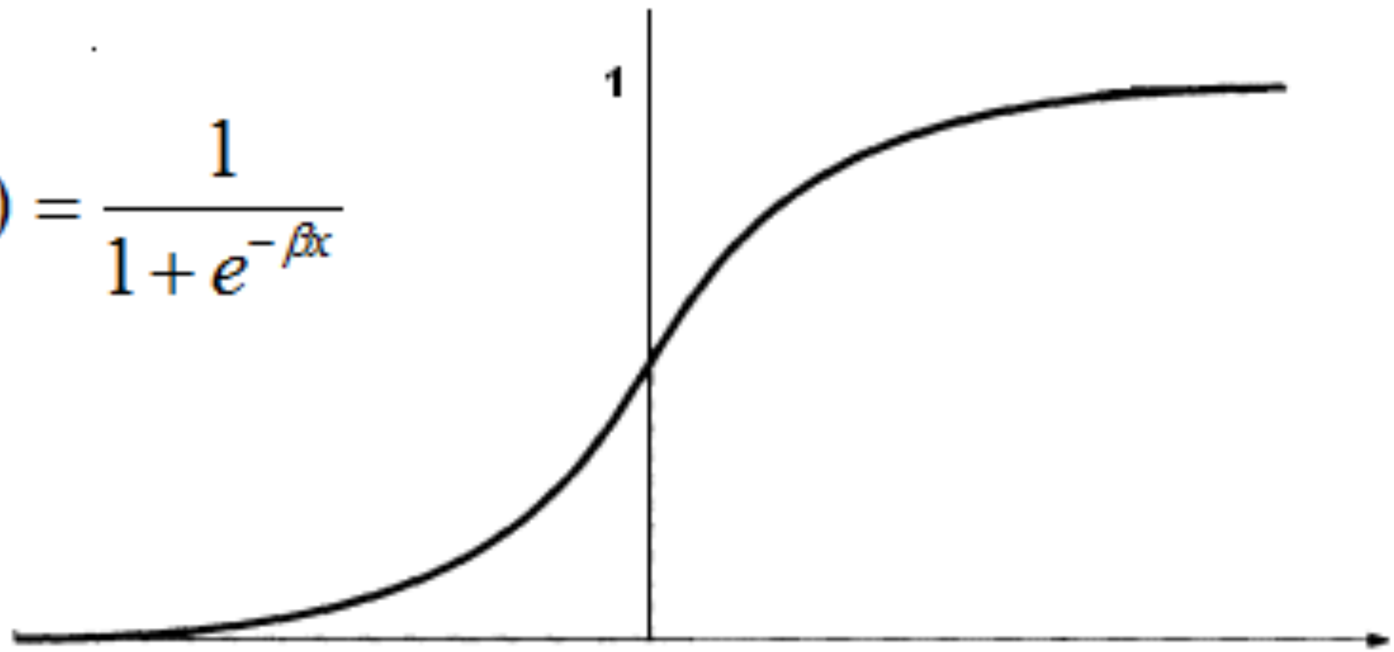

Input Nodes
A
B
Output Nodes



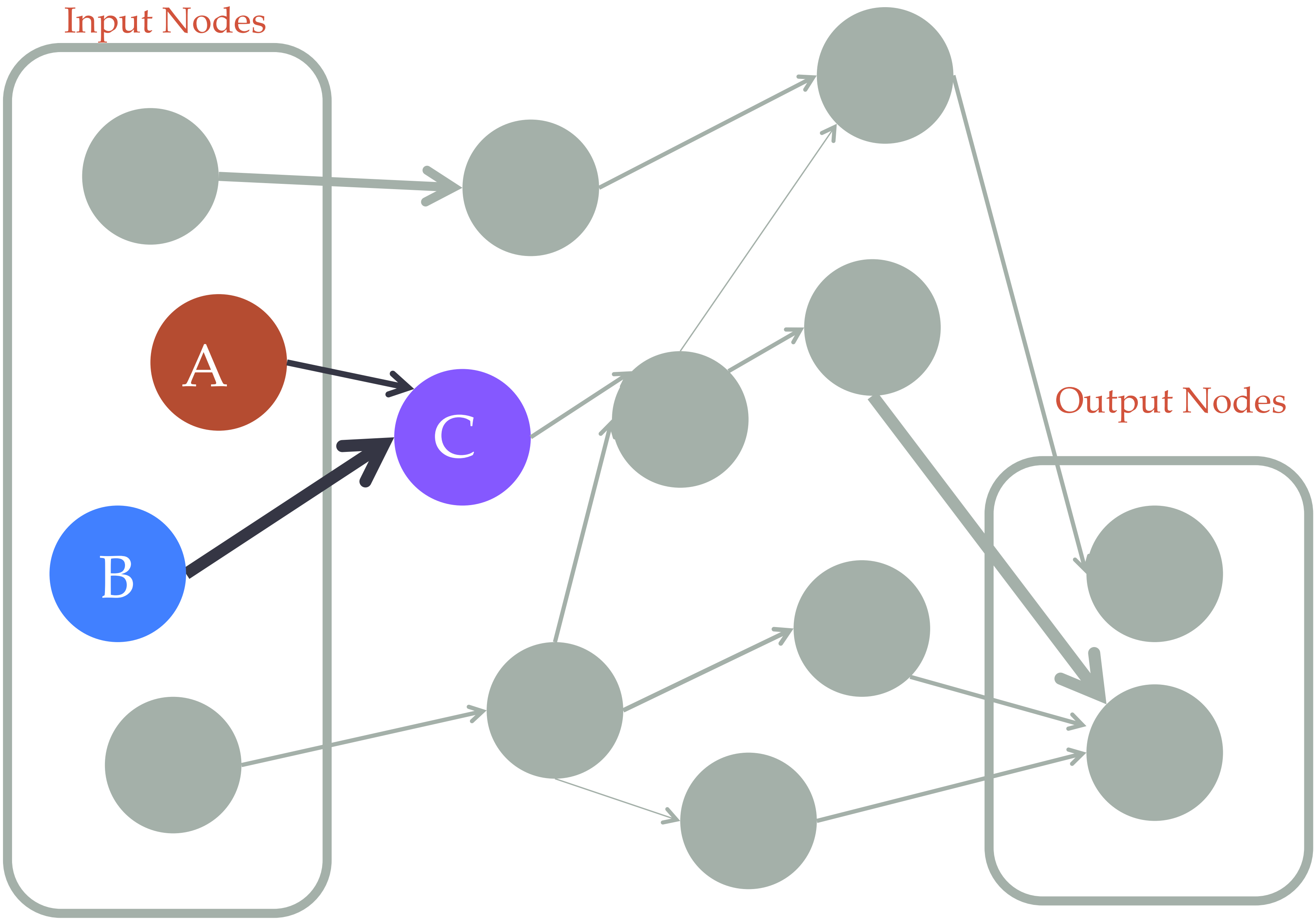
Input Nodes
A
B
C
Output Nodes

Sigmoid

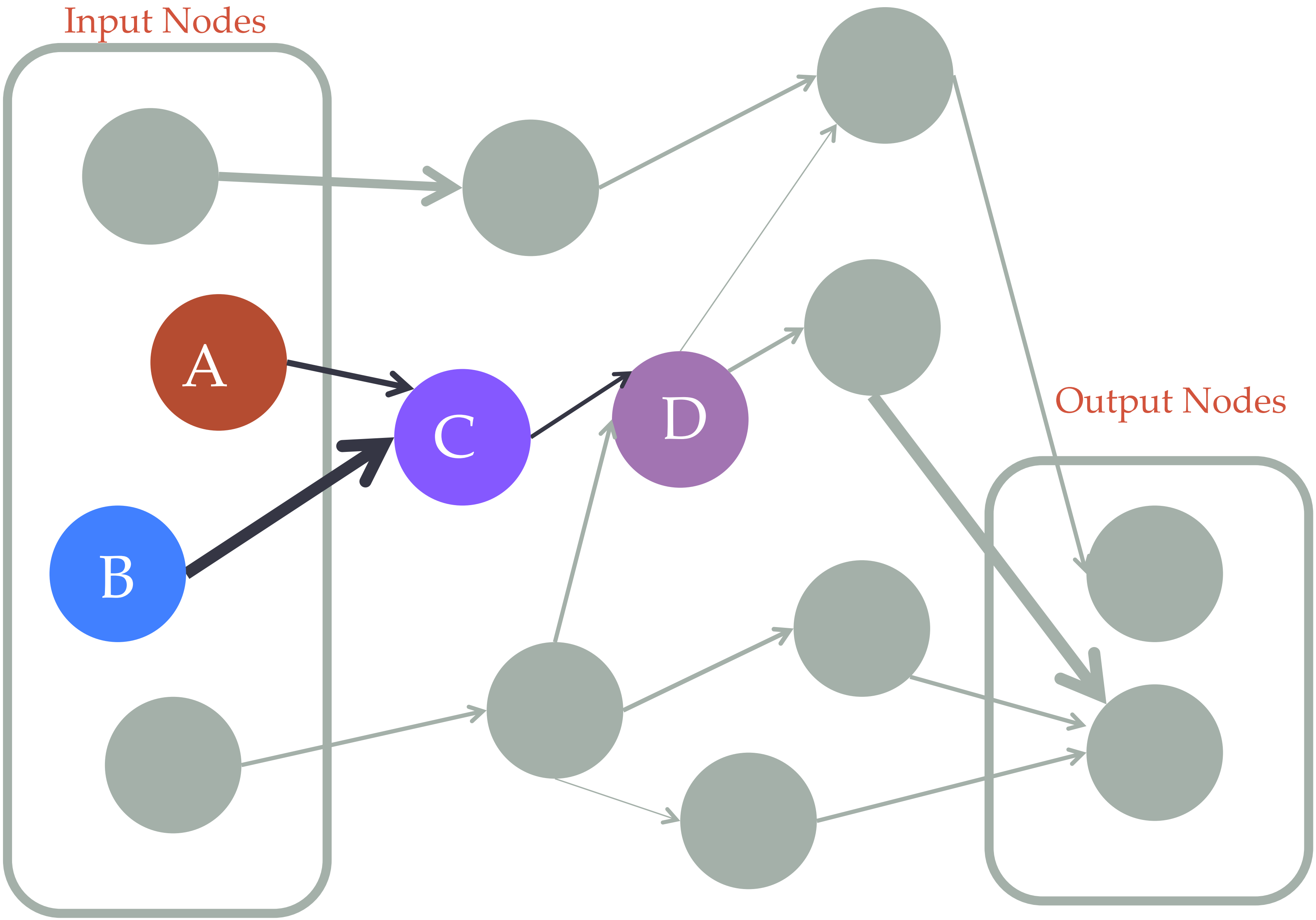
$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

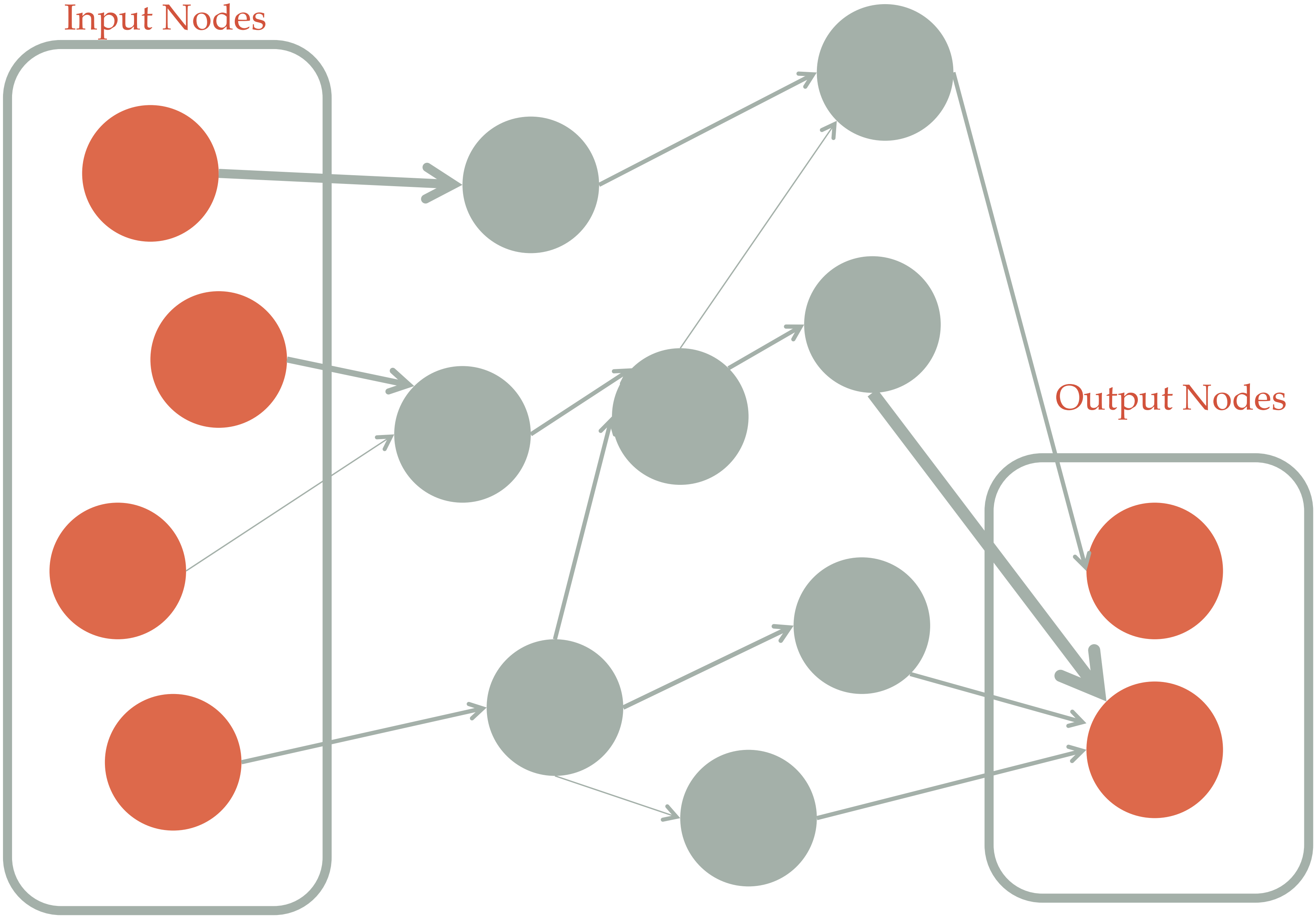


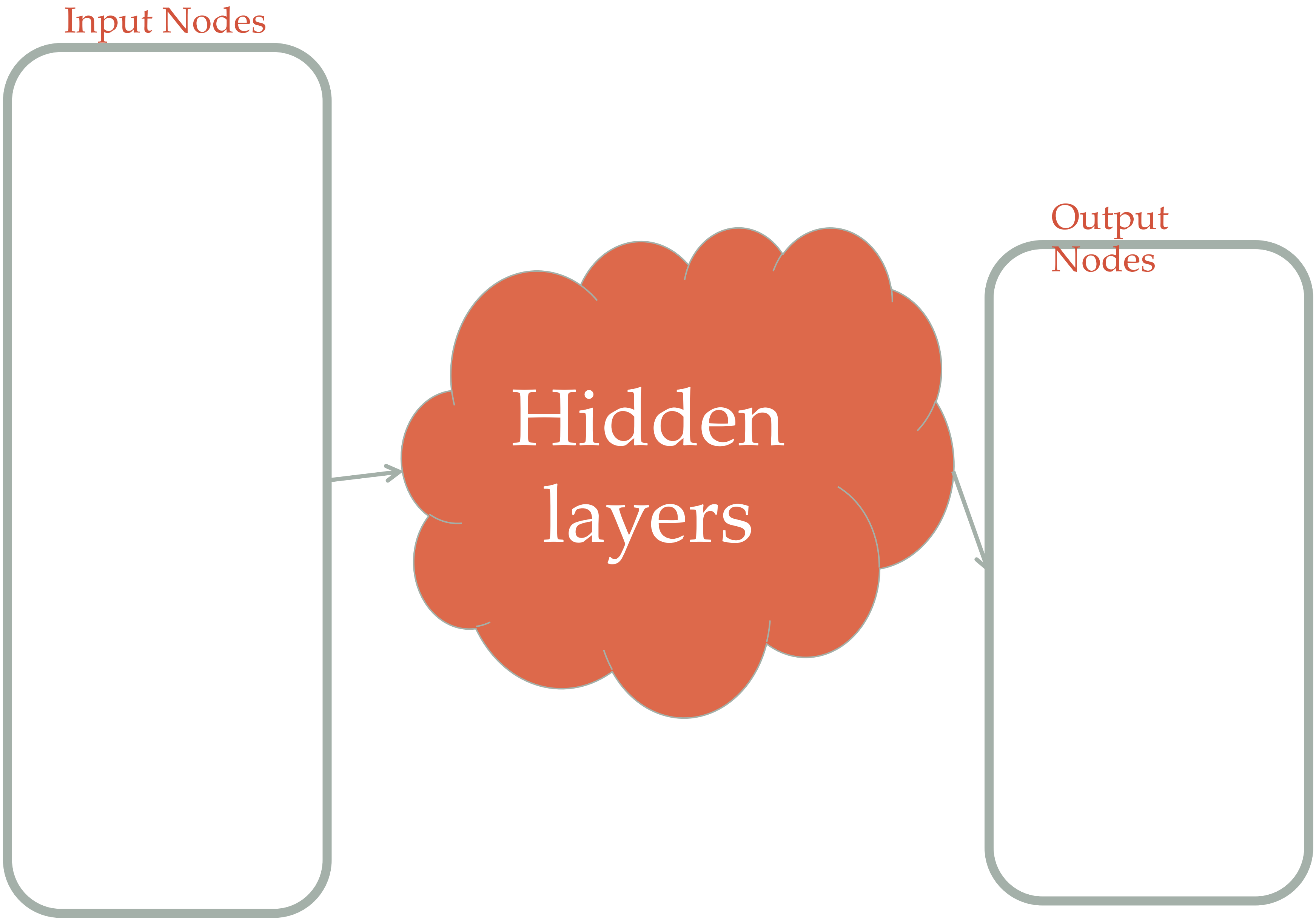
Wikipedia

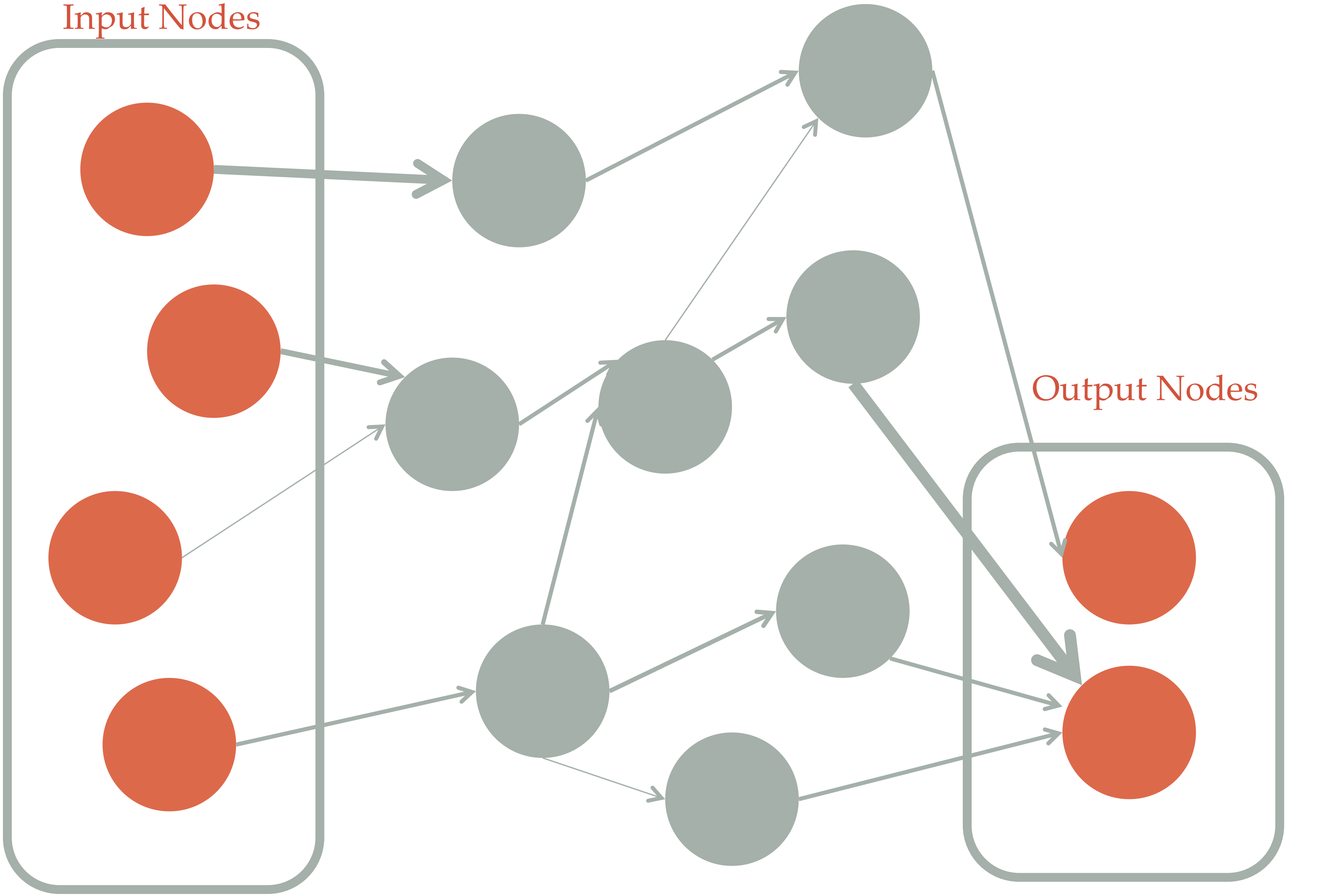


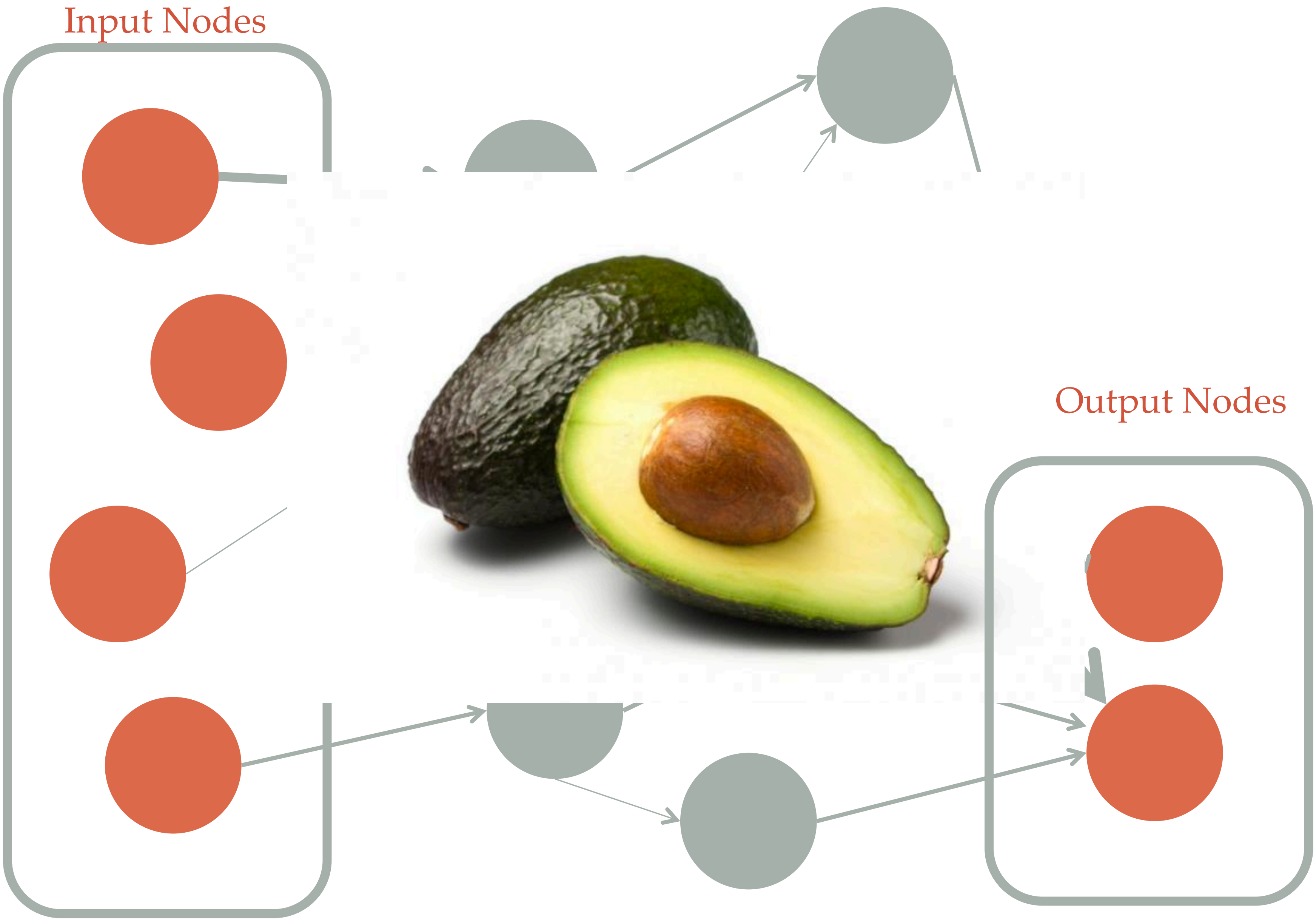
Input Nodes
A
B
C
Output Nodes

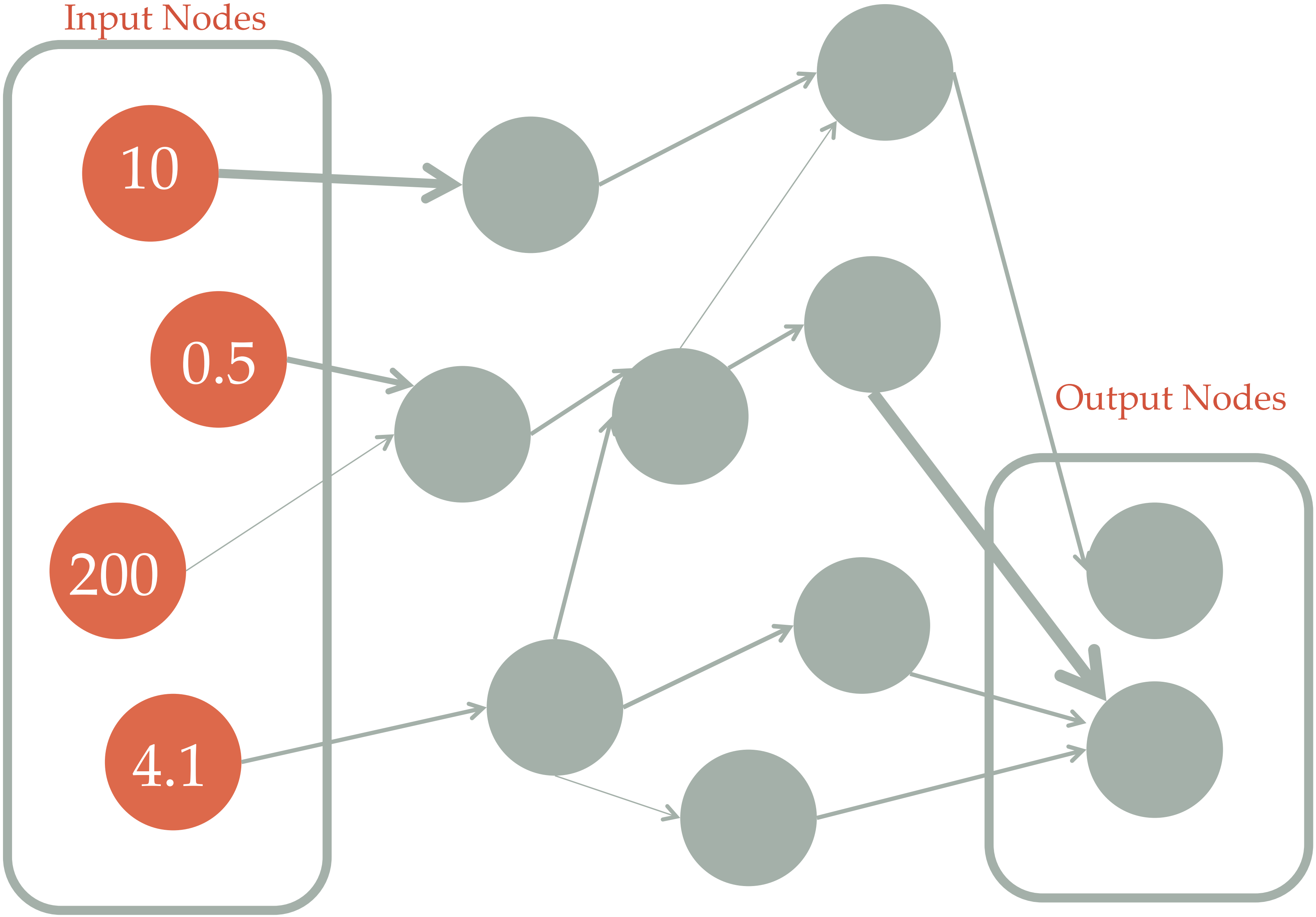

Input Nodes
A
B
C
D
Output Nodes


Input Nodes
Output Nodes

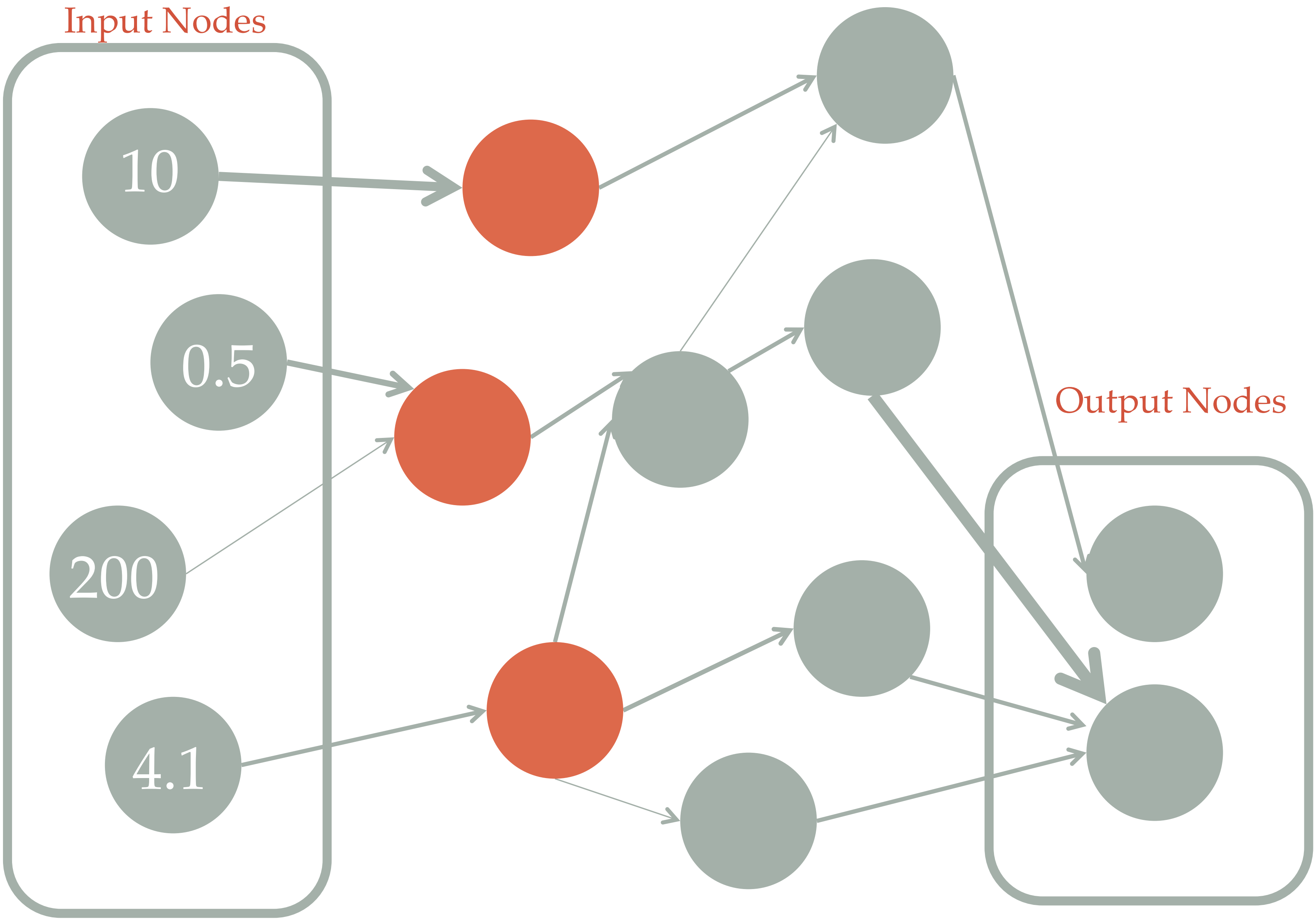

Input Nodes
Hidden layers
Output Nodes

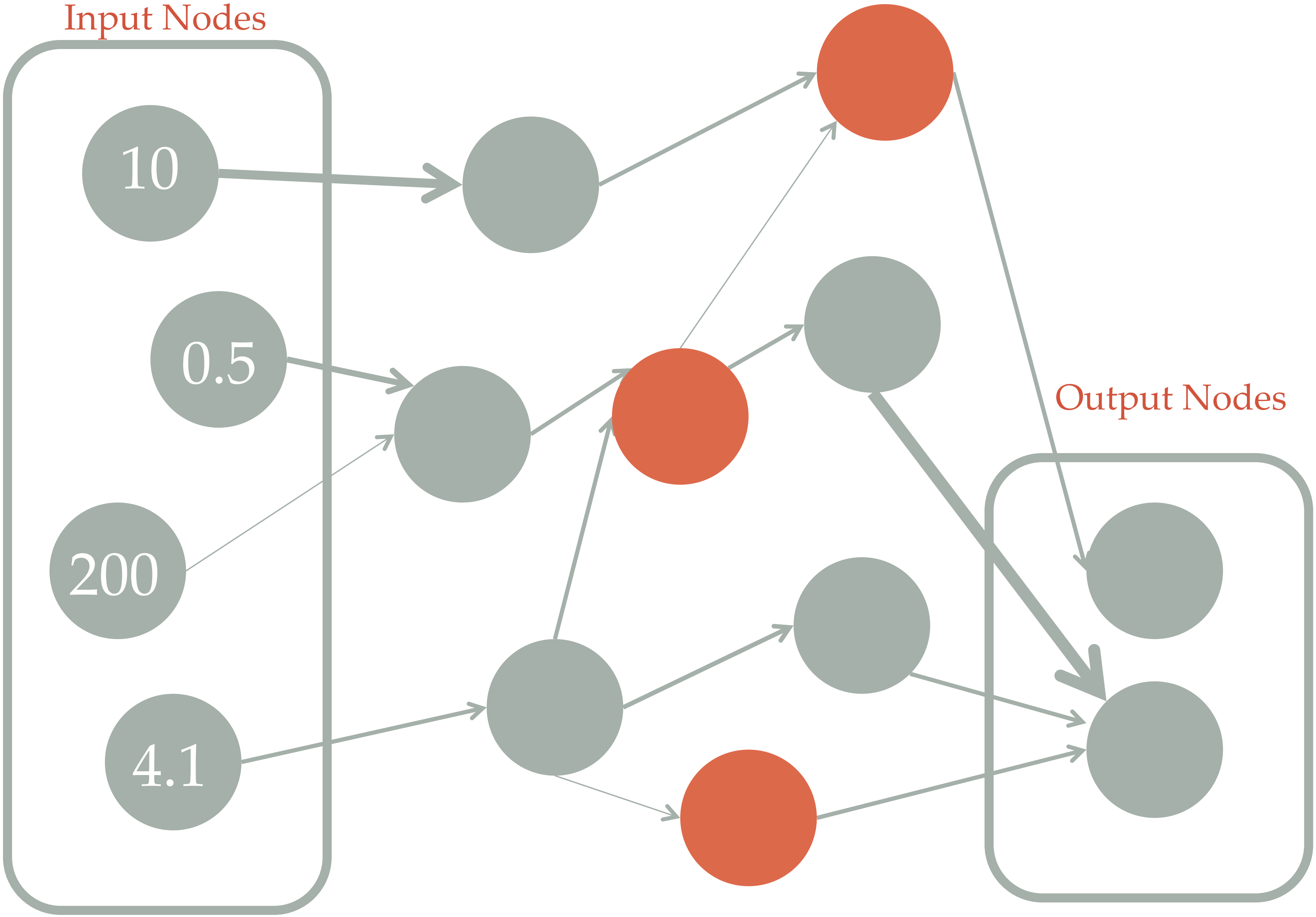

Input Nodes
Output Nodes

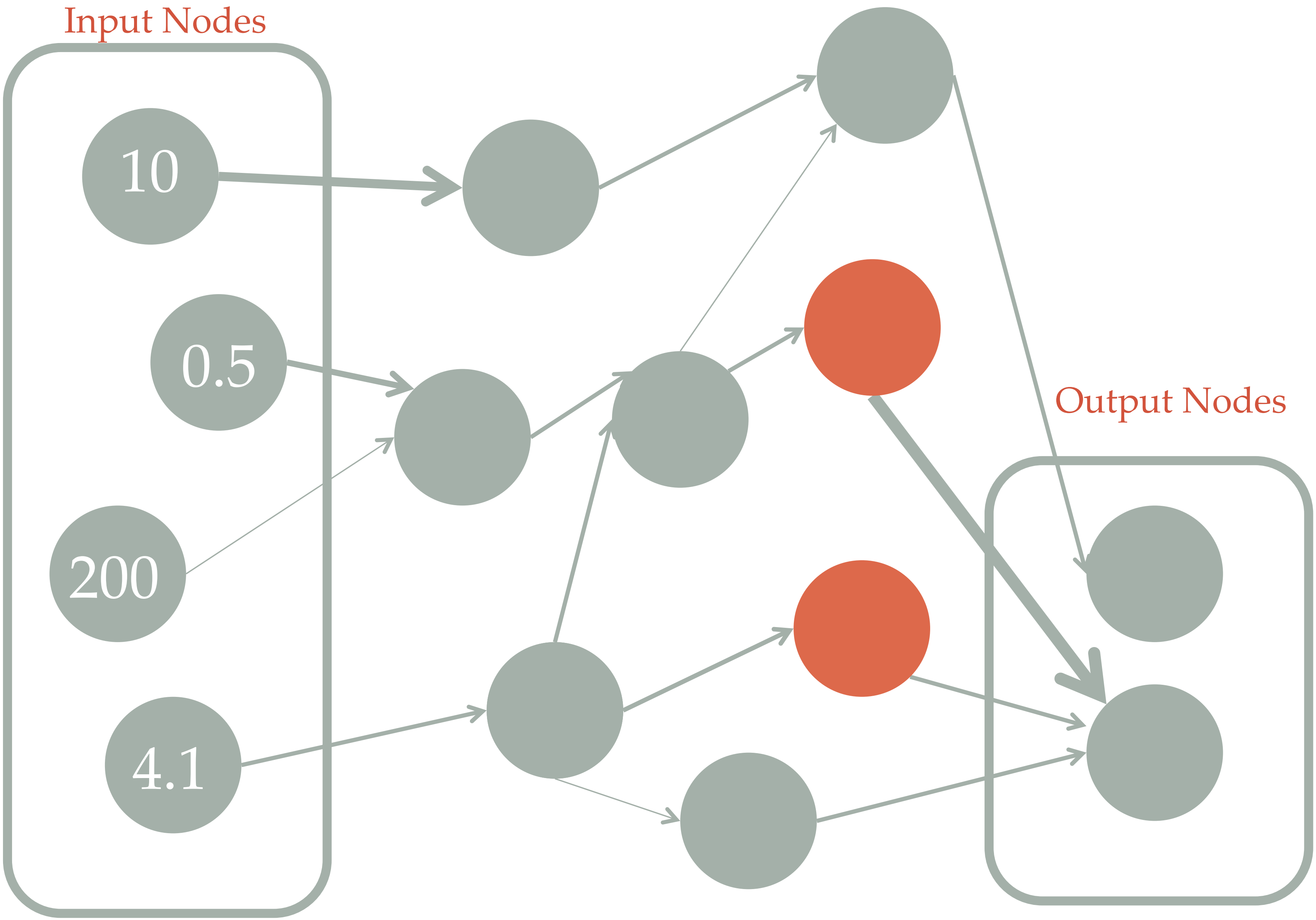

Input Nodes
Output Nodes



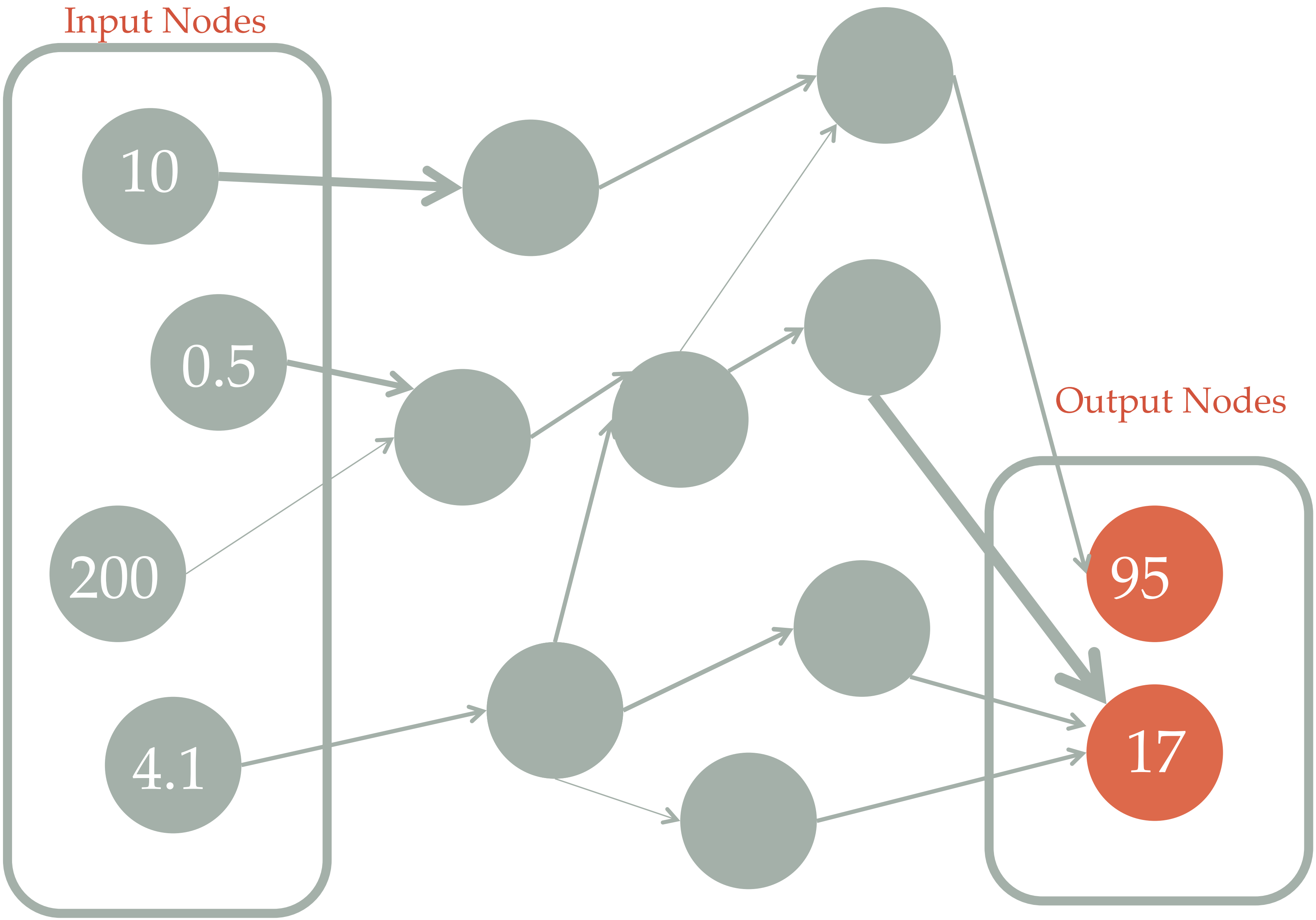
Input Nodes
10
0.5
200
4.1
Output Nodes


Input Nodes
10
0.5
200
4.1
Output Nodes

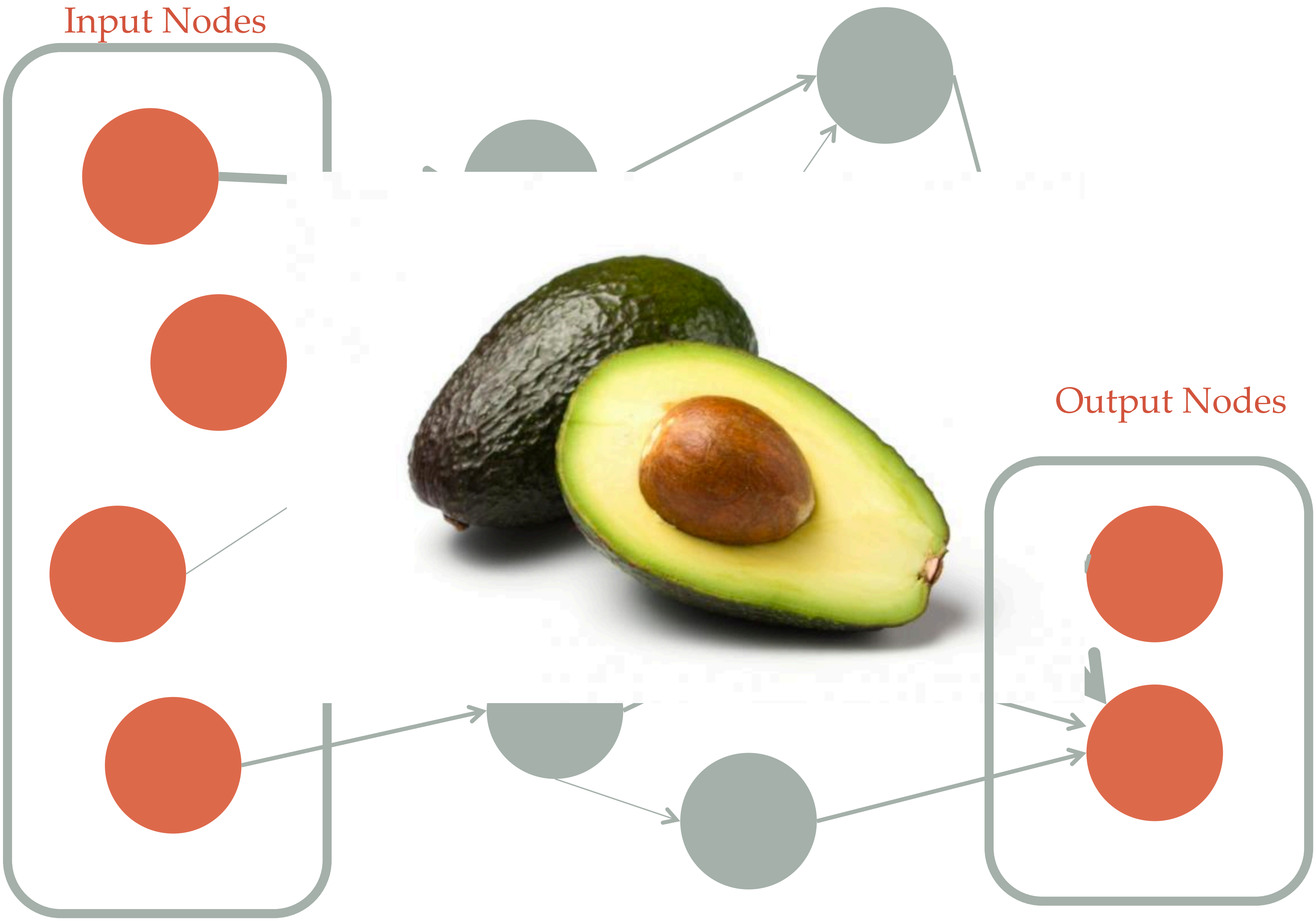

Input Nodes
10
0.5
200
4.1
Output Nodes

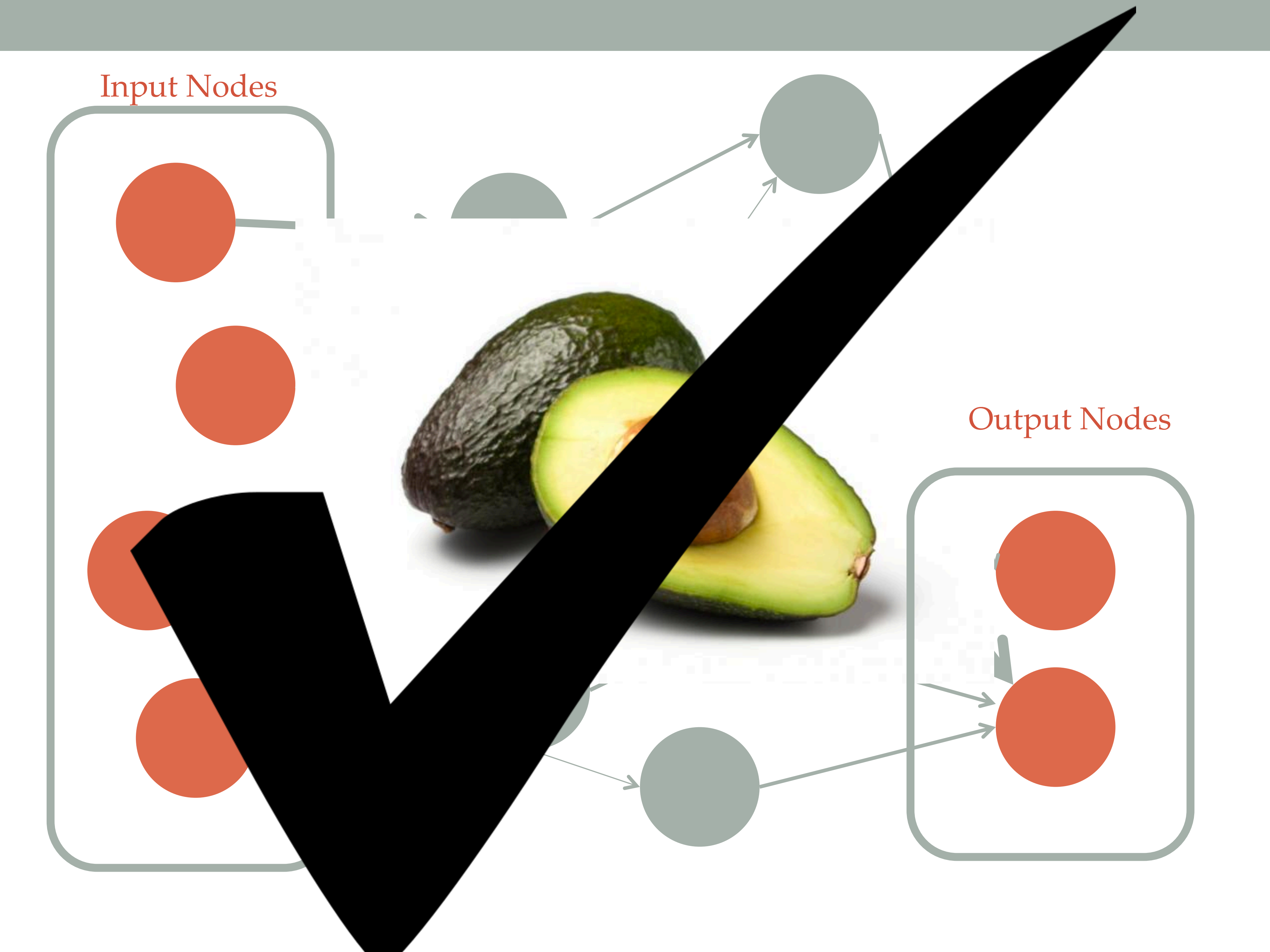


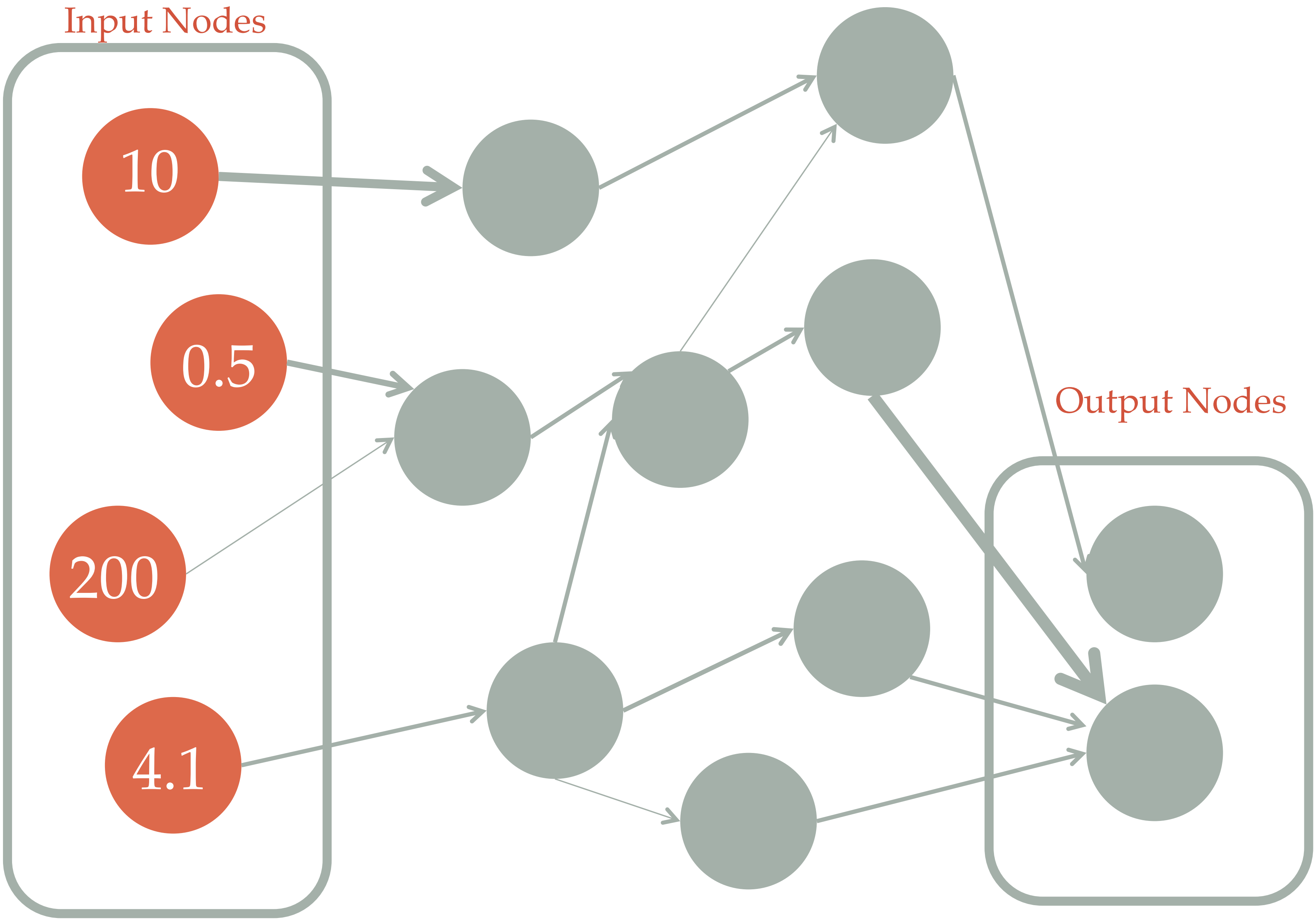
Input Nodes
10
0.5
200
4.1
Output Nodes


Input Nodes
10
0.5
200
4.1
Output Nodes
95
17

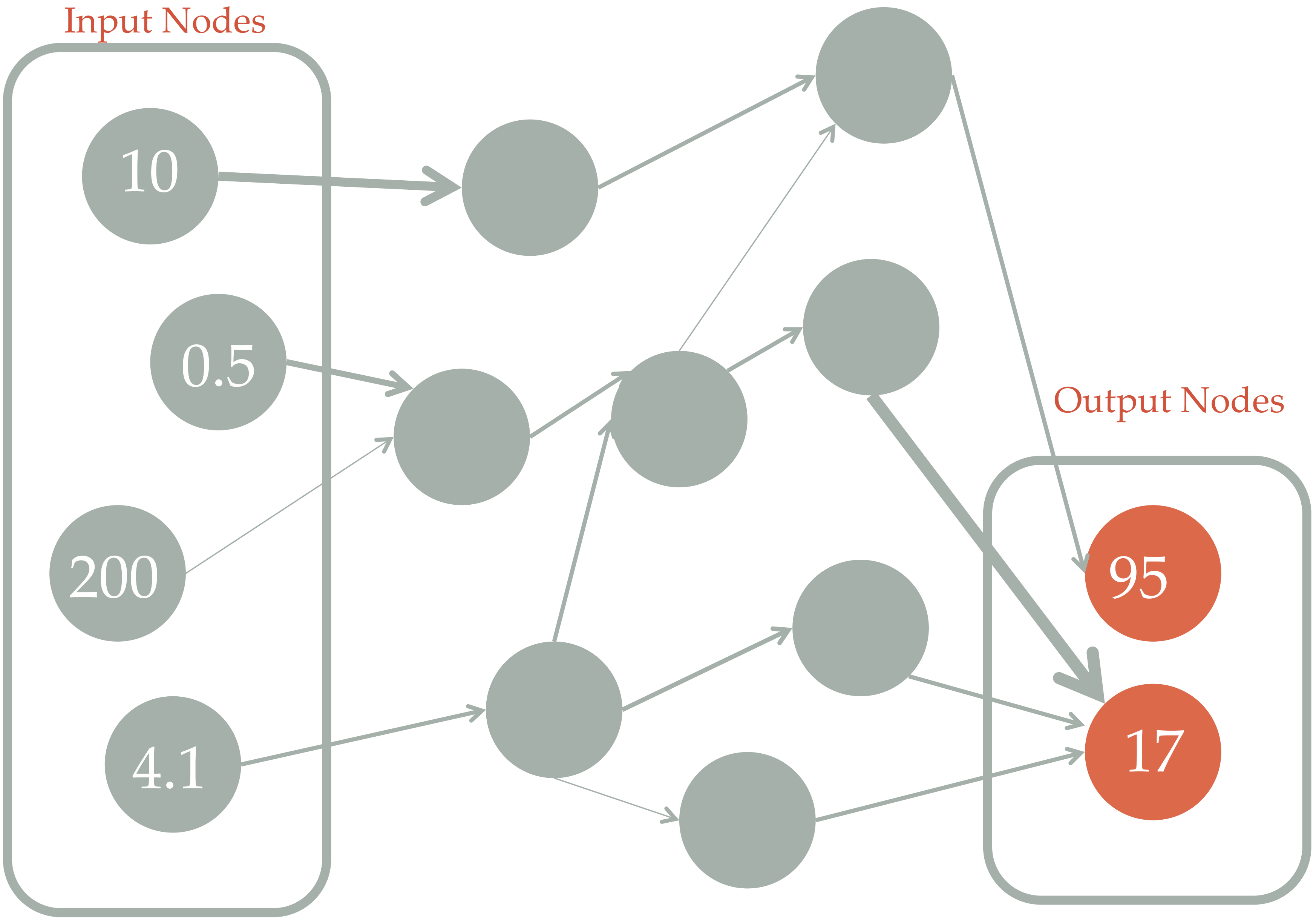
Forward propagation


Input Nodes
Output Nodes

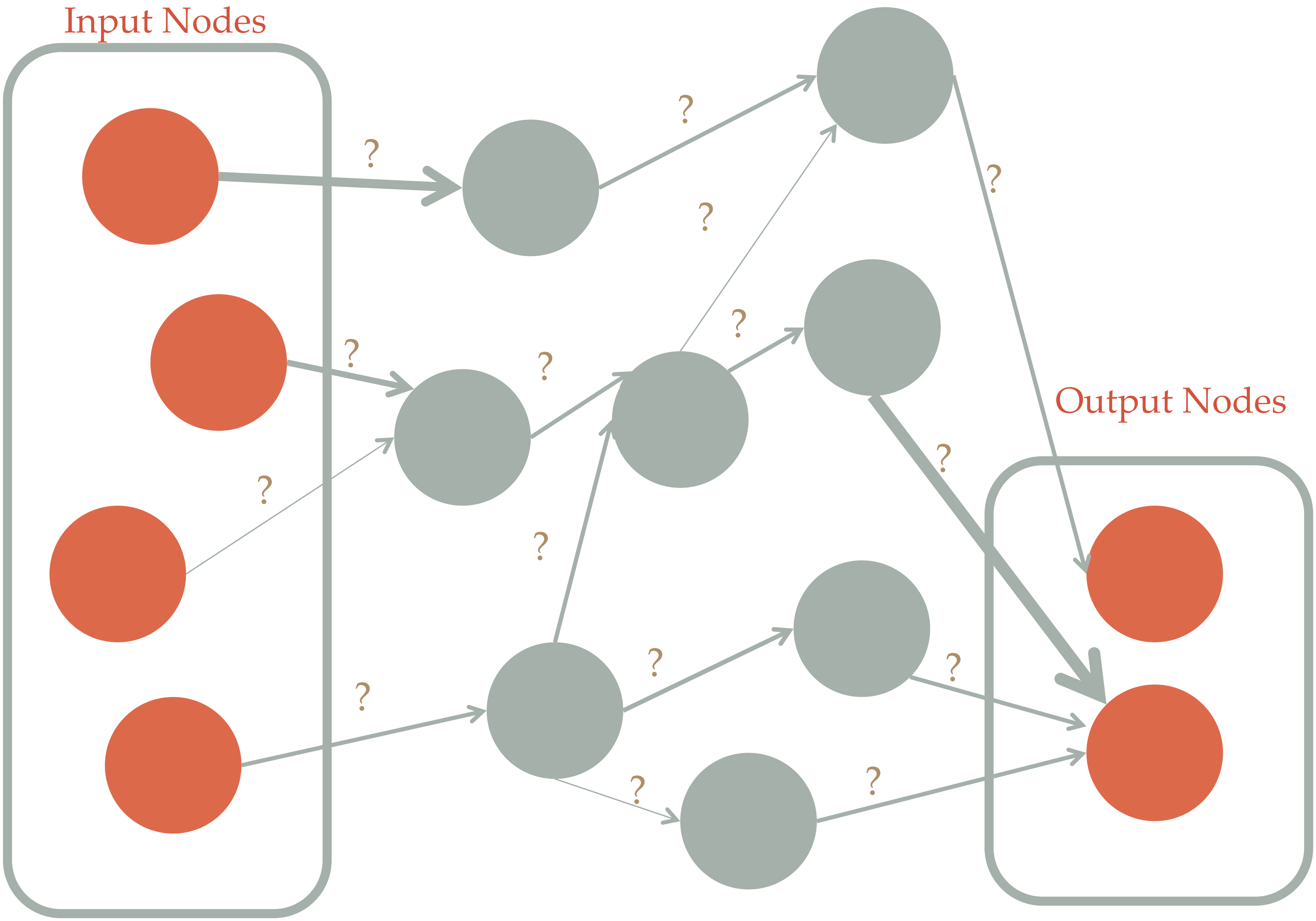

Input Nodes
Output Nodes



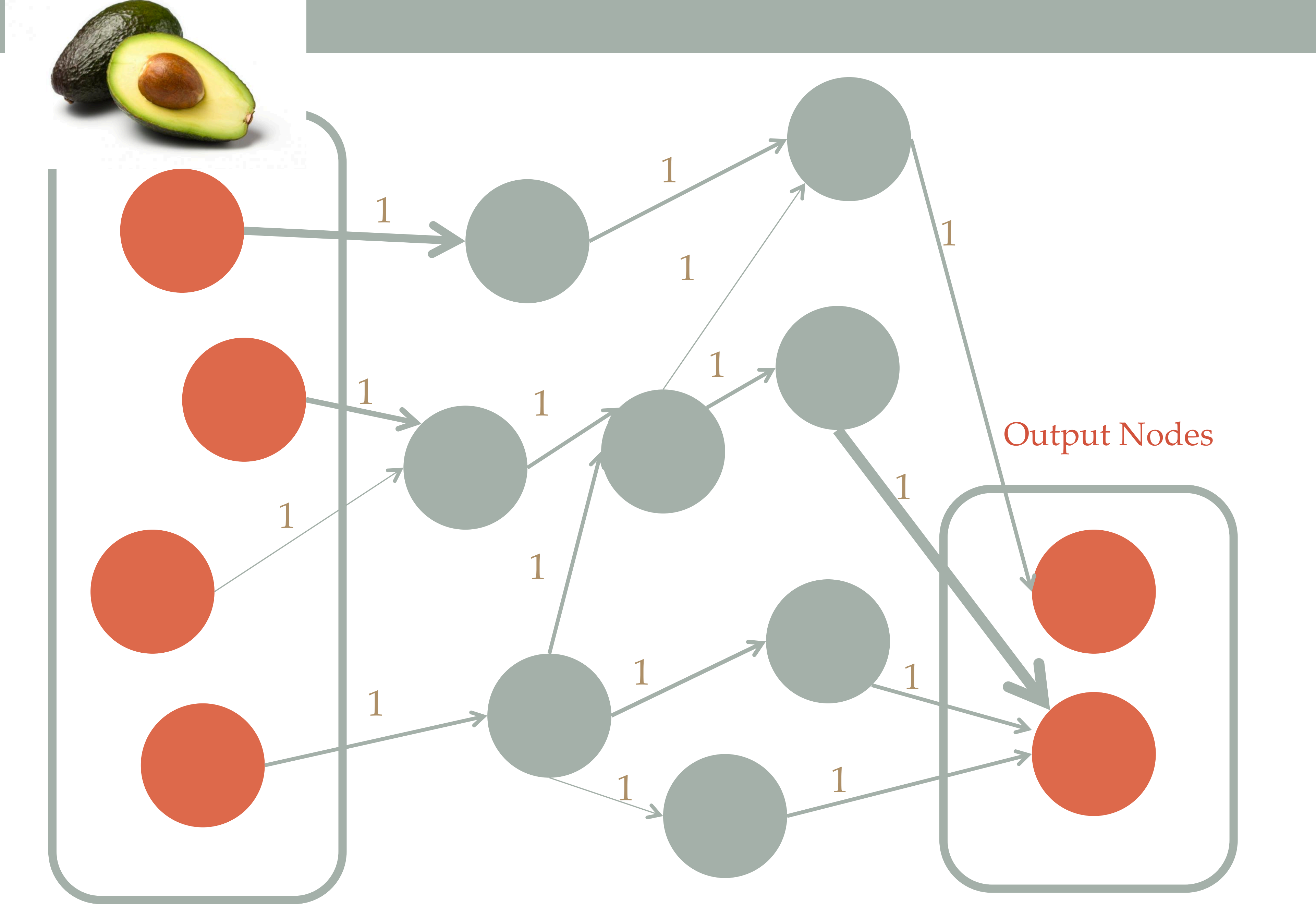
Input Nodes
10
0.5
200
4.1
Output Nodes


Input Nodes
10
0.5
200
4.1
Output Nodes
95
17

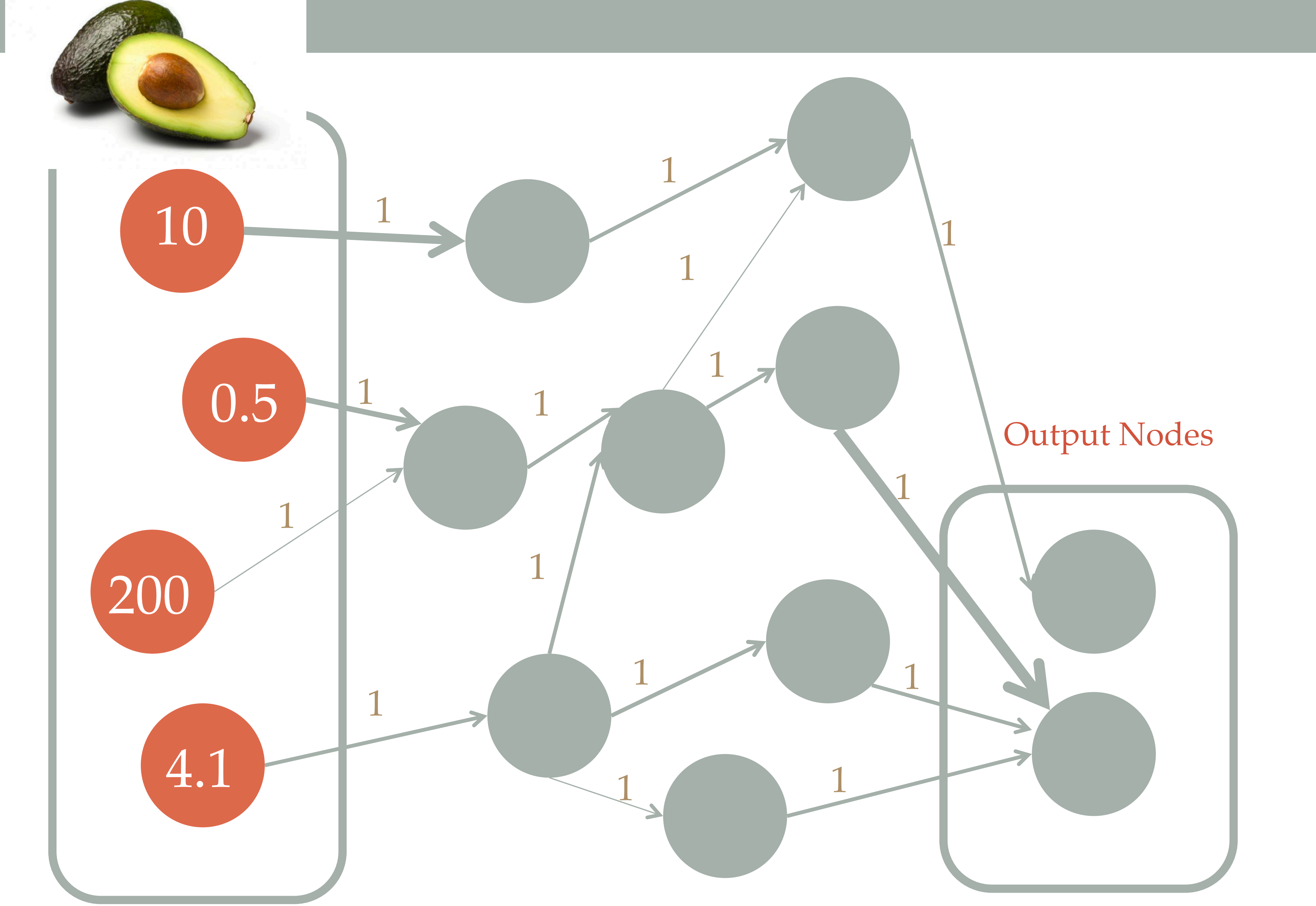
No randomness!

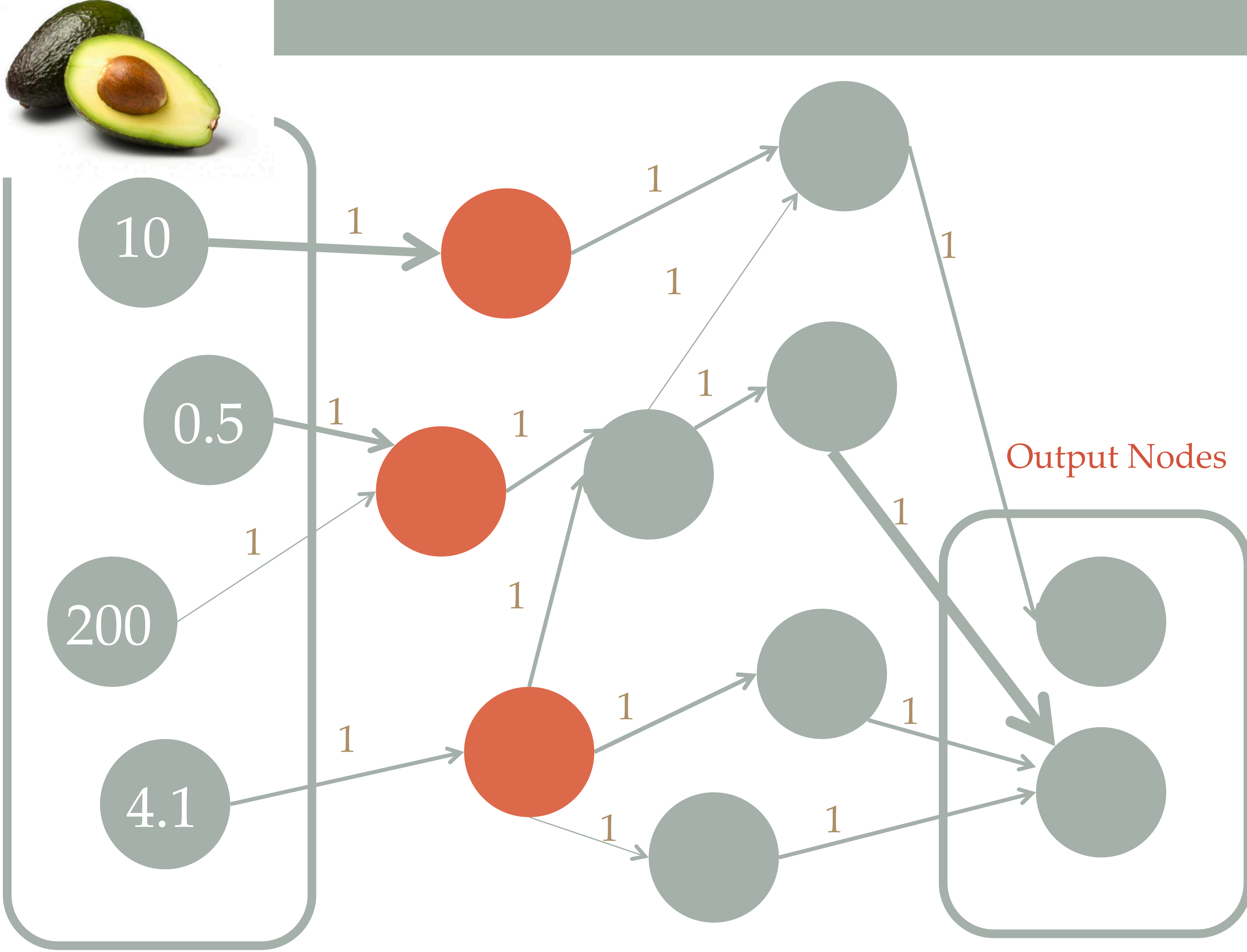

Input Nodes
Output Nodes
?
?
?
?
?
?
?
?
?
?
?
?
?
?
?

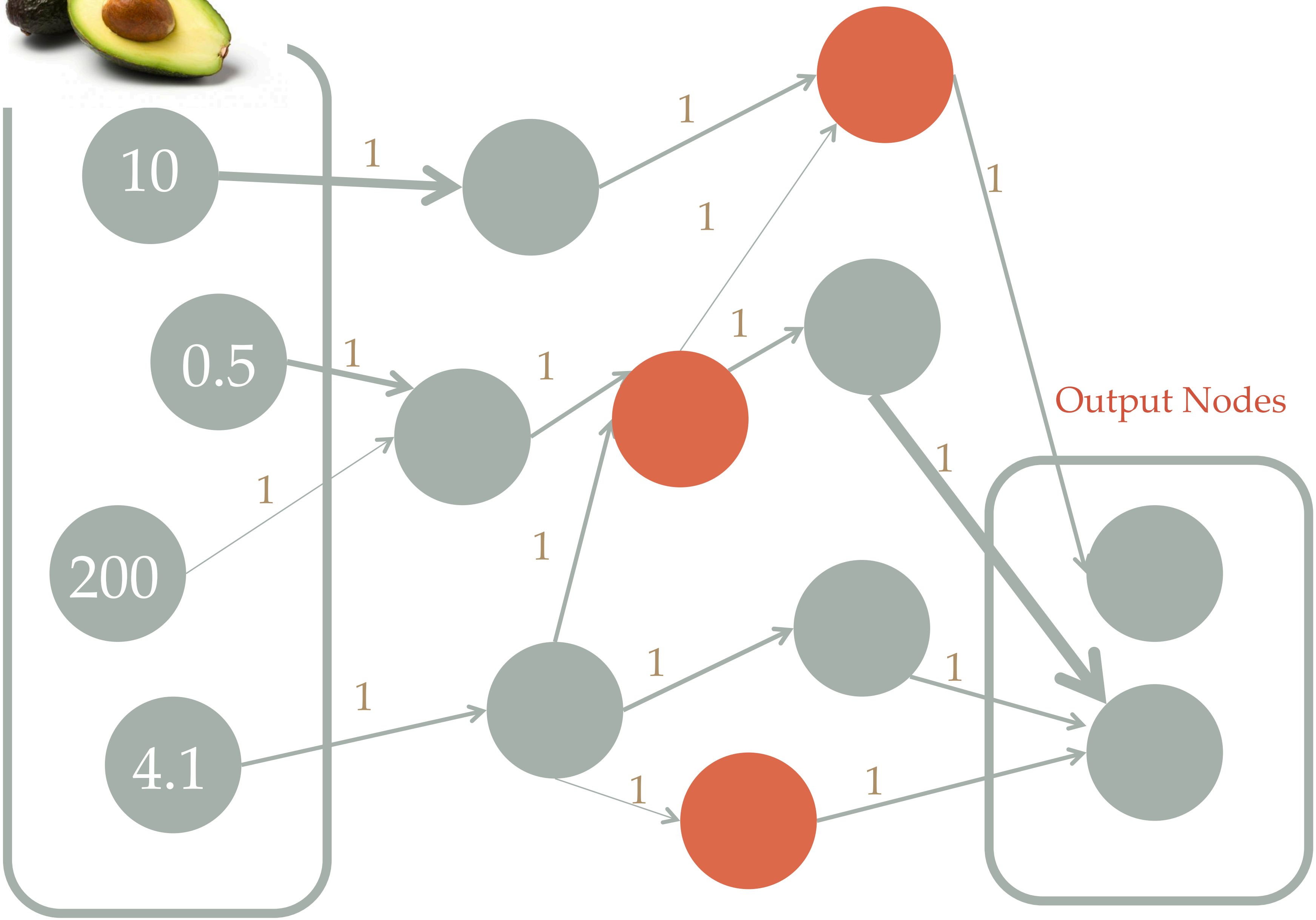
Backpropagation



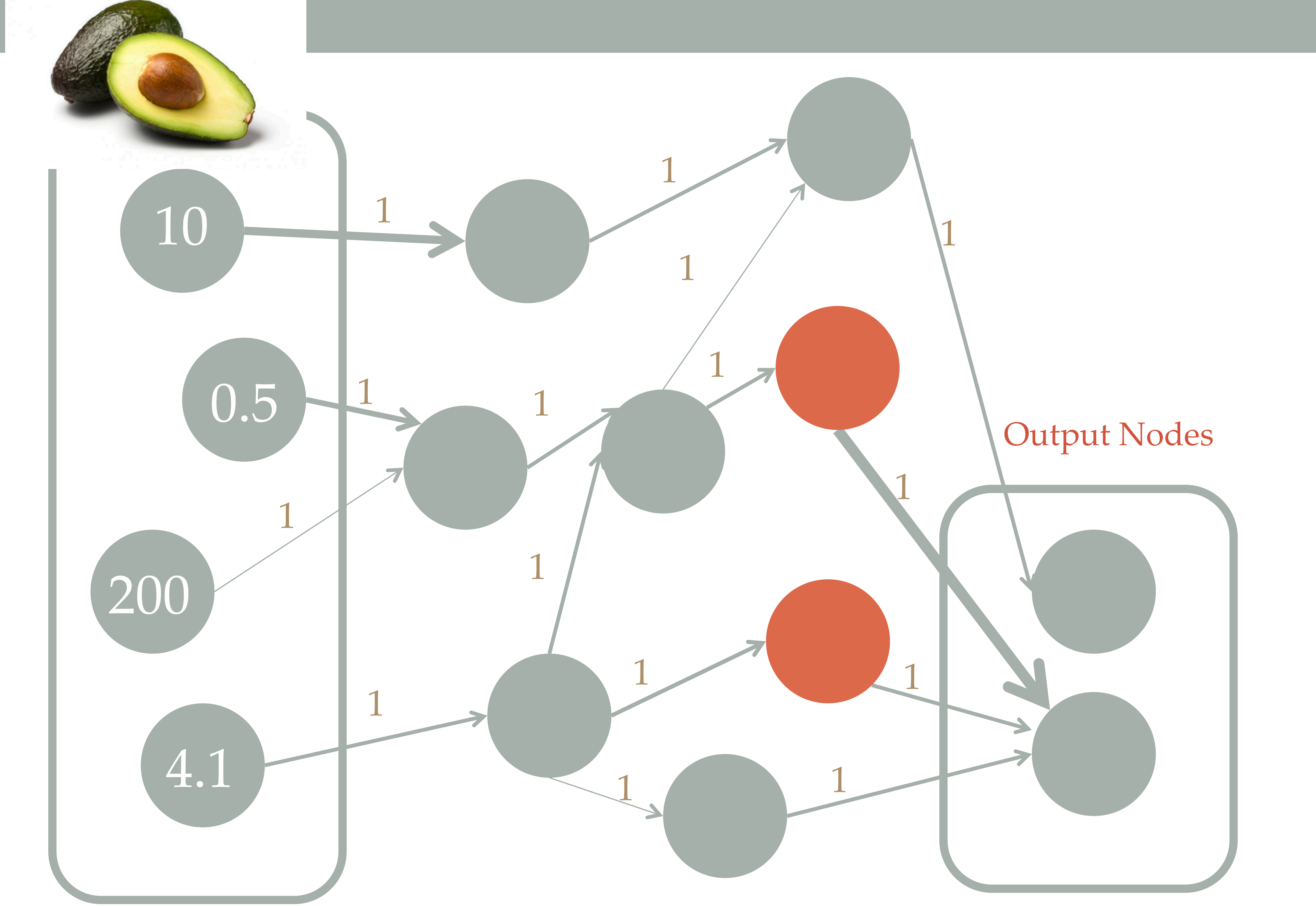
Output Nodes
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1


10
0.5
200
4.1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
Output Nodes

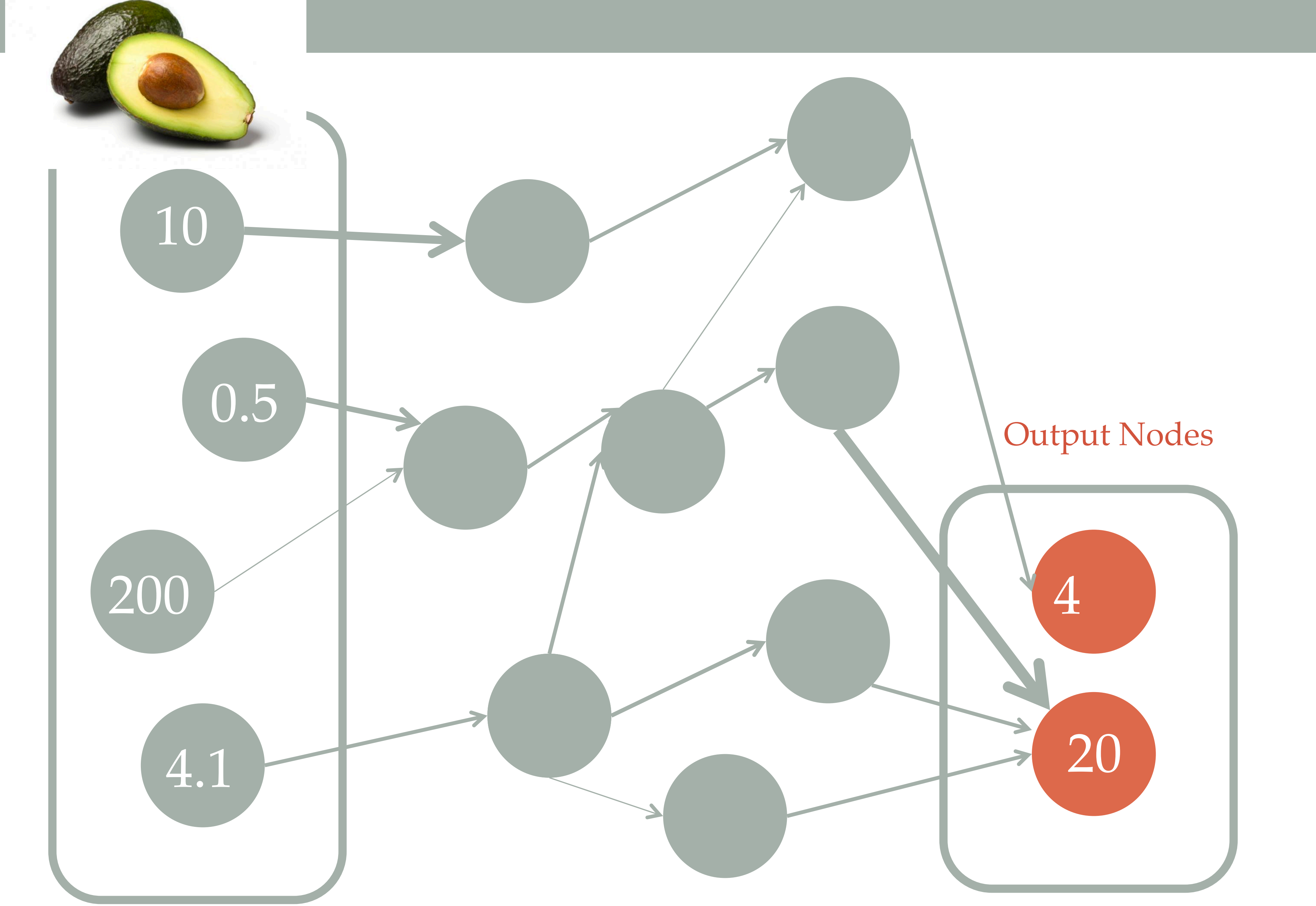

10
0.5
200
4.1
Output Nodes

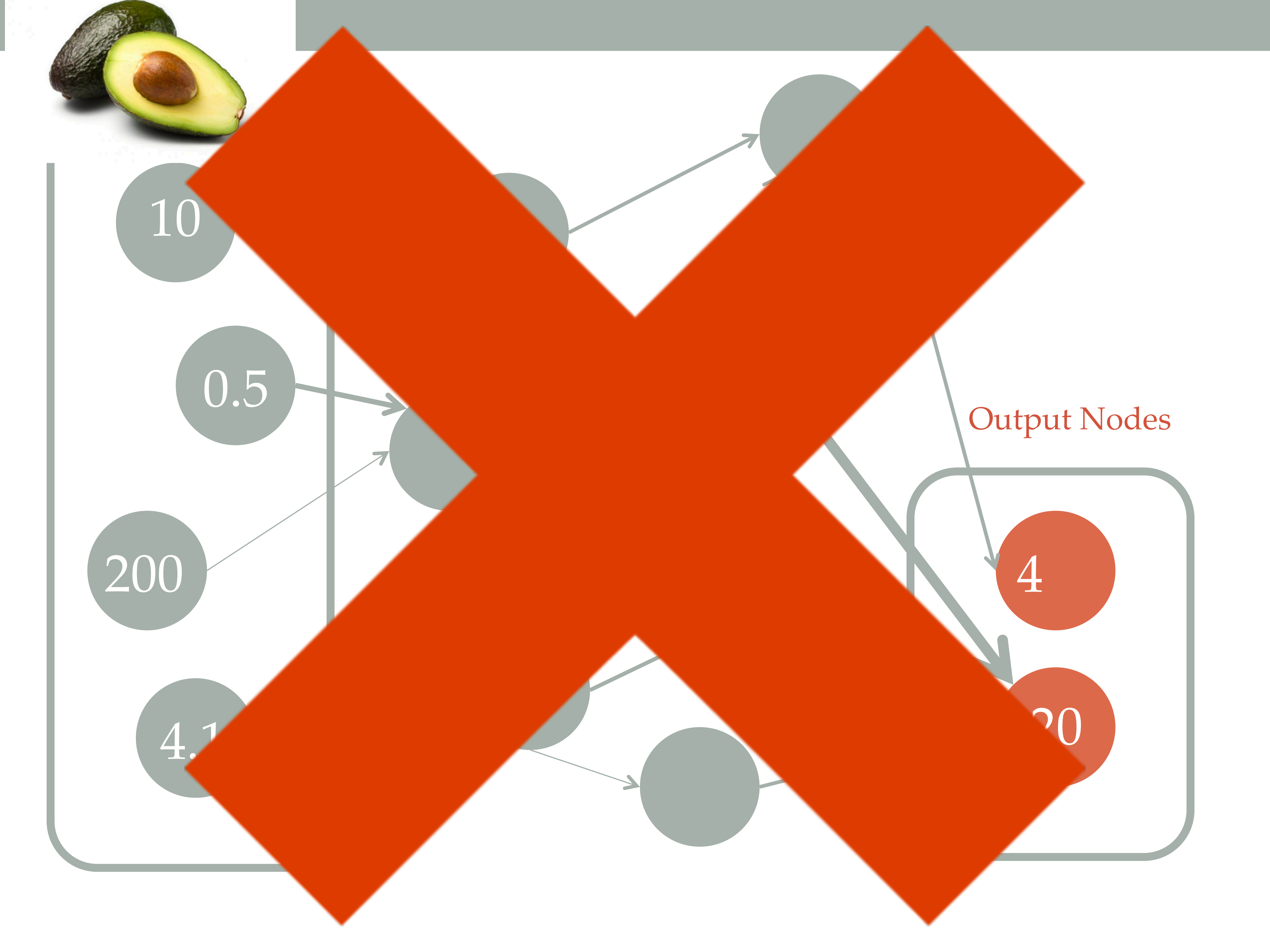


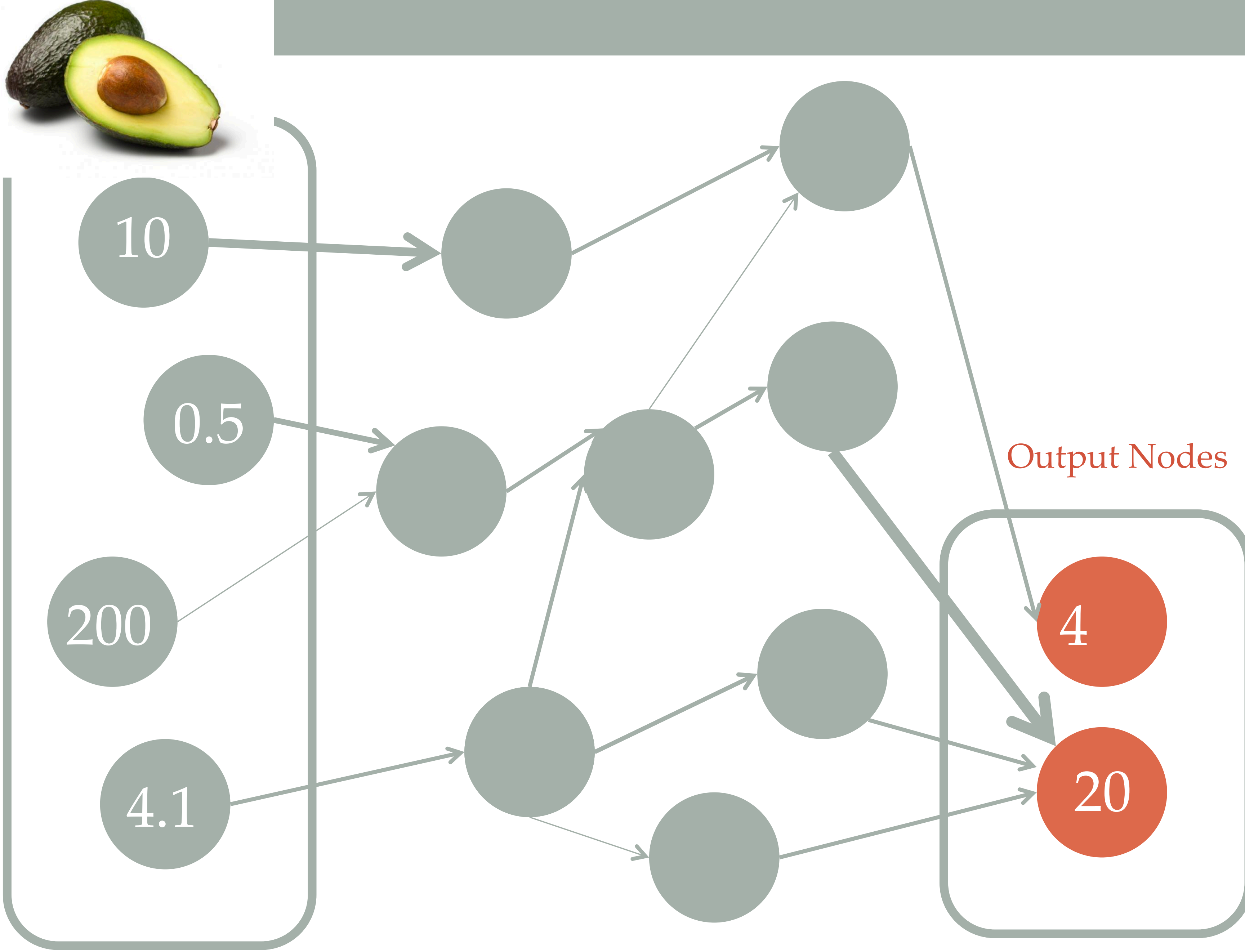
1
1
1
10
1
1
1
0.5
1
1
200
1
Output Nodes
1
1
1
4.1
1
1

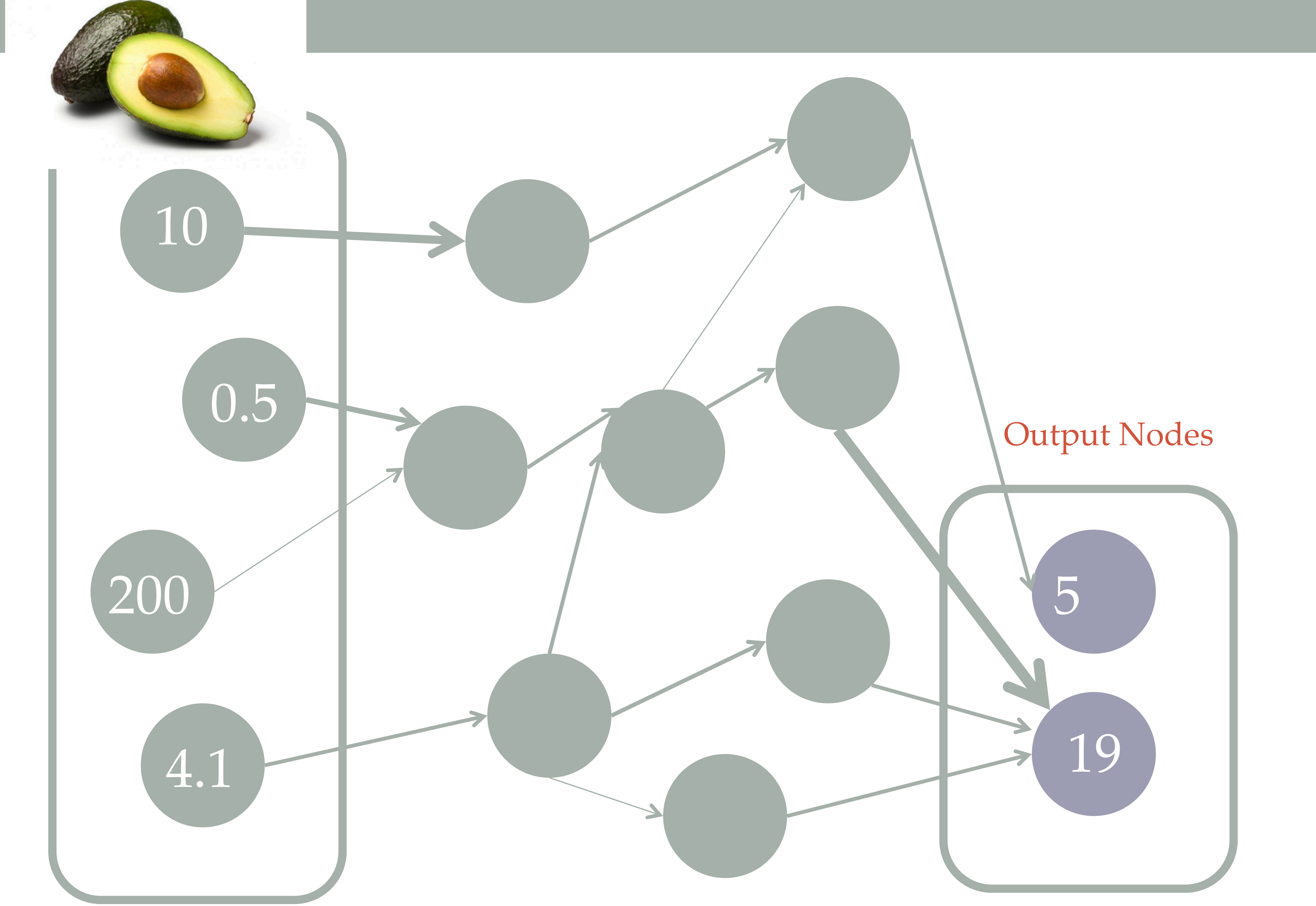


10
0.5
200
4.1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
Output Nodes


10
0.5
200
4.1
Output Nodes
4
20


10
0.5
200
4.1
Output Nodes
4
20


10
0.5
200
4.1
Output Nodes
4
20


10
0.5
200
4.1
Output Nodes
5
19

1. $\frac{\partial E}{\partial w_{ij}} = \delta_j * x_i$ for all weights and biases

2. $o_j' = o_j * (1 - o_j)$ for output layer nodes using softmax

3. $\varphi_j' = (1 - \varphi_j) * (1 + \varphi_j)$ for hidden layer nodes using tanh

4. $\varphi_j' = \varphi_j * (1 - \varphi_j)$ for hidden layer nodes using logistic sigmoid

5. $e_j = \left(o_j - t_j\right)$ for hidden and output layer nodes

6. $\delta_j = e_j * o_j{}'$ if *j* is an output node

7. $\delta_j = \left(\sum \delta_j w_j\right) * \varphi_j{}'$ if *j* is a hidden node

8. $\Delta w_{ij} = \alpha * \frac{\partial E}{\partial w_{ij}}$ delta for all weights and biases

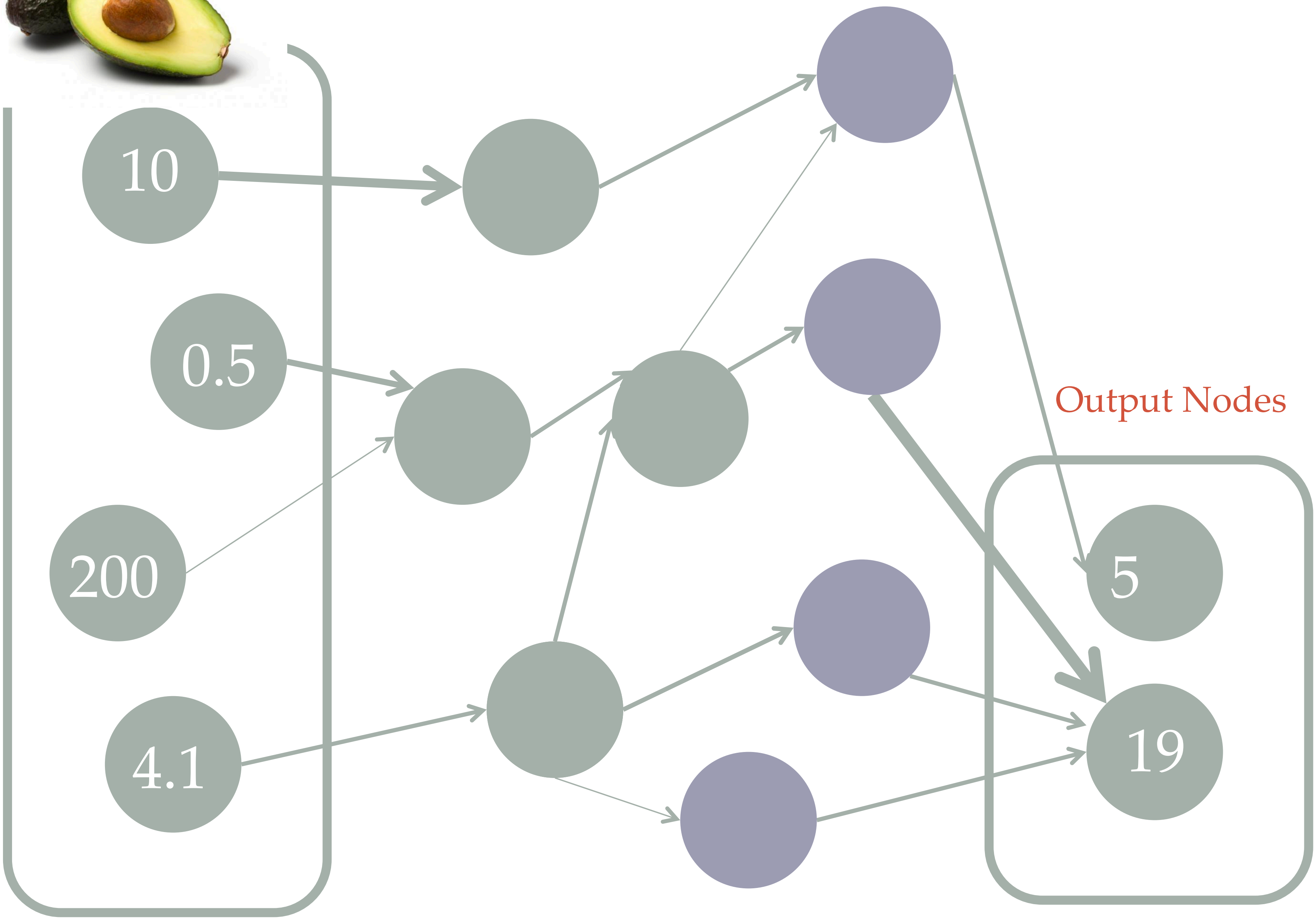
9. $w_{ij}{}' = w_{ij} + \Delta w_{ij}$ update for all weights and biases

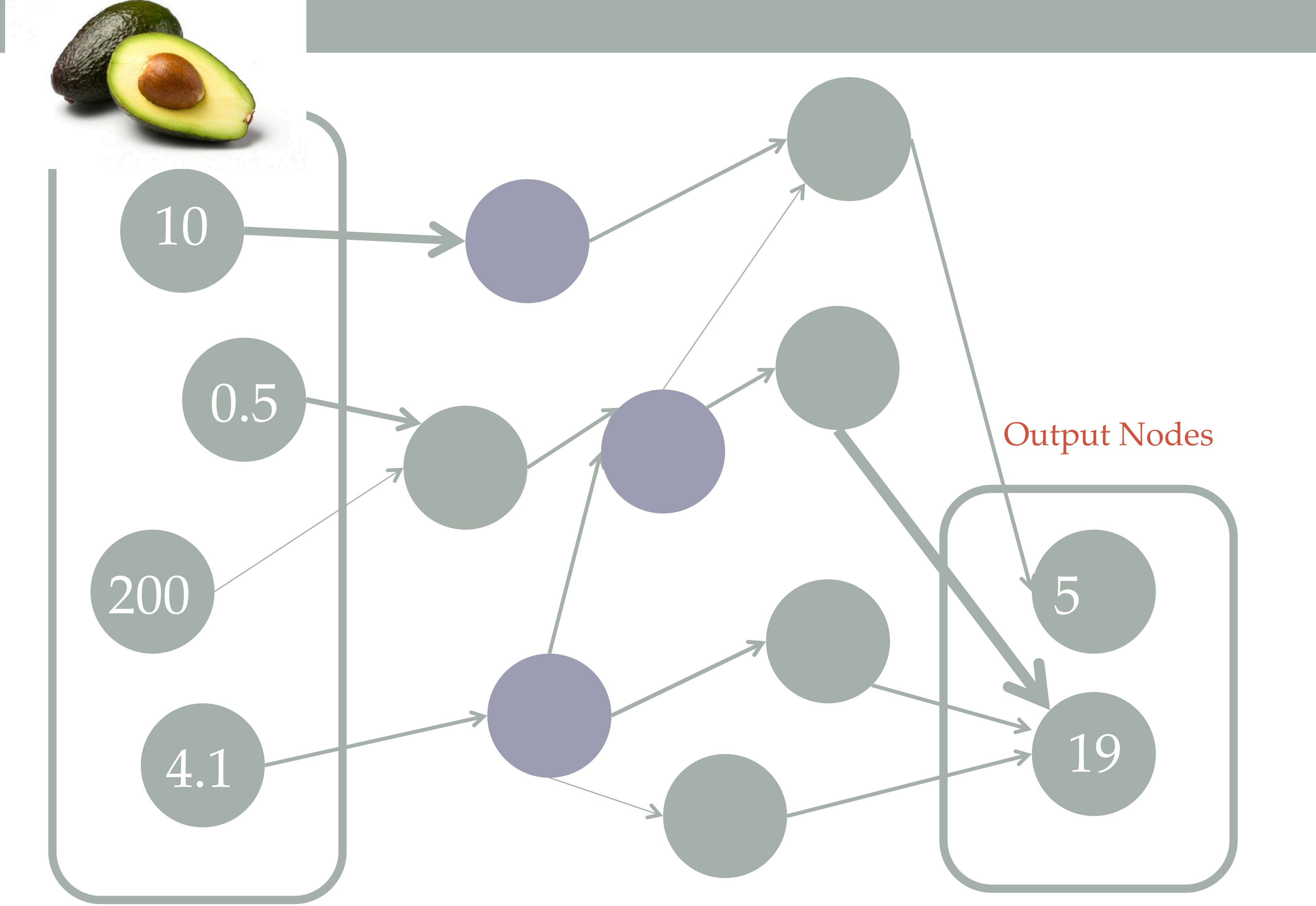
Values of the nodes

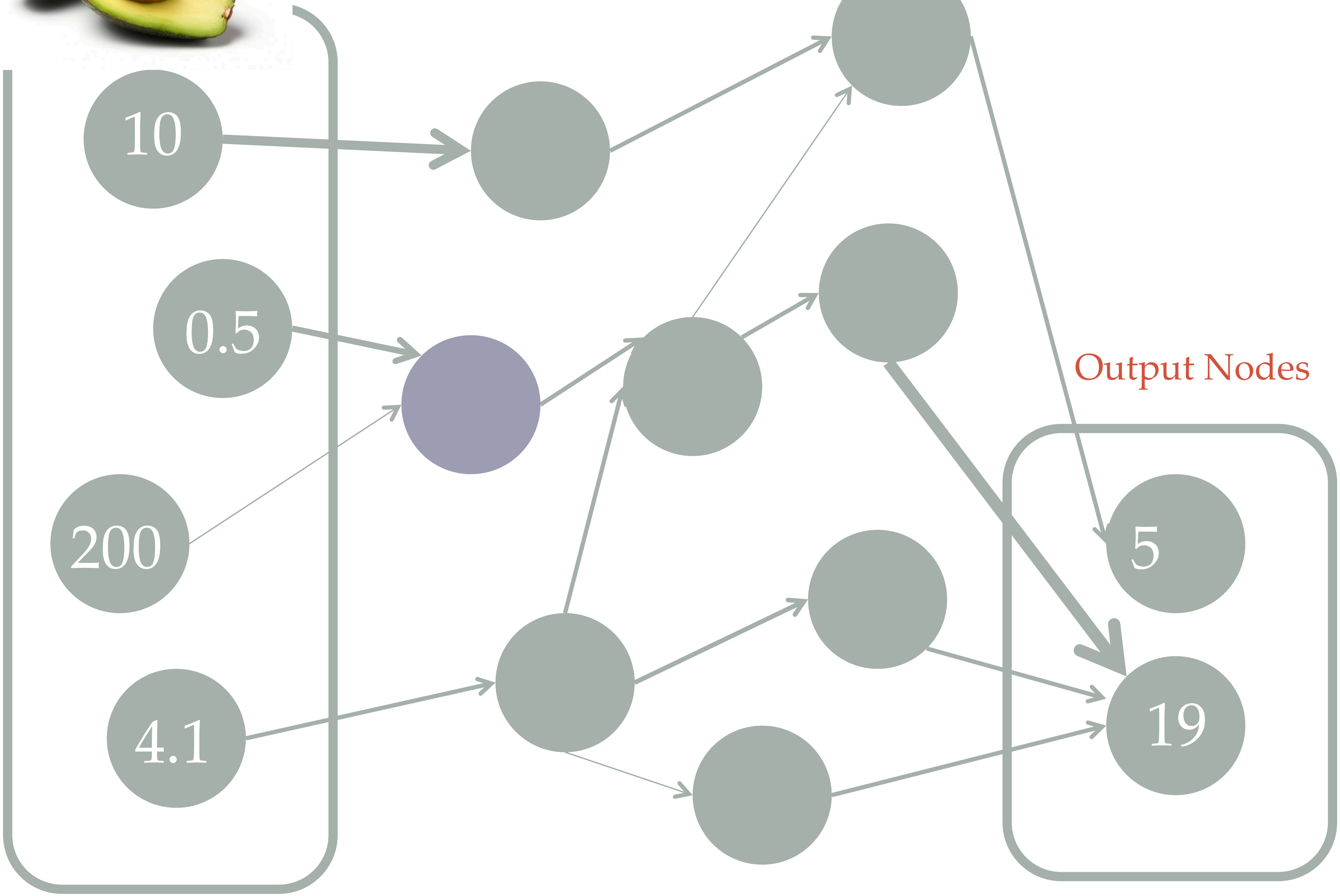
Amount of error

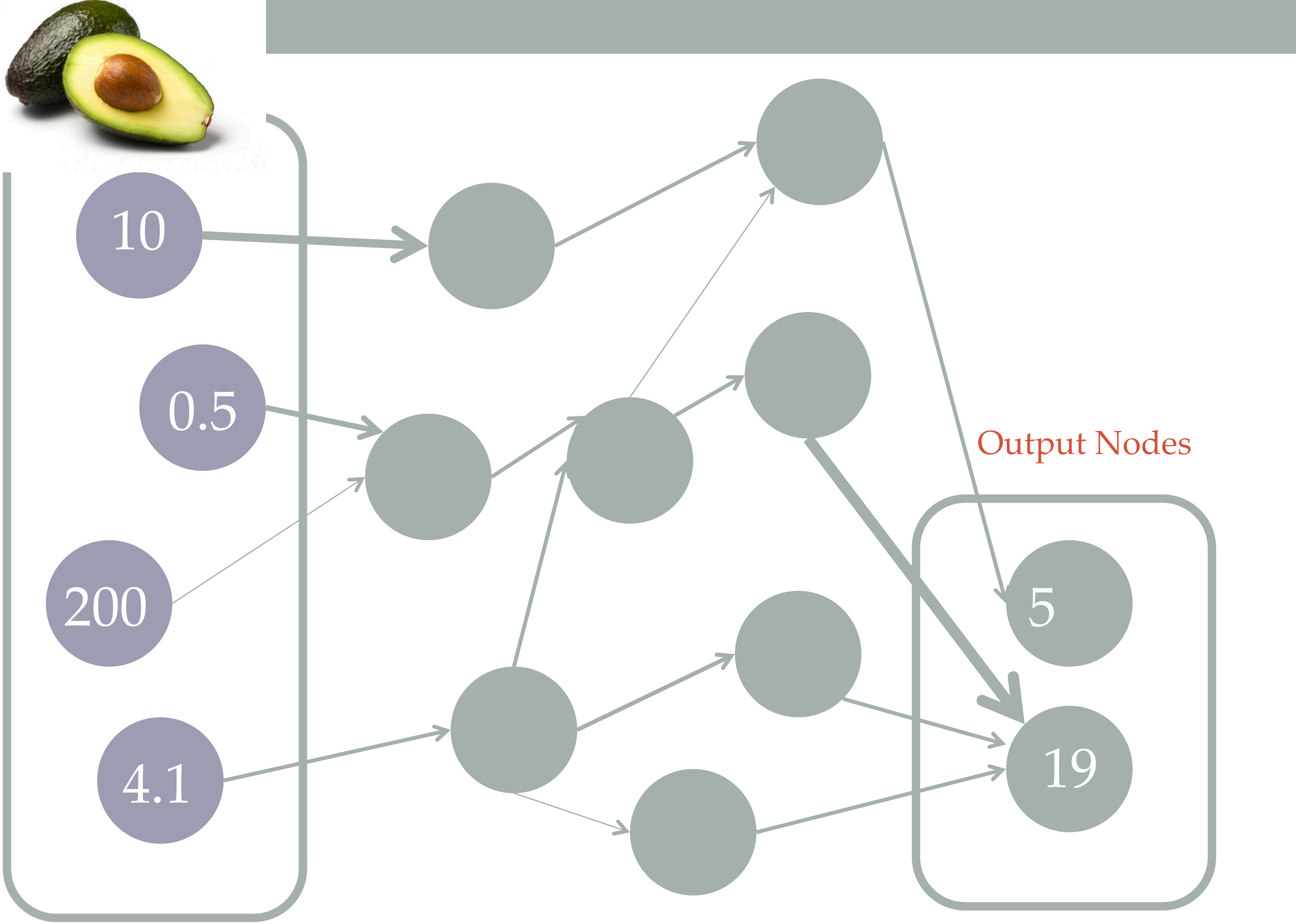
Weights of edges

Learning rate

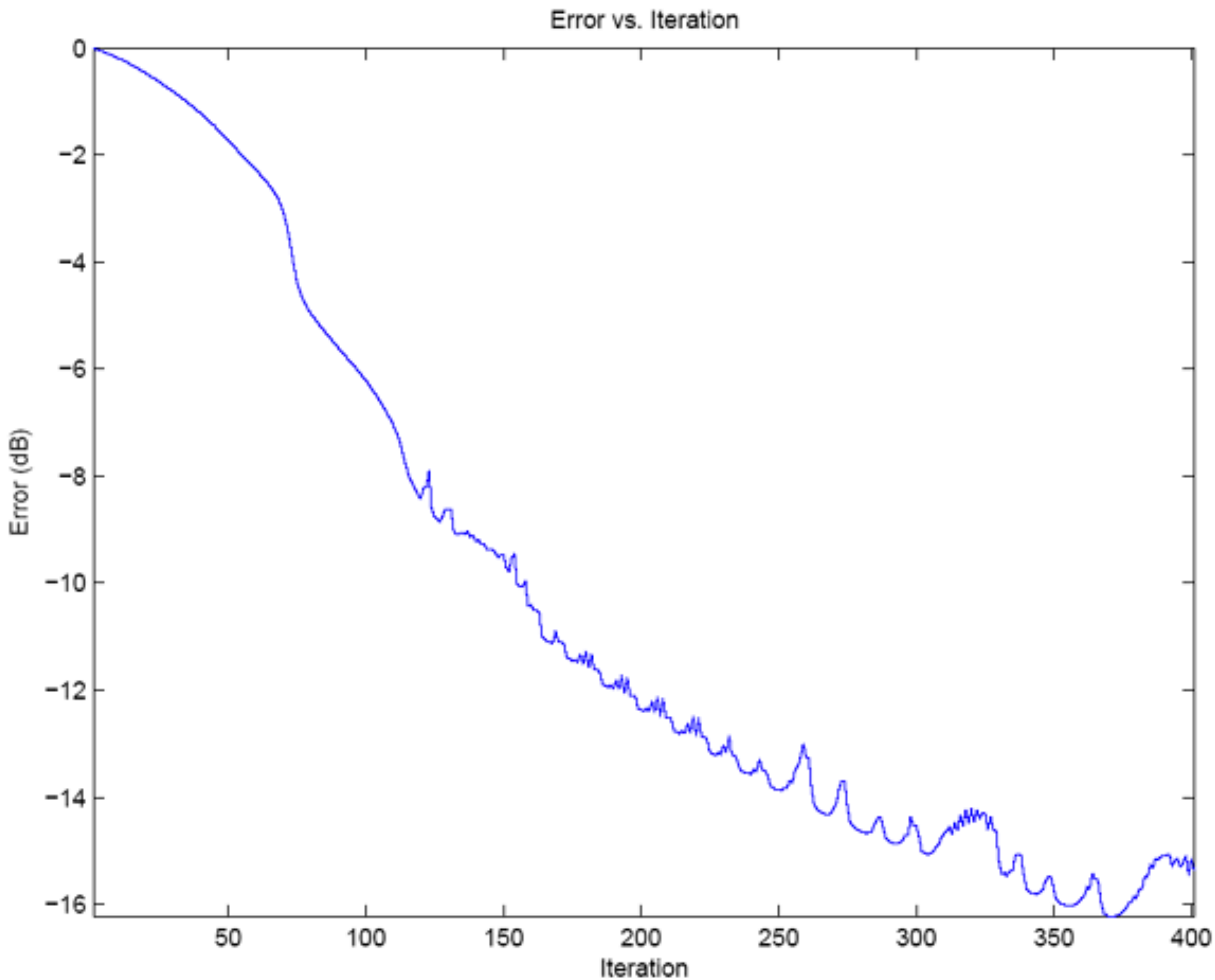

10
0.5
200
4.1
Output Nodes
5
19


10
0.5
200
4.1
Output Nodes
5
19


10
0.5
200
4.1
Output Nodes
5
19

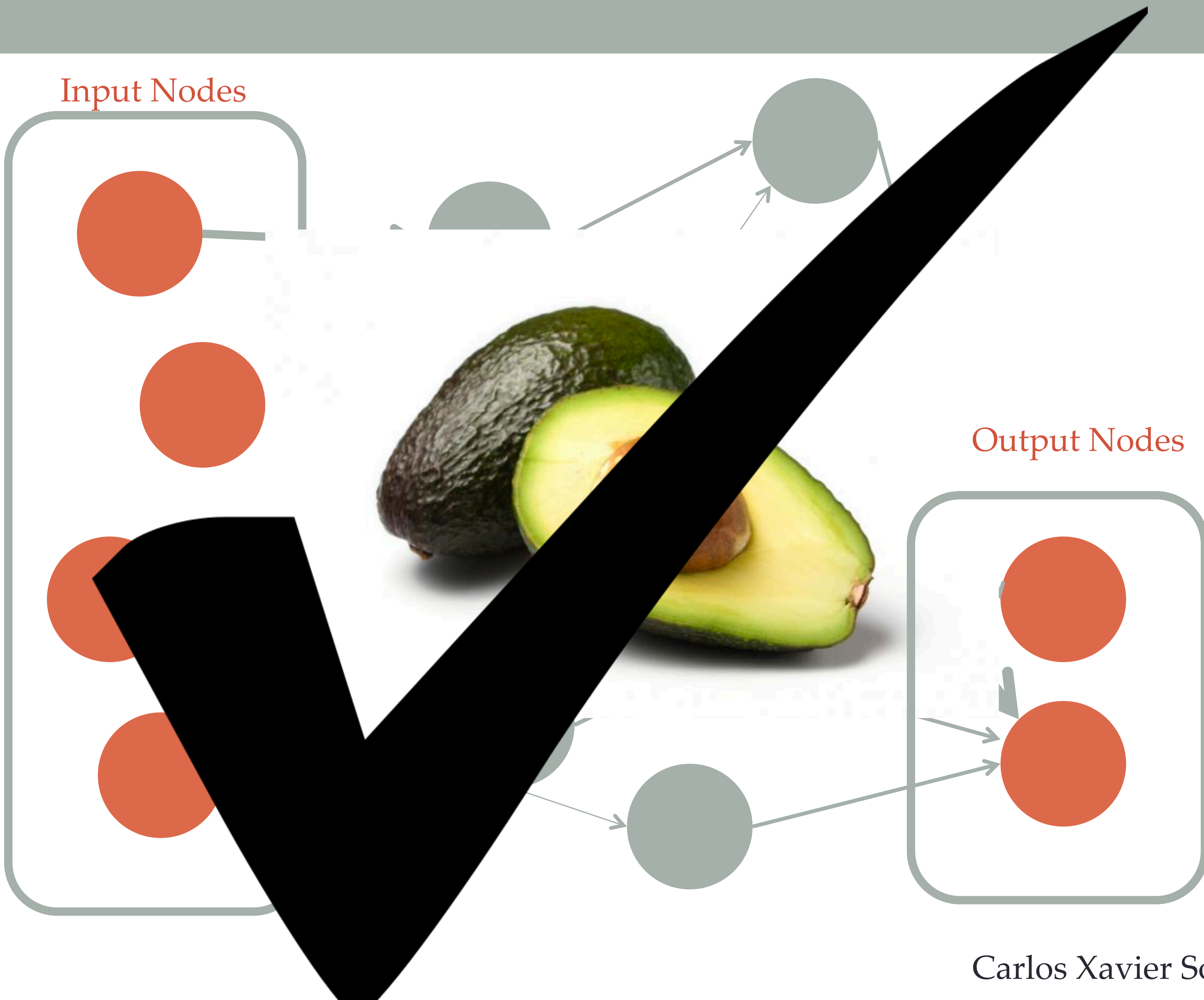


10
0.5
200
4.1
Output Nodes
5
19



Error vs. Iteration
Error (dB)
Iteration
0
−2
−4
−6
−8
−10
−12
−14
−16
50
100
150
200
250
300
350
400

Tuning parameters



Carlos Xavier Soto

Lesson 2: Neural networks can be trained on labeled data to classify avocados

DEEP NETS ON CAFFE

Scikit-learn

Caffe

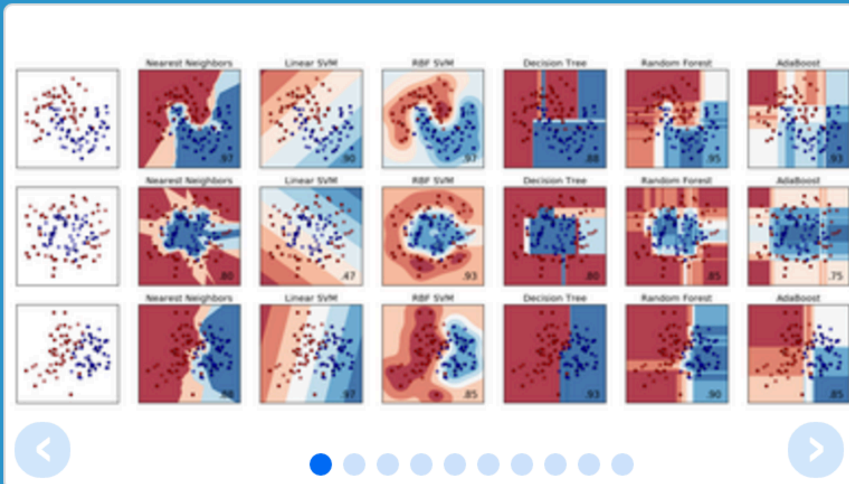
Theano

iPython Notebook

scikit learn

Home Installation Documentation Examples

Google™ Custom Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining a
- Accessible to everybody, and reusable in v
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD l

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping c sets.

Applications: Custo Grouping experimen
Algorithms: k-Mear mean-shift, ...

Dimensionality reduction

Reducing the number of random variables to

Model selection

Comparing, validating and choosing

Preprocessing

Feature extraction ar

Caffe

Deep learning framework by the BVLC

Created by
Yangqing Jia
Lead Developer
Evan Shelhamer

View On GitHub

Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

Check out our web image classification demo!

Why Caffe?

Expressive architecture encourages application and innovation. Models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices.

Extensible code fosters active development. In Caffe's first year, it has been forked by over 1,000 developers and had many significant changes contributed back. Thanks to these contributors the framework tracks the state-of-the-art in both code and models.

Speed makes Caffe perfect for research experiments and industry deployment. Caffe can process **over 60M images per day** with a single NVIDIA K40 GPU*. That's 1 ms/image for inference and 4 ms/image for learning. We believe that Caffe is the fastest convnet implementation available.

Community: Caffe already powers academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia. Join our community of brewers on

Welcome

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. Theano features:

- **tight integration with NumPy** – Use *numpy.ndarray* in Theano-compiled functions.
- **transparent use of a GPU** – Perform data-intensive calculations up to 140x faster than with CPU.(float32 only)
- **efficient symbolic differentiation** – Theano does your derivatives for function with one or many inputs.
- **speed and stability optimizations** – Get the right answer for `log(1+x)` even when `x` is really tiny.
- **dynamic C code generation** – Evaluate expressions faster.
- **extensive unit-testing and self-verification** – Detect and diagnose many types of errors.

Theano has been powering large-scale computationally intensive scientific investigations since 2007. But it is also approachable enough to be used in the classroom (University of Montreal's deep learning/machine learning classes).

News

- 2016/05/09: New technical report on Theano: Theano: A Python framework for fast computation of mathematical expressions. This is the new preferred reference.
- 2016/04/21: Release of Theano 0.8.2, adding support for CuDNN v5.
- 2016/03/29: Release of Theano 0.8.1, fixing a compilation issue on MacOS X with XCode 7.3.
- 2016/03/21: Release of Theano 0.8. Everybody is encouraged to update.
- Multi-GPU.
- We added support for **`CNMeM`** to speed up the GPU memory allocation.
- Theano 0.7 was released 26th March 2015. Everybody is encouraged to update.
- We support cuDNN if it is installed by the user.
- Open Machine Learning Workshop 2014 presentation.
- Colin Raffel tutorial on Theano.
- Ian Goodfellow did a 12h class with exercises on Theano.
- New technical report on Theano: Theano: new features and speed improvements

theano

Table Of Contents

Next topic

Release Notes

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

IP[y]: IPython

Interactive Computing

Install · Documentation · Project · Jupyter · News · Cite · Donate

The Jupyter Notebook

(Formerly known as the IPython Notebook)

The IPython Notebook is now known as the Jupyter Notebook. It is an interactive computational environment, in which you can combine code execution, rich text, mathematics, plots and rich media. For more details on the Jupyter Notebook, please see the Jupyter website.

Google™ Custom Search Search ×

NOTEBOOK VIEWER

Share your notebooks

COMMUNITY

Stack Overflow

Mailing list

File a bug

Reddit

Tweet

FOR DEVELOPERS

Mailing list

Scikit-learn

Caffe

Theano

iPython Notebook

Loading a pre-trained network into Caffe

Large Scale Visual Recognition Challenge 2010 (ILSVRC 2010)

10 million images
10,000 object classes
310,000 iterations

BVLC / caffe

Watch 1,408 | Star 10,496 | Fork 6,247

Code | Issues 421 | Pull requests 218 | Wiki | Pulse | Graphs

Branch: master | caffe / examples / 00-classification.ipynb | Find file | Copy path

longjon [example] improve classification notebook | bd6b03f Feb 24, 2016

3 contributors

781 lines (780 sloc) | 794 KB | Raw | Blame | History

Classification: Instant Recognition with Caffe

In this example we'll classify an image with the bundled CaffeNet model (which is based on the network architecture of Krizhevsky et al. for ImageNet).

We'll compare CPU and GPU modes and then dig into the model to inspect features and the output.

1. Setup

- First, set up Python, `numpy`, and `matplotlib`.

In [1]:

```
# set up Python environment: numpy for numerical routines, and matplotlib for plotting
import numpy as np
import matplotlib.pyplot as plt
```

2. Load net and set up input preprocessing

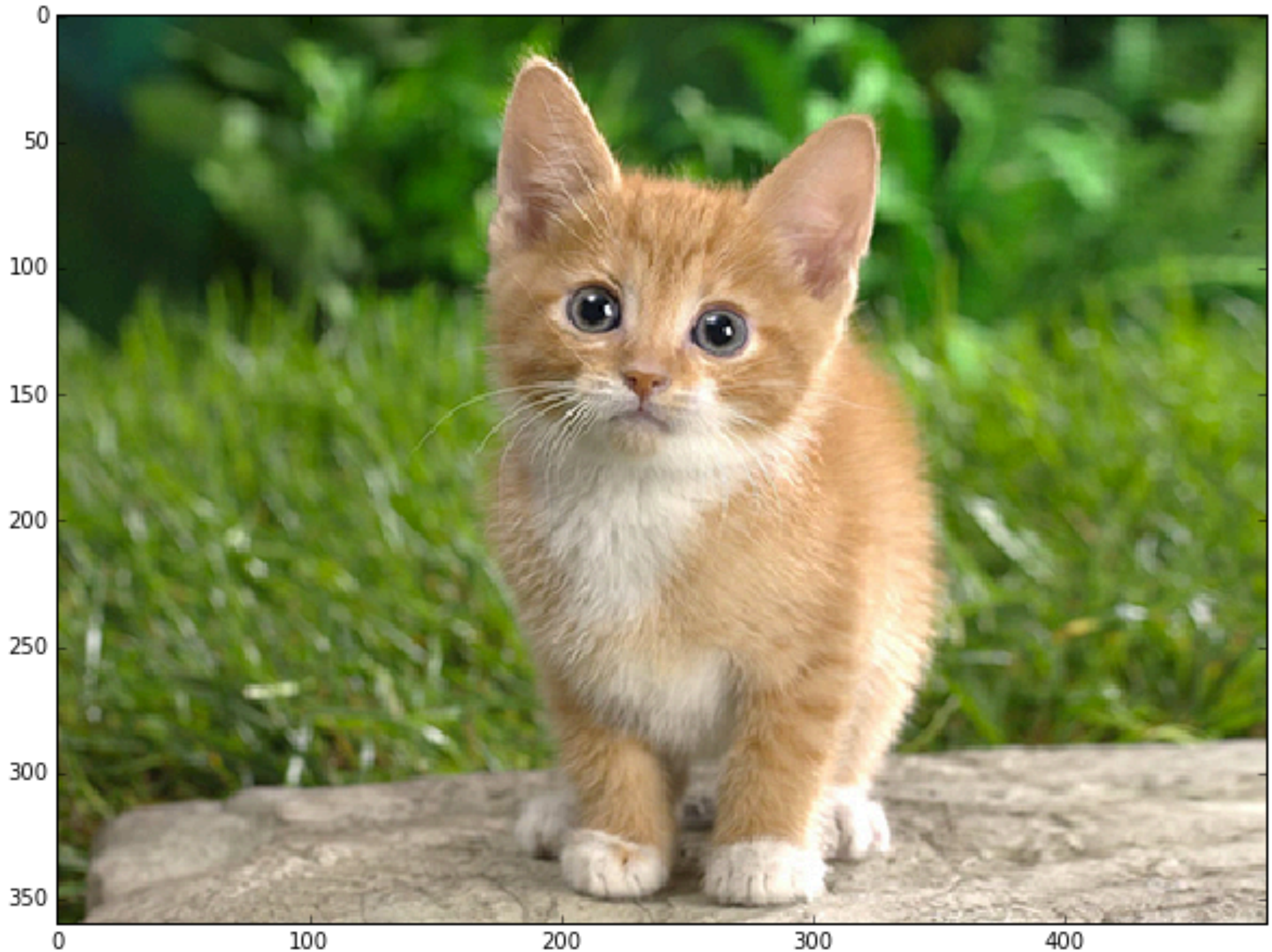
- Set Caffe to CPU mode and load the net from disk.

In [4]:
```
caffe.set_mode_cpu()

model_def = caffe_root + 'models/bvlc_reference_caffenet/deploy.prototxt'
model_weights = caffe_root + 'models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel'

net = caffe.Net(model_def,      # defines the structure of the model
                model_weights,  # contains the trained weights
                caffe.TEST)     # use test mode (e.g., don't perform dropout)
```

- Set up input preprocessing. (We'll use Caffe's `caffe.io.Transformer` to do this, but this step is independent of other parts of Caffe, so any custom preprocessing code may be used).

  Our default CaffeNet is configured to take images in BGR format. Values are expected to start in the range [0, 255] and then have the mean ImageNet pixel value subtracted from them. In addition, the channel dimension is expected as the first (*outermost*) dimension.

  As matplotlib will load images with values in the range [0, 1] in RGB format with the channel as the *innermost* dimension, we are arranging for the needed transformations here.


0
50
100
150
200
250
300
350
0
100
200
300
400

In [9]:
```python
# load ImageNet labels
labels_file = caffe_root + 'data/ilsvrc12/synset_words.txt'
if not os.path.exists(labels_file):
    !../data/ilsvrc12/get_ilsvrc_aux.sh

labels = np.loadtxt(labels_file, str, delimiter='\t')

print 'output label:', labels[output_prob.argmax()]
```

```
output label: n02123045 tabby, tabby cat
```

Tabby cat

- "Tabby cat" is correct! But let's also look at other top (but less confident predictions).

In [10]:
```python
# sort top five predictions from softmax output
top_inds = output_prob.argsort()[::-1][:5]  # reverse sort and take five largest items

print 'probabilities and labels:'
zip(output_prob[top_inds], labels[top_inds])
```

```
probabilities and labels:
```

Out[10]:
```
[(0.31243637, 'n02123045 tabby, tabby cat'),
 (0.2379719, 'n02123159 tiger cat'),
 (0.12387239, 'n02124075 Egyptian cat'),
 (0.10075711, 'n02119022 red fox, Vulpes vulpes'),
 (0.070957087, 'n02127052 lynx, catamount')]
```

Tabby cat
Tiger cat
Egyptian cat
Red fox
Lynx

Lesson 3: Caffe provides pre-trained networks to jumpstart learning

Today

- Lesson 1: Why now? Big data, big processing power, robust neural networks
- Lesson 2: Neural networks can be trained on labeled data to classify avocados
- Lesson 3: Caffe provides pre-trained networks to jumpstart learning

What do **you** go from here?

Today

- Lesson 1: Why now? Big data, big processing power, robust neural networks
- Lesson 2: Neural networks can be trained on labeled data to classify avocados
- Lesson 3: Caffe provides pre-trained networks to jumpstart learning

Cuda implementations

Theano, Tensorflow, etc

Today

- Lesson 1: Why now? Big data, big processing power, robust neural networks
- Lesson 2: Neural networks can be trained on labeled data to classify avocados
- Lesson 3: Caffe provides pre-trained networks to jumpstart learning

Restricted Boltzmann Machines

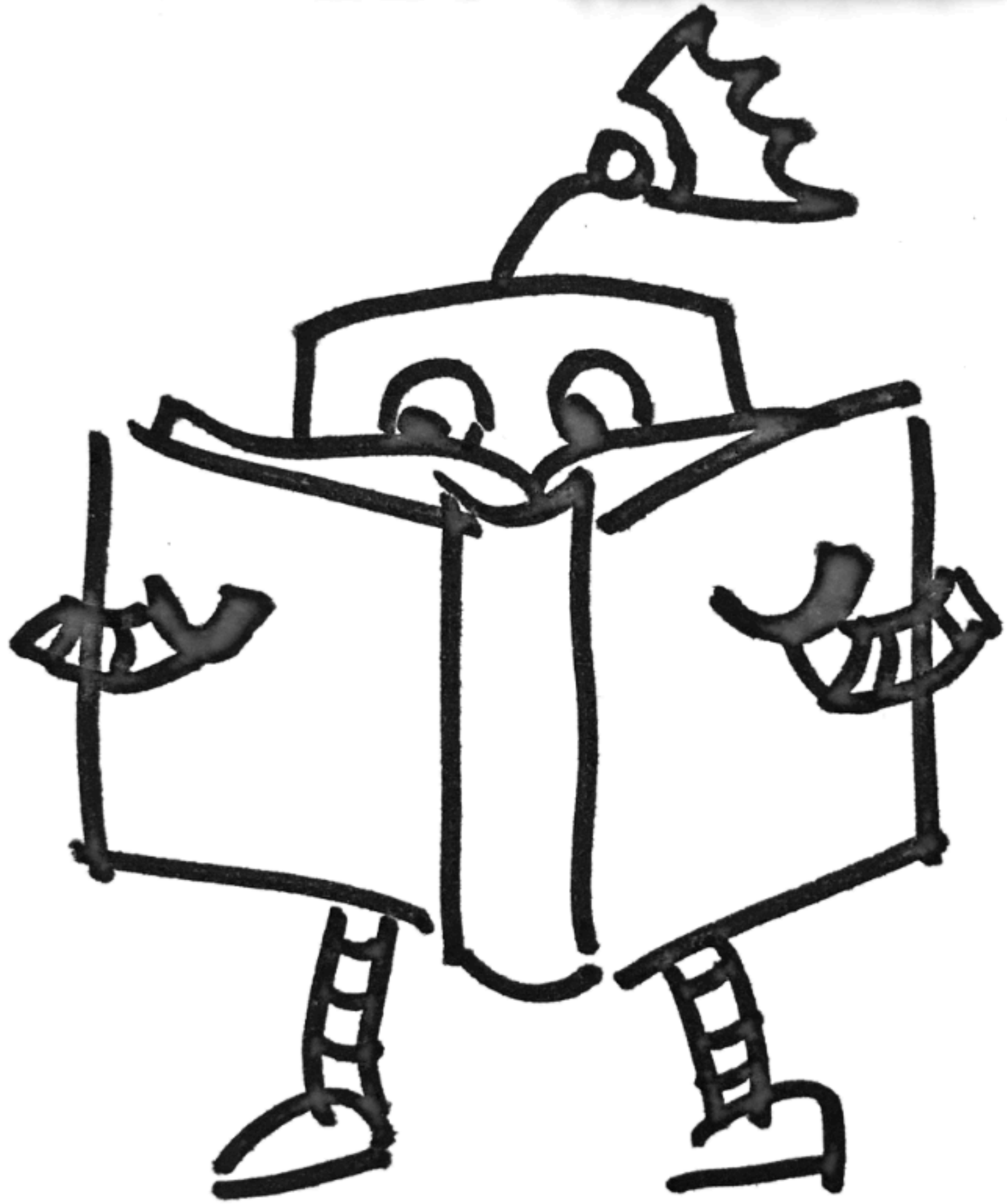
Recurrent network

Convolutional network

Today

- Lesson 1: Why now? Big data, big processing power, robust neural networks
- Lesson 2: Neural networks can be trained on labeled data to classify avocados
- Lesson 3: Caffe provides pre-trained networks to jumpstart learning

Caffe iPython notebooks
Kaggle competitions



Thank you!

irenetrampoline@gmail.com