Fundamentals of Deep Learning

Jeff Prosise
jeffpro@wintellect.com
@jprosise

Wintellect®

Wintellect NOW

The Big Picture

1950s – Researchers begin pondering whether computers can think like humans and devise **Symbolic AI** – e.g. chess programs

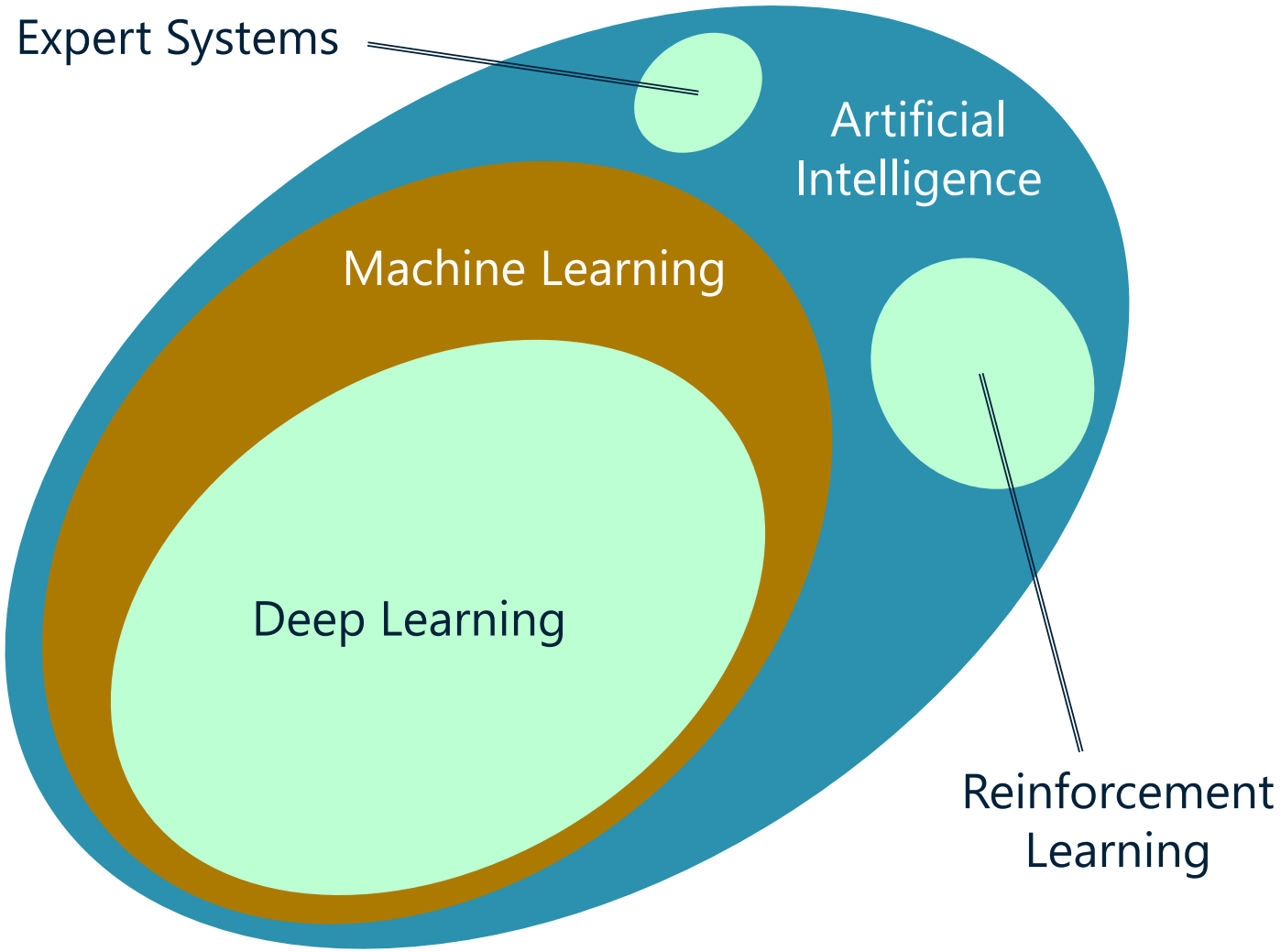
1960s – Early **neural networks** are devised, but training is impractical due to lack of computing power and access to compute resources

1980s – Researchers improve **back-propagation algorithms** for training neural networks and apply them to convolutional neural networks

1990s – Researchers at Bell Labs invent the modern **Support Vector Machine** (SVM) algorithm for discovering decision boundaries

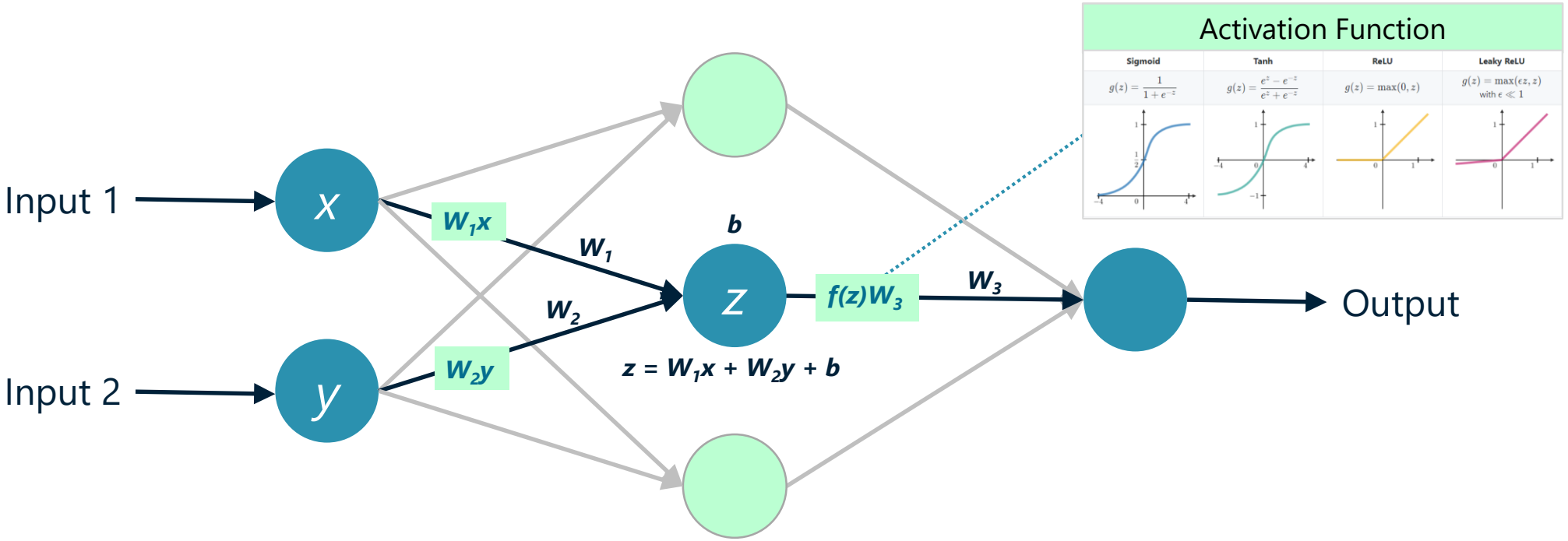
2000s – Algorithmic advances in machine learning give rise to **decision trees, random forests**, **gradient-boosting machines**, and more

2010s – ML and AI explode due to faster computing hardware (especially **GPUs**), expanded availability of data, and increased research funding

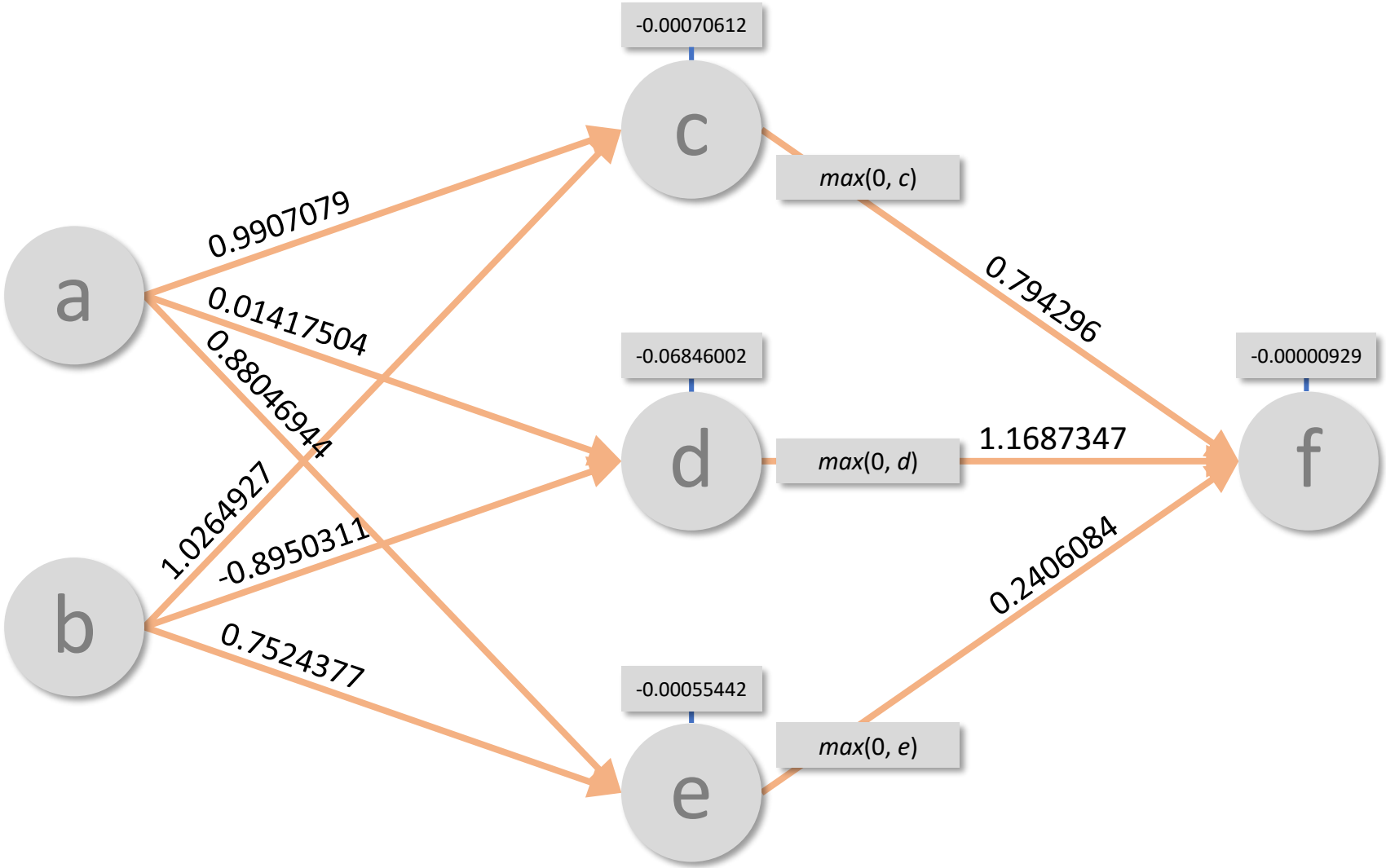


Neural Networks

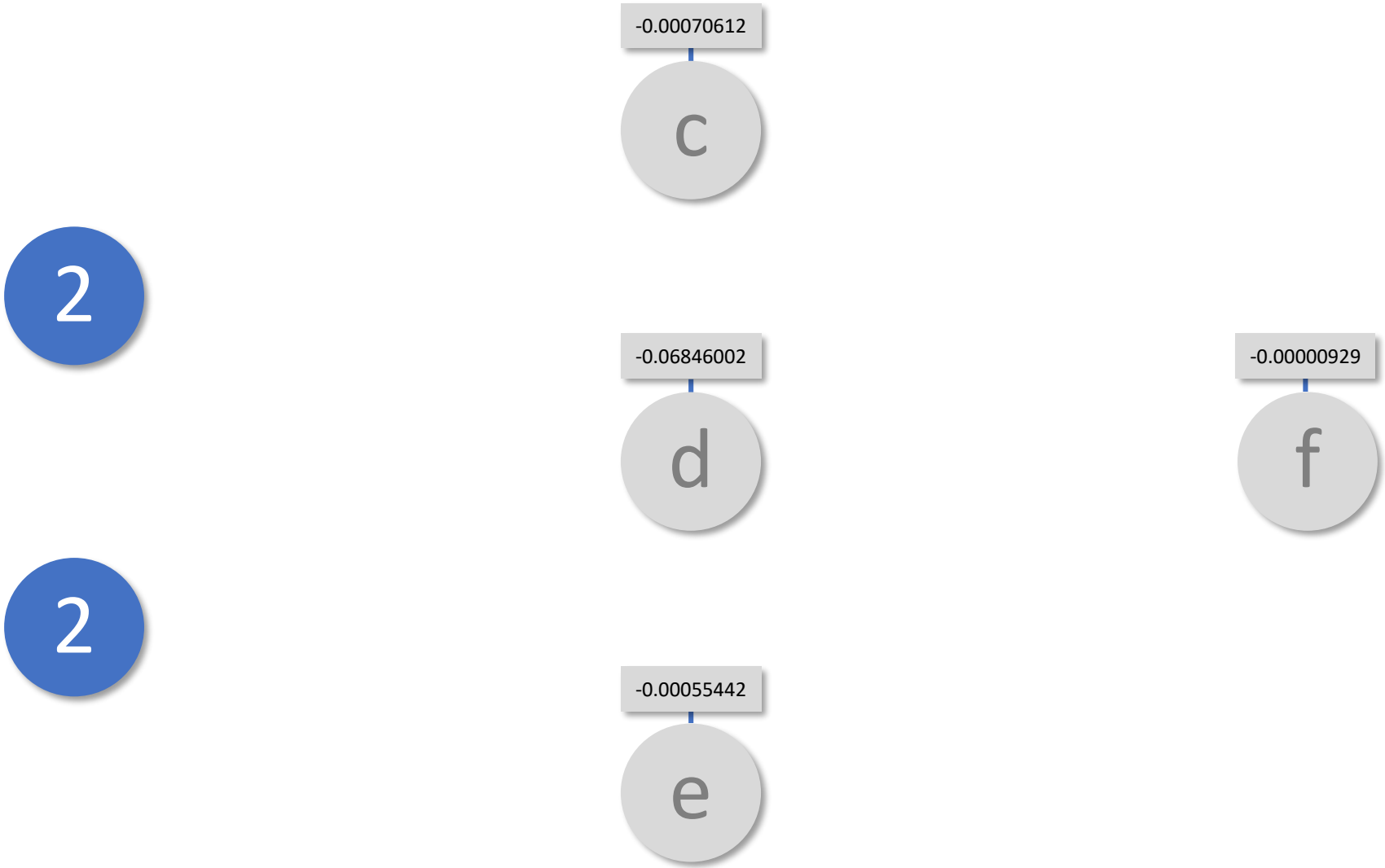
- Contain layers of *neurons* connected by paths assigned *weights*
- Data propagates forward by summing the products of values and weights and using activation functions to add non-linearity



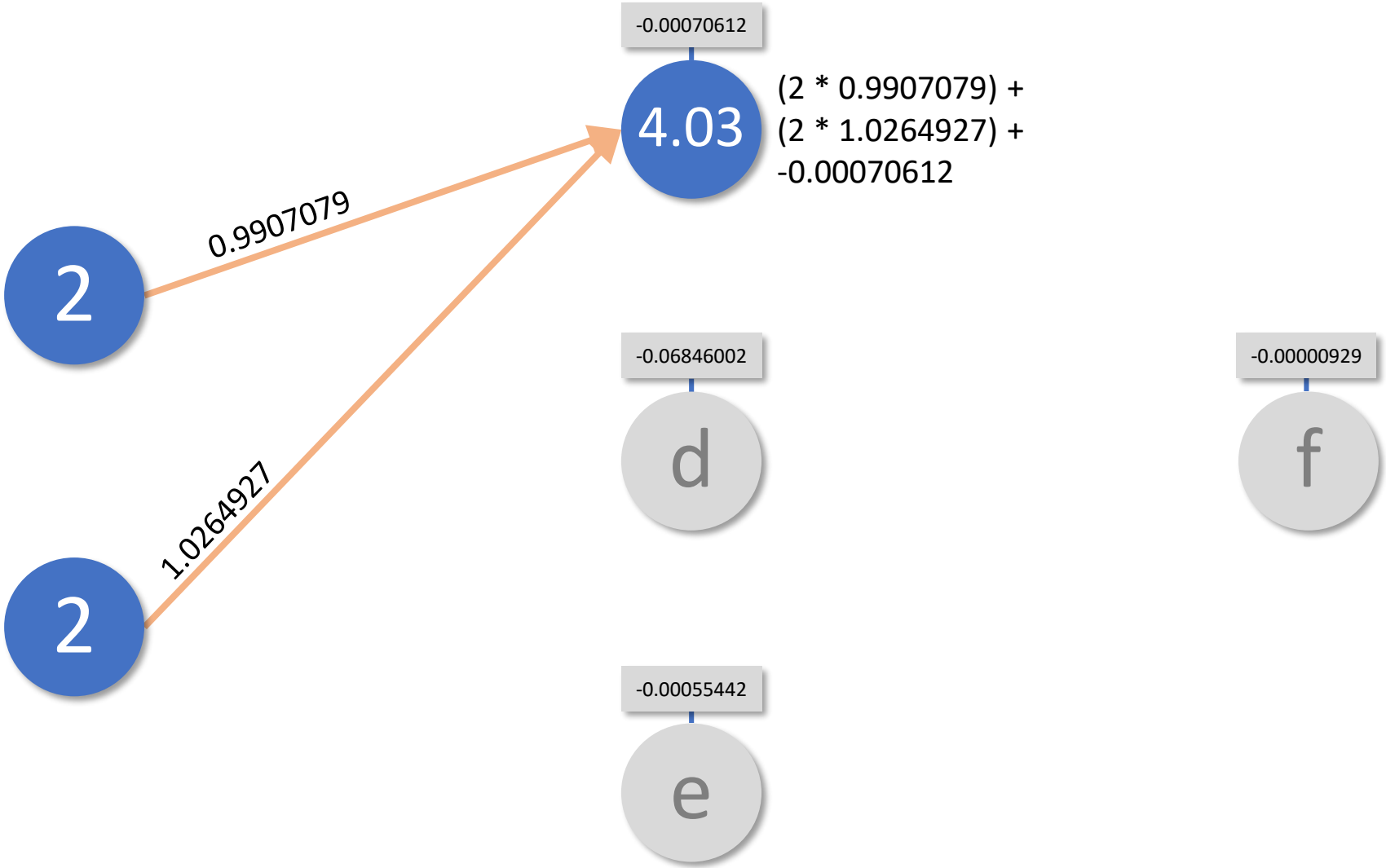
How Neural Networks Work



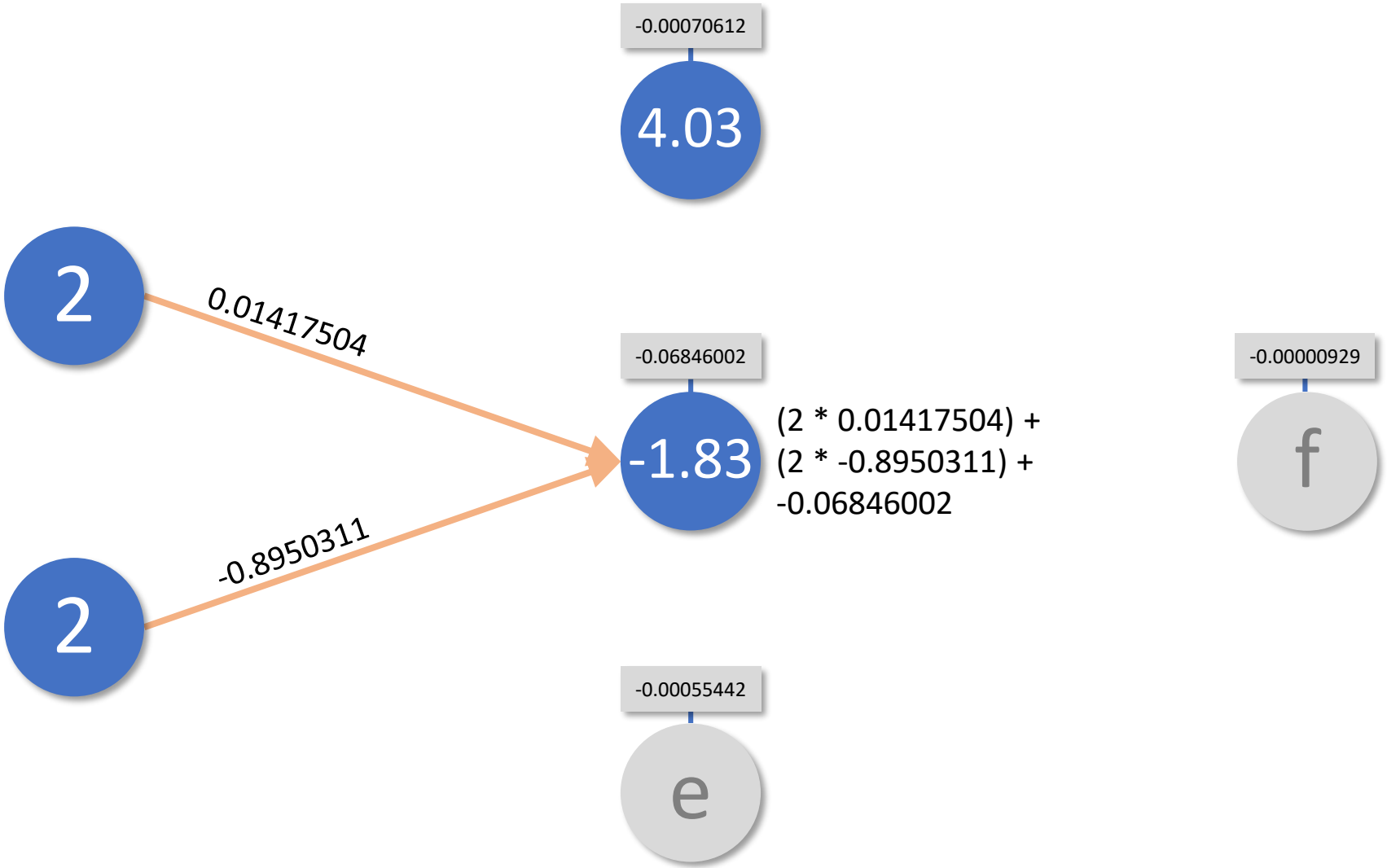
How Neural Networks Work, Cont.



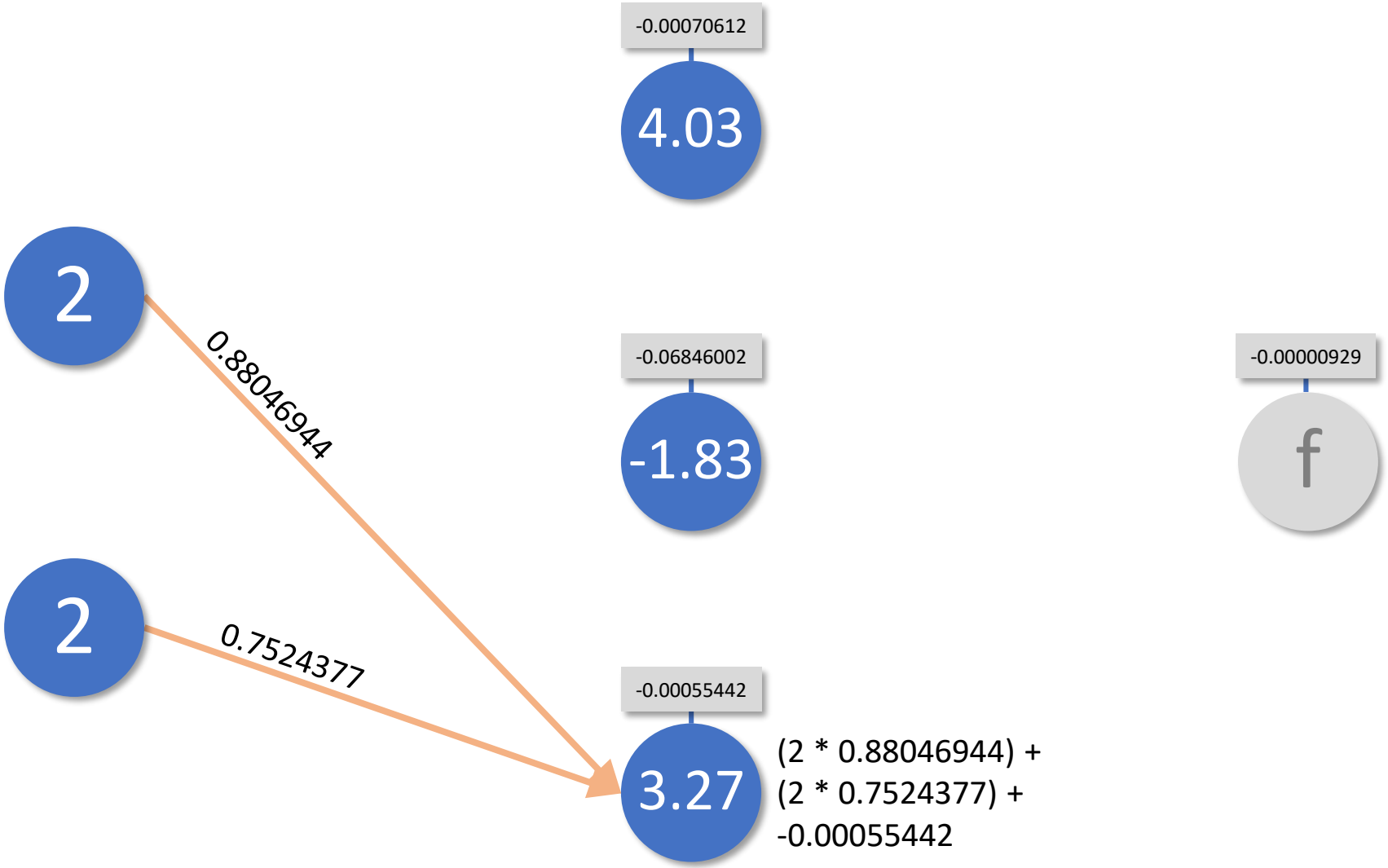
How Neural Networks Work, Cont.



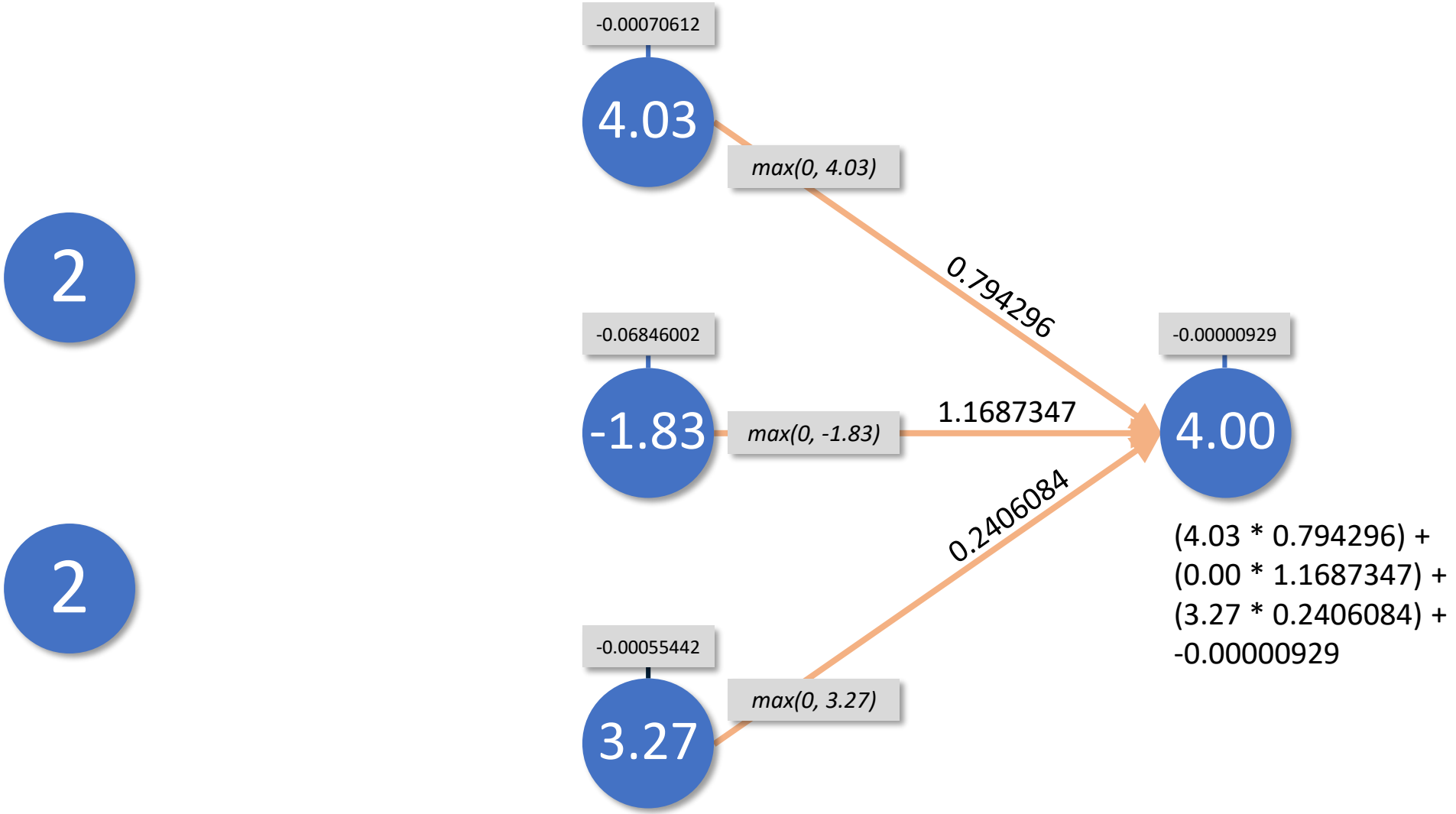
How Neural Networks Work, Cont.



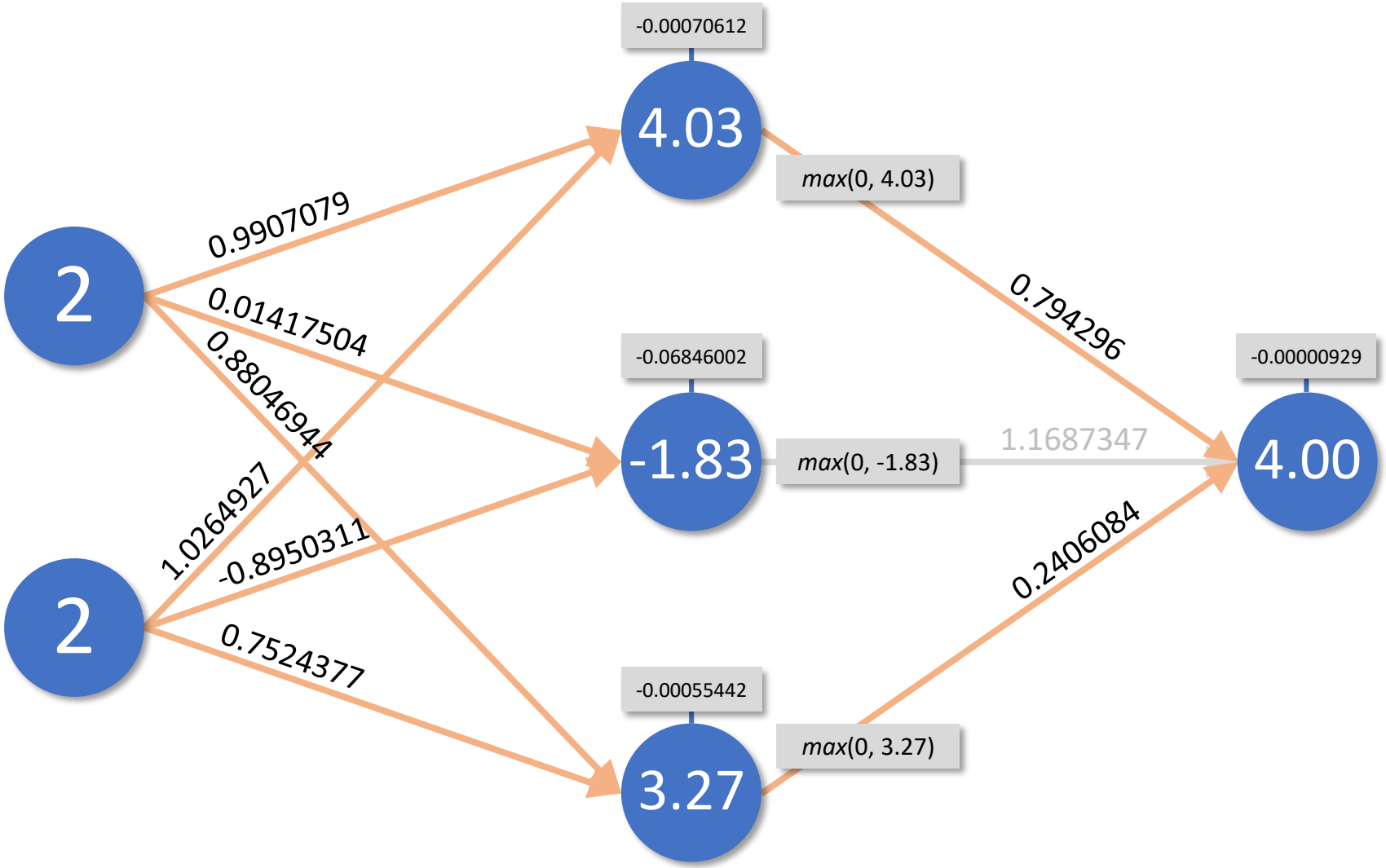
How Neural Networks Work, Cont.



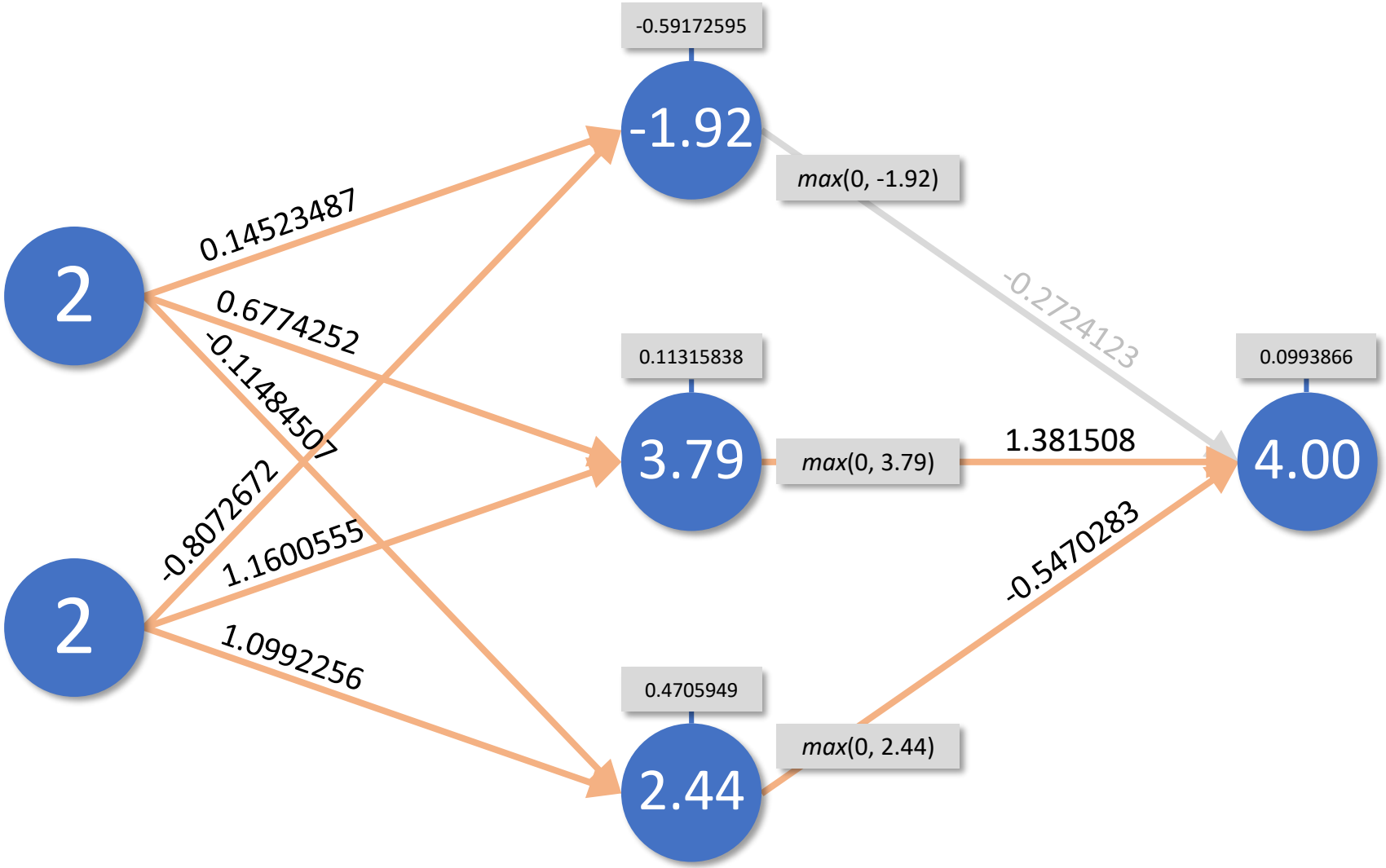
How Neural Networks Work, Cont.



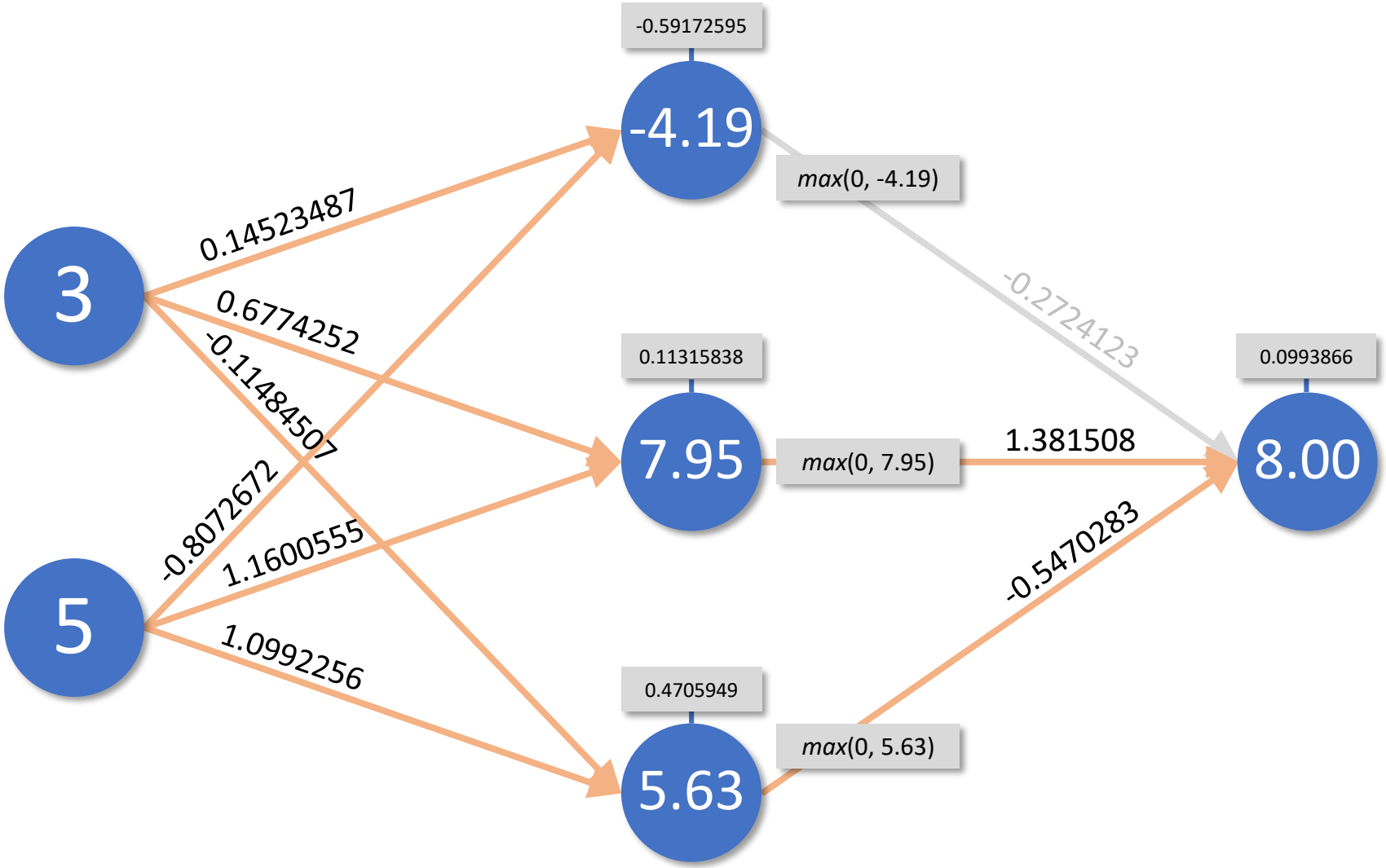
How Neural Networks Work, Cont.



How Neural Networks Work, Cont.



How Neural Networks Work, Cont.



Deep-Learning Libraries

Library	Language(s)	GitHub URL
TensorFlow	Python, C++, JavaScript	https://github.com/tensorflow/tensorflow
Microsoft Cognitive Toolkit (CNTK)	Python, C++, C#	https://github.com/Microsoft/CNTK
Theano	Python	http://deeplearning.net/software/theano/
Keras	Python	https://github.com/keras-team/keras
Caffe / Caffe2	Python, C++	https://github.com/caffe2/caffe2
PyTorch	Python	https://github.com/pytorch/pytorch
Apache MXNet	Python, C++, JavaScript, R, Julia, Scala, Perl	https://github.com/apache/incubator-mxnet
DeepLearning4J (DL4J)	Java, Scala, and other JVM languages	https://github.com/deeplearning4j/deeplearning4j
Core ML (iOS)	Swift, Objective-C	N/A

Building a Neural Network with Keras

```
from keras.layers import Dense
from keras.models import Sequential

model = Sequential()
model.add(Dense(16, activation='relu', input_dim=2)) # Each input contains 2 values
model.add(Dense(16, activation='relu'))
model.add(Dense(1)) # Single numeric output
model.compile(loss='mae', optimizer='adam', metrics=['mae'])
```

Training a Neural Network

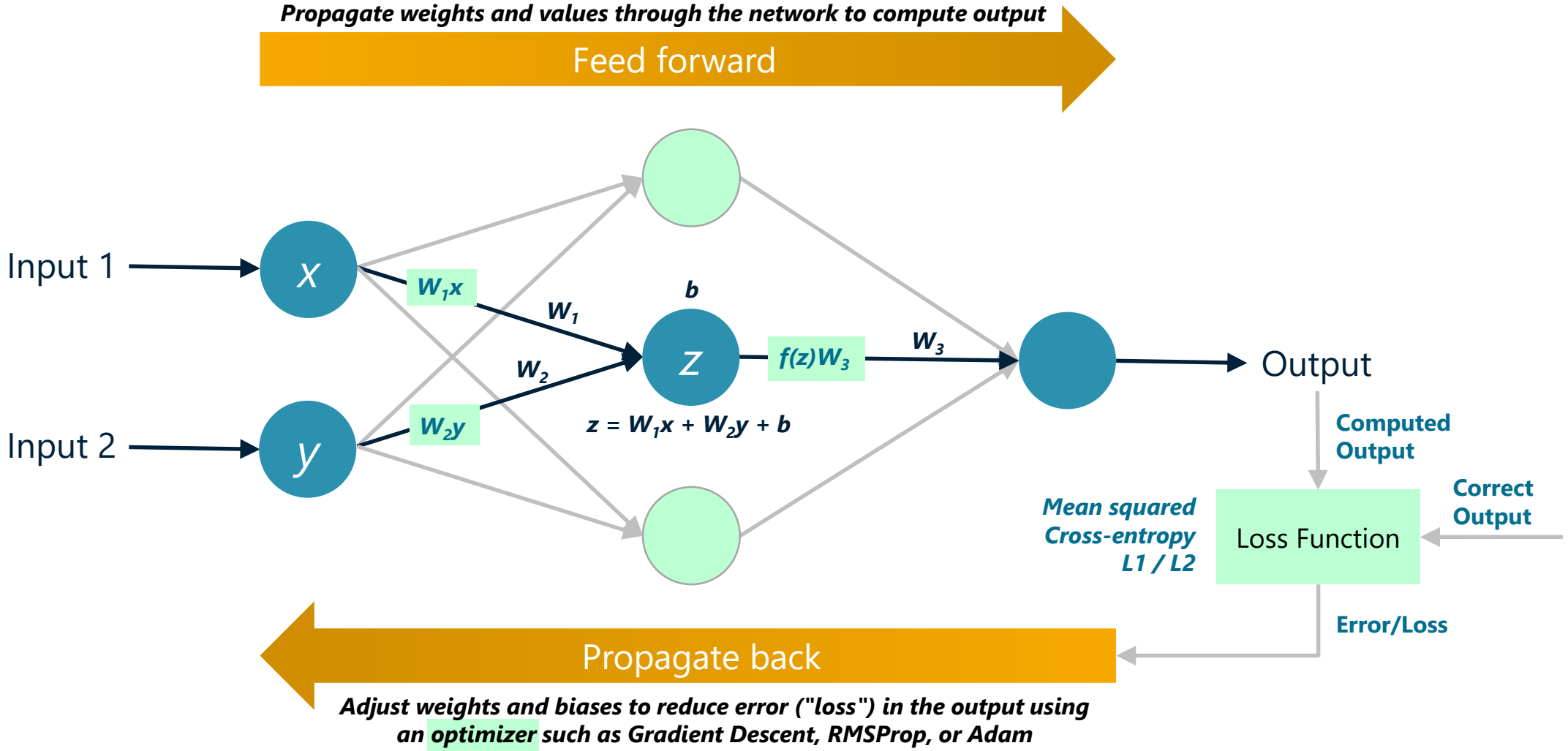
```
# Train without validation
model.fit(x, y, epochs=10, batch_size=128)

# Train with 80% of the input data and validate with 20%
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=128)

# Train using provided test data for validation
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=128)

# Evaluate the model's accuracy separately
scores = model.evaluate(x_test, y_test, verbose=0)
```

Backpropagation



Making a Prediction

```
import numpy as np

model.predict(np.array([[1.0, 2.0]])) # Assumes network expects two values in each input
```

Demo

Your First Neural Network

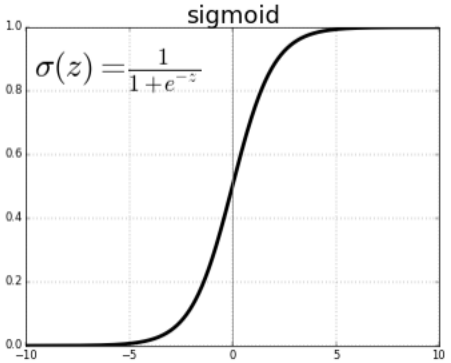
Classification

- Determine to which class (category) an input belongs
  - Which character of the alphabet does a hand-written character represent?
  - Is a credit-card transaction fraudulent or not fraudulent?
  - Is an e-mail spam or not spam?
- Binary classification
  - Sigmoid activation on output, binary_crossentropy loss function
- Multiclass classification
  - Softmax activation on output, categorical_crossentropy loss function

Building and Training a Binary Classifier

```
from keras.layers import Dense
from keras.models import Sequential

model = Sequential()
model.add(Dense(16, activation='relu', input_dim=(...)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid')) # Probability from 0.0 to 1.0
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=128)
```



Building and Training a Multiclass Classifier

```
from keras.layers import Dense
from keras.models import Sequential

model = Sequential()
model.add(Dense(16, activation='relu', input_dim=(...)))
model.add(Dense(16, activation='relu'))
model.add(Dense(10, activation='softmax')) # 10 possible classes
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=128)
```

Making a Prediction with a Classifier

```
import numpy as np

# Get the probabilities for each class
model.predict(np.array([[...]]))

# Get the predicted class
model.predict_classes(np.array([...]))[0]
```

Demo

Multiclass Classification

Working with Text

- How do you build machine-learning models that classify text?
  - Classify e-mails as spam or not spam
  - Score text for sentiment (sentiment analysis)
- Raw text must be prepared for use in machine learning
  - Remove numbers, symbols, punctuation characters, and stop words
  - Stem the words (e.g., turn "cats" into "cat") and convert to lowercase
- Processed text must then be vectorized
  - Use Keras's **Tokenizer** class to do the vectorizing

Vectorizing Text

```
documents = [
    'Quick brown fox', # Stop words removed
    'Jumps over lazy dog'
]

tokenizer = Tokenizer()
tokenizer.fit_on_texts(documents)
vectors = tokenizer.texts_to_matrix(documents)
```

Vocabulary

quick	1
brown	2
fox	3
jumps	4
over	5
lazy	6
dog	7

	quick	brown	fox	jumps	over	lazy	dog
0	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1

Turning Text into Sequences

```
documents = [
    'Quick brown fox', # Stop words removed
    'Jumps over lazy dog'
]

tokenizer = Tokenizer()
tokenizer.fit_on_texts(documents)
sequences = tokenizer.texts_to_sequences(documents)
padded_sequences = pad_sequences(sequences, 5)
```

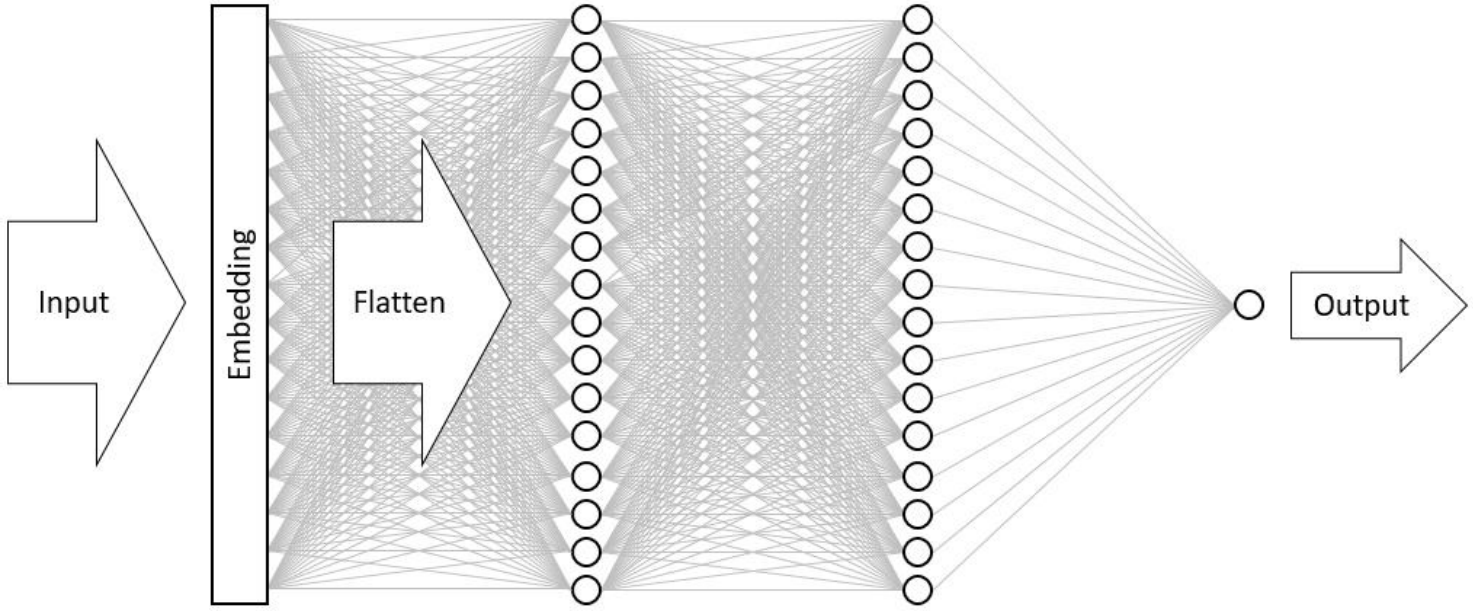
Vocabulary

quick	1
brown	2
fox	3
jumps	4
over	5
lazy	6
dog	7

0	0	1	2	3
0	4	5	6	7

Word Embeddings

- Neural networks use word-embedding layers to turn text sequences containing word indexes into dense vectors of a specified size
- Keras makes this easy with its **Emdedding** class



Classifying Text Using Word Embeddings

```
model = Sequential()
model.add(Embedding(10000, 32, input_length=500))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=5, batch_size=32)
```

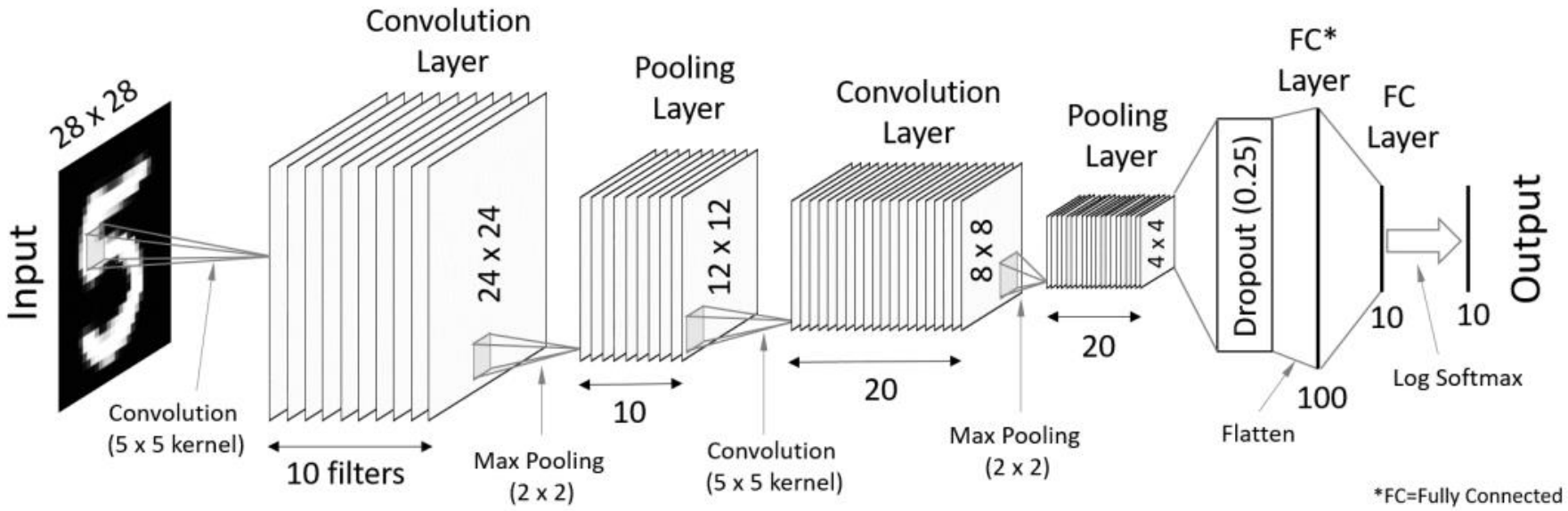
Vocabulary length = 10,000
Output dimension = 32
Length of each sequence = 500

Demo

Binary Classification

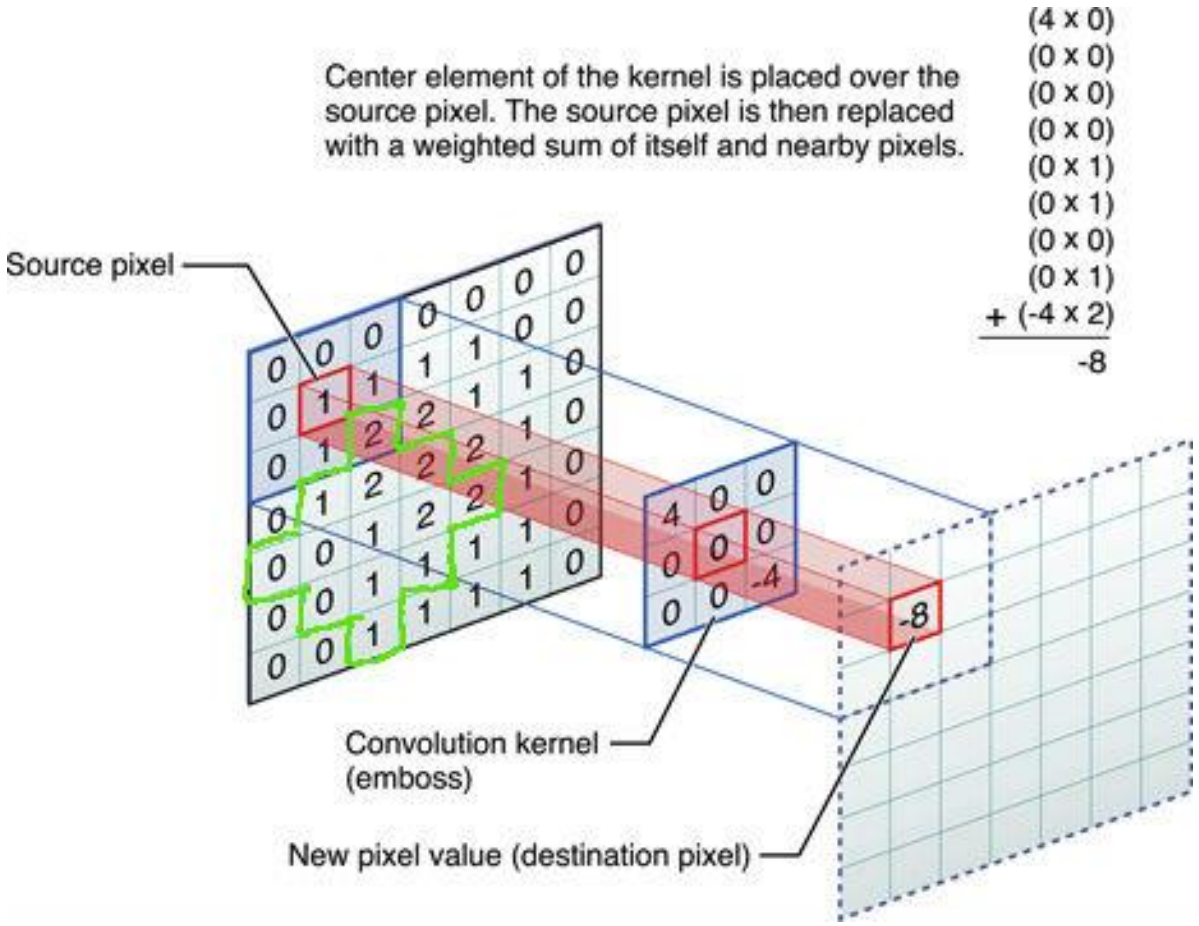
Convolutional Neural Networks

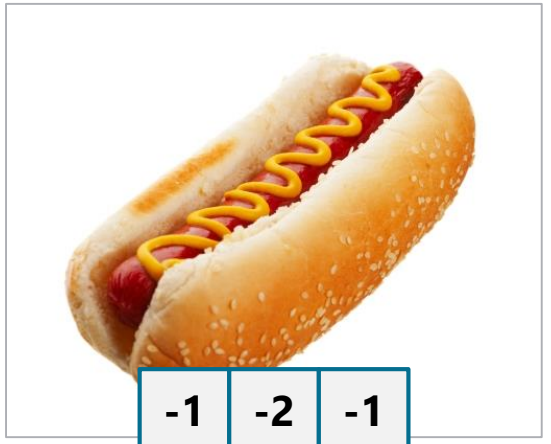
- Excel at tasks involving computer vision and sequence processing
- Use convolution layers and convolution kernels to create feature maps
- Use pooling layers to subsample feature maps and generalize features



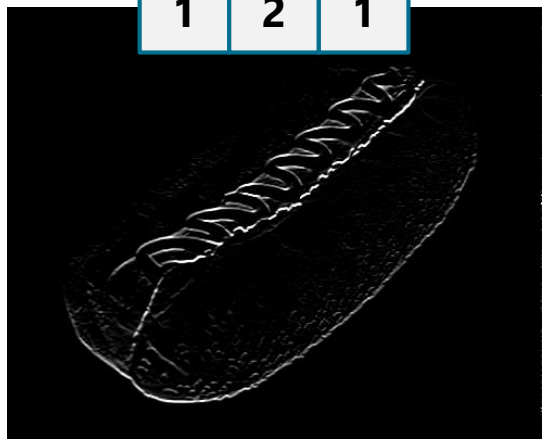
Source: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Feature Mapping with Convolution Kernels





-1	-2	-1
0	0	0
1	2	1



Source: https://stats.stackexchange.com/questions/235032/any-use-of-non-rectangular-shaped-kernels-in-convolutional-neural-networks-espe

Building and Training a CNN

```
model = Sequential()
model.add(Conv2D(10, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(20, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten()) # Reshape output from previous layer for input to next layer
model.add(Dropout(0.25)) # Regularize by randomly dropping 25% of the inputs in each epoch
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=128)
```

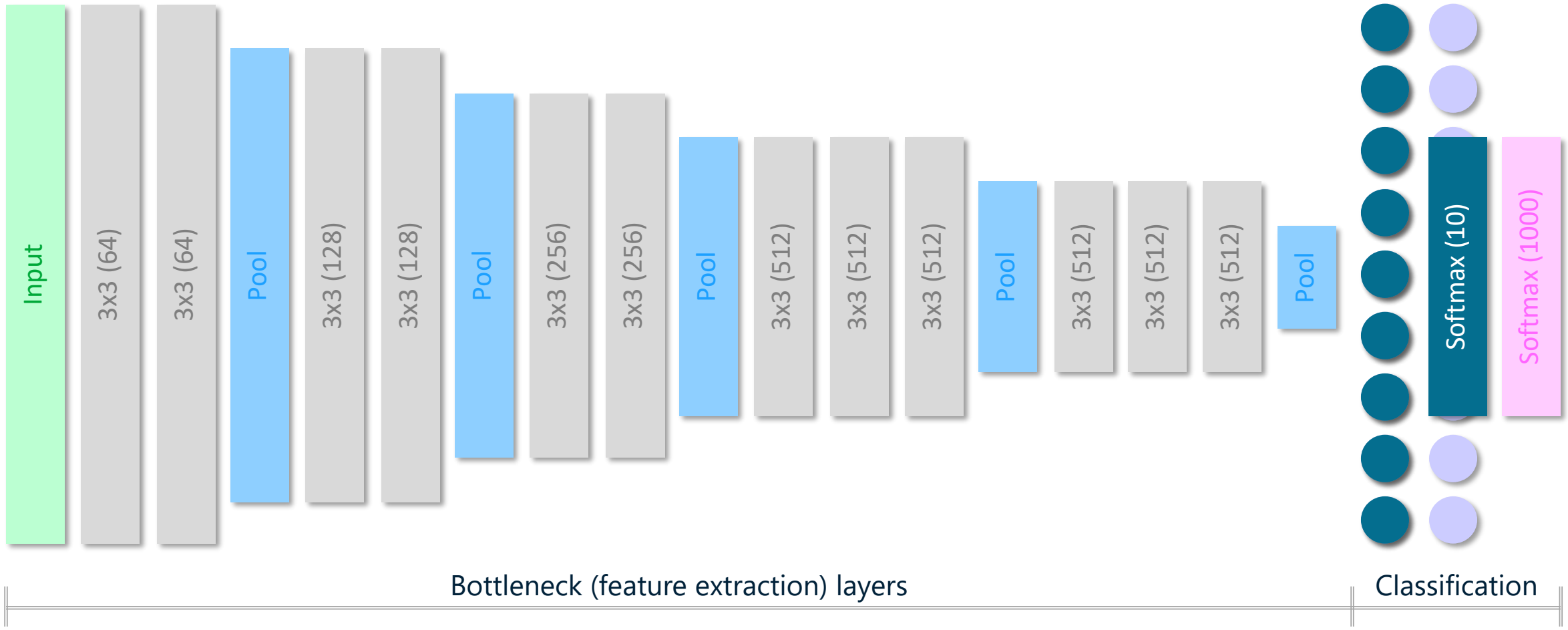
Demo

Image Classification

Transfer Learning

- Leverages pretrained CNNs to achieve acceptable accuracy with exponentially less data, compute power, and training time
  - Replaces fully connected classification layers in pretrained model with new layers, reusing pretrained model's convolutional base for feature extraction
  - Allows image-classification models to be trained with as few as 50-100 images
  - Eliminates need for GPU-equipped HPC clusters (train on a laptop)
- Pretrained CNNs available from Microsoft, Google, and others
  - Frequently made available through GitHub
- Keras includes popular pretrained CNNs

How Transfer Learning Works



Pretrained CNNs Included with Keras

Model	Accuracy	Versions
DenseNet	Up to 93.6%	DenseNet121, DenseNet169, and DenseNet201
EfficientNet	N/A	EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, EfficientNetB5, EfficientNetB6, and EfficientNetB7
Inception	Up to 95.3%	InceptionV3 and InceptionResNetV2
MobileNet	Up to 90.1%	MobileNet and MobileNetV2
NASNet	Up to 96.0%	NASNetMobile and NASNetLarge
ResNet	Up to 94.2%	ResNet50, ResNet50V2, ResNet101, ResNet101V2, ResNet152, and ResNet152V2
VGG	Up to 90.1%	VGG16 and VGG19
Xception	94.5%	Xception

https://keras.io/api/applications/

Using a Pretrained CNN to Classify Images

```
# Instantiate the model
model = ResNet50V2(weights='imagenet', input_shape=(224, 224, 3))

# Load and preprocess the image to be classified
x = image.load_img('IMAGE_PATH', target_size=(224, 224))
x = image.img_to_array(x)
x = np.expand_dims(x, axis=0) # Converts (224, 224, 3) to (1, 224, 224, 3)
x = preprocess_input(x)/255 # Unique to each pretrained CNN

# Use the model to classify the image
predictions = model.predict(x)
print(decode_predictions(predictions, top=5)[0])
```

"Retraining" a Pretrained CNN (Transfer Learning)

```
# Instantiate the model (minus the classification layers) and freeze the layers
base_model = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

# Add and train new classification layers
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=8)
```

Fast Transfer Learning

```
# Instantiate the model (minus the classification layers)
base_model = ResNet50V2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Run the images through the base model
x = base_model.predict(x)

# Build a network for classification and train it with the output
model = Sequential()
model.add(Flatten(input_shape.x.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=8)
```

Demo

Transfer Learning

Get the Code

https://github.com/jeffprosise/Deep-Learning