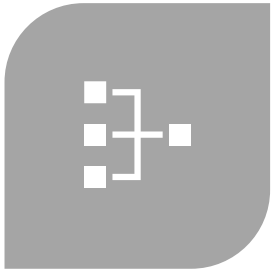
Building a Convolutional Neural Network from Scratch Using Numpy

Caroline Clark

Development Process

Research



Planning

Modular Development

Assemble and Test

Training a Neural Network

1. Sample data
2. Compute forward calculations and calculate loss
3. Compute backward derivative calculations
4. Update gradients
5. Repeat

Getting a Neural Network to Converge

...have enough data

...have enough compute

...use appropriate activation functions

...choose the right architecture

...initialize weights properly

...choose good hyperparameters

…preprocess data

...normalize activations

...regularize

...use good optimizer

...wait

CNNs in Theory and in Practice

Theory

CONV => RELU => POOL => FLATTEN => DENSE => SOFTMAX

Practice

→ Create layers using Python classes
→ Code math for each layer's forward calculation
→ Store required variables in cache to be used during backprop
→ Code math for each layer's derivative
→ Use up the cached variables during backpropagation

Convolutions are Element-Wise Multiplication and Summation

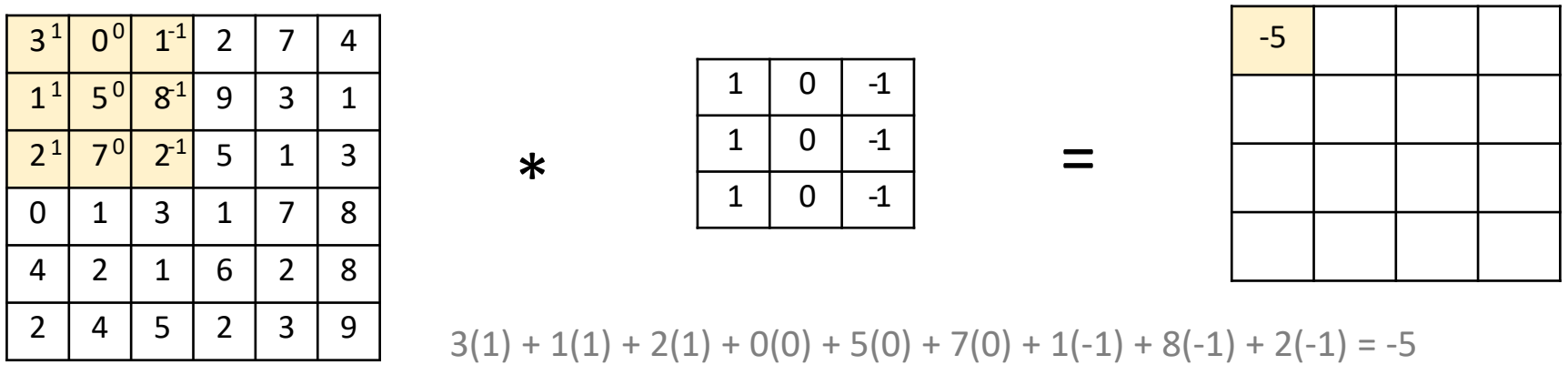
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

1	0	-1
1	0	-1
1	0	-1

=

Convolutions are Element-Wise Multiplication and Summation



3(1) + 1(1) + 2(1) + 0(0) + 5(0) + 7(0) + 1(-1) + 8(-1) + 2(-1) = -5

Convolutions are Element-Wise Multiplication and Summation

3^{1}	0^{0}	1^{-1}	2	7	4
1^{1}	5^{0}	8^{-1}	9	3	1
2^{1}	7^{0}	2^{-1}	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

1	0	-1
1	0	-1
1	0	-1

=

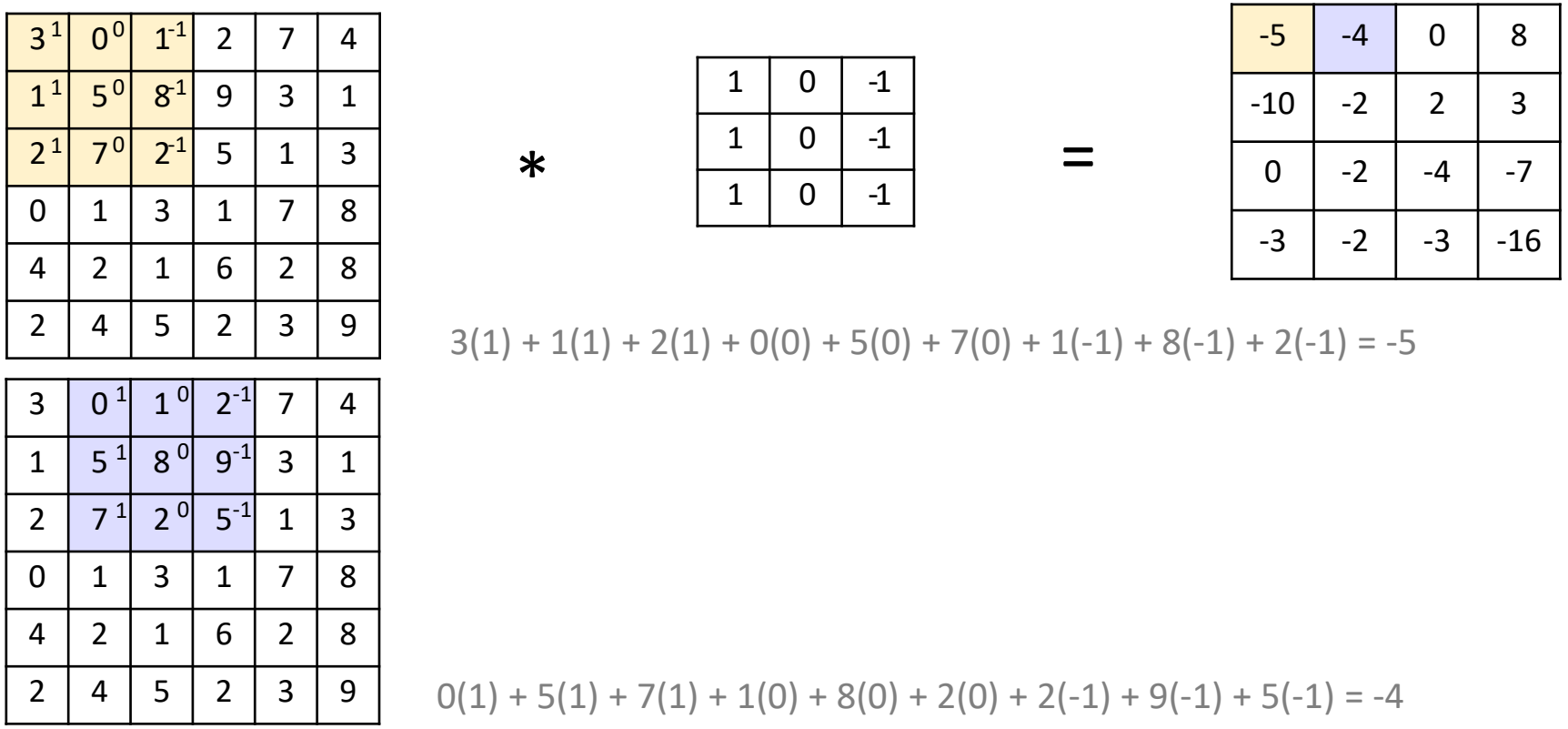
-5	-4		

$3(1) + 1(1) + 2(1) + 0(0) + 5(0) + 7(0) + 1(-1) + 8(-1) + 2(-1) = -5$

3	0^{1}	1^{0}	2^{-1}	7	4
1	5^{1}	8^{0}	9^{-1}	3	1
2	7^{1}	2^{0}	5^{-1}	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$0(1) + 5(1) + 7(1) + 1(0) + 8(0) + 2(0) + 2(-1) + 9(-1) + 5(-1) = -4$

Convolutions are Element-Wise Multiplication and Summation



3(1) + 1(1) + 2(1) + 0(0) + 5(0) + 7(0) + 1(-1) + 8(-1) + 2(-1) = -5

0(1) + 5(1) + 7(1) + 1(0) + 8(0) + 2(0) + 2(-1) + 9(-1) + 5(-1) = -4

…and so on until you've covered the input volume

Convolutions are Element-Wise Multiplication and Summation

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

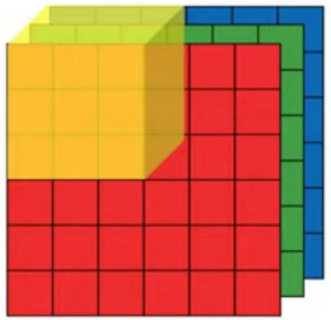
*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

In 3D



Forward Propagation

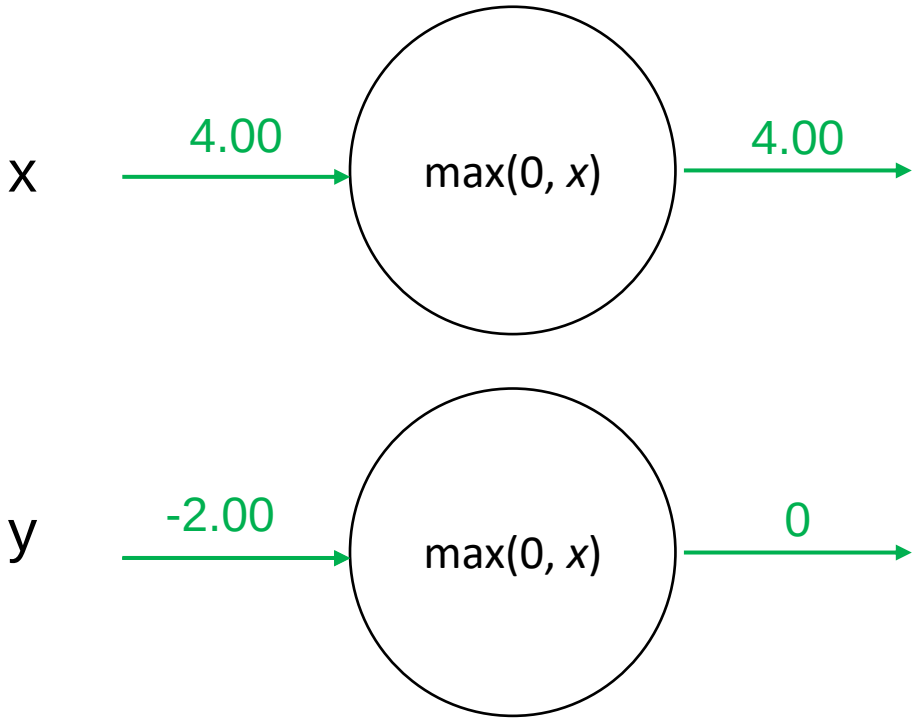
Convolution Output Dimensions | *n* x *n* image, *f* x *f* filter, padding *p*, stride *s*

$$\left[\frac{n+2p-f}{s}+1\right] \text{ x } \left[\frac{n+2p-f}{s}+1\right] \text{ x num\_filters}$$

Backward Propagation

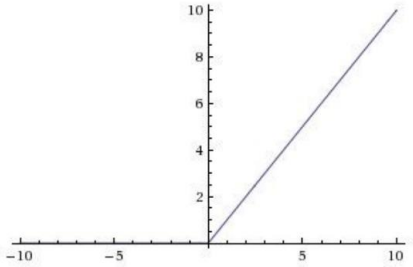
CONV layer parameters W and b are updated after each pass

ReLU Acts as a Differentiable Gate

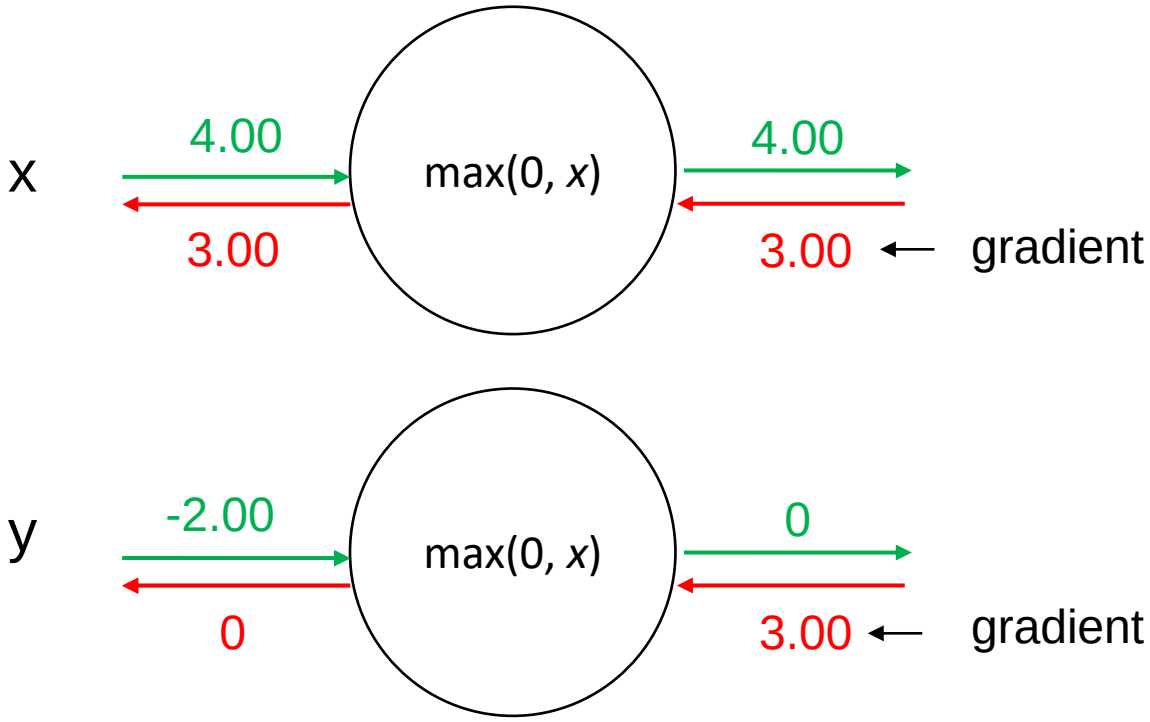


Rectified Linear Unit

$f(x) = \max(0, x)$

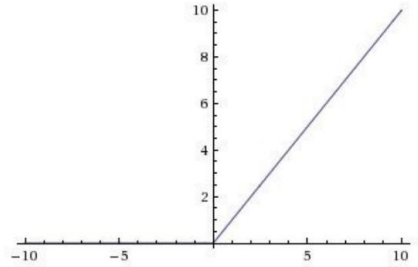


ReLU Acts as a Differentiable Gate



Rectified Linear Unit

$f(x) = \max(0, x)$



Forward Propagation

```
np.where(Z < 0, 0, Z)
```

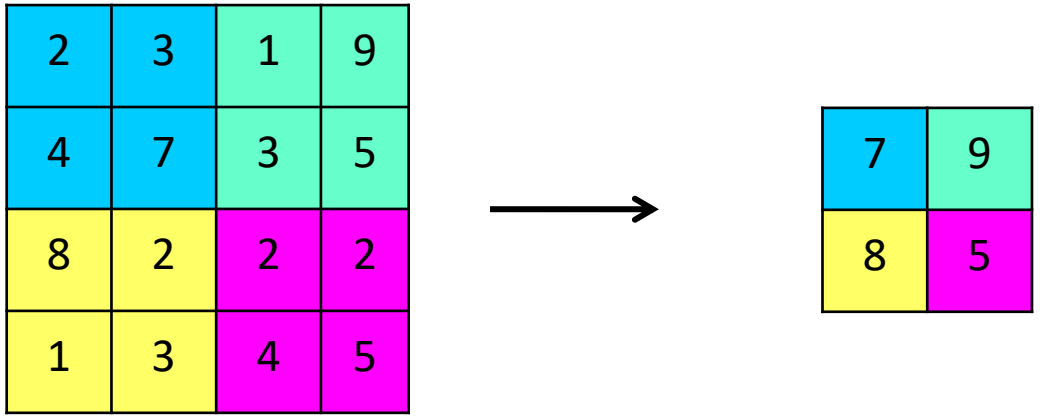
Backward Propagation

```
dA * np.where(Z < 0, 0, 1)
```

Max Pooling Shrinks Height and Width, Preserves Strongest Activations

Forward Propagation

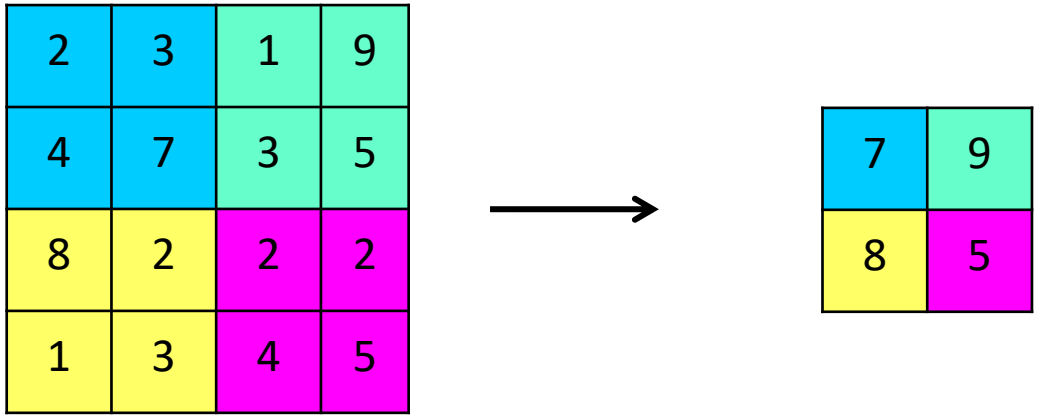
Pass max value of region defined by *f* filter size and stride *s*



Max Pooling Shrinks Height and Width, Preserves Strongest Activations

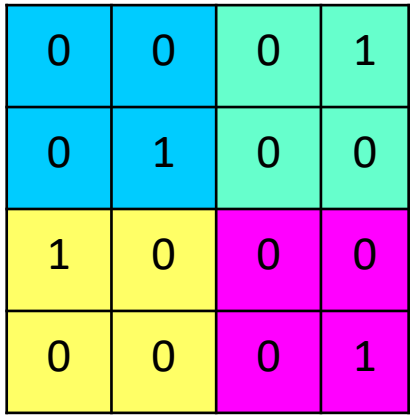
Forward Propagation

Pass max value of region defined by *f* filter size and stride *s*

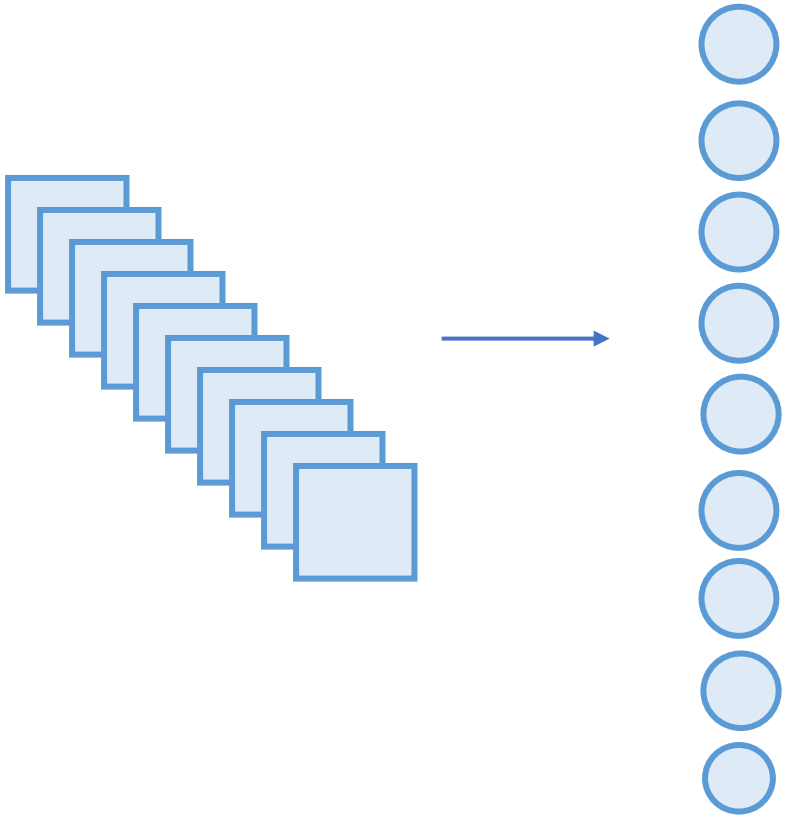


Backward Propagation

Create Boolean mask of max value to determine where gradients flow



Flattening Required Before Dense Layers



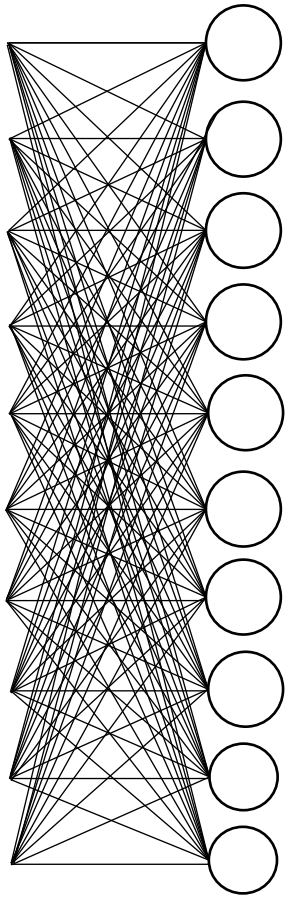
Forward Propagation

```
Z.reshape(-1, shape[0]
```

Backward Propagation

```
Z =Z.T
Z.reshape(shape)
```

Dense Layers in CNNs are Typical Fully-Connected Layers



Forward Propagation

Output values of given layer L:

$$Z^{[L]} = W^{[L]} A^{[L-1]} + B^{[L]}$$

Backward Propagation

```
dW = np.dot(dA, X.T) / batch_size
db = np.sum(dA, axis=1, keepdims=True)
```

Dense layer parameters W and b are updated after each pass

* The last dense layer must have same number of units as you have classes

Softmax Activation Layer used for Classification of 2+ Classes

Softmax interprets score vector values as **unnormalized log probabilities** of each class

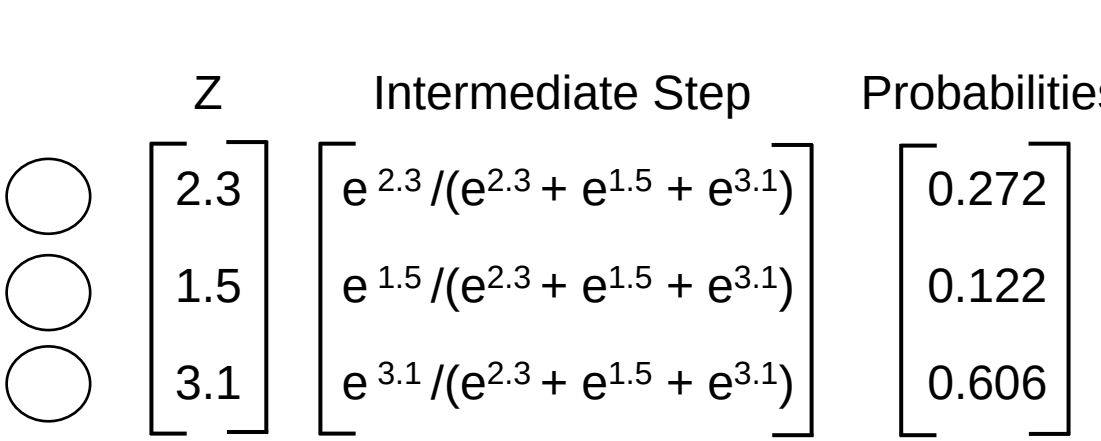
To get probabilities of each class, exponentiate and normalize

	Z	Intermediate Step	Probabilities
◯	2.3	$e^{2.3}/(e^{2.3} + e^{1.5} + e^{3.1})$	0.272
◯	1.5	$e^{1.5}/(e^{2.3} + e^{1.5} + e^{3.1})$	0.122
◯	3.1	$e^{3.1}/(e^{2.3} + e^{1.5} + e^{3.1})$	0.606

Softmax Activation Layer used for Classification of 2+ Classes

Softmax interprets score vector values as **unnormalized log probabilities** of each class

To get probabilities of each class, exponentiate and normalize



$$\frac{e_k^s}{\sum_j e_j^s}$$

Forward Propagation

```
expZ = np.exp(Z - np.max(Z))
expZ / np.sum(expZ, axis=0)
```

Backward Propagation

```
dA * Z * (1 - Z)
```

Softmax to Categorical Cross-Entropy Loss

Maximize probability of correct class by minimizing negative log likelihood of correct class

Loss for one example:

$$L_i = -log \frac{e^s_{y_i}}{\sum_j e^s_j}$$

Cost over training set:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Softmax to Categorical Cross-Entropy Loss

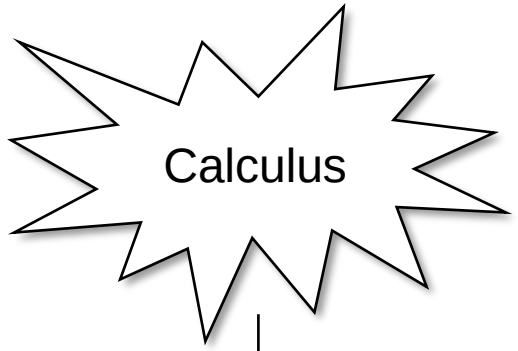
Maximize probability of correct class by minimizing negative log likelihood of correct class

Loss for one example:

$$L_i = -log\frac{e^s_{y_i}}{\sum_j e^s_j}$$

Cost over training set:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$



$$dZ^{[L]} = \hat{Y} - Y = \frac{\partial J}{\partial Z^{[L]}}$$

Now we can kick off backprop

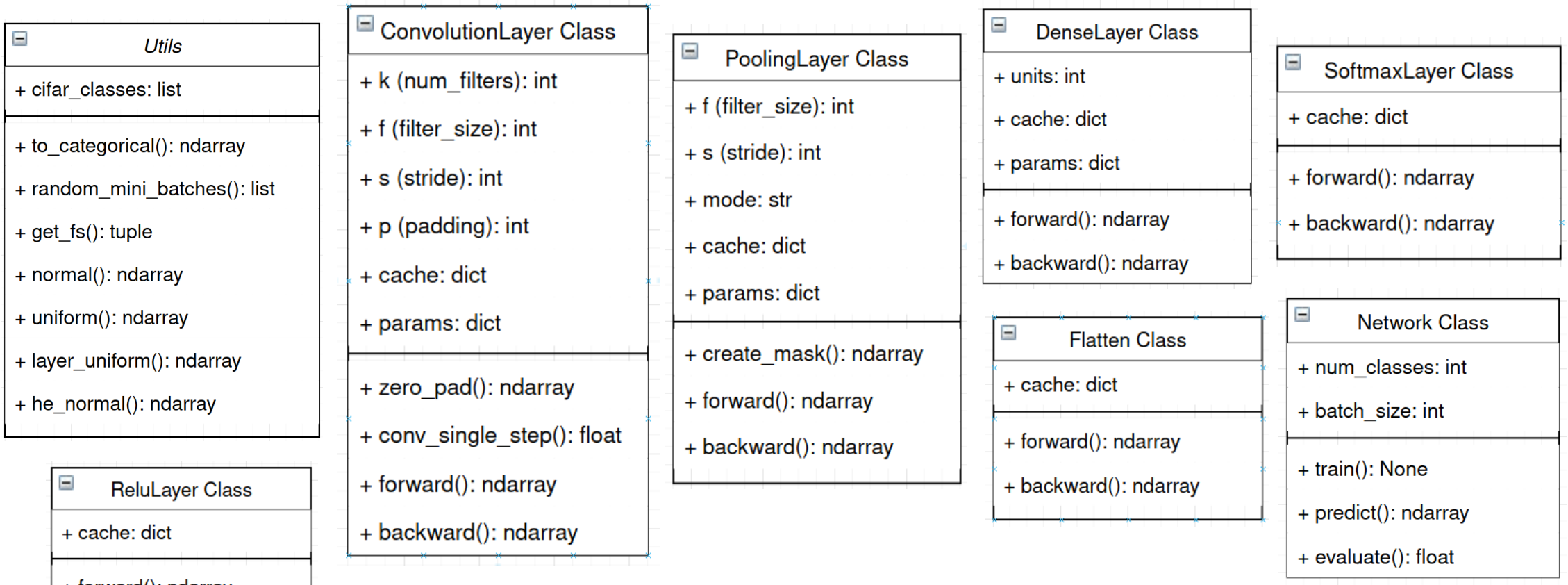
Partial derivative of cost function with respect to $Z^{[L]}$

Stanford CS231n: Convolutional Neural Networks for Visual Recognition.
Deeplearning.ai: Deep Learning Specialization, Andrew Ng.

Example Forward and Backward Loop

Layer	Input Shape	Output Shape
Convolution Forward *f=3, f_size=3, p=1, s=1, m=256*	(256, 32, 32, 3)	(256, 32, 32, 3)
Pooling Forward *f_size=2, s=2*	(256, 32, 32, 3)	(256, 16, 16, 3)
Flatten Forward	(256, 16, 16, 3)	(768, 256)
Dense Forward	(768, 256)	(10, 256)
Softmax Forward	(10, 256)	(10, 256)
Softmax Backward	(10, 256)	(10, 256)
Dense Backward	(10, 256)	(768, 256)
Flatten Backward	(768, 256)	(256, 16, 16, 3)
Pooling Backward	(256, 16, 16, 3)	(256, 32, 32, 3)
Convolution Backward	(256, 32, 32, 3)	(256, 32, 32, 3)

Python Package Diagrams



CNN Library Status

- Model trains!
- Some bugs with scaling
- Running slowly on CPU
- Next Steps
  - Debug scaling issue
  - Publish to PyPi
- Future
  - Build out optimizers as class
  - Regularization, Batch Normalization layers
  - Port to Numba to leverage GPUs
  - Generalize to other classification problems

```python
# Import the required modules.
from hive_ml.layers.activation import ReluLayer
from hive_ml.layers.convolution import ConvolutionLayer
from hive_ml.layers.pooling import PoolLayer
from hive_ml.layers.flatten import Flatten
from hive_ml.layers.dense import DenseLayer
from hive_ml.layers.loss import SoftmaxLayer
from hive_ml.network import Model
```

```python
# Instantiate a model.
model = Model(
    ConvolutionLayer(filters=3, filter_size=3, padding=1, stride=1),
    ReluLayer(),
    PoolLayer(filter_size=2, stride=2, mode='max'),
    Flatten(),
    DenseLayer(units=100),
    ReluLayer(),
    DenseLayer(units=10),
    SoftmaxLayer()
)
```

```python
# Train the model.
model.train(X_train, y_train, 0.003, batch_size=64, epochs=1)
```

Demo