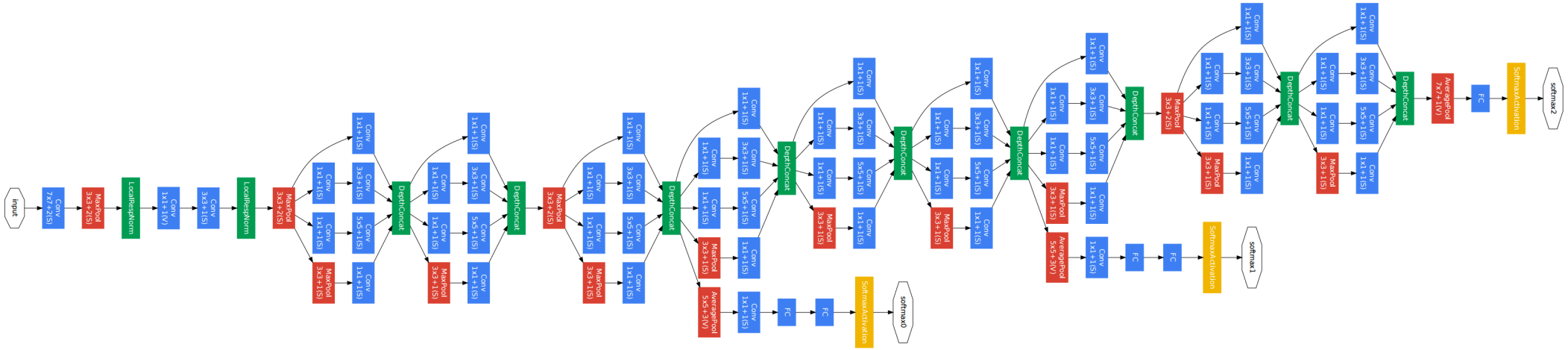
Symbolic Compute Graphs

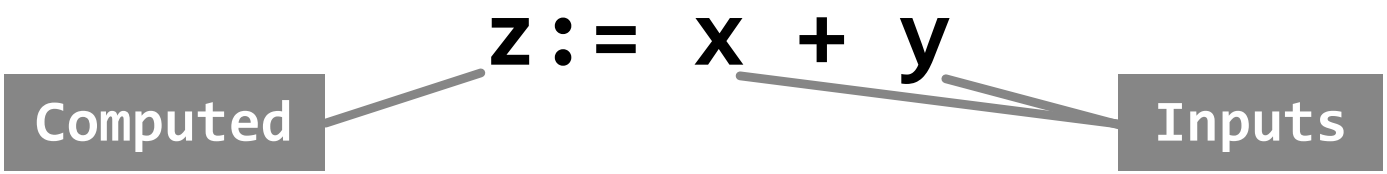
GoogLeNet

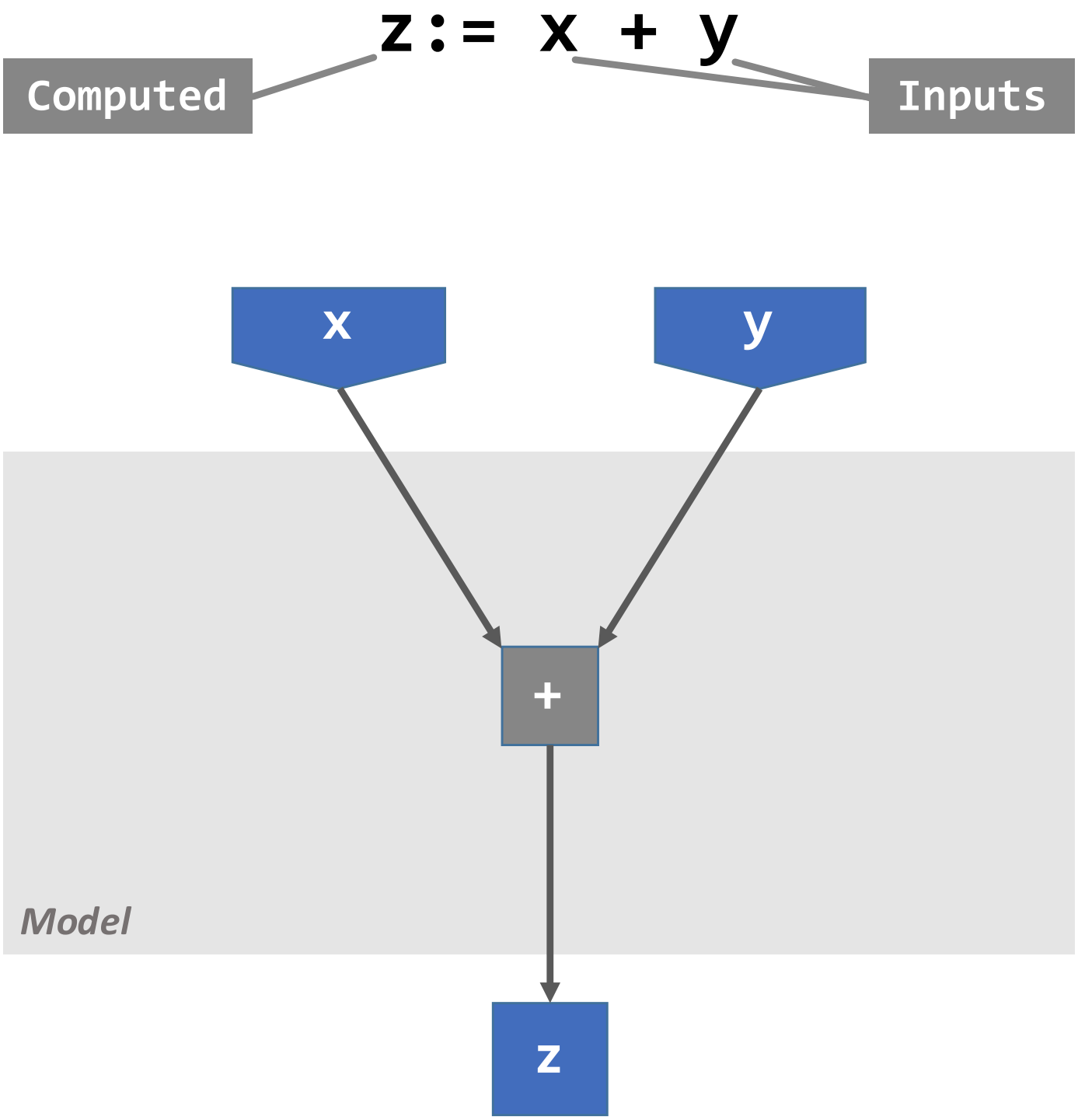

input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
Conv 1x1+1(S)
Conv 1x1+1(S)
FC
FC
SoftmaxActivation
softmax0
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
AveragePool 7x7+1(V)
FC
SoftmaxActivation
softmax2
FC
FC
SoftmaxActivation
softmax1

Compute Graphs

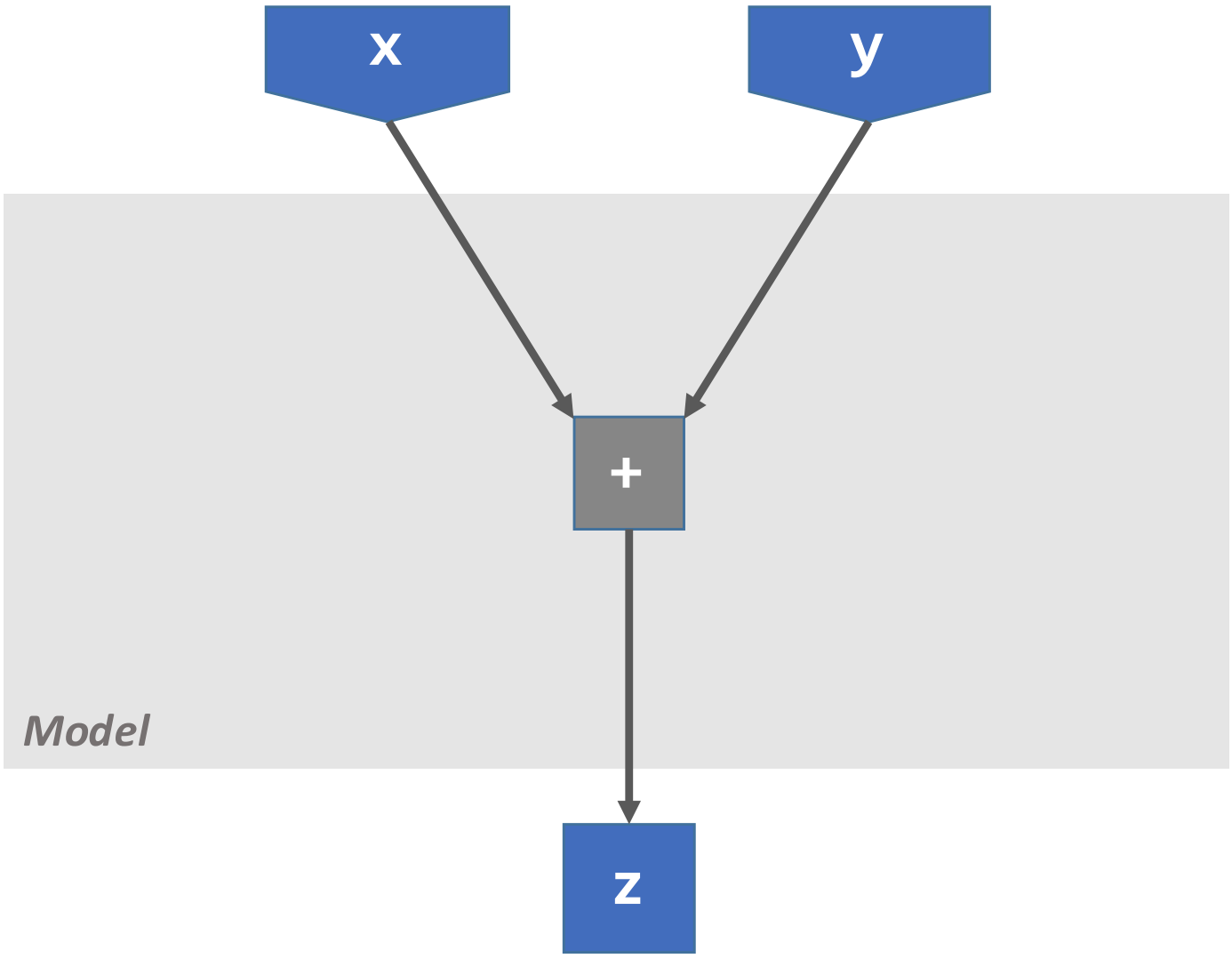
- Deep networks are static
- Math can be represented as a graph

```
z:= x + y
```


z:= x + y
Computed
Inputs

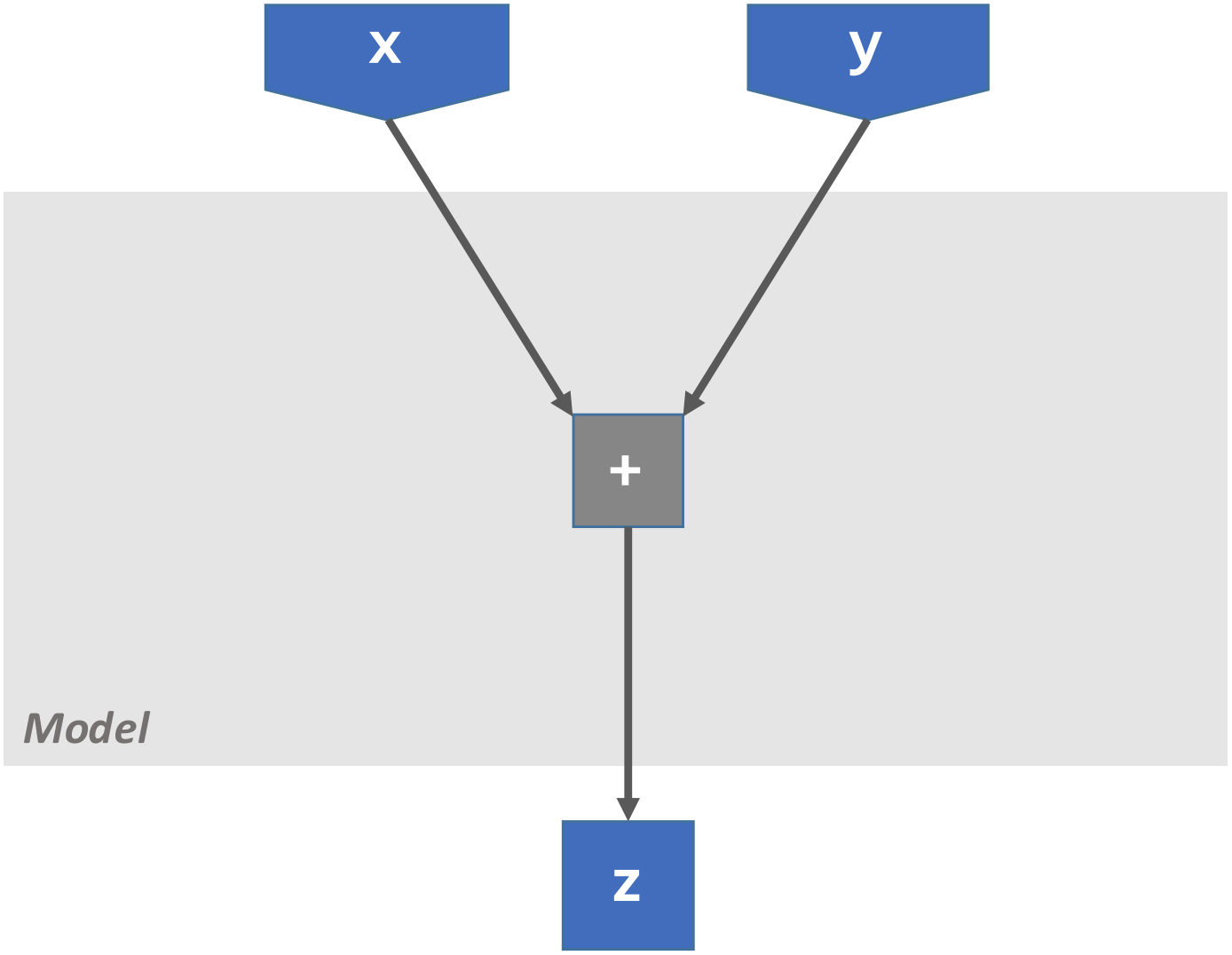

z := x + y
Computed
Inputs
x
y
+
Model
z

$$z := a*x + b*y + c$$



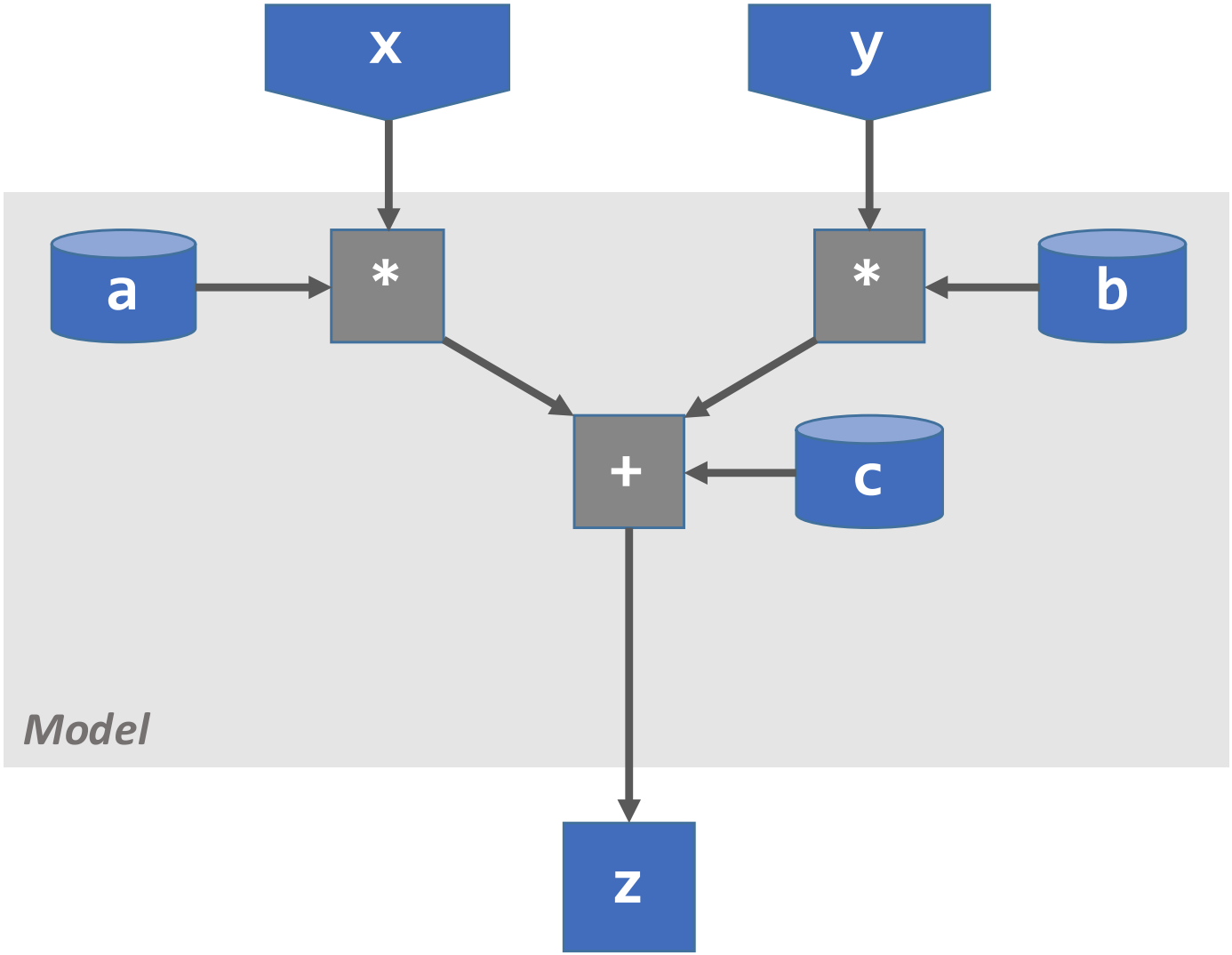
$$z := a*x + b*y + c$$

Parameters

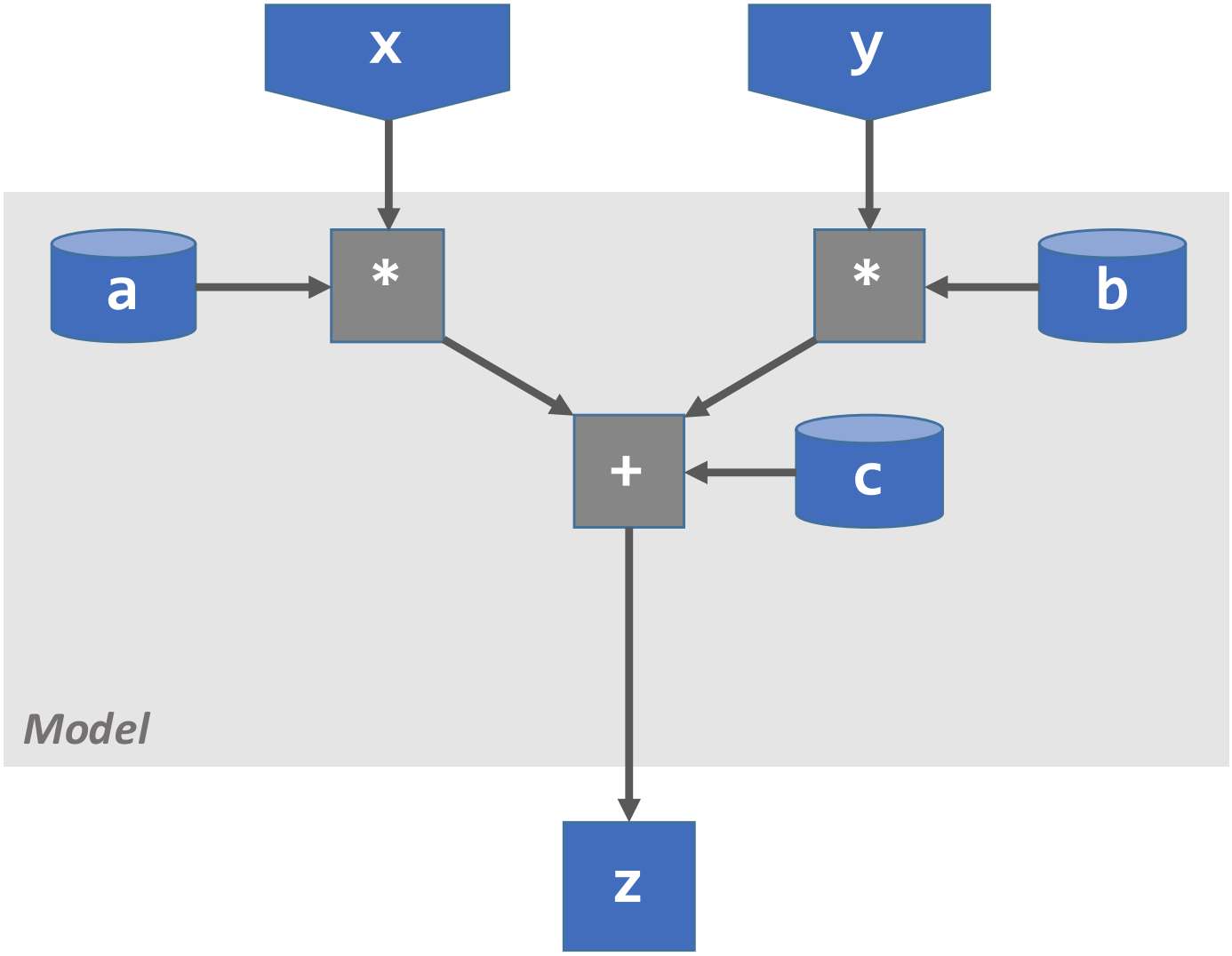


$$z := a*x + b*y + c$$

Parameters

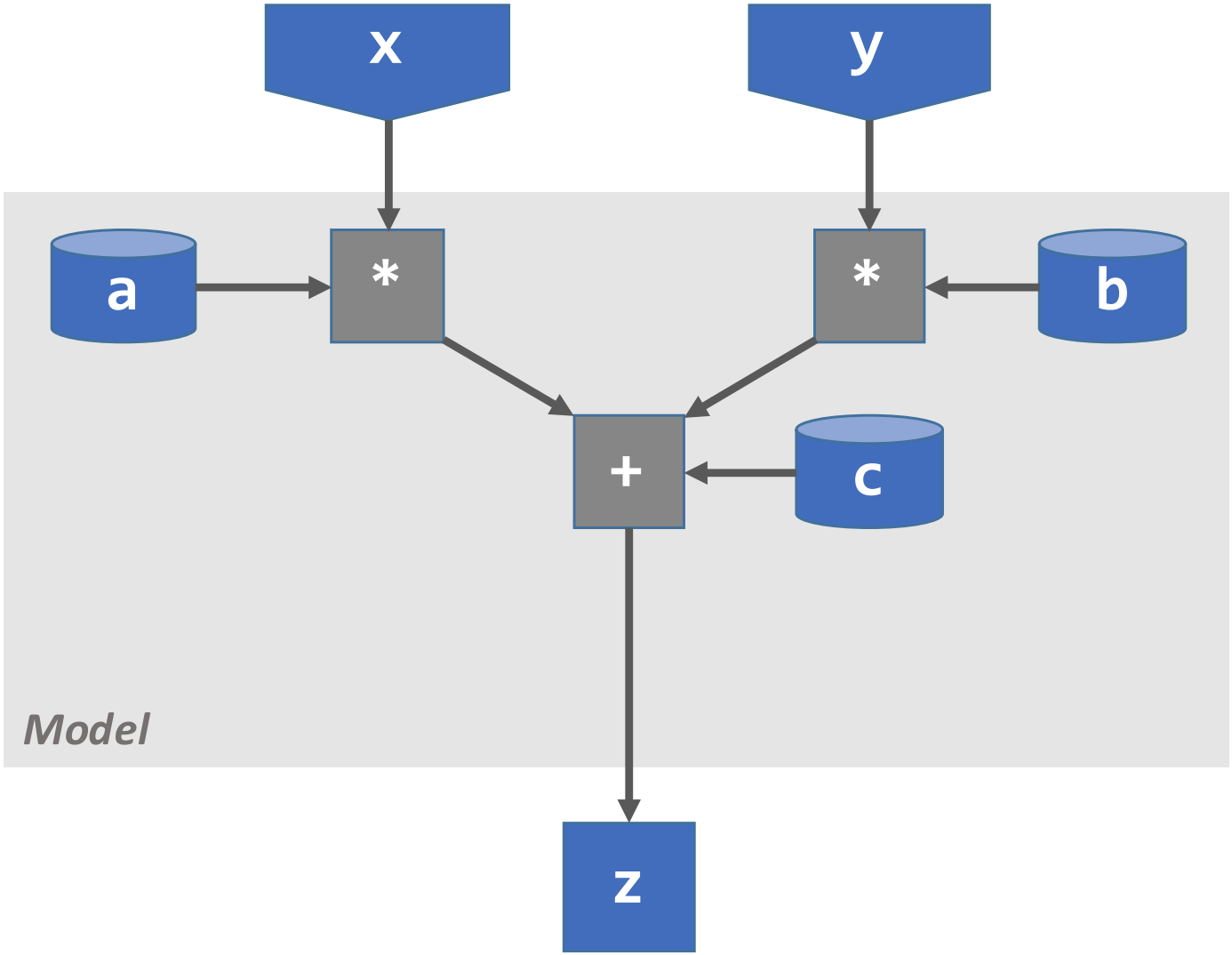


```
z:= Exp(a*x + b*y + c)
```



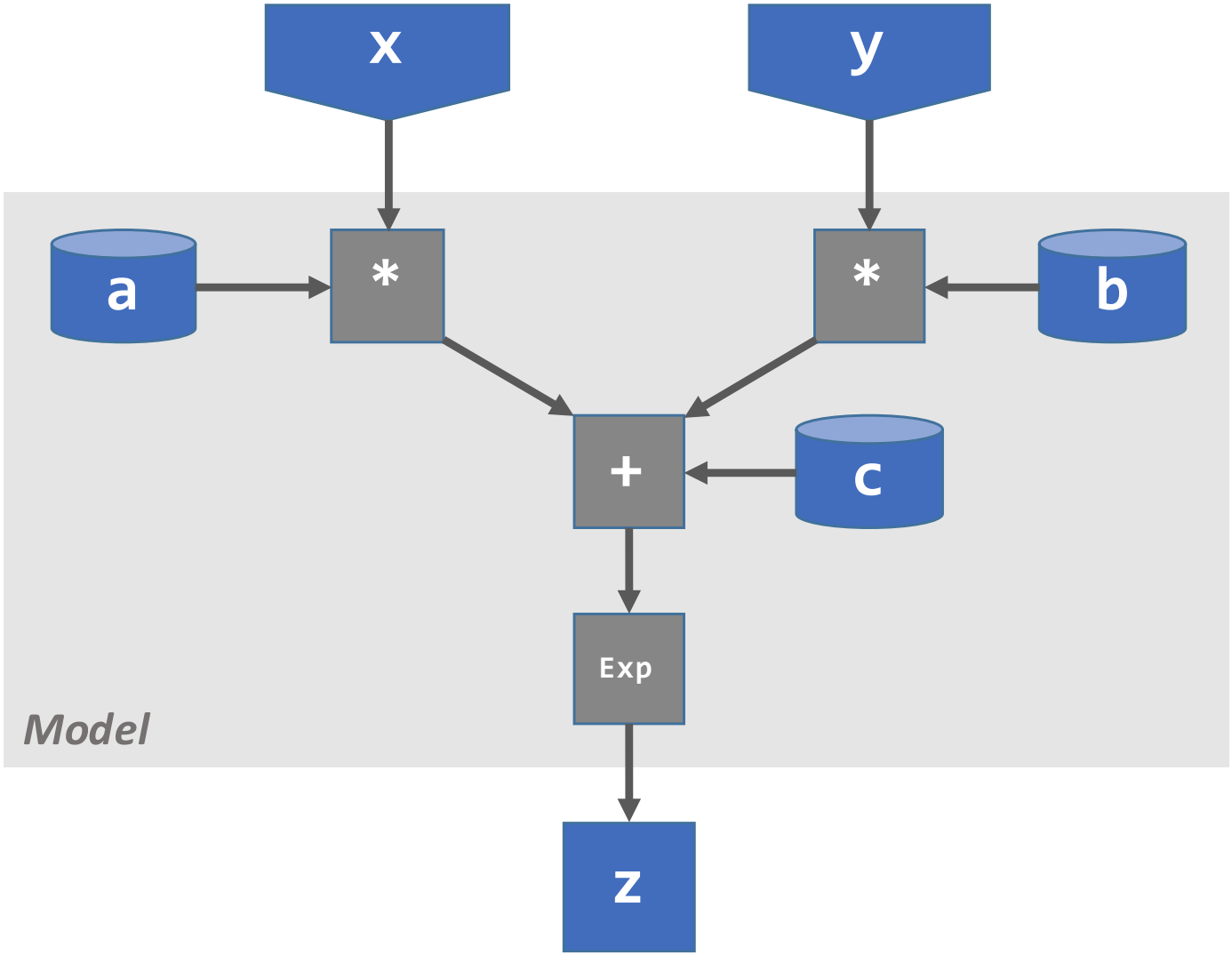
```
z:= Exp(a*x + b*y + c)
```

Function

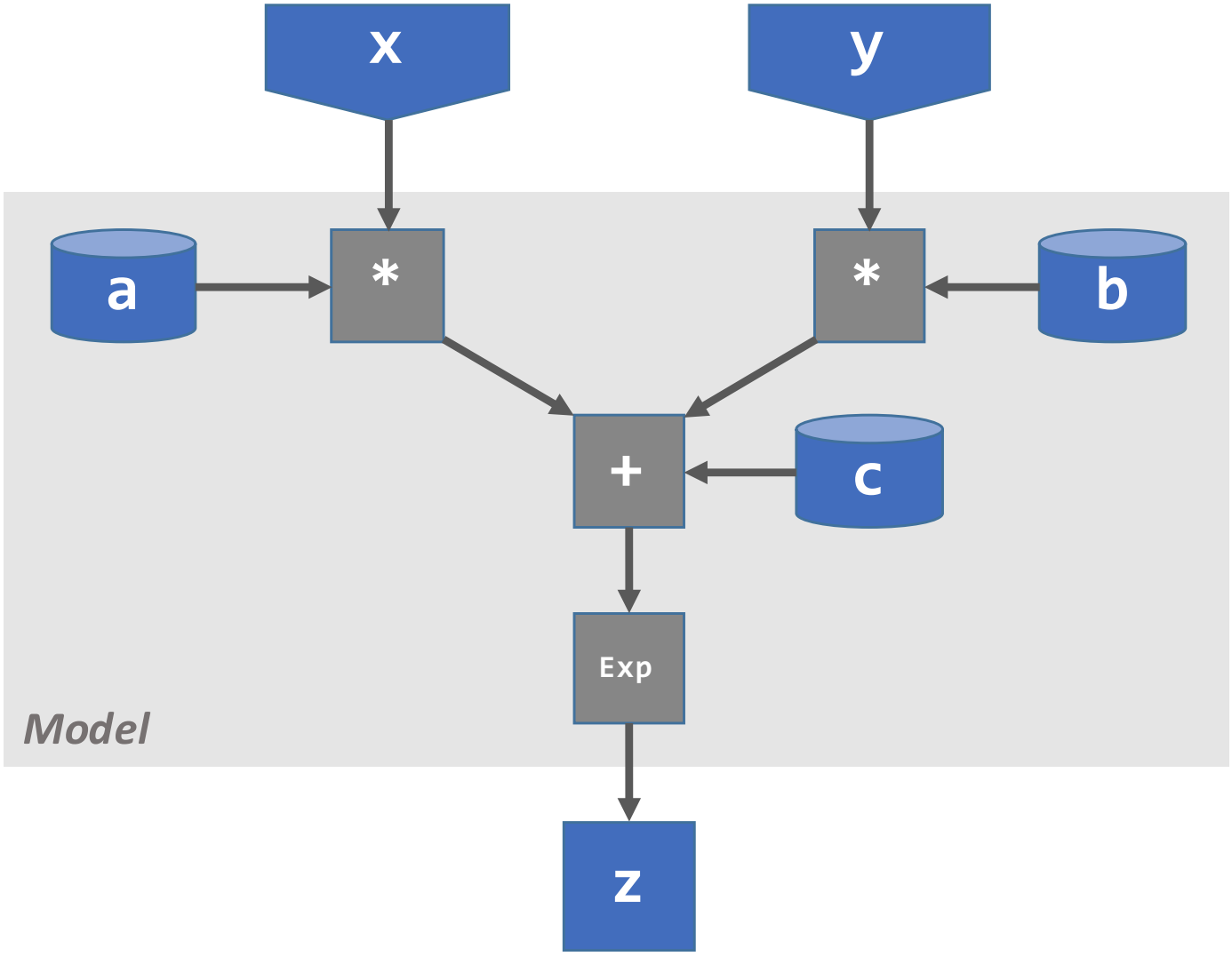


```
z:= Exp(a*x + b*y + c)
```

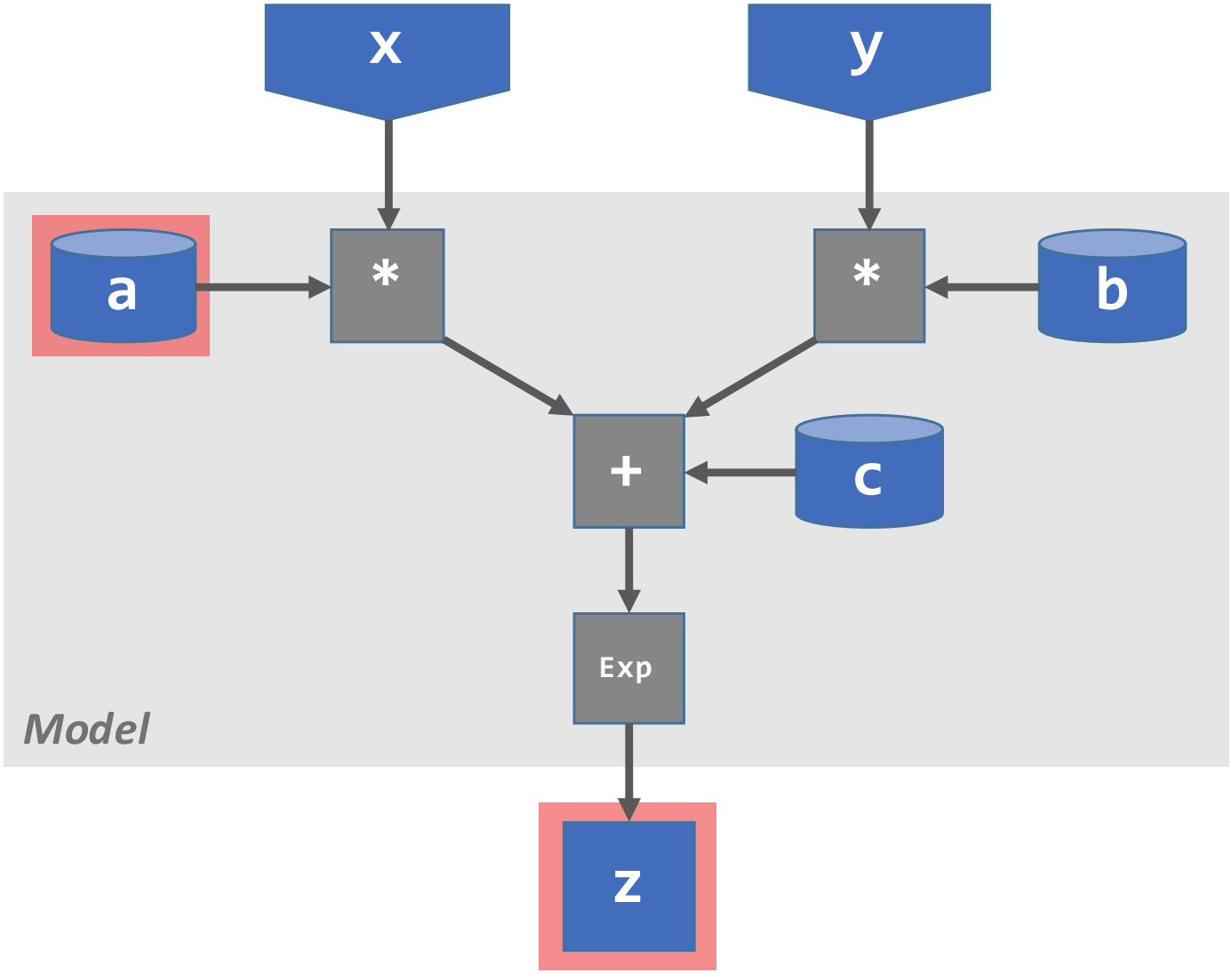
Function



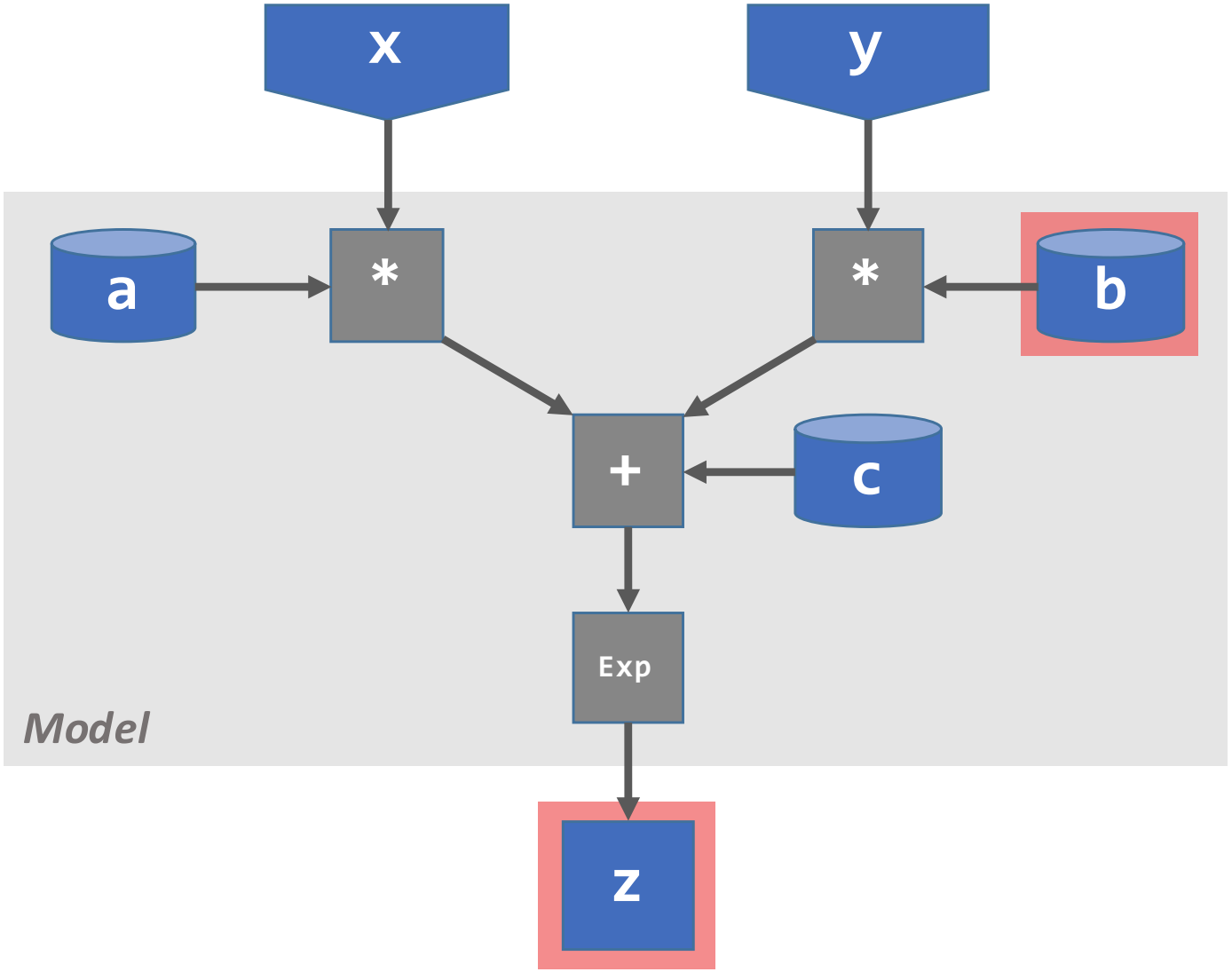
What is ∂z/∂a ?



What is ∂z/∂a ?

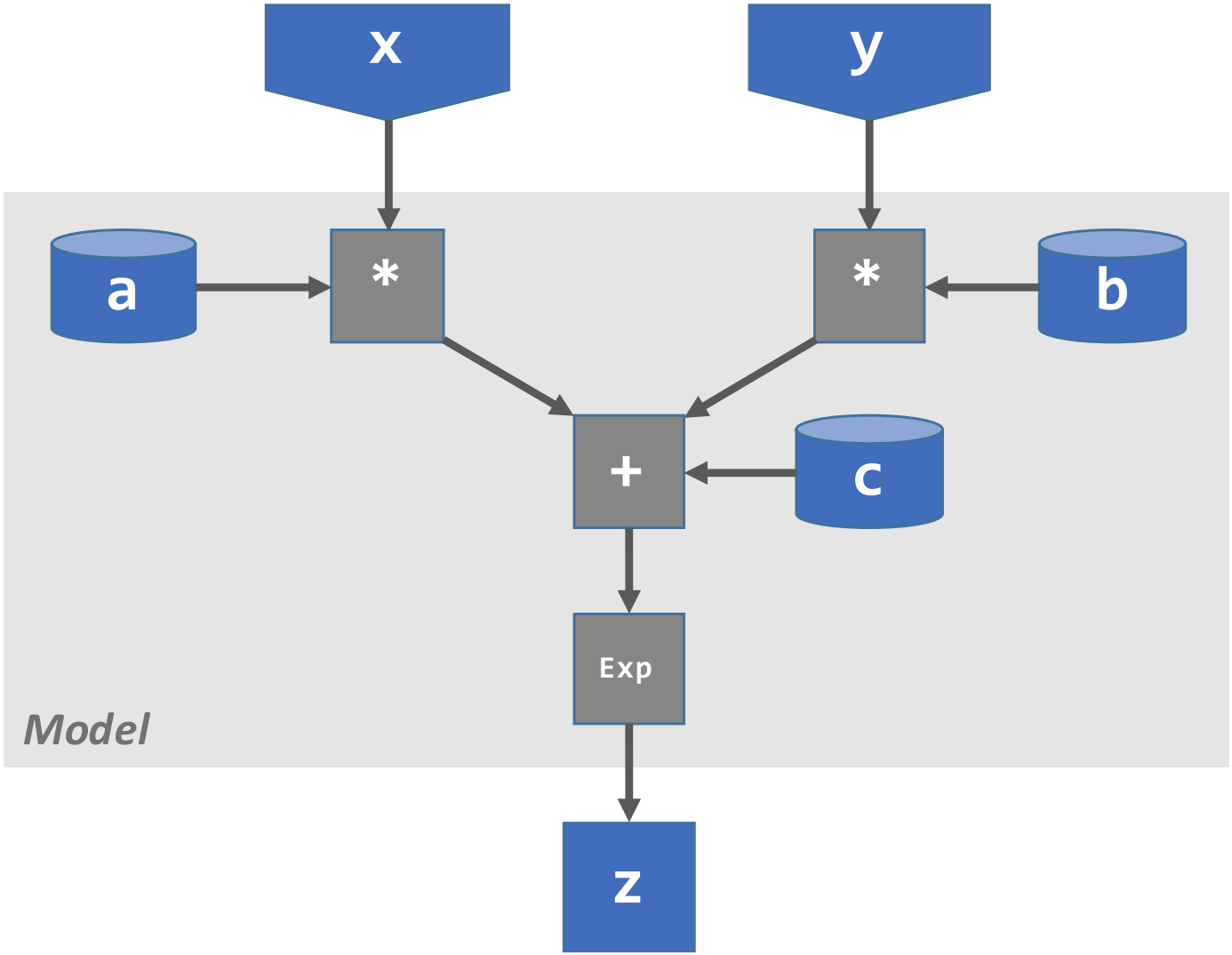


What is ∂z/∂a ? ∂z/∂y ?

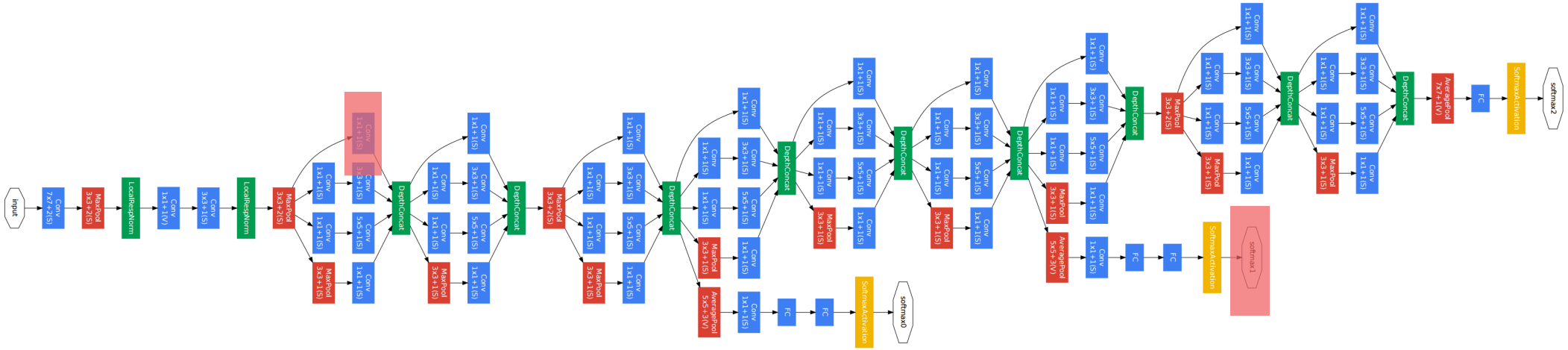


What is ∂z/∂a ? ∂z/∂y ?

How about ∂+/∂b ?

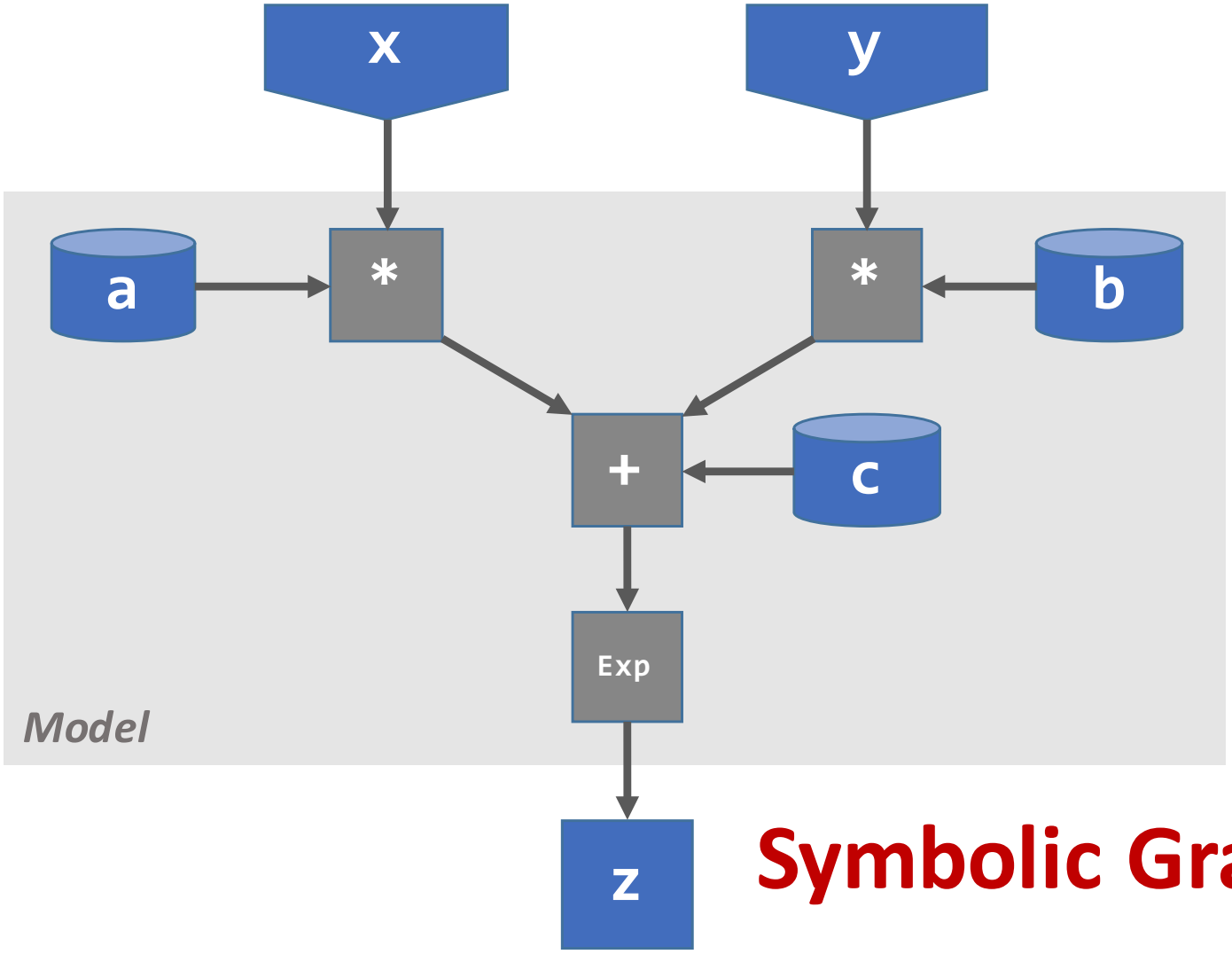


How about here?



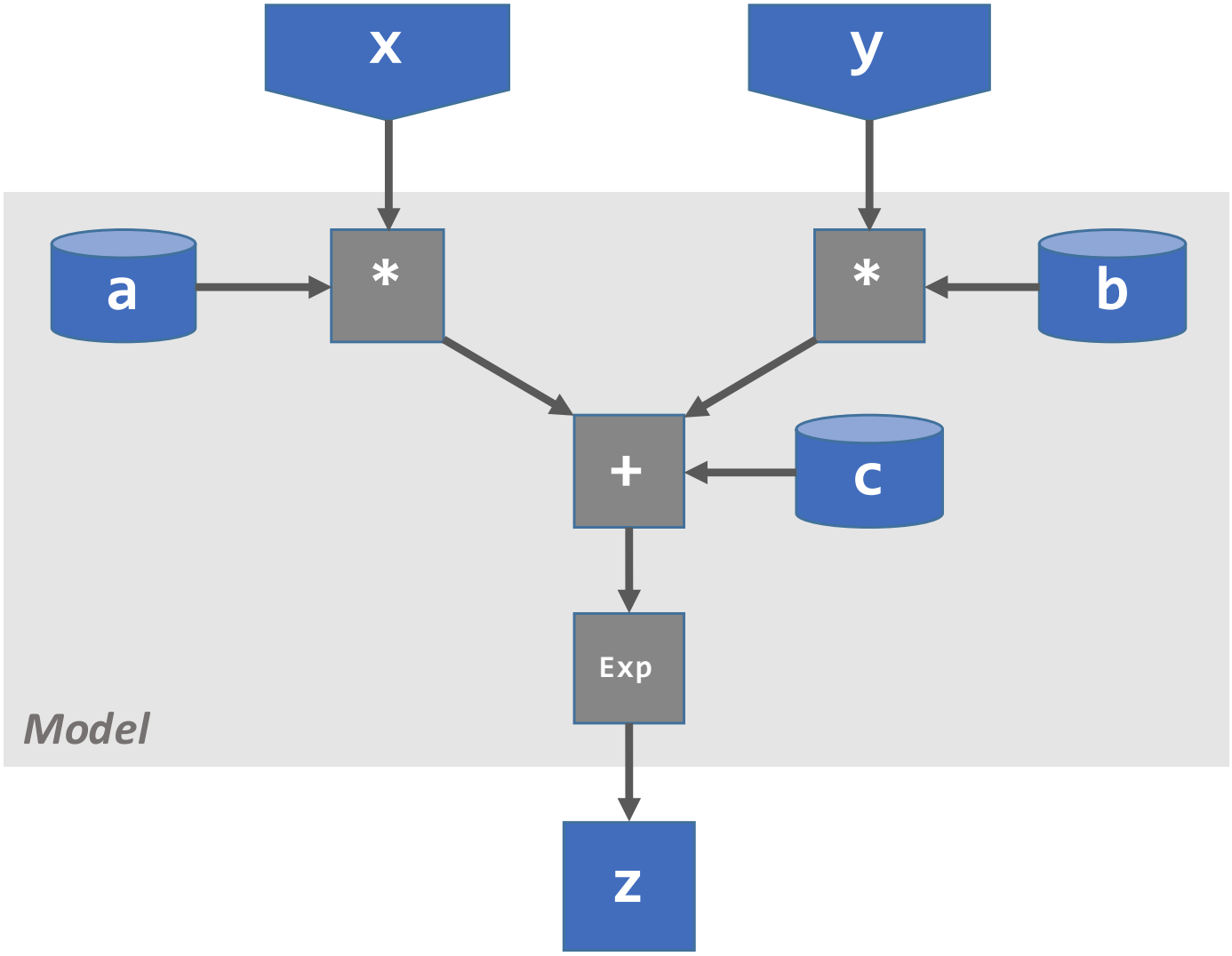
What is ∂z/∂a ? ∂z/∂y ?

How about ∂+/∂b ?

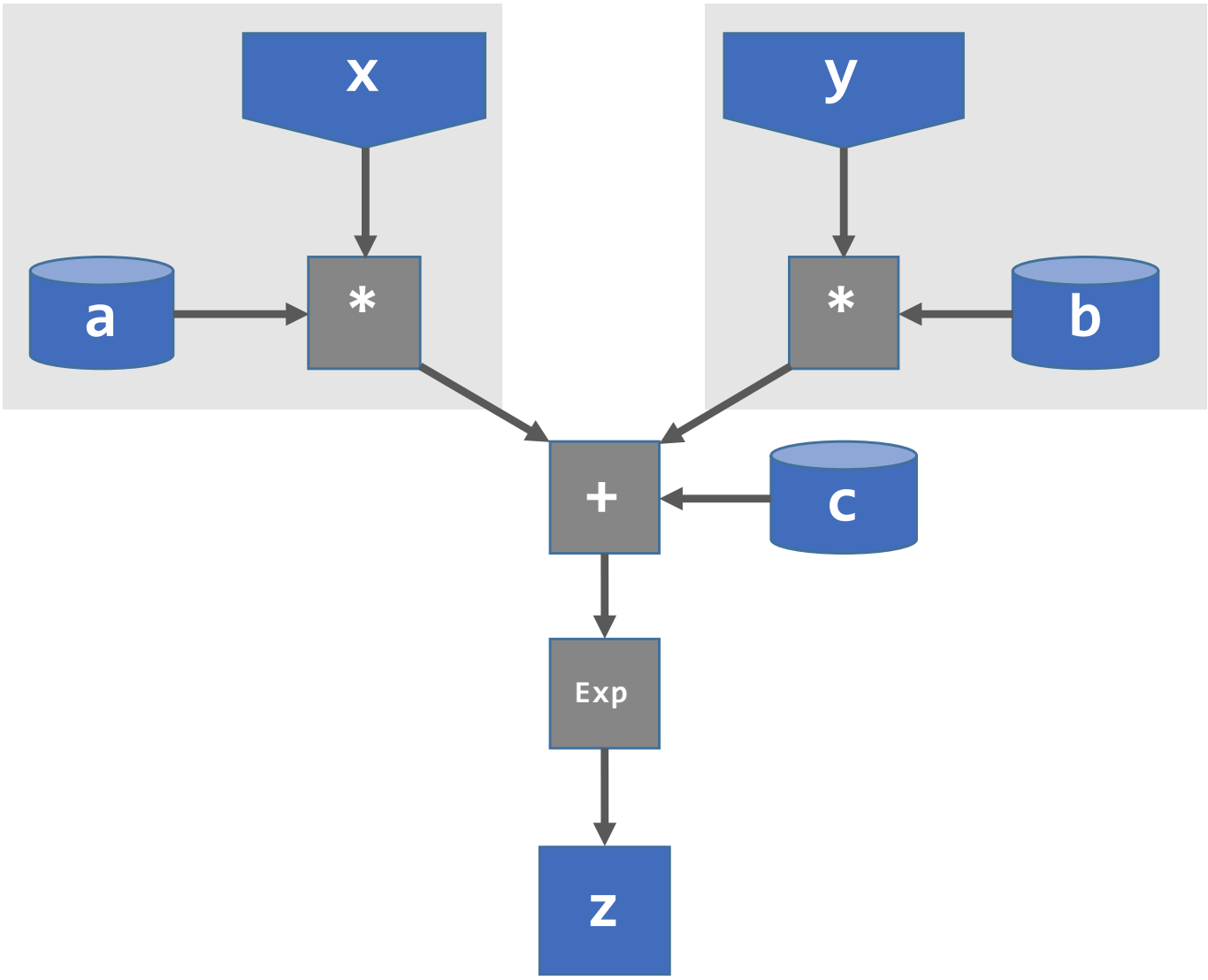


Symbolic Gradients!

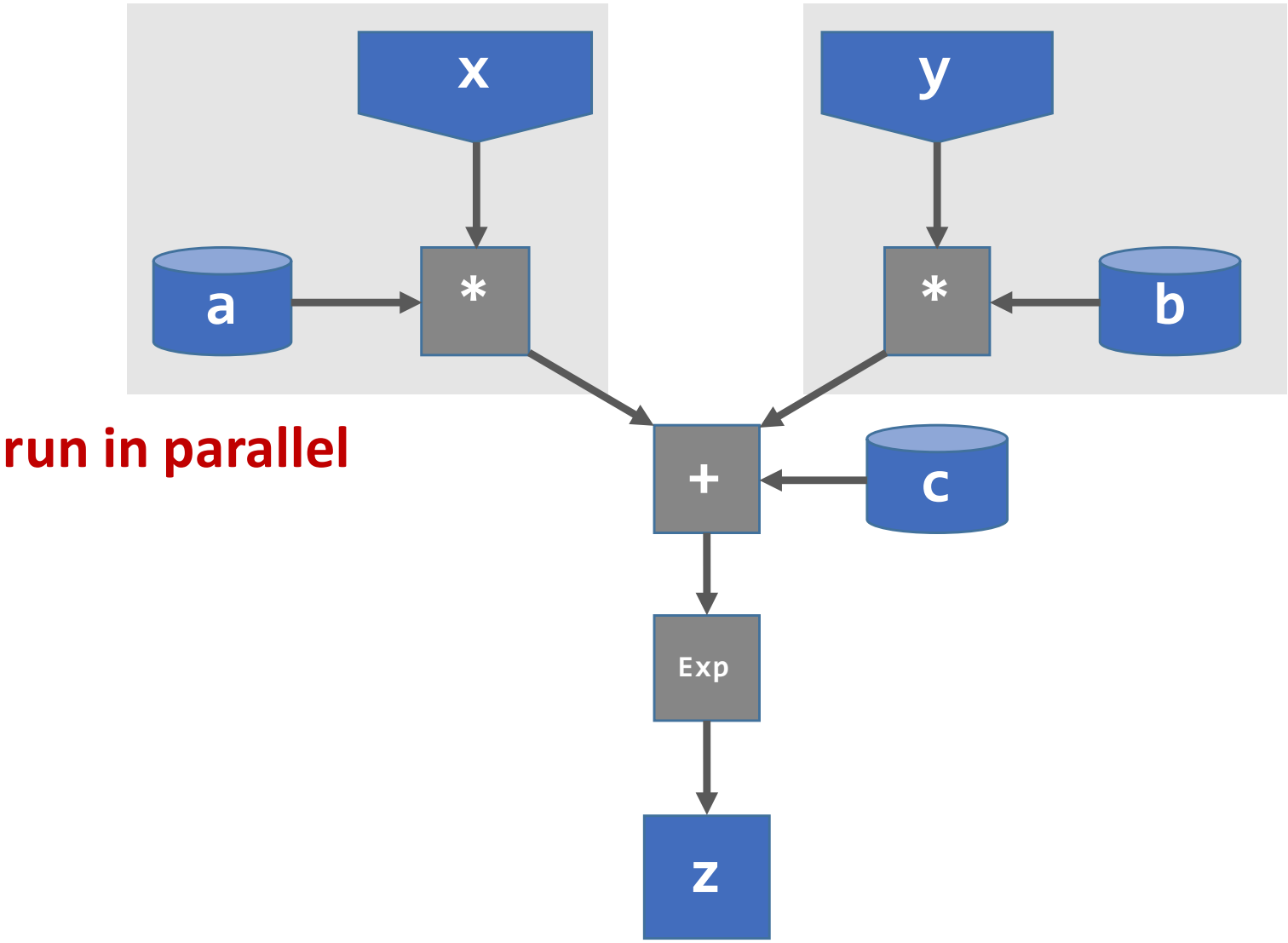
What about optimization?



What about optimization?

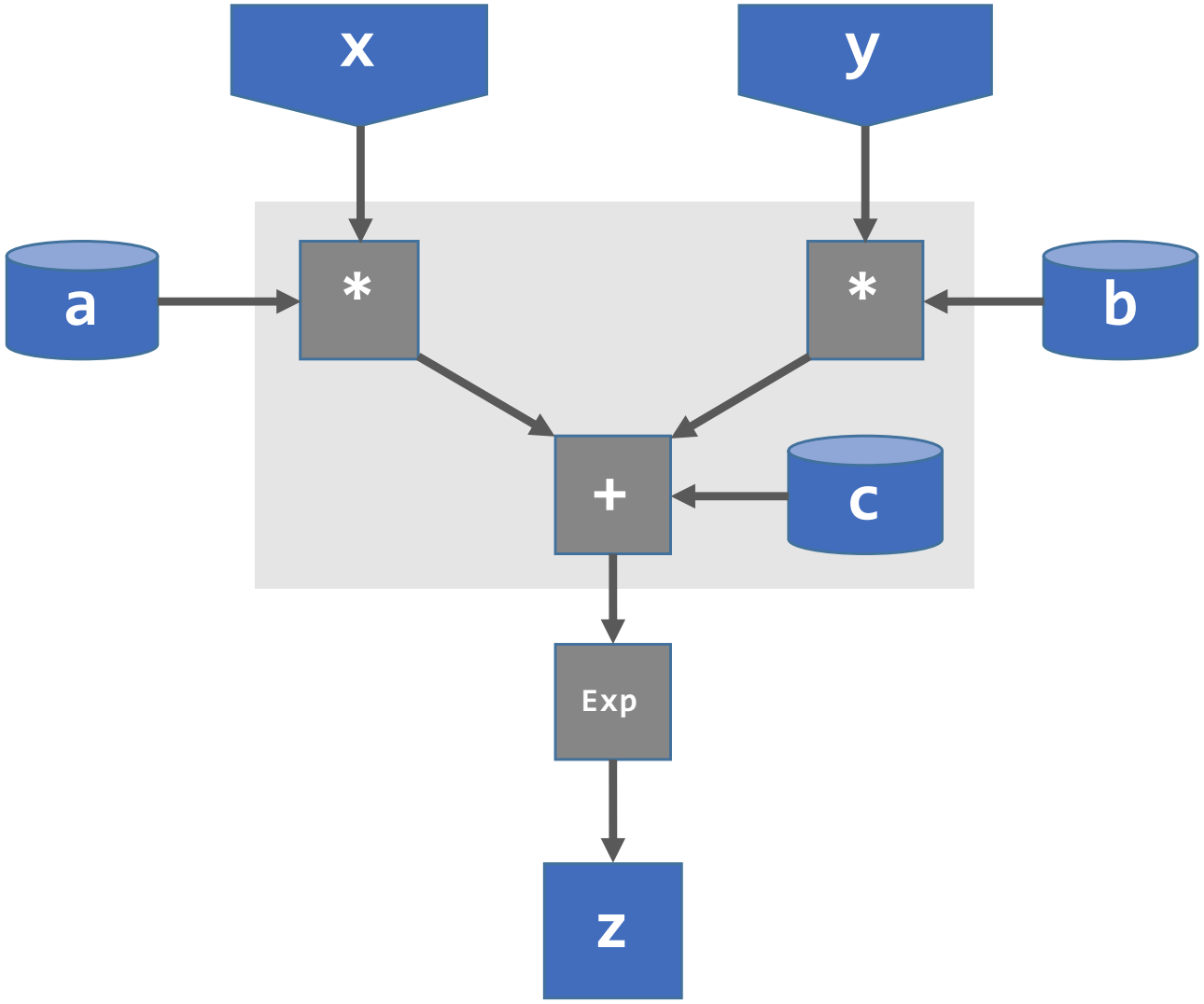


What about optimization?

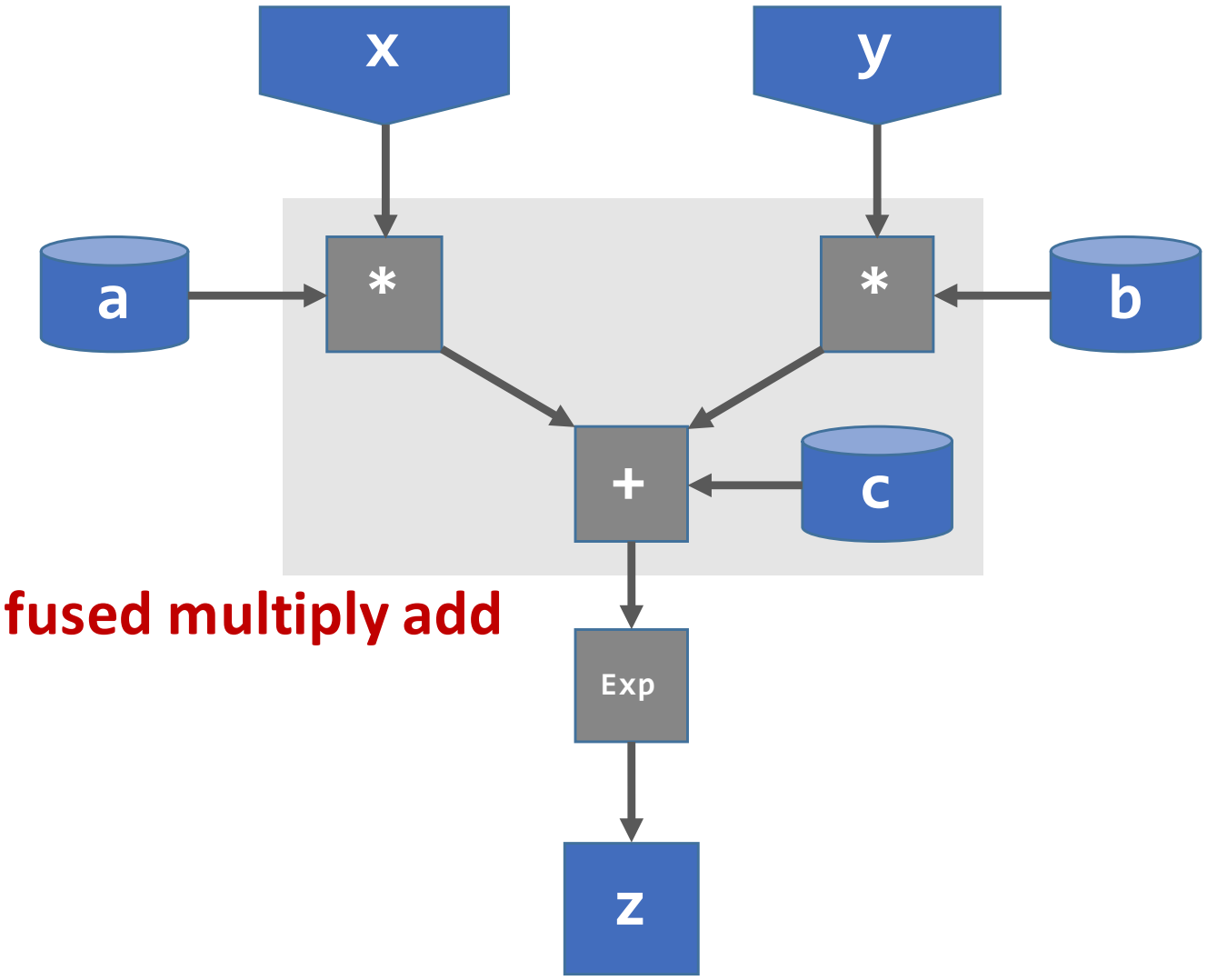


run in parallel

What about optimization?



What about optimization?



Compute Graphs

- Deep networks are static
- Math can be represented as a graph

Compute Graphs

- **Deep networks are static**
- **Math can be represented as a graph**
- **This allows to automate**
  - Differentiation
  - Optimization
  - Parallelization

How do I define a graph?

How do I define a graph?

- You don’t have to worry about this!
- Mostly transparent to the user
- Math abstracted into network layers

```
X, Y = matrix(), vector()
```

Inputs

```
X, Y = matrix(), vector()
```

```
net = InputLayer(X)
net = DenseLayer(net, num_units=64)
net = DenseLayer(net, num_units=10,
                      nonlinearity=softmax)

prediction = net
parameters = get_all_params(net)
```

Network

```
X, Y = matrix(), vector()

net = InputLayer(X)
net = DenseLayer(net, num_units=64)
net = DenseLayer(net, num_units=10,
                      nonlinearity=softmax)

prediction = net
parameters = get_all_params(net)

loss = mean(categorical_crossentropy(prediction, Y)
accuracy = mean(eq(prediction, Y))
```

Cost

```
X, Y = matrix(), vector()

net = InputLayer(X)
net = DenseLayer(net, num_units=64)
net = DenseLayer(net, num_units=10,
                      nonlinearity=softmax)

prediction = net
parameters = get_all_params(net)

loss = mean(categorical_crossentropy(prediction, Y)
accuracy = mean(eq(prediction, Y))

gradient = grad(loss, parameters)
train_op = sgd(grad, parameters)
```

Optimization

```
gradient = grad(loss, parameters)
train_op = sgd(grad, parameters)
```

```
{{ train loop }}

x_mb, y_mb = get_minibatch()
run([train_op], feed={X:x_mb, Y:y_mb})

{{ end }}
```

Train

```
gradient = grad(loss, parameters)
train_op = sgd(grad, parameters)


{{ train loop }}

x_mb, y_mb = get_minibatch()
run([train_op], feed={X:x_mb, Y:y_mb})

{{ end }}
```

Train

```
x_v, y_v = get_validation_set()
valid_loss, valid_acc = run([loss, accuracy],
                            feed={X:x_v, Y:y_v})
```

Validation

```
gradient = grad(loss, parameters)
train_op = sgd(grad, parameters)
```

```
{{ train loop }}

x_mb, y_mb = get_minibatch()
run([train_op], feed={X:x_mb, Y:y_mb})

{{ end }}
```

Train

```
x_v, y_v = get_validation_set()
valid_loss, valid_acc = run([loss, accuracy],
                            feed={X:x_v, Y:y_v})
```

Validation

```
x_new = get_unlabeled_data()
y_new = run([predictions], feed={X:x_new})
```

Prediction

```
gradient = grad(loss, parameters)
train_op = sgd(grad, parameters)
```

```
{{ train loop }}

x_mb, y_mb = get_minibatch()
run([train_op], feed={X:x_mb, Y:y_mb})

{{ end }}
```

Train

```
x_v, y_v = get_validation_set()
valid_loss, valid_acc = run([loss, accuracy],
                            feed={X:x_v, Y:y_v})
```

Validation

```
x_new = get_unlabeled_data()
y_new = run([predictions], feed={X:x_new})
```

Prediction

It's just python!