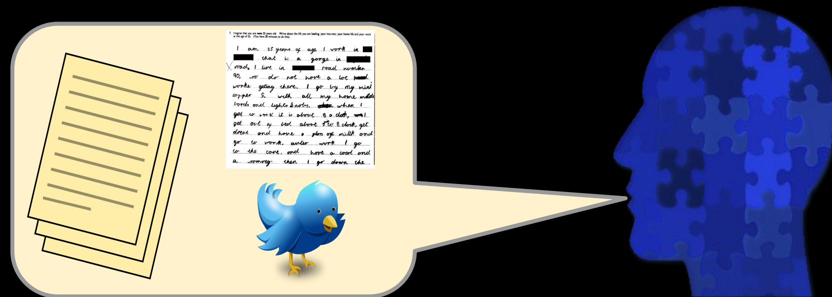
TensorFlow and Recurrent Neural Networks

CSE392 - Spring 2019
Special Topic in CS

Task



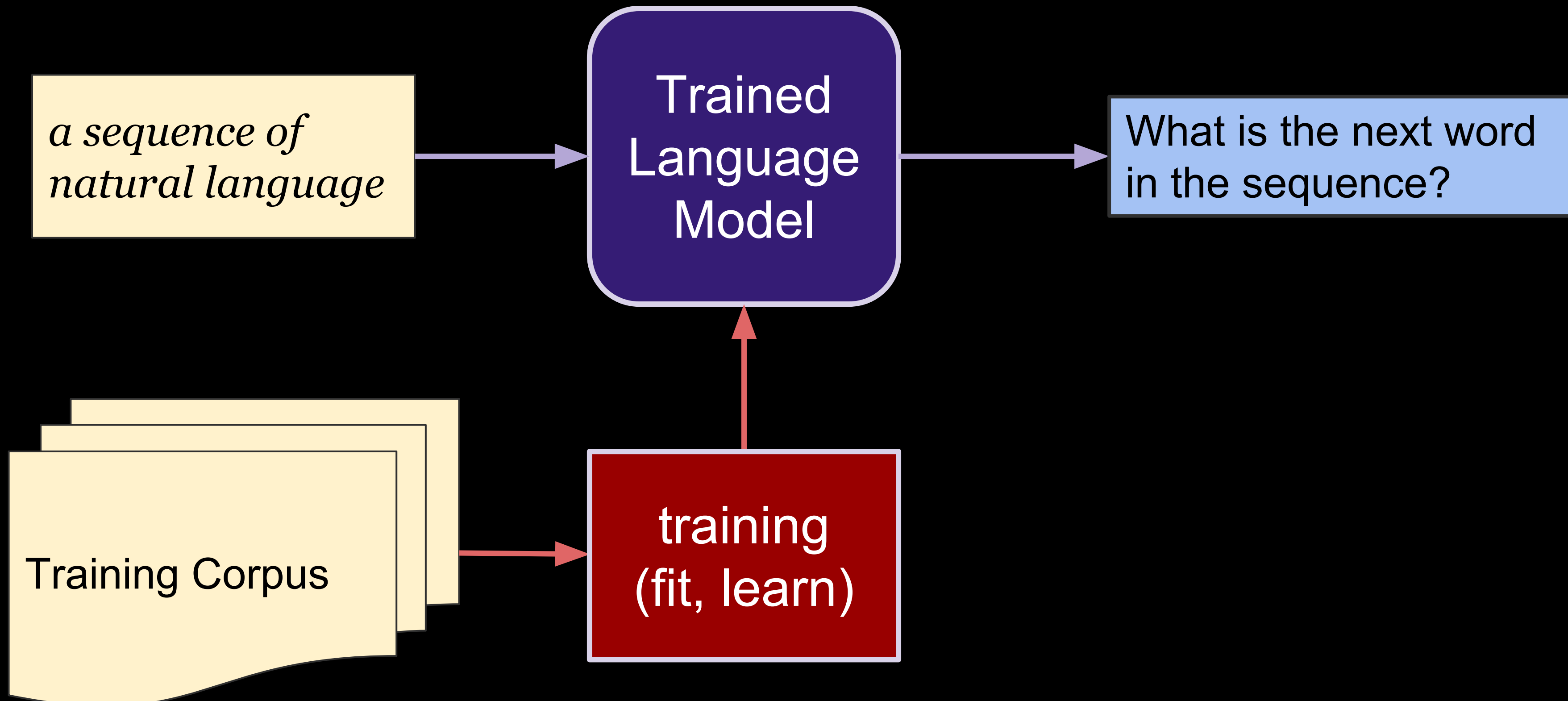
- Language Modeling and (Most Tasks)

how?

- Recurrent Neural Network
  - Implementation toolkit: TensorFlow

Language Modeling

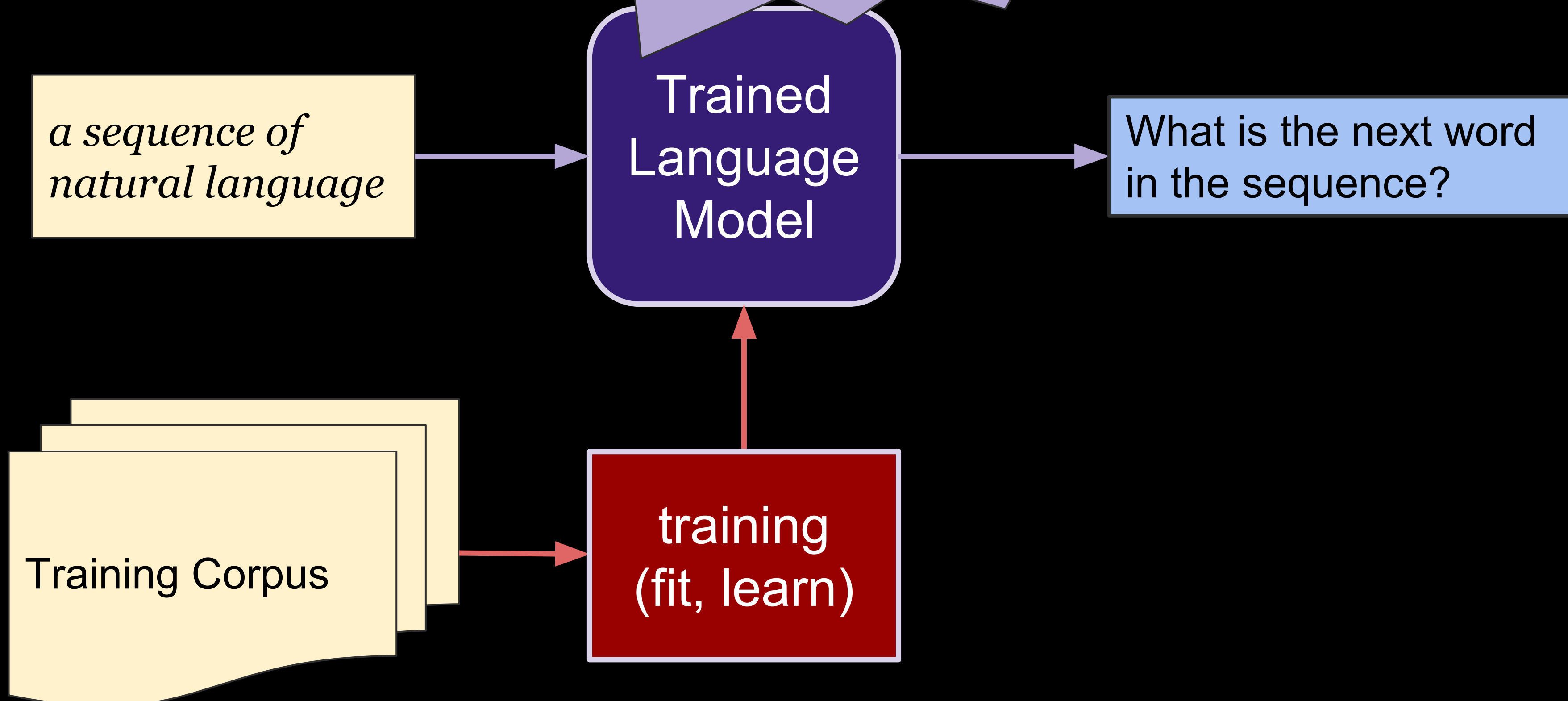
Building a model (or system / API) that can answer the following:



Language Modeling

Building a model (or s… …er the following:

To fully capture natural language, models get very complex!



Two Topics

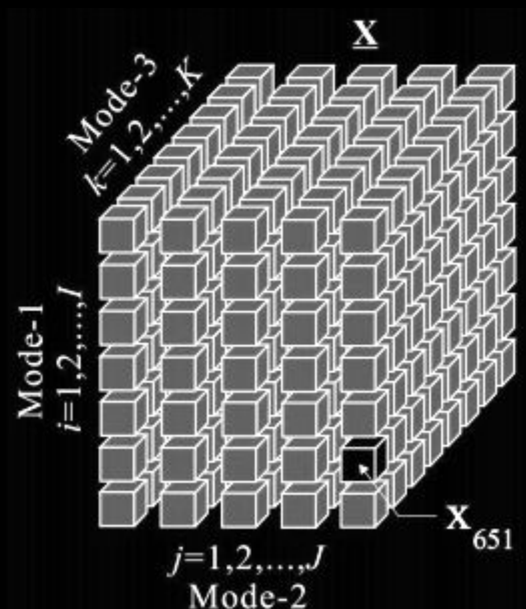
1. A Concept in Machine Learning: **Recurrent Neural Networks (RNNs)**

2. A Toolkit or Data WorkFlow System: **TensorFlow**
   Powerful for implementing RNNs

TensorFlow

A workflow system catered to numerical computation.

Basic idea: defines a graph of operations on tensors



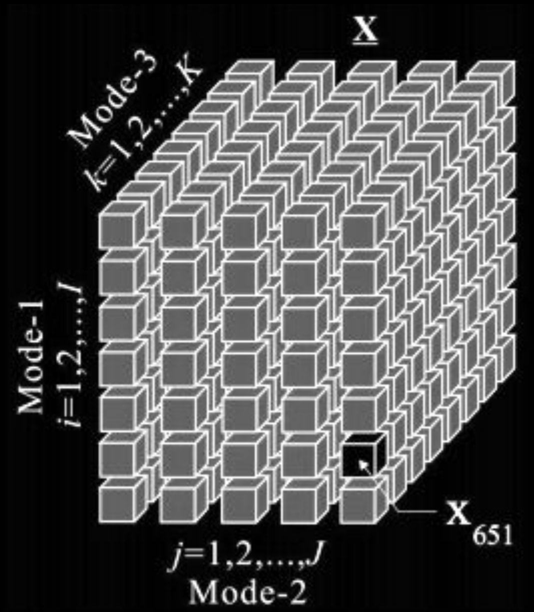
(i.stack.imgur.com)

TensorFlow

A workflow system catered to numerical computation.

Basic idea: defines a graph of operations on tensors

A multi-dimensional matrix



(i.stack.imgur.com)

TensorFlow

A workflow system catered to numerical computation.

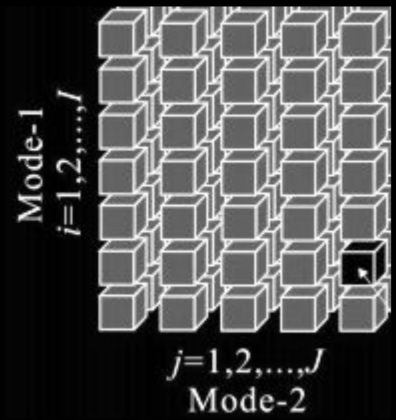
Basic idea: defines a graph of operations on tensors

A multi-dimensional matrix

A 2-d tensor is just a matrix.

1-d: vector

0-d: a constant / scalar



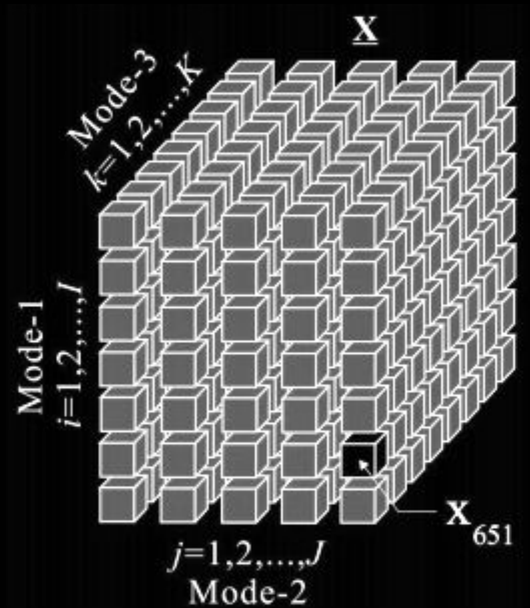
(i.stack.imgur.com)

TensorFlow

A workflow system catered to numerical computation.

Basic idea: defines a graph of operations on tensors

A multi-dimensional matrix



(i.stack.imgur.com)

A 2-d tensor is just a matrix.

1-d: vector

0-d: a constant / scalar

Linguistic Ambiguity:
"ds" of a Tensor =/= Dimensions of a Matrix

TensorFlow

A workflow system catered to numerical computation.

Basic idea: defines a graph of operations on tensors

Why?

Efficient, high-level built-in **linear algebra** and **machine learning optimization** *operations* (i.e. transformations).

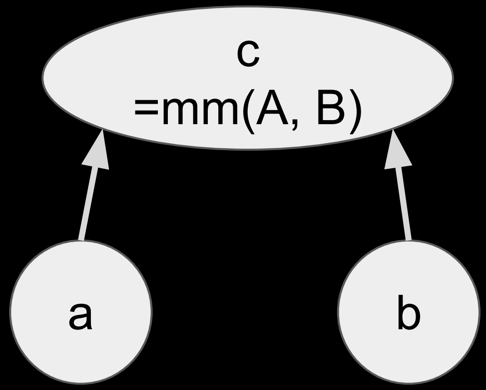
enables complex models, like deep learning

Tensor**Flow**

Operations on tensors are often conceptualized as graphs:

A simple example:

c = tensorflow.matmul(a, b)

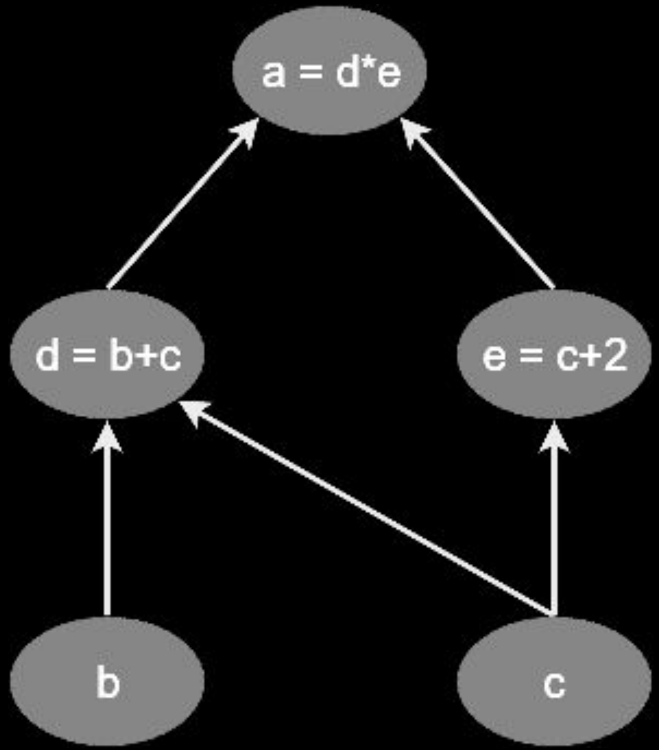


TensorFlow

Operations on tensors are often conceptualized as graphs:

example:

d=b+c
e=c+2
a=d∗e



(Adventures in Machine Learning. *Python TensorFlow Tutorial*, 2017)

Ingredients of a TensorFlow

tensors*
variables - persistent mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation (e.g. matrix multiply, add) executed by device *kernels*

graph

session
defines the environment in which operations *run*.
(like a Spark context)

devices
the specific devices (cpus or gpus) on which to run the session.

Ingredients of a TensorFlow

tensors*
variables - persistent
mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation
(e.g. matrix multiply, add)
executed by device *kernels*

- tf.Variable(initial_value, name)
- tf.constant(value, type, name)
- tf.placeholder(type, shape, name)

graph

session
defines the environment in which operations *run*.
(like a Spark context)

devices
the specific devices (cpus or gpus) on which to run the session.

Operations

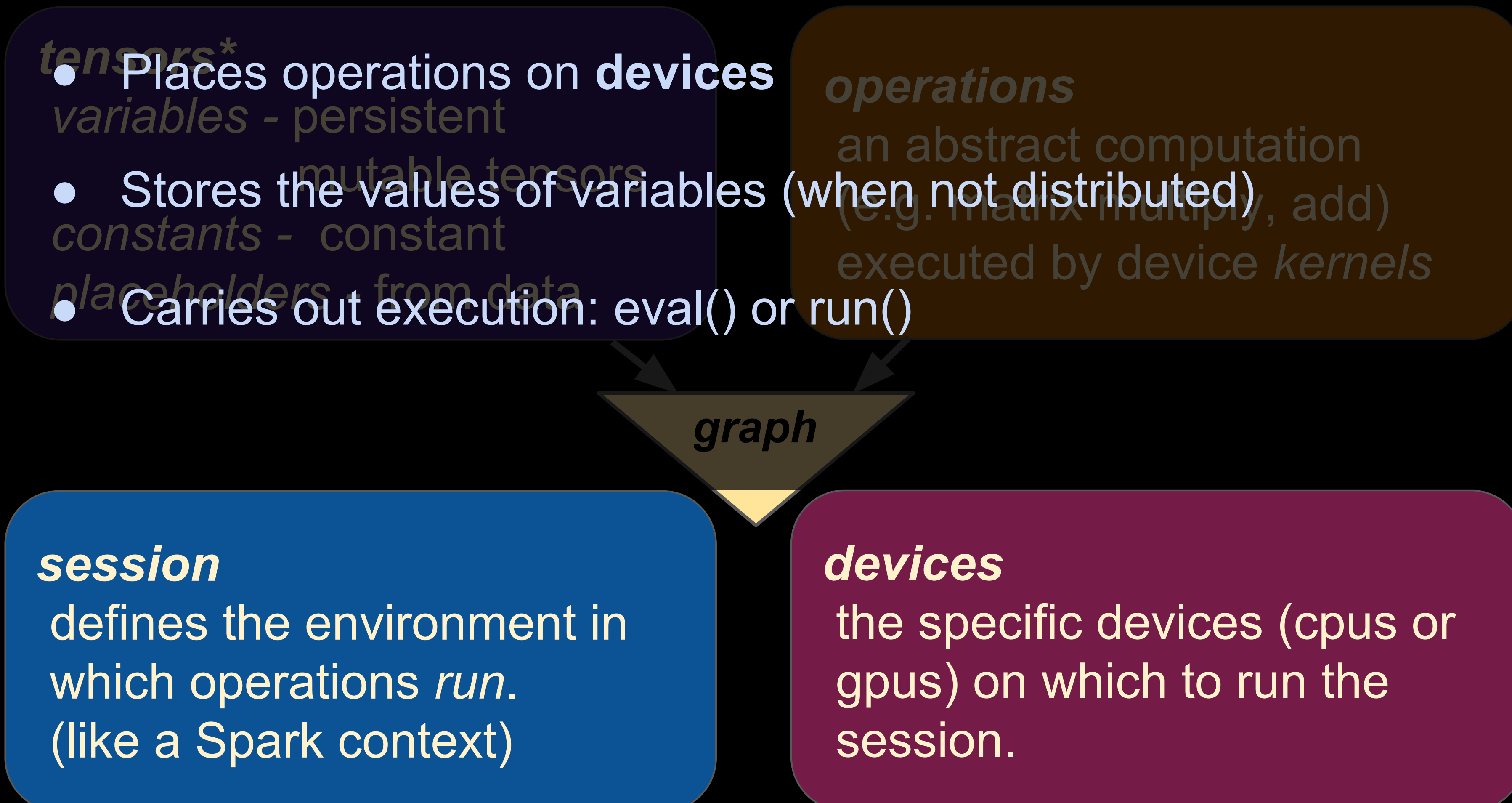
tensors*
variables - persistent
mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation
(e.g. matrix multiply, add)
executed by device *kernels*

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Sessions

- Places operations on **devices**
- Stores the values of variables (when not distributed)
- Carries out execution: eval() or run()



Ingredients of a TensorFlow

tensors*
variables - persistent
mutable tensors
constants - constant
placeholders - from data

operations
an abstract computation
(e.g. matrix multiply, add)
executed by device *kernels*

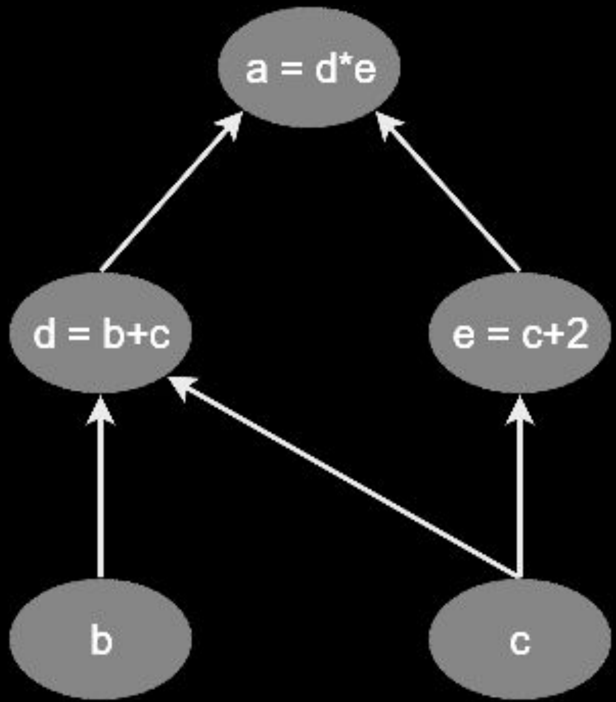
graph

session
defines the environment in
which operations *run*.
(like a Spark context)

devices
the specific devices (cpus or
gpus) on which to run the
session.

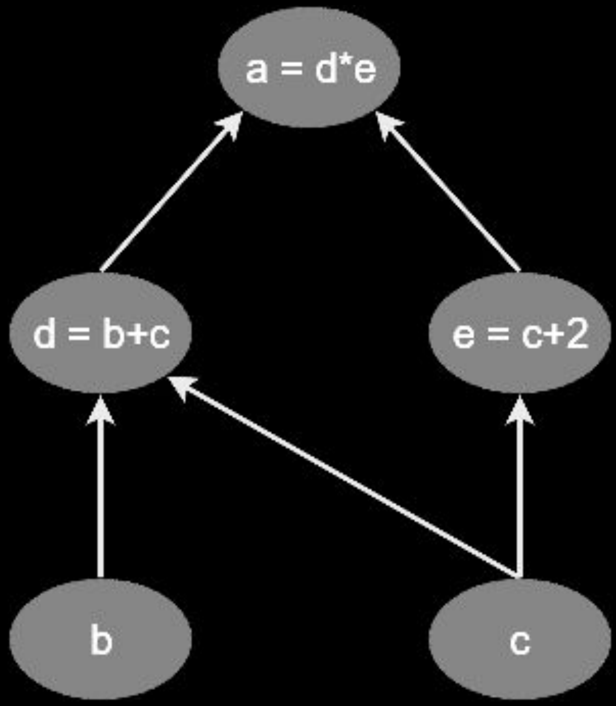
Example

```
import tensorflow as tf
b = tf.constant(1.5, dtype=tf.float32, name="b")
c = tf.constant(3.0, dtype=tf.float32, name="c")




d = b+c
e = c+2

a = d*e
```



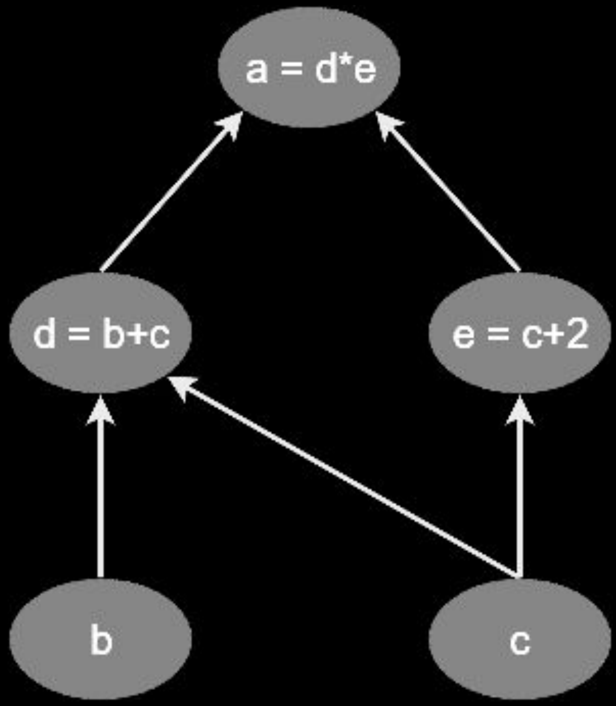
Example

```
import tensorflow as tf
b = tf.constant(1.5, dtype=tf.float32, name="b")
c = tf.constant(3.0, dtype=tf.float32, name="c")




d = b+c  #1.5 + 3
e = c+2  #3+2

a = d*e  #4.5*5 = 22.5
```



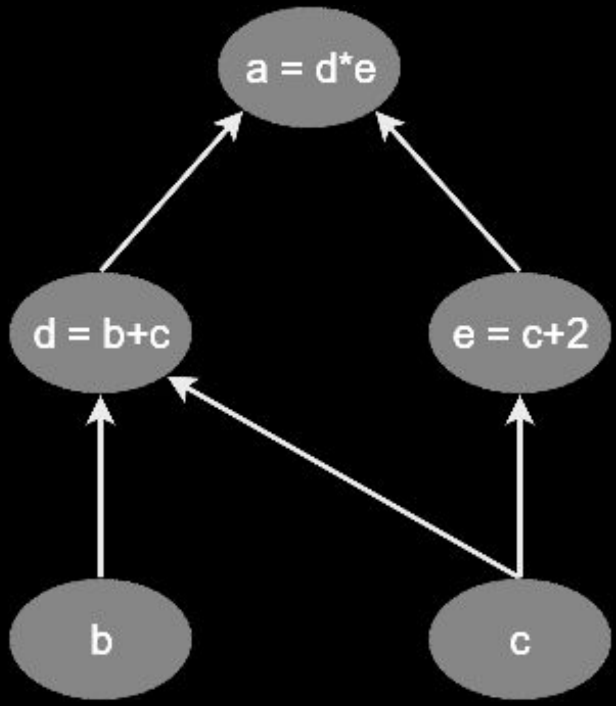
Example (working with 0-d tensors)

```
import tensorflow as tf
b = tf.constant(1.5, dtype=tf.float32, name="b")
c = tf.constant(3.0, dtype=tf.float32, name="c")




d = b+c  #1.5 + 3
e = c+2  #3+2

a = d*e  #4.5*5 = 22.5
```



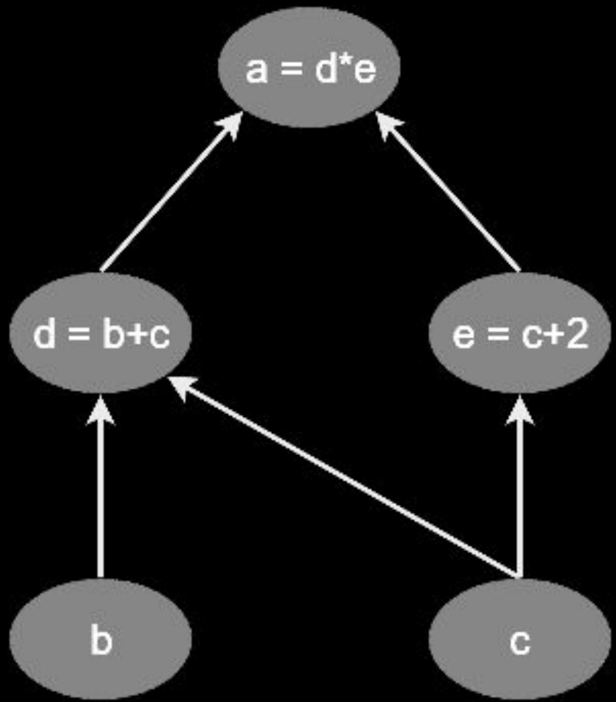
Example: now a 1-d tensor

```
import tensorflow as tf
b = tf.constant([1.5, 2, 1, 4.2],
        dtype=tf.float32, name="b")
c = tf.constant([3, 1, 5, 10],
        dtype=tf.float32, name="c")

d = b+c
e = c+2

a = d*e
```



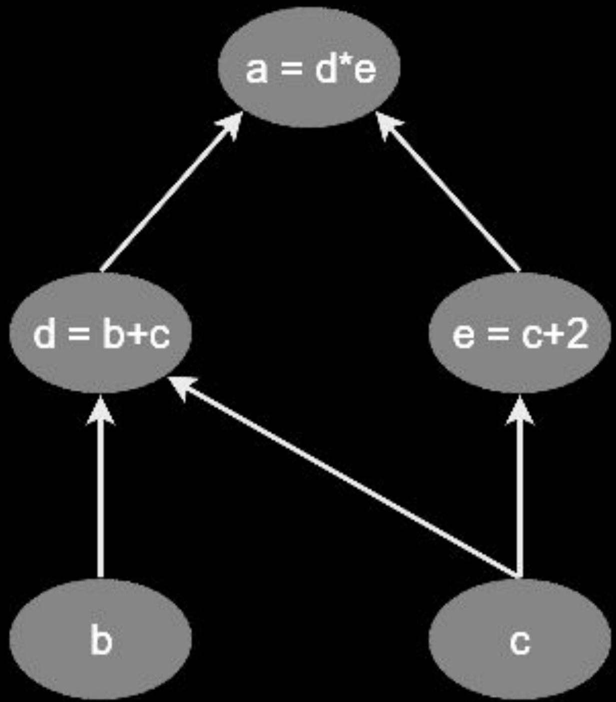
Example: now a 1-d tensor

```
import tensorflow as tf
b = tf.constant([1.5, 2, 1, 4.2],
         dtype=tf.float32, name="b")
c = tf.constant([3, 1, 5, 10],
         dtype=tf.float32, name="c")

d = b+c  #[4.5,  3, 6, 14.2]
e = c+2  #[5, 4, 7, 12]

a = d*e  #??
```



Example: now a 2-d tensor

```
import tensorflow as tf
b = tf.constant([[...], [...]],
         dtype=tf.float32, name="b")
c = tf.constant([[...], [...]],
         dtype=tf.float32, name="c")

d = b+c
e = c+2

a = tf.matmul(d,e)
```



Example: Logistic Regression

```
X = tf.constant([[...], [...]],
        dtype=tf.float32, name="X")
y = tf.constant([...],
        dtype=tf.float32, name="y")
# Define our beta parameter vector:
beta = tf.Variable(tf.random_uniform([featuresZ_pBias.shape[1], 1], -1.,
1.), name = "beta")
```

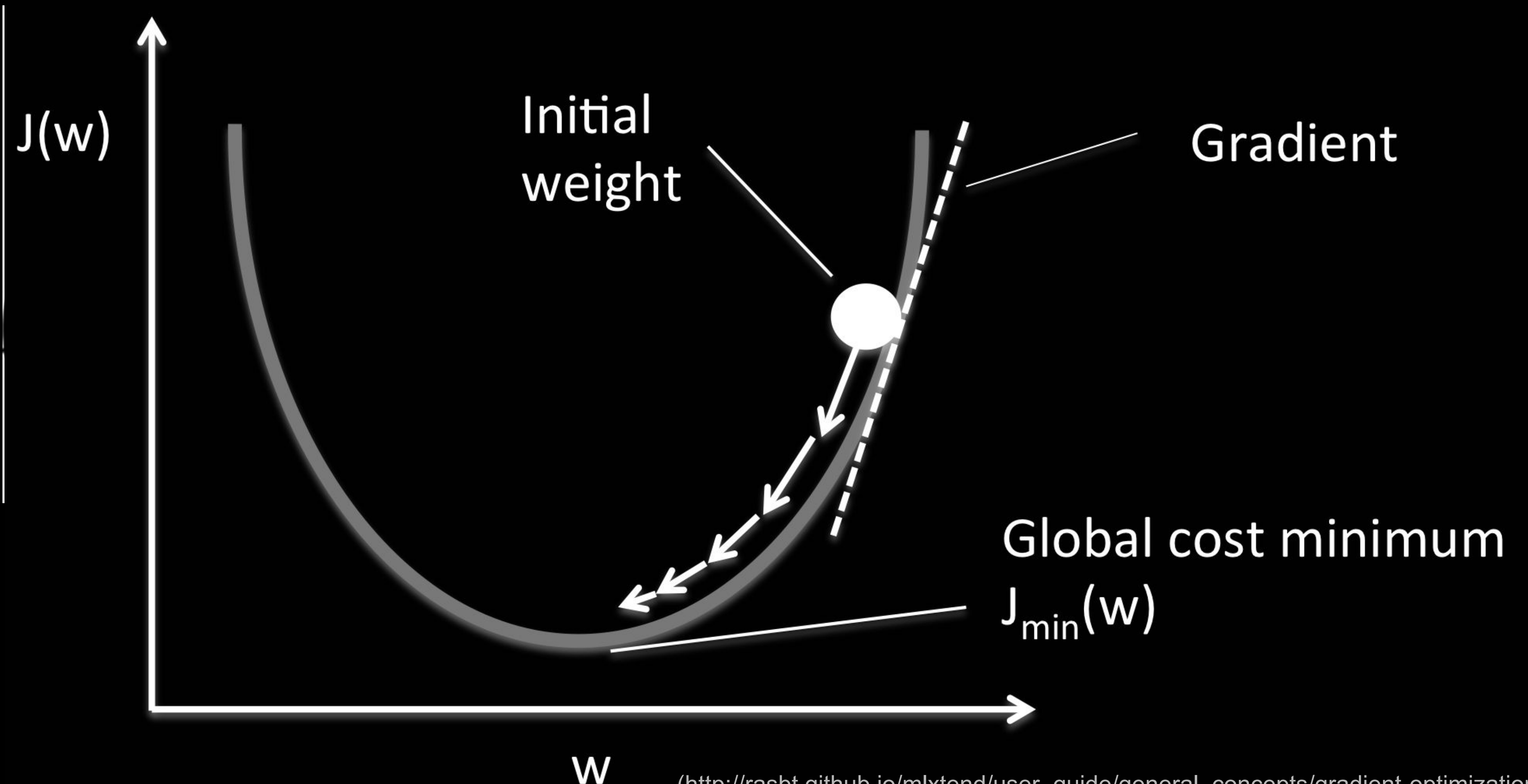
Example: Logistic Regression

```
X = tf.constant([[...], [...]],
        dtype=tf.float32, name="X")
y = tf.constant([...],
        dtype=tf.float32, name="y")
# Define our beta parameter vector:
beta = tf.Variable(tf.random_uniform([featuresZ_pBias.shape[1], 1], -1.,
1.), name = "beta")

#then setup the prediction model's graph:
y_pred = tf.softmax(tf.matmul(X, beta), name="predictions")
```

Example: Logistic Regression

```
X = tf.constant([[...], [...]],
        dtype=tf.float32, name="X")
y = tf.constant([...],
        dtype=tf.float32, name="y")
# Define our beta parameter vector:
beta = tf.Variable(tf.random_uniform([featuresZ_pBias.shape[1], 1], -1.,
1.), name = "beta")

#then setup the prediction model's graph:
y_pred = tf.softmax(tf.matmul(X, beta), name="predictions")

#Define a *cost function* to minimize:
penalizedCost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred),
reduction_indices=1)) #conceptually like |y - y_pred|
```

Optimizing Parameters -- derived from **gradients**



(http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/)

Example: Logistic Regression

```
X = tf.constant([[...], [...]],
        dtype=tf.float32, name="X")
y = tf.constant([...],
        dtype=tf.float32, name="y")
# Define our beta parameter vector:
beta = tf.Variable(tf.random_uniform([featuresZ_pBias.shape[1], 1], -1.,
1.), name = "beta")

#then setup the prediction model's graph:
y_pred = tf.softmax(tf.matmul(X, beta), name="predictions")

#Define a *cost function* to minimize:
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred),
reduction_indices=1))
```

Example: Logistic Regression

```
X = tf.constant([[...], [...]], dtype=tf.float32, name="X")
y = tf.constant([...],      dtype=tf.float32, name="y")
# Define our beta parameter vector:
beta = tf.Variable(tf.random_uniform([featuresZ_pBias.shape[1], 1], -1., 1.), name = "beta")
#then setup the prediction model's graph:
y_pred = tf.softmax(tf.matmul(X, beta), name="predictions")
#Define a *cost function* to minimize:
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred), reduction_indices=1))

#define how to optimize and initialize:
optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()
```

Example: Logistic Regression

```
X = tf.constant([[...], [...]], dtype=tf.float32, name="X")
y = tf.constant([...],     dtype=tf.float32, name="y")
# Define our beta parameter vector:
beta = tf.Variable(tf.random_uniform([featuresZ_pBias.shape[1], 1], -1., 1.), name = "beta")
#then setup the prediction model's graph:
y_pred = tf.softmax(tf.matmul(X, beta), name="predictions")
#Define a *cost function* to minimize:
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred), reduction_indices=1))

#define how to optimize and initialize:
optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate)
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()

#iterate over optimization:
with tf.Session() as sess:
 sess.run(init)
 for epoch in range(n_epochs):
  sess.run(training_op)
 #done training, get final beta:
 best_beta = beta.eval()
```

Neural Networks: Graphs of Operations

Neural Networks: Graphs of Operations (excluding the optimization nodes)

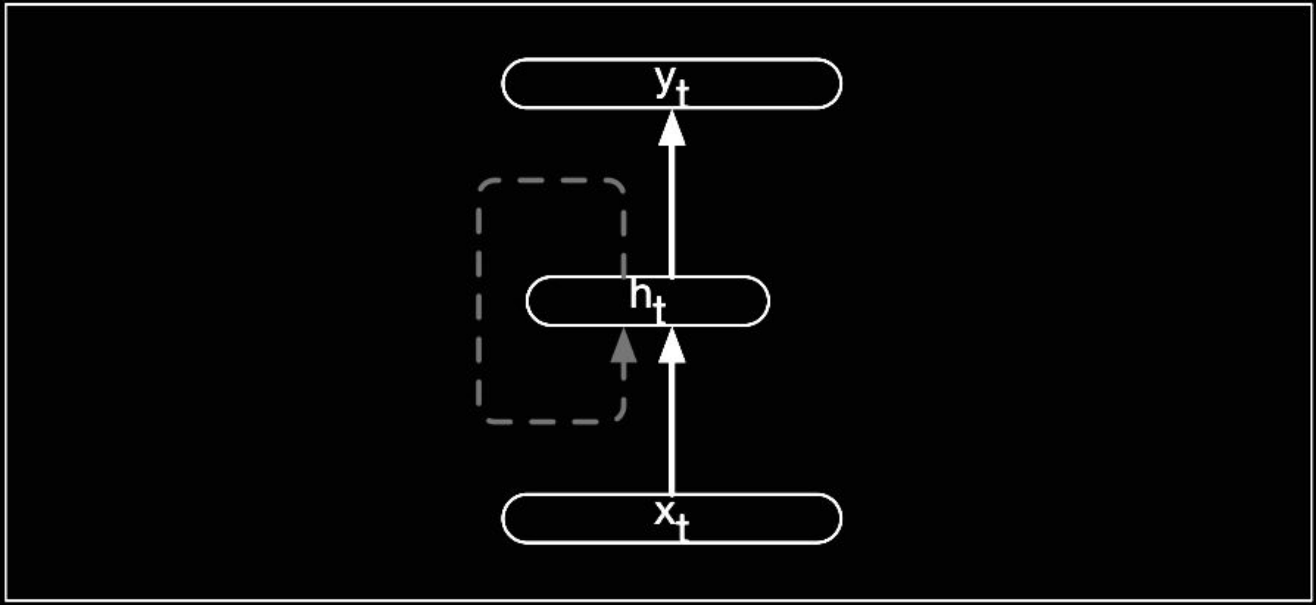


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

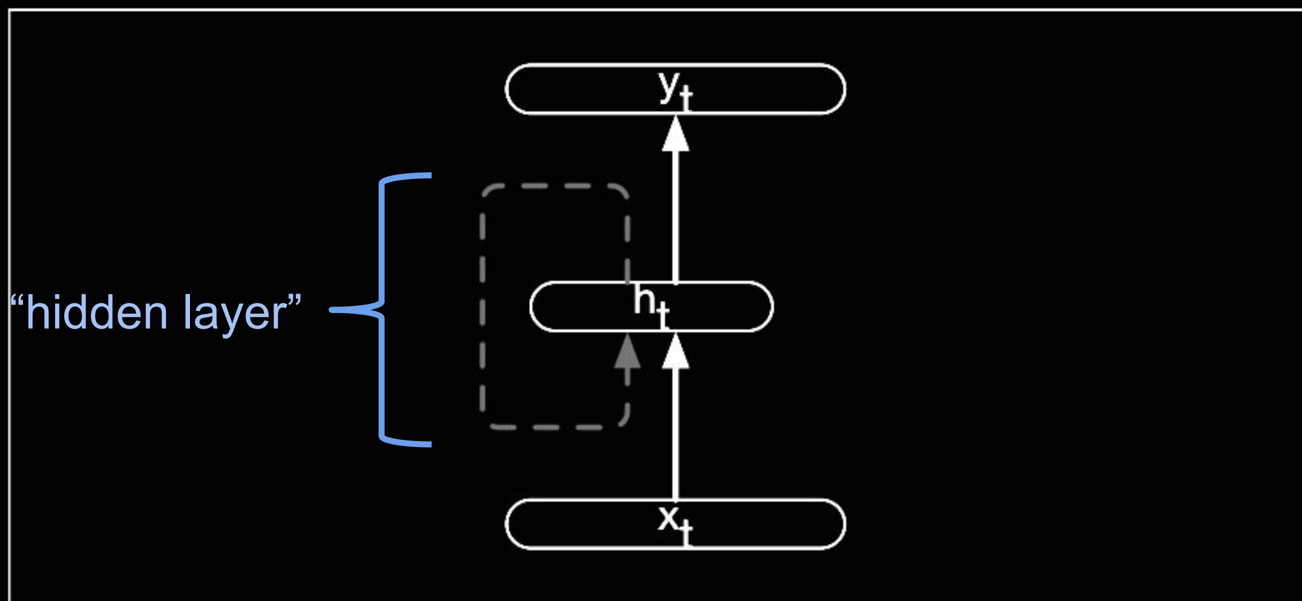
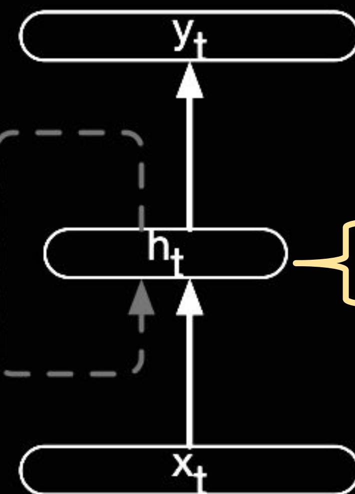


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations
(excluding the optimization nodes)



"hidden layer"

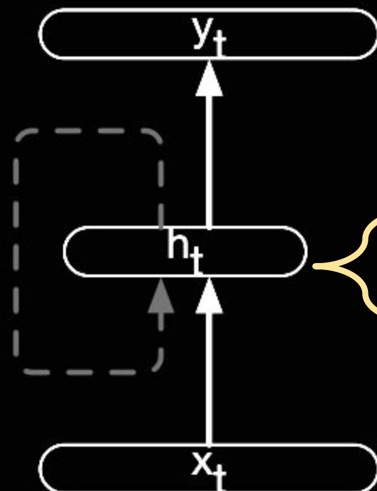
$$h_t = g(vecmul(h_{t-1}, U) + vecmul(x_t, V))$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

“hidden layer”



$$h_t = g(h_{t-1}U + x_tV)$$

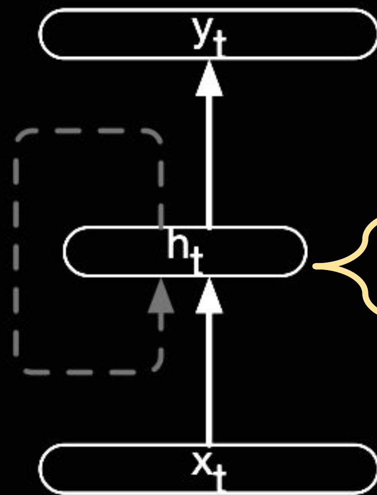
short hand for vector/ matrix multiply

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)

“hidden layer”



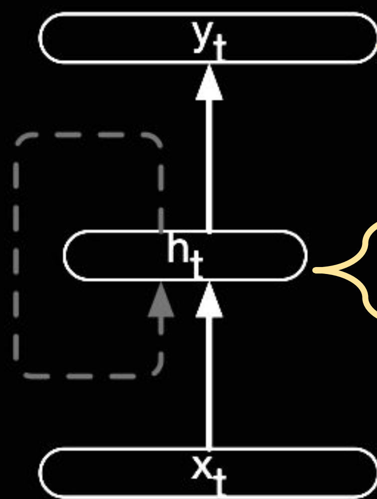
Activation Function

$$h_{(t)} = g(h_{(t-1)}U + x_{(t)}V)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations
(excluding the optimization nodes)



"hidden layer"

$$y_{(t)} = f(h_{(t)}W)$$

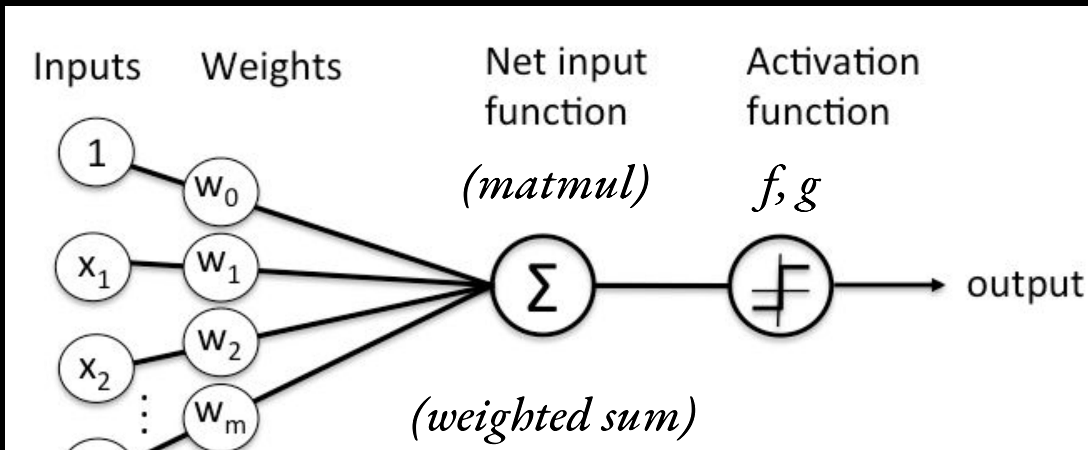
Activation Function

$$h_{(t)} = g(h_{(t-1)}U + x_{(t)}V)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Neural Networks: Graphs of Operations (excluding the optimization nodes)



(skymind, AI Wiki)

$$y_{(t)} = f(h_{(t)}W)$$

Activation Function

$$h_{(t)} = g(h_{(t-1)}U + x_{(t)}V)$$

Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

(Jurafsky, 2019)

Common Activation Functions

$z = h_{(t)}W$

Logistic: $\sigma(z) = 1 / (1 + e^{-z})$

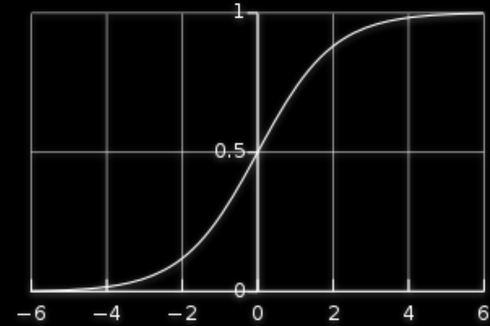
Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$

Common Activation Functions

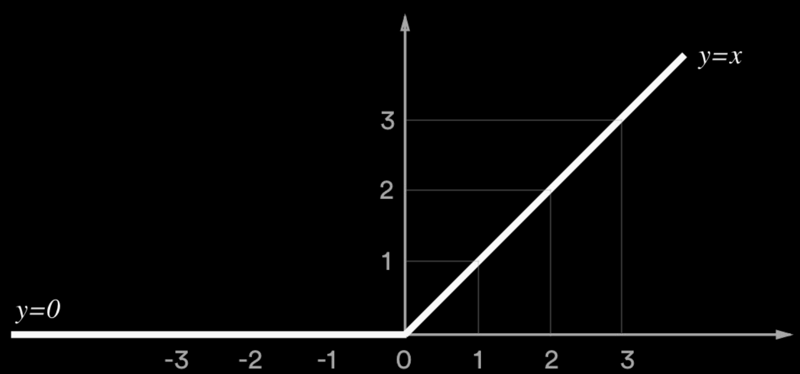
$z = h_{(t)}W$

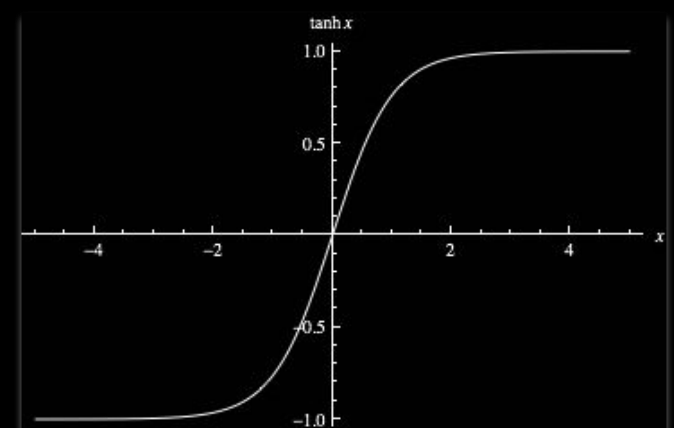
Logistic: $\sigma(z) = 1 \,/\, (1 + e^{-z})$



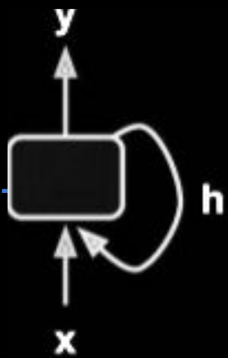
Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) \,/\, (e^{2z} + 1)$

Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$





Example: Forward Pass



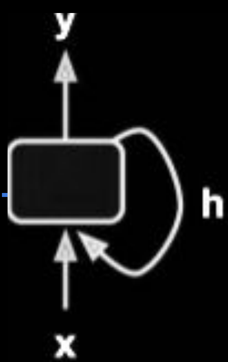
(Geron, 2017)

```
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
     h(i) = g(U h(i-1) + W x(i)) #update hidden state
     y(i) = f(V h(i)) #update output
```

Example: Forward Pass

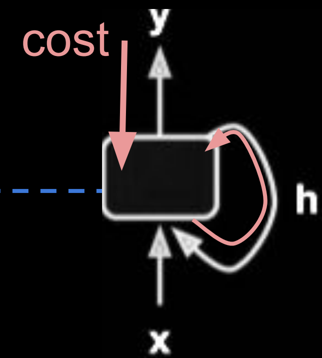

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden state
    y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
```

Example: Forward Pass




```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden
state
    y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
...
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

Optimization:

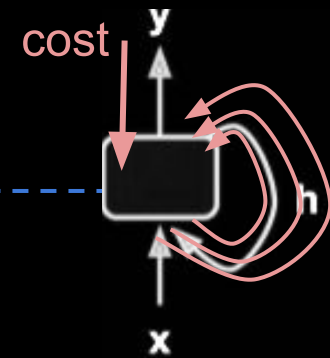


Backward Propagation

To find the gradient for the overall graph, we use **back propogation,** which *essentially* chains together the gradients for each node (function) in the graph.

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden state
    y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
...
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

Optimization:



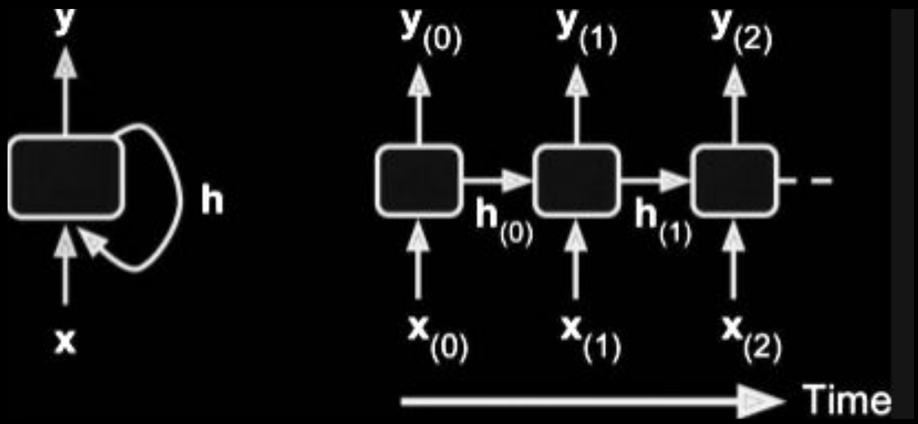
Backward Propagation

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,
state
    y(i) = tf.softmax(tf.matmul
...
cost = tf.reduce_mean(-tf.redu
```

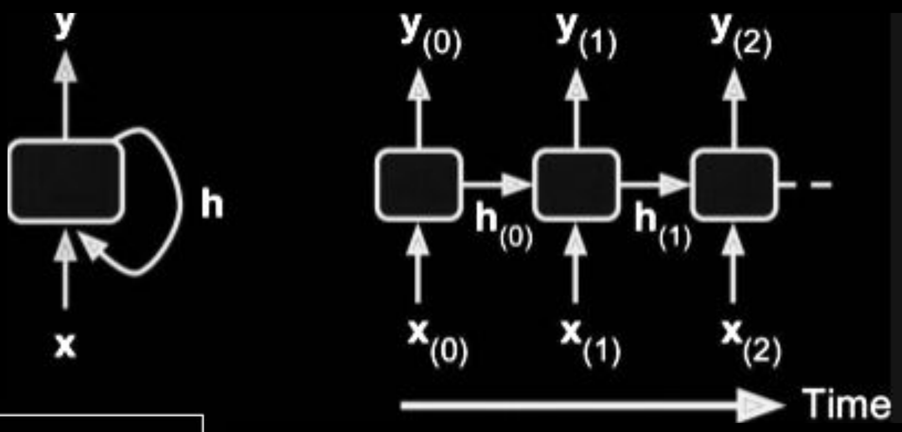
To find the gradient for the overall graph, we use **back propogation,** which *essentially* chains together the gradients for each node (function) in the graph.

With many recursions, the gradients can vanish or explode (become too large or small for floating point operations).

Solution: Unrolling



Solution: Unrolling



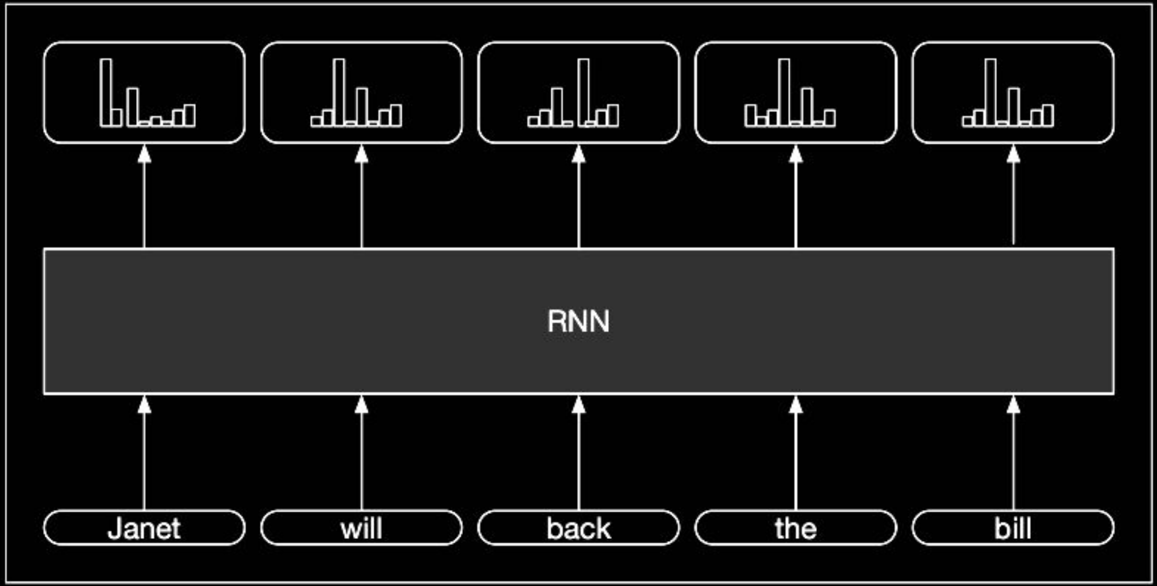
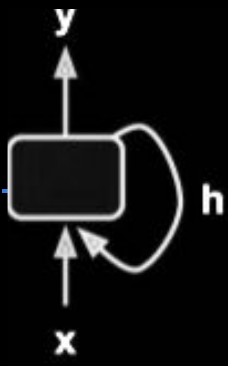
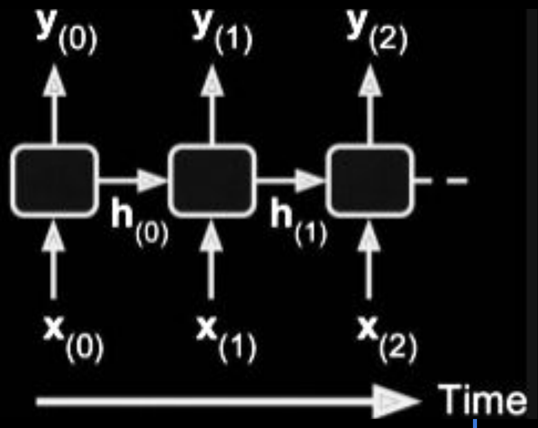


Figure 9.8 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

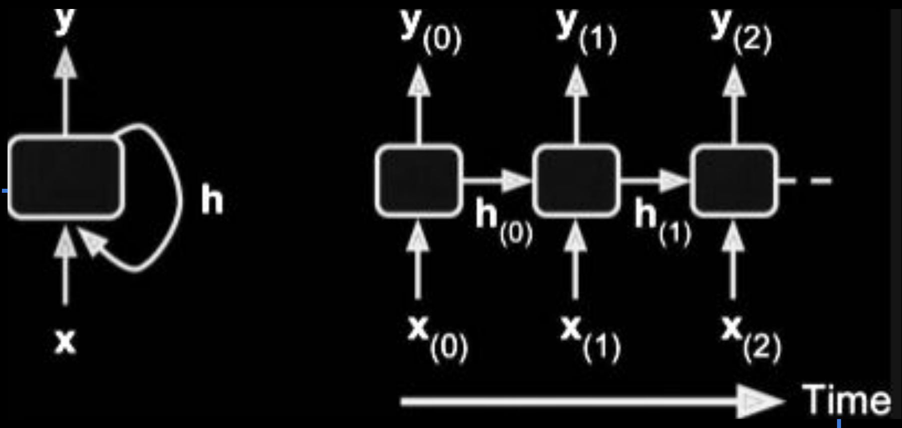
Example: Forward Pass





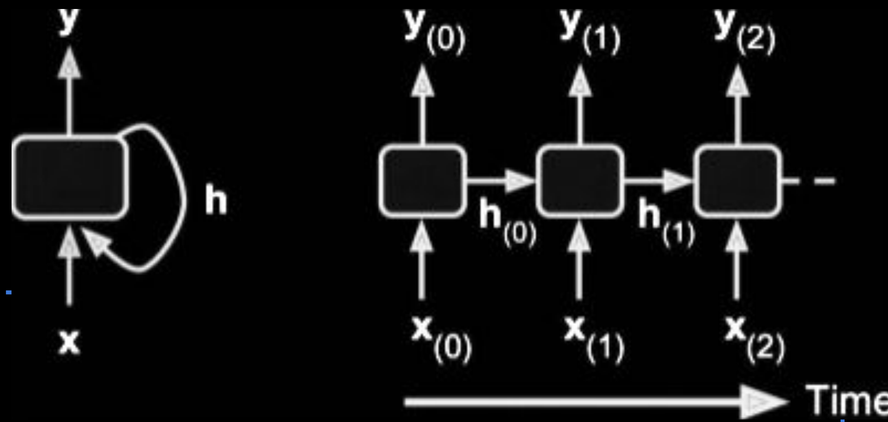

```
#define forward pass graph:

h(i) = tf.nn.relu(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden state
y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
```

Example: Forward Pass



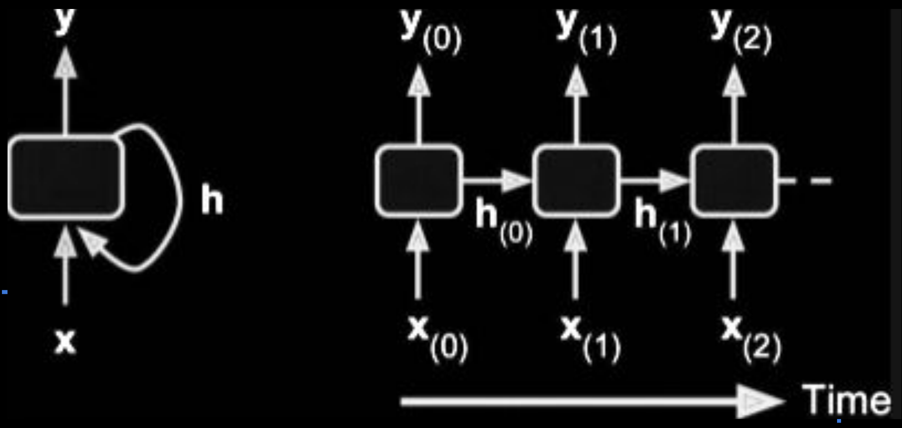

```
hidden_size, output_size = 5, 1



#define forward pass graph:

h(i) = tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu)
y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
```

Example: Forward Pass



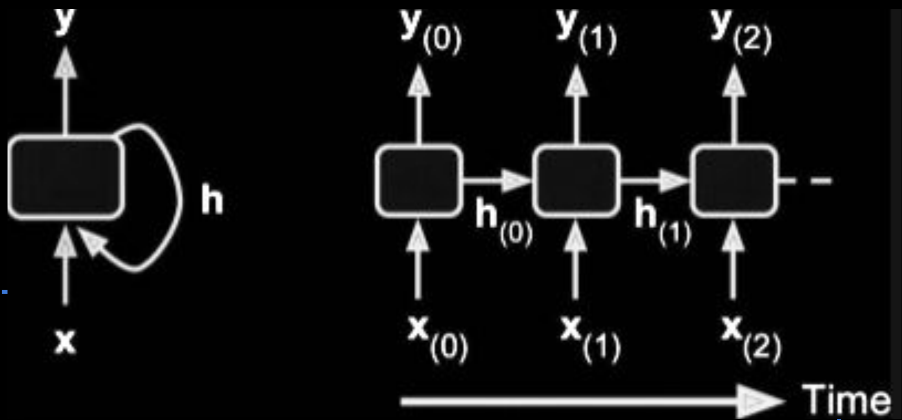

```
hidden_size, output_size = 5, 1



#define forward pass graph:


h_(i) = tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu)
y_(i) = tf.softmax(tf.matmul(V, h_(i))) #update output
```

Example: Forward Pass



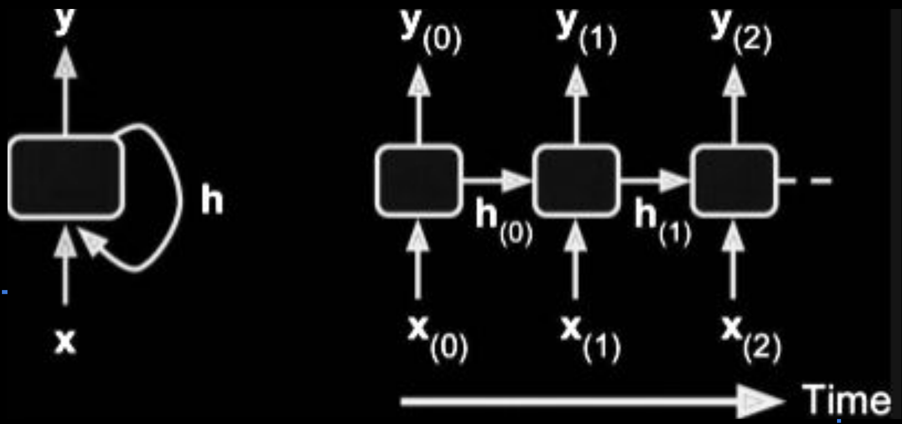

```
hidden_size, output_size = 5, 1



#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWrapper(
     tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu),
     output_size = output_size
y(t) = tf.softmax(tf.matmul(V, h(t))) #update output
```

Example: Forward Pass



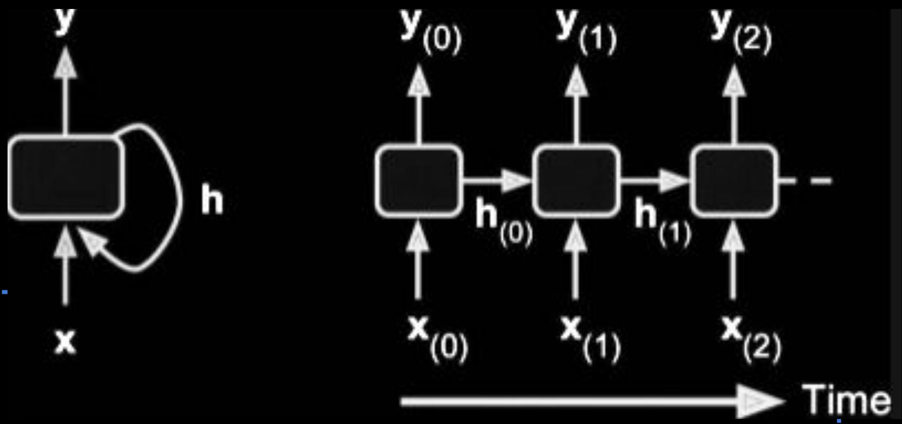

```
hidden_size, output_size = 5, 1



#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWrapper(
     tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu),
     output_size = output_size
```

Example: Forward Pass



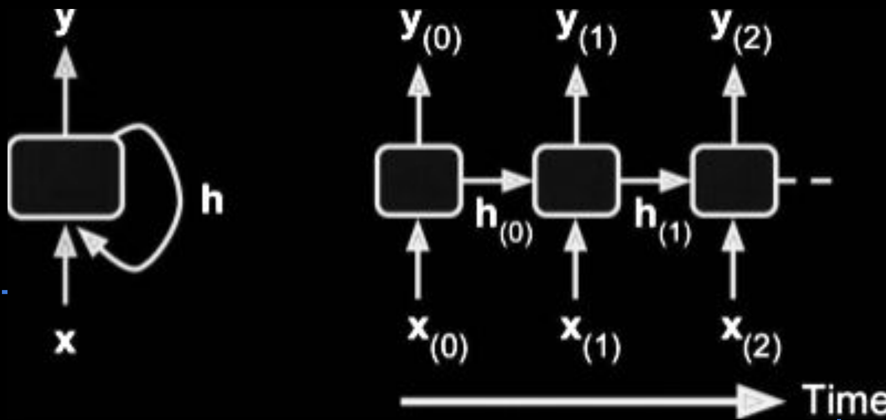

```
hidden_size, output_size = 5, 1



#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWrapper(
     tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu),
     output_size = output_size
#define training parameters:
learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(outputs))  #softmax cost
optimizer = tf.train.AdamOptimizer(learing_rate=learning_rate)
```

Example: Forward Pass



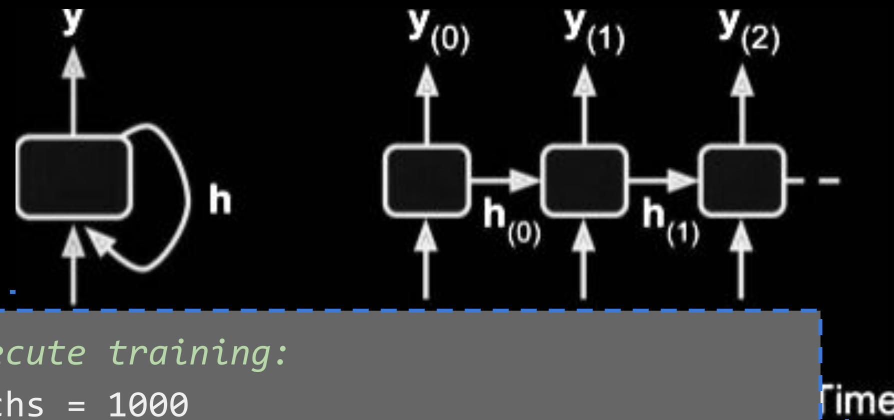

```
hidden_size, output_size = 5, 1



#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWrapper(
     tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu),
     output_size = output_size
#define training parameters:
learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(outputs))  #softmax cost
optimizer = tf.train.AdamOptimizer(learing_rate=learning_rate)
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()
```

Example: Forward Pass




```
hidden_size, output_size = 5, 1
input_size, unroll_steps = 10, 20
X = tf.placeholder(tf.float32, [None, unroll_steps, input_size])
y = tf.placeholder(tf.float32, [None, unroll_steps, output_size])
#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWrapper(
     tf.contrib.BasicRNNCell(num_units=hidden_size, activation = tf.nn.relu),
     output_size = output_size
#define training parameters:
learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(outputs))  #softmax cost
optimizer = tf.train.AdamOptimizer(learing_rate=learning_rate)
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()
```

Example: Forward Pass

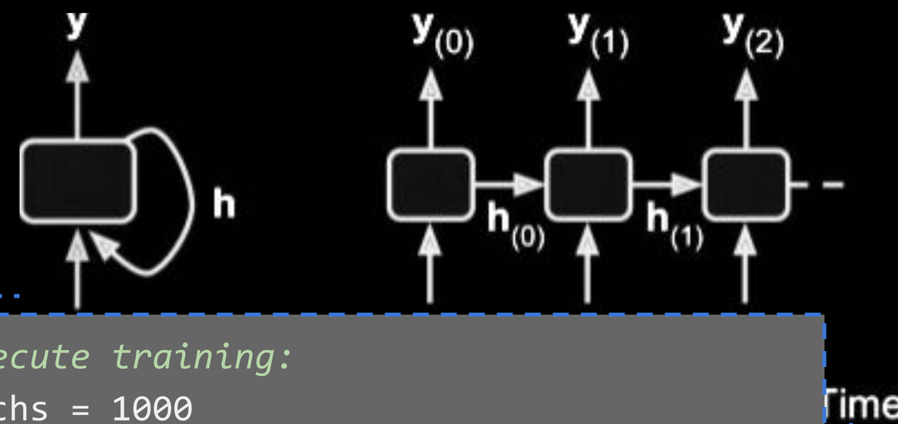



```
hidden_size, output_size = 5, 1
input_size, unroll_steps = 10, 20
X = tf.placeholder(tf.float32, [None, unr
y = tf.placeholder(tf.float32, [None, unr
#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWra
     tf.contrib.BasicRNNCell(num_units=hi
     output_size = output_size
#define training parameters:
learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(y*t
optimizer = tf.train.AdamOptimizer(leari
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()
```

```
#execute training:
epochs = 1000
batch_size = 50
with tf.Session() as sess:
  init.run()
```

(Geron, 2017)

Example: Forward Pass

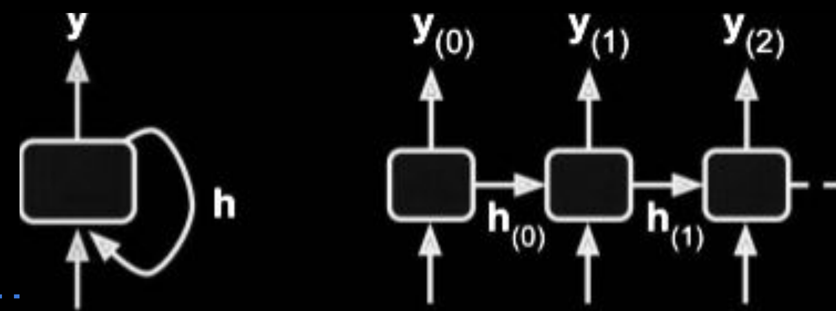



```
hidden_size, output_size = 5, 1
input_size, unroll_steps = 10, 20
X = tf.placeholder(tf.float32, [None, unr
y = tf.placeholder(tf.float32, [None, unr
#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWra
     tf.contrib.BasicRNNCell(num_units=hi
     output_size = output_size
#define training parameters:
learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(y*tf
optimizer = tf.train.AdamOptimizer(learin
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()
```

```
#execute training:
epochs = 1000
batch_size = 50
with tf.Session() as sess:
  init.run()
  for iter in range(epochs)
     X_batch, y_batch = ...#fetch next batch
     sess.run(training_op, feed_dict=\
              {X:X_batch, y:y_batch})
```

(Geron, 2017)

Example: Forward Pass




```
hidden_size, output_size = 5, 1
input_size, unroll_steps = 10, 20
X = tf.placeholder(tf.float32, [None, unr
y = tf.placeholder(tf.float32, [None, unr
#define forward pass graph:
cell = tf.contrib.rnn.OutputProjectionWra
     tf.contrib.BasicRNNCell(num_units=hi
     output_size = output_size
#define training parameters:
learning_rate = 0.001
cost = tf.reduce_mean(-tf.reduce_sum(y*t
optimizer = tf.train.AdamOptimizer(learin
training_op = optimizer.minimize(cost)
init = tf.global_variables_initializer()
```

```
#execute training:
epochs = 1000
batch_size = 50
with tf.Session() as sess:
  init.run()
  for iter in range(epochs)
     X_batch, y_batch = …#fetch next batch
     sess.run(training_op, feed_dict=\
              {X:X_batch, y:y_batch})
     if iter % 100 == 0:
          c = cost.eval(feed_dict=\
              {X:X_batch, y:y_batch})
          print(iter, “\tcost: ”, c)
```

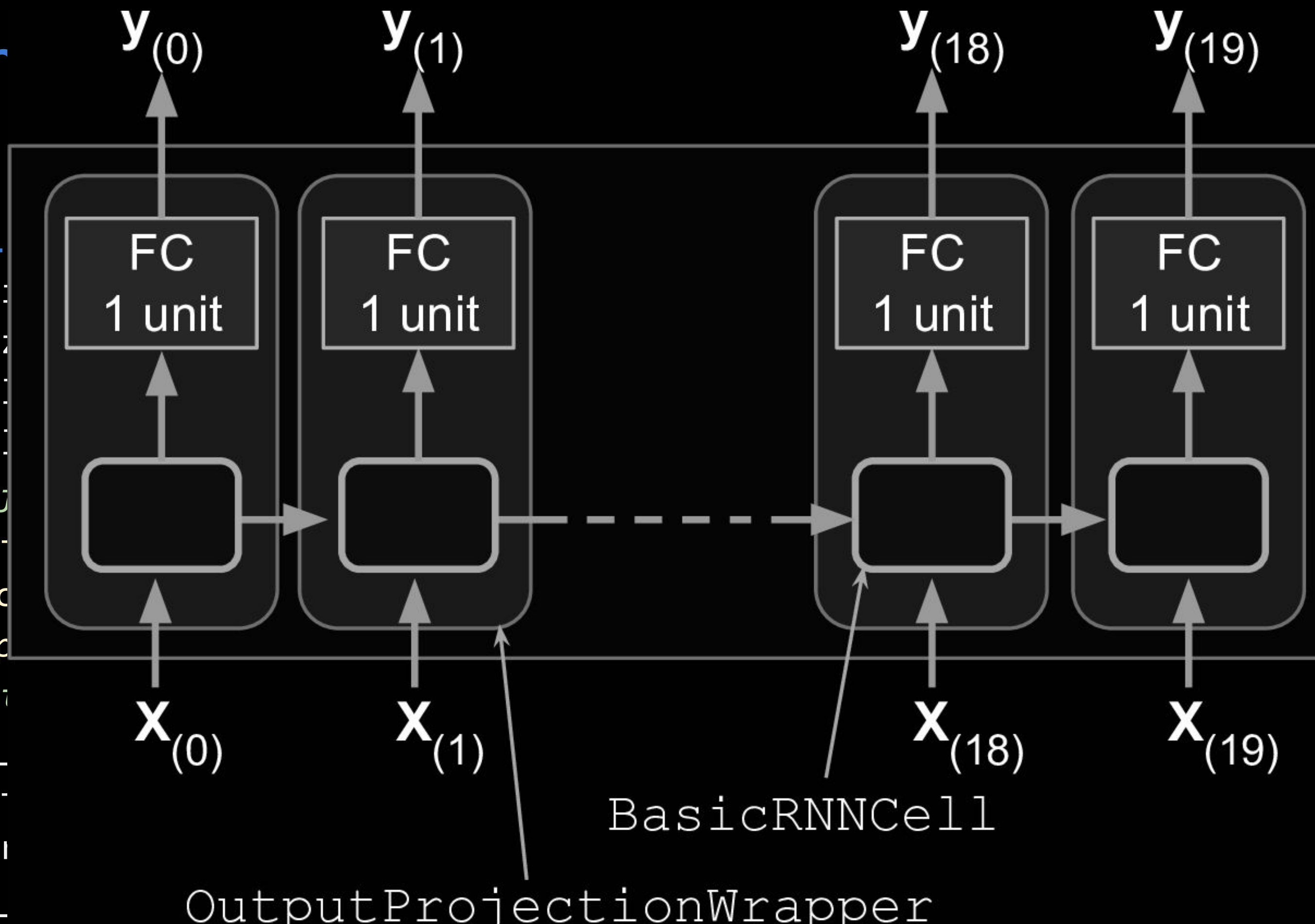
(Geron, 2017)

Time

Exar

```
hidden_si
input_si
X = tf.p
y = tf.p
#define j
cell = t
     tf.c
     outp
#define t
learning_
cost = t
optimizer
training_
init = tf.global_variables_initializer()
```

```
ext batch
ct=\
ch})
\
ch})
c)
```



(Geron, 2017)