DL4nlp - Deep Learning for Natural Language Processing

Ander Barrena Madinabeitia
ander.barrena@ehu.eus - @4nderB
Hitz Zentroa - Ixa Taldea

Tensor Flow!

Introducing TensorFlow

- Deep Learning Frameworks
  - Tensor operations made easy
  - … with interface for parallelization in GPUs
  - Ready-to-use functions
  - Auto-gradient of functions
  - Allows to share models

- **TensorFlow** (Keras), **Pytorch** to name a few
  - Different advantages, paradigms, levels of abstraction, programming languages , etc.
  - TF ~ deployment & production
  - Pytorch ~ research

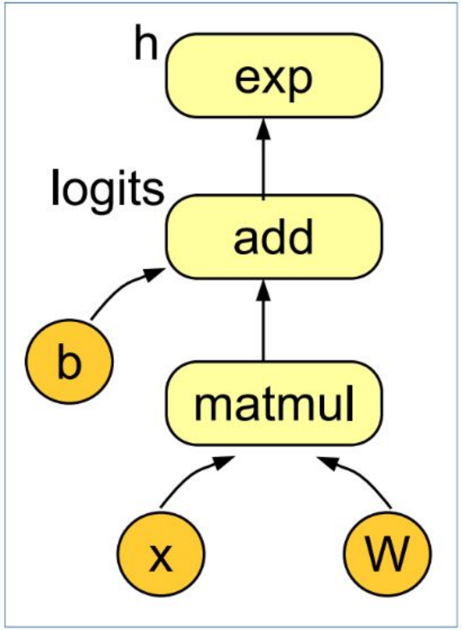
Introducing TensorFlow

- Most widely used (???)
- Google Brain, then put open source
- Interface to express machine learning algorithms PLUS an implementation
- **KEY IDEA** Numeric computation as data flow graphs
    - Nodes are mathematical operations, with inputs and outputs
    - Edges are tensors between nodes

Introducing TensorFlow

How to build your LR classifier

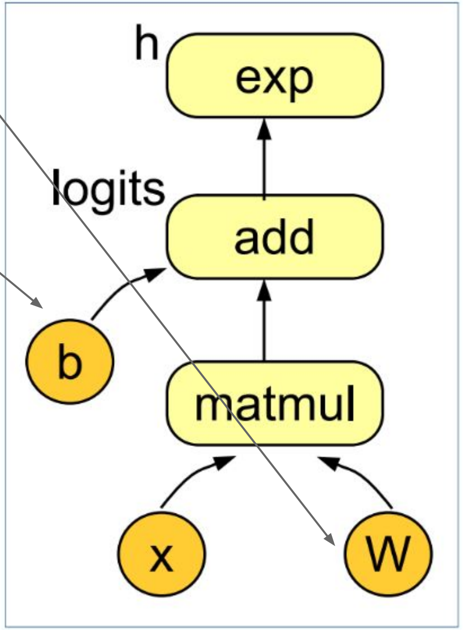
$$h = \exp(xW + b)$$



Introducing TensorFlow

Variables: nodes which output their current value (**Parameters**)

$$h = \exp(xW + b)$$

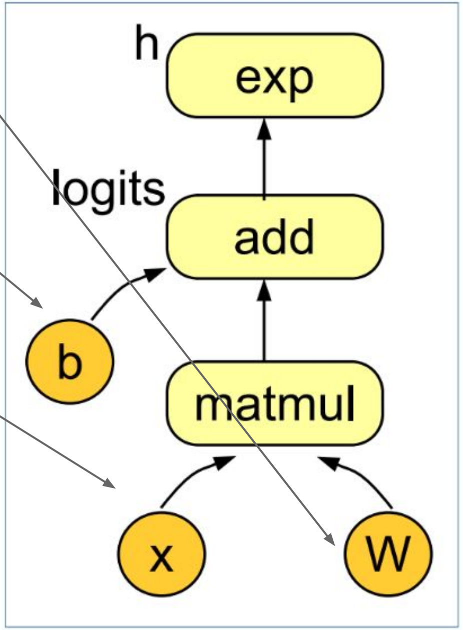


Introducing TensorFlow

Variables: nodes which output their current value (**Parameters**)

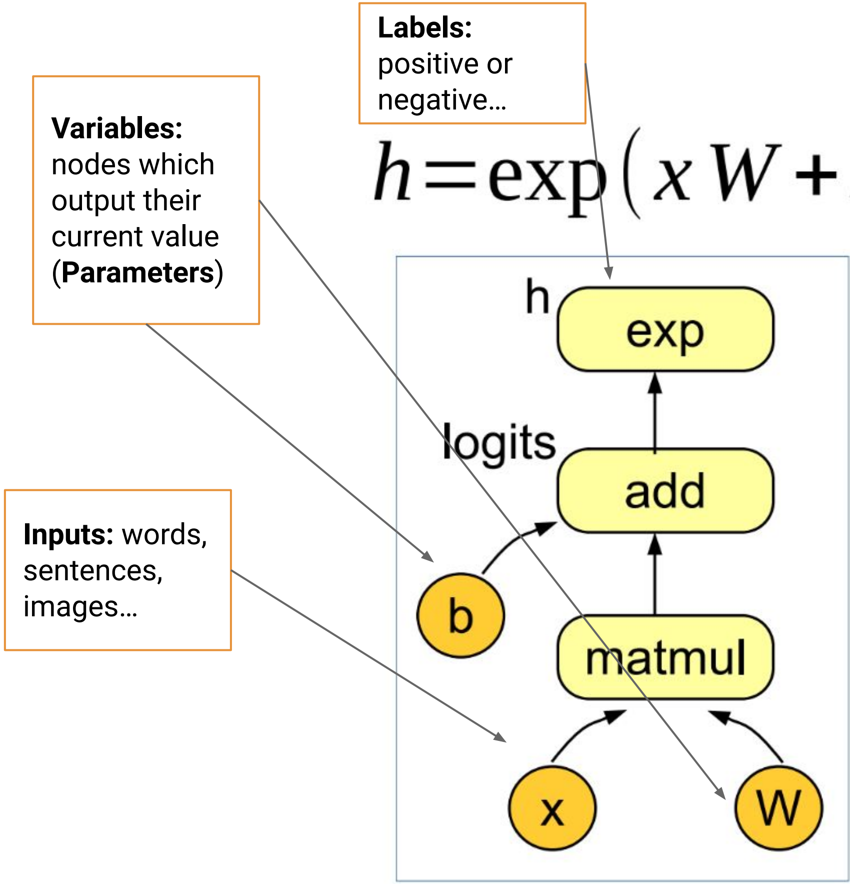
Inputs: words, sentences, images…

$$h = \exp(xW + b)$$



Introducing TensorFlow

$$h = \exp(xW + b)$$



Labels: positive or negative…

Variables: nodes which output their current value (**Parameters**)

Inputs: words, sentences, images…

Introducing TensorFlow

Sentiment analysis:

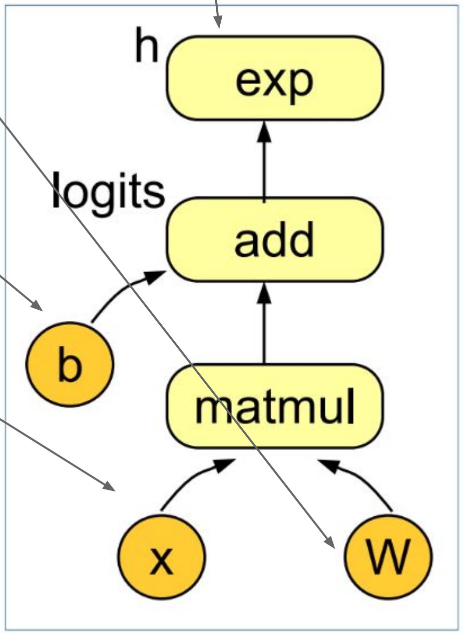
- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

Labels: positive or negative…

$$h = \exp(xW + b)$$

Variables: nodes which output their current value (**Parameters**)

Inputs: words, sentences, images…



Introducing TensorFlow

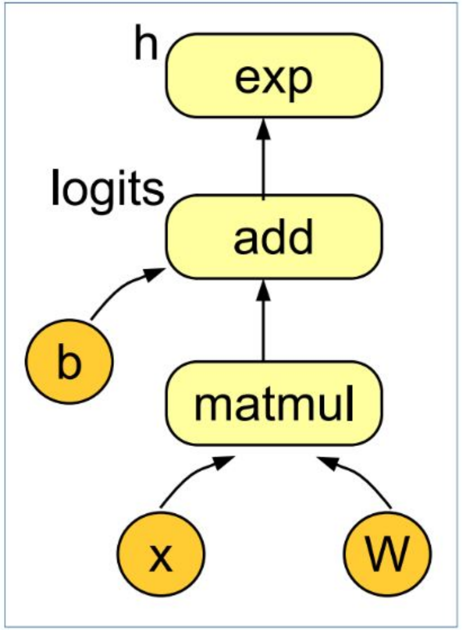
Sentiment analysis:

- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

$$h = \exp(xW + b)$$

Mathematical operations:

Matmul: matrix multiplication
Add: add elementiwse (with broadcasting)
Exp: exponential elementwise

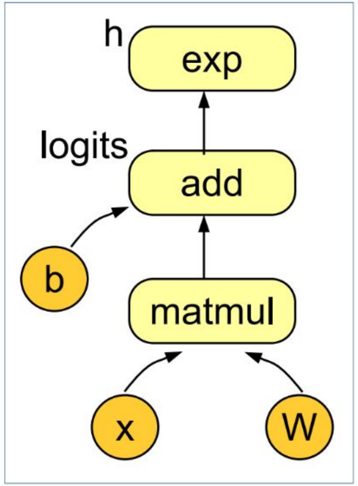


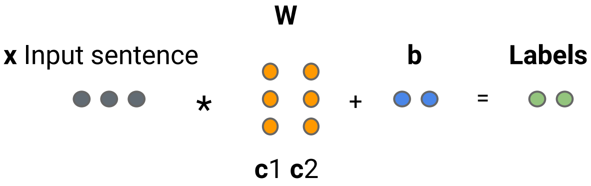
Introducing TensorFlow

Sentiment analysis:

- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

$$h = \exp(xW + b)$$



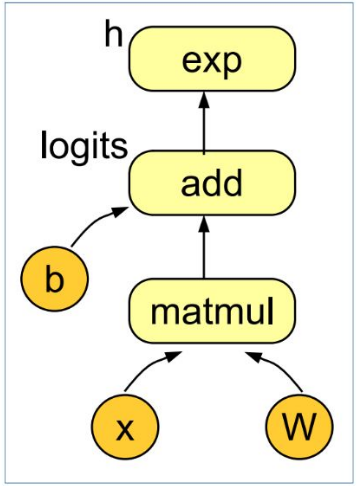


Introducing TensorFlow

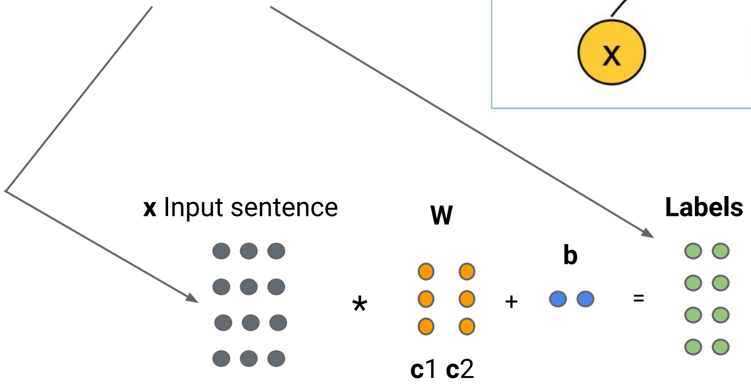
Sentiment analysis:

- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

$$h = \exp(xW + b)$$



minibatch! -> how many?

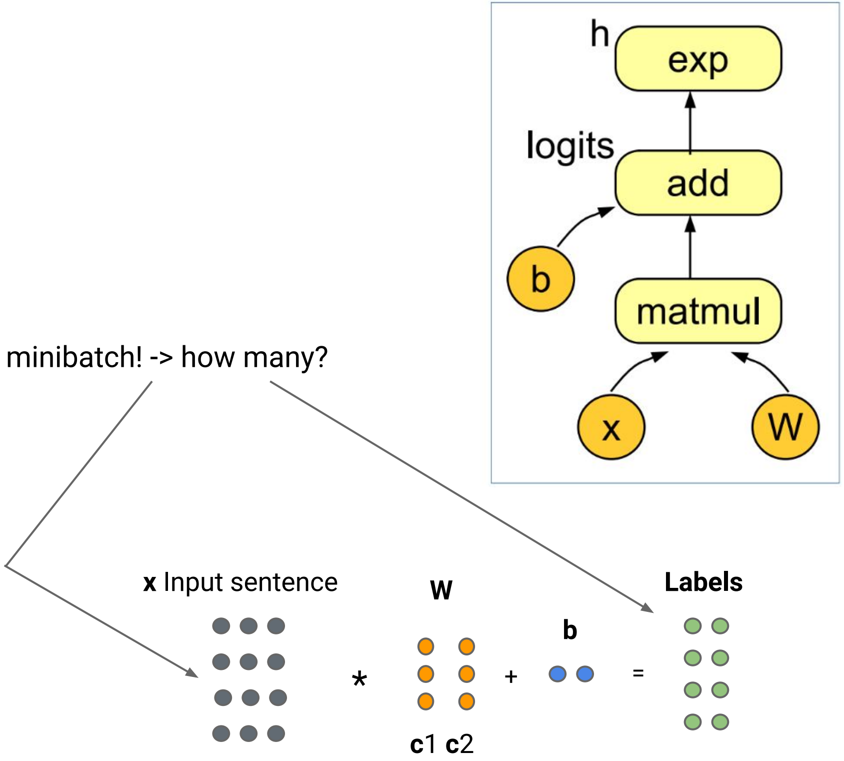


Introducing TensorFlow

Sentiment analysis:

- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

$$h = \exp(xW + b)$$



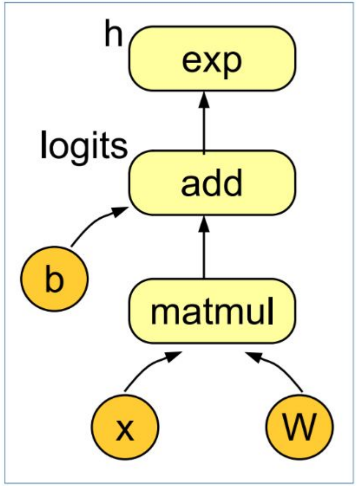
broadcasting! -> [1,1] + 2 = [3,3]

Introducing TensorFlow

Sentiment analysis:

- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

$$h = \exp(xW + b)$$



TF Code

```
import tensorflow as tf
x = loadtrainexamples()
W = tf.Variable(tf.zeros([3, 2]))
b = tf.Variable(tf.zeros([2]))

def model(x):
logits = tf.matmul(x, W) + b
h = tf.exp(logits)
```

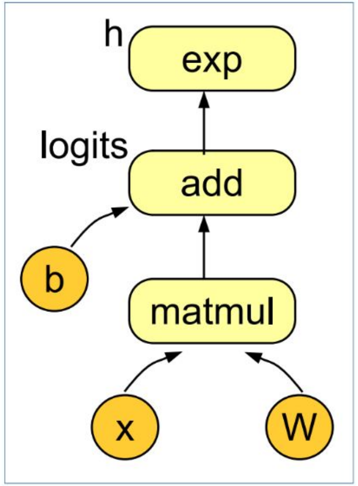
Introducing TensorFlow

Sentiment analysis:

- Input: sentence (3 features)
- Labels: polarity (2 classes p|n)
- Variables:
  - W (3x2)
  - b (2)

TF has constructed a graph! (visualize with TensorBoard)

$$h = \exp(xW + b)$$



TF Code

```
import tensorflow as tf
x = loadtrainexamples()
W = tf.Variable(tf.zeros([3, 2]))
b = tf.Variable(tf.zeros([2]))

def model(x):
logits = tf.matmul(x, W) + b
h = tf.exp(logits)
```

Introducing TensorFlow

- Steps to implement a NN:
  - Prepare the input and specify dimensions
  - Define the architecture (graph)
  - **Train**
    - Specify Optimizer & loss
    - Manage epochs and mini-Batches
  - **Test** & evaluate

Introducing TensorFlow

Training

- Define loss

```
def myCrossEntropy(logits,y)
return tf.reduce_mean(
tf.nn.sparse_softmax_cross_entropy_with_logits(
logits=logits,labels=y))
```

- Compute loss

```
y = loadtrainlabels()
cost=crossEntropy(logits,y)
```

- Compute gradients and optimize
    - No need to do derivates manually :)

```
with tf.GradientTape() as tape:
        logits = model(x)
        cost = myCrossEntropy(logits,y)
gradients = tape.gradient(cost, [W,b])

optimizer = tf.optimizers.SGD(learning_rate)
optimizer.apply_gradients(zip(gradients, [W,b]))
```

Introducing TensorFlow

Training & Test

- Manage epochs and mini-batches

```
#train
for i in range(1000): #epochs!
        batch_x,batch_y = data.next_batch(x,y) #batches!
        with tf.GradientTape() as tape:
                logits = tf.matmul(x, W) + b
                cost = tf.reduce_mean(
                tf.nn.sscewlogits(logits=logits, labels=y))
        gradients=tape.gradient(cost,[W,b])
        optimizer.apply_gradients(zip(gradients,[W,b]))

#test
logits = model(new_examples)
np.argmax(logits, axis=1) #nicer output
#compute accuracy
```

This is already done in the first assignment! check it out!

Introducing TensorFlow

Training & Test

- Summary of steps
  - Prepare the input data and specify dimensions
  - Define model architecture (graph)
    - Weights and operations
  - Train
    - Optimizer & loss
    - Manage epochs and mini-batches
  - Test & eval

Introducing TensorFlow

Training & Test

- Summary of steps
  - **Prepare the input** data and spicify dimensions
  - **Define model** architecture (graph)
    - Weights and operations
  - **Train**
    - Optimizer & loss
    - Manage epochs and mini-batches
  - **Test** & eval

Thanks!