Deep Learning Basics

Slides to accompany the Pytorch exercises

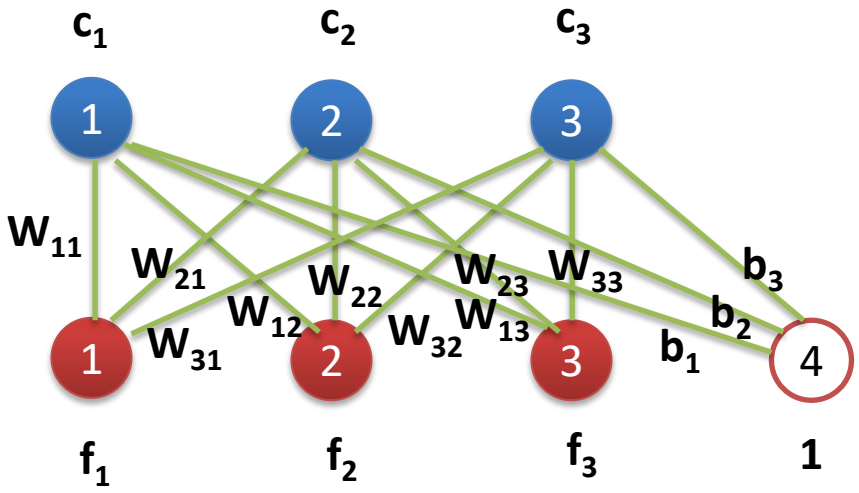
What is deep learning?

- **Deep learning** refers to the use of **artificial neural networks** with **more than one layer** of neurons (interconnections).

features

Neural Networks

Outputs = **c** ; Inputs = **f ; Neurons = W**



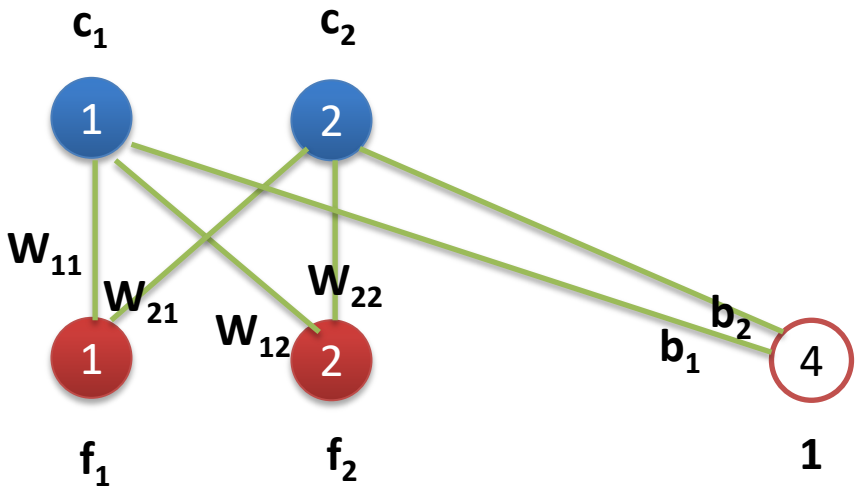
Operations:

1. each neuron (interconnection) has a **weight** = W
2. it contributes the **weighted** input value f to the output => f * W
3. each output is the **sum** of the contributions of all incoming neurons …

   c = sum of neuron contributions = sum of f * W

Neural Networks Example

Outputs = **c** ; Inputs = **f ; Neurons = W**

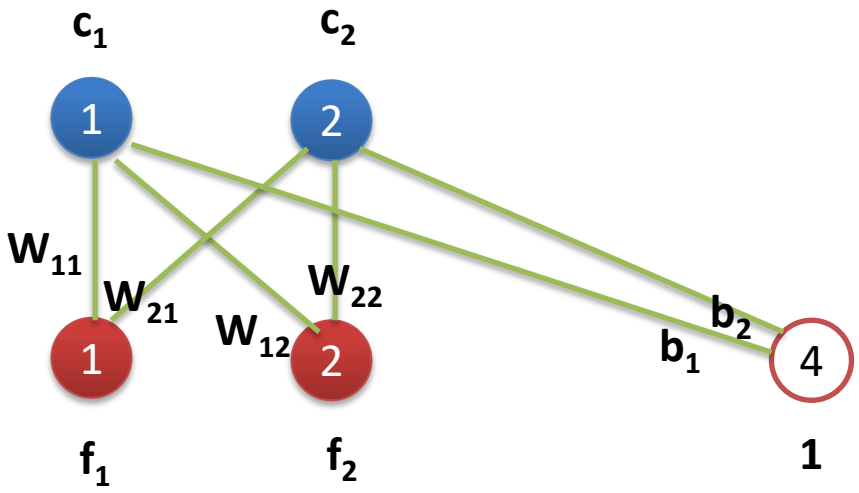


$f_1 = 1$ $f_2 = 2$ $W_{11} = 3$ $W_{21} = 7$ $W_{12} = 4$ $W_{22} = 1$ $b_1 = 0.5$ $b_2 = 0.3$

What are c_1 and c_2?

Neural Networks Example

Outputs = **c** ; Inputs = **f ; Neurons = W**



$f_1 = 1$ $f_2 = 2$ $W_{11} = 3$ $W_{21} = 7$ $W_{12} = 4$ $W_{22} = 1$ $b_1 = 0.5$ $b_2 = 0.3$

What are c_1 and c_2?

$c_1 = 1 * 3 + 2 * 4 + 0.5 = 11.5$

$c_2 = 1 * 7 + 2 * 1 + 0.3 = 9.3$

5

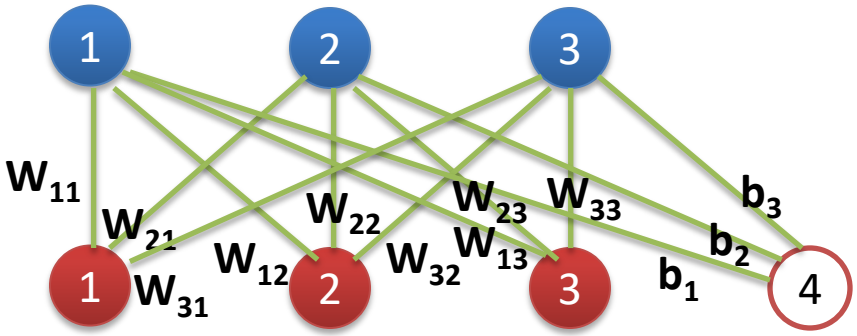
Neural Networks

- Now look at those equations you solved carefully.
- Do you see a set of linear equations?

Neural Networks

Outputs = **c** ; Inputs = **f ; Neurons = W**

Classes



Features

$$c_1 = f_1 W_{11} + f_2 W_{12} + f_3 W_{13} + b_1$$

$$c_2 = f_1 W_{21} + f_2 W_{22} + f_3 W_{23} + b_2$$

$$c_3 = f_1 W_{31} + f_2 W_{32} + f_3 W_{33} + b_3$$

Neural Networks

Outputs **c** are a linear combination of inputs **f** …

$$c_1 = f_1 W_{11} + f_2 W_{12} + b_1$$
$$c_2 = f_1 W_{21} + f_2 W_{22} + b_2$$

$f_1 = 1$ $f_2 = 2$ $W_{11} = 3$ $W_{21} = 7$ $W_{12} = 4$ $W_{22} = 1$ $b_1 = 0.5$ $b_2 = 0.3$

$c_1 = 1 * 3 + 2 * 4 + 0.5 = 11.5$
$c_2 = 1 * 7 + 2 * 1 + 0.3 = 9.3$

Neural Networks

Outputs **c** are a linear combination of inputs **f** …

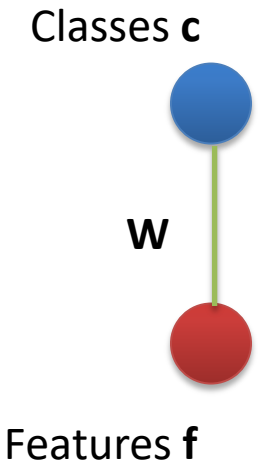
$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

$$\begin{bmatrix} f_1 = 1 & f_2 = 2 \end{bmatrix} \begin{bmatrix} W_{11} = 3 & W_{21} = 7 \\ W_{12} = 4 & W_{22} = 1 \end{bmatrix} \begin{bmatrix} b_1 = 0.5 & b_2 = 0.3 \end{bmatrix}$$

$$\begin{bmatrix} c_1 = 1 * 3 + 2 * 4 + 0.5 = 11.5 & c_2 = 1 * 7 + 2 * 1 + 0.3 = 9.3 \end{bmatrix}$$

Neural Networks

Outputs **c** are a linear combination of inputs **f** …



$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

See how simple this looks →

$$c = f\,W + b$$

Neural Networks

Outputs **c** are a linear combination of inputs **f** …

$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_2 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \\ W_{13} & W_{23} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

$$\mathbf{c} = \mathbf{fW} + \mathbf{b}$$

$f_1 = 1 \quad f_2 = 2 \quad f_3 = 3$

$W_{11} = 3 \quad W_{21} = 7$
$W_{12} = 4 \quad W_{22} = 1$
$W_{13} = 1 \quad W_{23} = 2$

$b_1 = 0.5 \quad b_2 = 0.3$

$c_1 = ?$
$c_2 = ?$

Try this out!

Also do this in Pytorch – exercise 210.

Machine Learning

- There are three broad categories of machine learning algorithms:
  - Supervised
  - Unsupervised
  - Reinforcement-Learning

Machine Learning

- Supervised learning is where you're given inputs and corresponding output labels picked from a finite set of labels (and these labels are not part of the inputs).
- Unsupervised learning is where you have the inputs but have to predict some part of the inputs given the other parts, or some grouping of the inputs, etc. You just don't have any labels or external output values in your training data.
- Reinforcement learning is where the feedback is limited to a reward (or a whack in the rear). You're not told what the right answer was.

Machine Learning

Categories of Machine Learning >>

Supervised

Unsupervised

Reinforcement

Supervised Machine Learning

Categories of Supervised Machine Learning >>

Classification

We're going to do this now!

Regression

The **Classifier**

What is a Classifier?

Something that performs classification.

Classification = categorizing
Classification = deciding
Classification = labelling

Classification = Deciding = Labelling

Classification = Deciding = Labelling

5’ 8”

6’2”

5’11”

5’11”

5’8”

6’6”

5’ 2”

6’9”

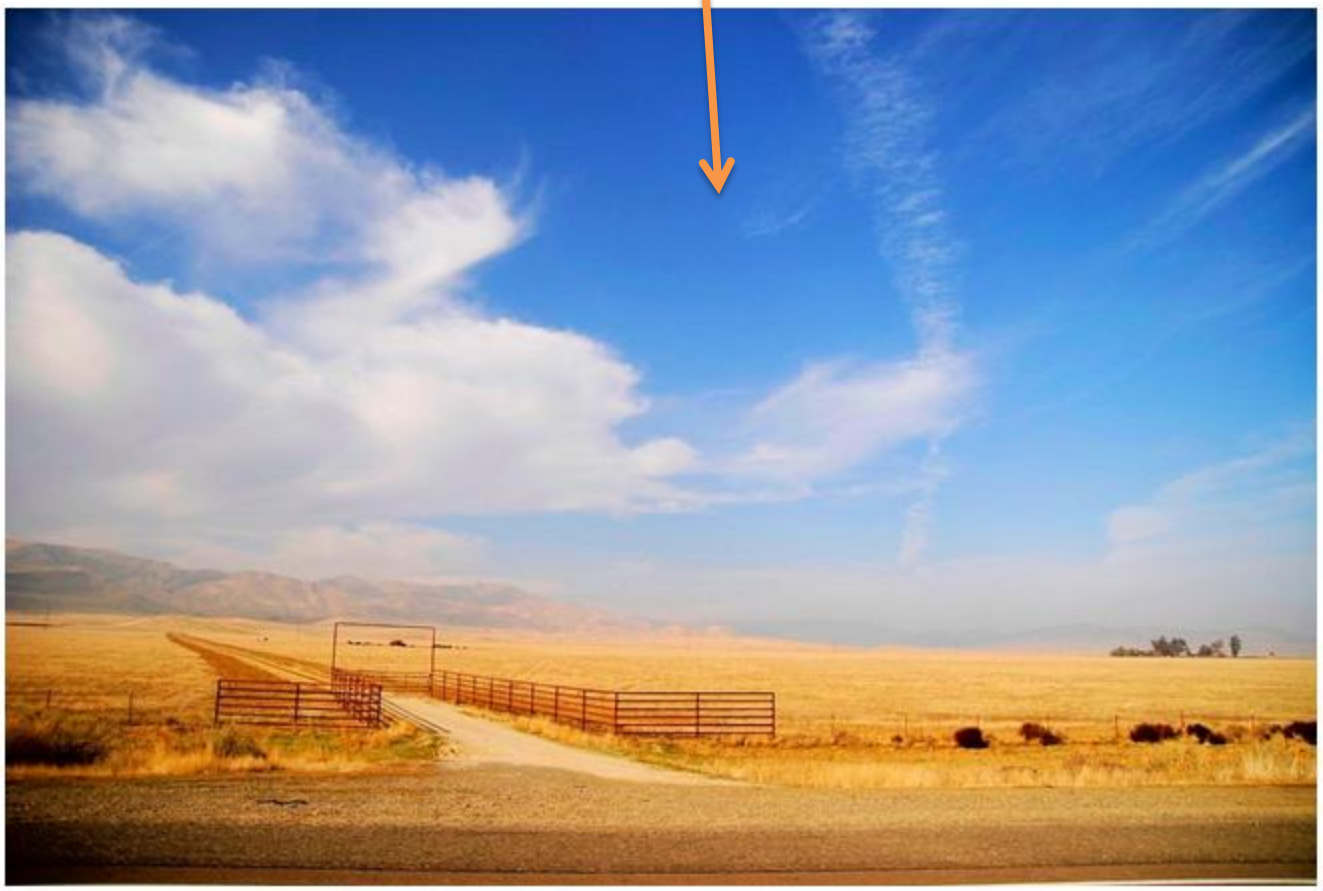
6’8”

6’10”

Classify these door heights as: Short or Tall ?

What is Classification?

What colour do you see here?



Russell Mondy/Flickr

What is Classification?

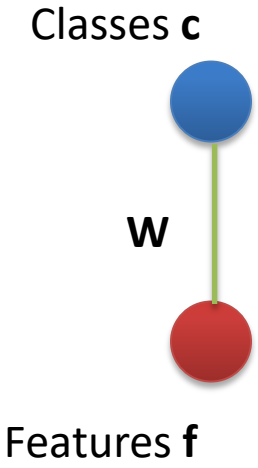
Classification = Categorizing = Labelling = Deciding

Russell Mondy/Flickr

Neural Networks as Classifiers

Since classification is deciding …

Could you use the outputs **c** to make a hard decision? How would you do it?



$$\begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \\ W_{13} & W_{23} & W_{33} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

These are the equations you have already gone over.

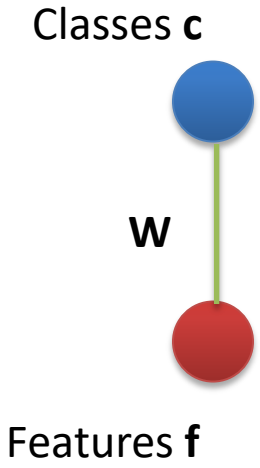
You take some inputs $f_1\ f_2\ f_3$ …

and get some real numbers as outputs .. $c_1\ c_2\ c_3$.

Can you use the values of $c_1\ c_2\ c_3$ to make a decision?

Neural Networks as Classifiers

Could you use the outputs **c** to make a hard decision? How would you do it?

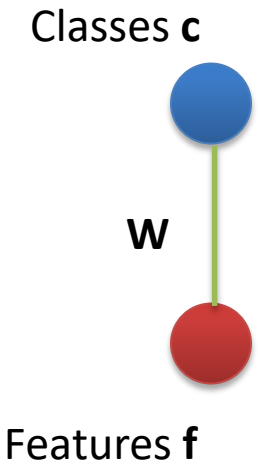


$$\begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \\ W_{13} & W_{23} & W_{33} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

Can you use the values of $c_1\ c_2\ c_3$ to make a decision?
Hint: $c_1\ c_2\ c_3$ stand for category 1, category2, category 3.

Neural Network Classifiers

Yes, you can. If the outputs **c** are preferences for categories …

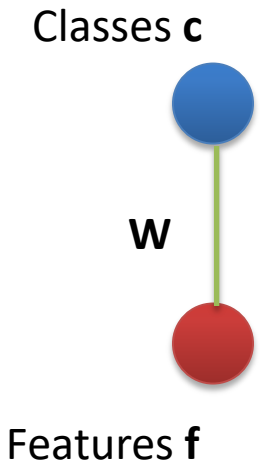


$$\begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \\ W_{13} & W_{23} & W_{33} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

You can just say that the category (output) favoured by the neural network is the one with the highest value. So, the category output by the classifier is $\text{argmax}_n\ (c_n)$

Neural Network Classifiers

Outputs **c** are a linear combination of inputs **f** …



$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_2 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \\ W_{13} & W_{23} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix}$$

The category output by the classifiers is $\text{argmax}_n\ (c_n)$

$f_1 = 1 \quad f_2 = 2 \quad f_3 = 3$

$W_{11} = 3 \quad W_{21} = 7$
$W_{12} = 4 \quad W_{22} = 1$
$W_{13} = 1 \quad W_{23} = 2$

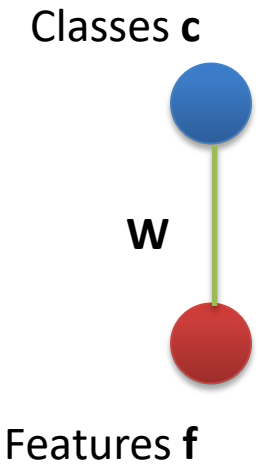
$b_1 = 0.5 \quad b_2 = 0.3$

$c_1 = ?$
$c_2 = ?$

Which category did the classifier output?

Do this in Pytorch – exercise 310.

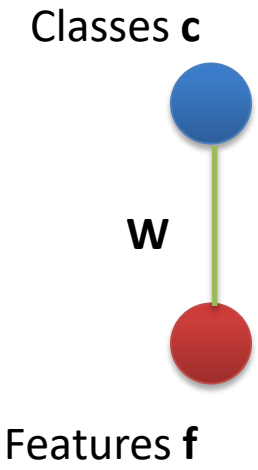
Neural Network Classifiers Problem 1



$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \end{bmatrix}$$

Come up with weights such that if $f_1 > f_2$ the classifier will select c_1 else c_2 !

Neural Network Classifiers Problem 1



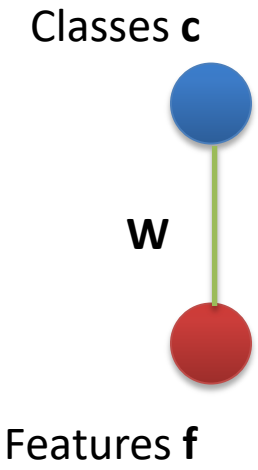
Example 1: since f1 < f2 below,

c1 must be less than c2.

$f_1 = 1 \quad f_2 = 2$ $\qquad\qquad$ $c_1 < c_2$

When $f_1 < f_2$ … $c_1 < c_2$ so $\text{argmax}_n\ (c_n) = 2$ (the categories being 1 & 2).

Neural Network Classifiers Problem 1



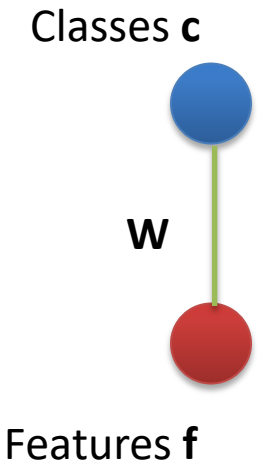
Example 2: if f1 > f2, then

c1 must be more than c2.

$f_1 = 4$ $f_2 = 2$ $c_1 > c_2$

When $f_1 > f_2$ … $c_1 > c_2$ so $\text{argmax}_n (c_n) = 1$ (the categories being 1 & 2).

Neural Network Classifiers Problem 1



Come up with weights such that if $f_1 > f_2$ the classifier will select c_1 else c_2 !

$W_{11} = ?$ $W_{21} = ?$

$W_{12} = ?$ $W_{22} = ?$

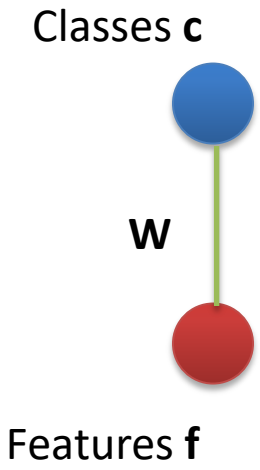
Find the weights so that $f_1 < f_2$ => $c_1 < c_2$

Try this in Pytorch – exercise 350.

Classification

- Here's the second fun problem!!!

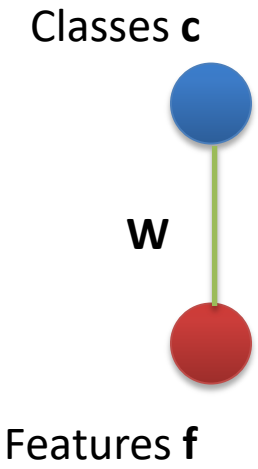
Neural Network Classifiers Problem 2



$$\begin{bmatrix} c_1 & c_2 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_2 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \\ W_{13} & W_{23} \end{bmatrix}$$

Come up with weights such that if $f_1 > f_2 + f_3$ the classifier will select c_1 else c_2 !

Neural Network Classifiers Problem 2



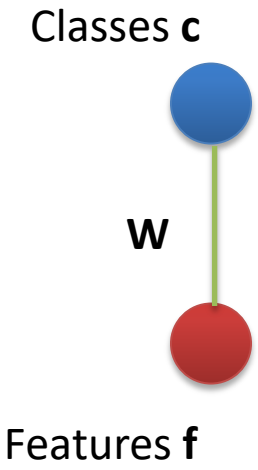
Example: since f1 < f2 + f3

below, c1 must be less than c2.

$f_1 = 4 \quad f_2 = 2 \quad f_3 = 3 \qquad c_1 < c_2$

Since $f_1 < f_2 + f_3$, $c_1 < c_2$ so $\text{argmax}_n\ (c_n) = 2$
(the categories being 1 & 2)

Neural Network Classifiers Problem 2



Come up with weights such that if $f_1 > f_2 + f_3$ the classifier will select c_1 else c_2 !

$$W_{11} = ? \quad W_{21} = ?$$
$$W_{12} = ? \quad W_{22} = ?$$
$$W_{13} = ? \quad W_{23} = ?$$

Find the weights so that $f_1 < f_2 + f_3 \Rightarrow c_1 < c_2$

Try this in Pytorch – exercise 380.

Classifiers

- So now you know how a classifier works!
- But the numbers you're feeding in as input are not devoid of meaning.
- Let's take a look at the inputs in the last problem …

$$f_1 = 4 \quad f_2 = 2 \quad f_3 = 3$$

- They could represent a text document!!!
- Want to know how?

Neural Network Document Classifiers

Say a language has only the words “a”, “b” and “c”. Now, in a document …

$\mathbf{f_1}$ = count of “a” $\mathbf{f_2}$ = count of “b” $\mathbf{f_3}$ = count of “c”
$\mathbf{category_1}$ = Sports $\mathbf{category_2}$ = Politics

- Question: What are f1, f2 and f3 for each of the following documents?
- a b a c a b c a c
- a b a a c

Neural Network Document Classifiers

Say a language has only the words “a”, “b” and “c”. Now, in a document …

$\mathbf{f_1}$ = count of “a” $\mathbf{f_2}$ = count of “b” $\mathbf{f_3}$ = count of “c”
$\mathbf{category_1}$ = Sports $\mathbf{category_2}$ = Politics

- Answer:

- a b a c a b c a c — $\mathbf{f_1} = 4, \mathbf{f_2} = 2, \mathbf{f_3} = 3$
- a b a a c — $\mathbf{f_1} = 3, \mathbf{f_2} = 1, \mathbf{f_3} = 1$

Neural Network Document Classifiers

- a b a c a b c a c — $\mathbf{f_1} = 4, \mathbf{f_2} = 2, \mathbf{f_3} = 3$
- a b a a c — $\mathbf{f_1} = 3, \mathbf{f_2} = 1, \mathbf{f_3} = 1$

- Now, if you apply the weights you came up with for Problem 2 to these inputs, what classes would you find these documents belonging to?

$f_1 > f_2 + f_3$ => c1

$f_1 <= f_2 + f_3$ => c2

Neural Network Document Classifiers

$f_1 > f_2 + f_3$ => c1

$f_1 <= f_2 + f_3$ => c2

$W_{11} = 1$ $W_{21} = 0$
$W_{12} = 0$ $W_{22} = 1$
$W_{13} = 0$ $W_{23} = 1$

Compute c1 and c2 then do argmax(c1, c2)

- **class of $f_1 = 4, f_2 = 2, f_3 = 3$** => **?**
- **class of $f_1 = 3, f_2 = 1, f_3 = 1$** => **?**

Neural Network Document Classifiers

$f_1 > f_2 + f_3$ => c1

$f_1 <= f_2 + f_3$ => c2

- $f_1 = 4, f_2 = 2, f_3 = 3$ => **category$_2$**
- $f_1 = 3, f_2 = 1, f_3 = 1$ => **category$_1$**

Neural Network Document Classifiers

$\mathbf{f_1}$ = count of “a” $\mathbf{f_2}$ = count of “b” $\mathbf{f_3}$ = count of “c”
$\mathbf{c_1}$ = Sports $\mathbf{c_2}$ = Politics

- a b a c a b c a c => Politics
- a b a a c => Sports

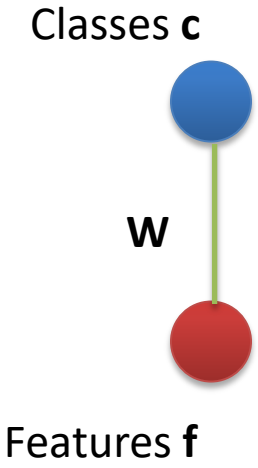
Text Classification

- So you've learnt how text can be represented as a vector of numbers.

$$f_1 = 4 \quad f_2 = 2 \quad f_3 = 3$$

- You've also learnt to do topic classification.
- Provided someone gives you the weights!
- But documents have a vocabulary of thousands of words, so the weight matrix can be very large. It would be very difficult to come up with a good one manually.
- Is there a better way to come up with a weight matrix?
- Can you show the neural network some examples and ask it to come up with a weight matrix?
- Yes, you can. That's called training a neural network.

Training a Neural Network



- How do you learn the weights automatically?
  - Step 1: Get some **training data**
  - Step 2: Create a **loss function** reflecting the badness of the neural net
  - Step 3: Adjust the weights to **minimize the loss (error)** on the training data

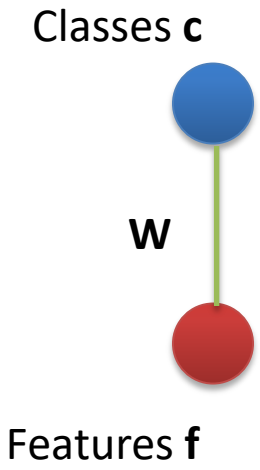
Let the **machine learn** weights such that if $f_1 > f_2 + f_3$ the classifier will select c_1 else c_2 !
Give it examples (training data) + tell it which is the right way up + let it climb up on its own.

$f_1 = 4 \quad f_2 = 2 \quad f_3 = 3$

$W_{11} = ? \quad W_{21} = ?$
$W_{12} = ? \quad W_{22} = ?$
$W_{13} = ? \quad W_{23} = ?$

$c_1 < c_2$
So $\text{argmax}_n\ (c_n) = 2$

Step 1: Get some training data

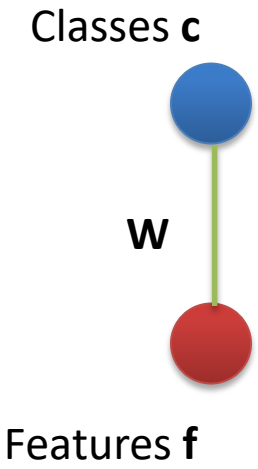


- That's easy (for us).
- We can generate it!

c = 1	$f_1 = 4$	$f_2 = 2$	$f_3 = 3$
c = 0	$f_1 = 2$	$f_2 = 1$	$f_3 = -3$
c = 0	$f_1 = -2$	$f_2 = -1$	$f_3 = -3$
c = 1	$f_1 = 3$	$f_2 = 1$	$f_3 = 7$
c = 1	$f_1 = -3$	$f_2 = 1$	$f_3 = 0$
c = 0	$f_1 = 3$	$f_2 = 1$	$f_3 = 1$

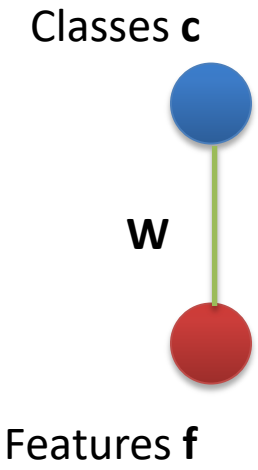
Try this in Pytorch – exercise 410.

Step 2: Create a loss function



- A loss function is a function that reflects the degree of incorrectness of the machine learning algorithm.
- To make Step 3 easy, this loss function thing has to be differentiable.

Step 2: Create a loss function



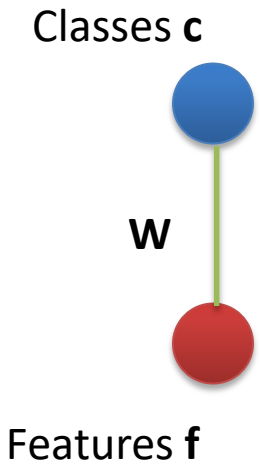
- There're many such functions and one that's usually used with classifiers is "cross-entropy".

$$H(p, q) = -\sum_{x} p(x) \log q(x).$$

p is the one-hot encoding of the correct class

q is the softmax of the classifier's output

Step 2: Create a loss function



Cross entropy:

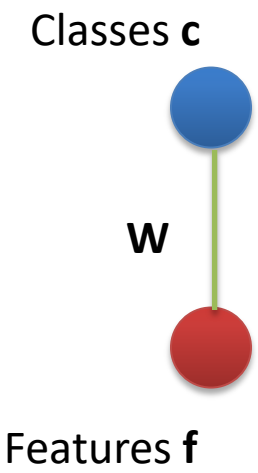
$$H(p,q) = -\sum_x p(x)\ \log q(x).$$

For classifiers:

p is the one-hot encoding of the correct class

q is the softmax of the classifier's output

Step 2: Create a loss function



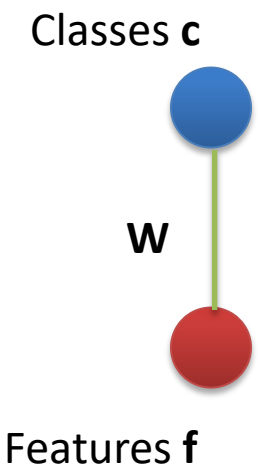
<u>One-hot encoding of a number</u>

It is just a vector with a 1 in the position of that number and 0 in the positions of all other numbers.

<u>One-hot encoding of a category number</u>

A vector with a 1 in the position of that category and 0s at the positions of all other categories.

Step 2: Create a loss function



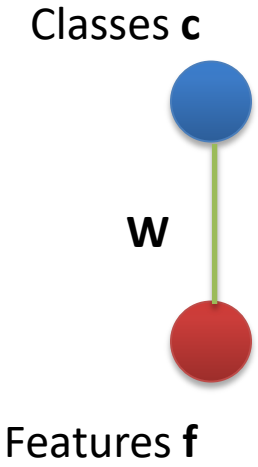
Say we are deciding between 3 classes.

The one hot encoding of category 0 is [1, 0, 0]

The one hot encoding of category 1 is [0, 1, 0]

What is the one-hot encoding of category 2?

Step 2: Create a loss function



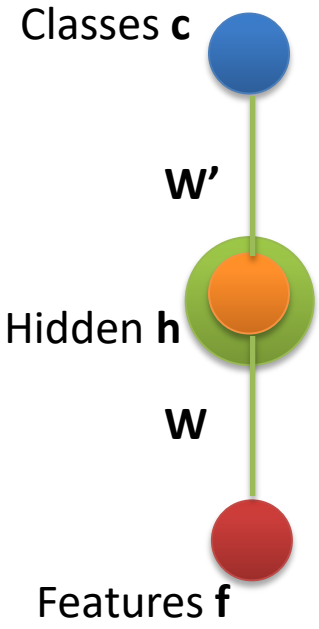
Cross entropy:

$$H(p, q) = -\sum_{x} p(x) \log q(x).$$

For classifiers, **q** is the softmax of the classifier's output

$$\mathbf{q}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Softmax



Softmax

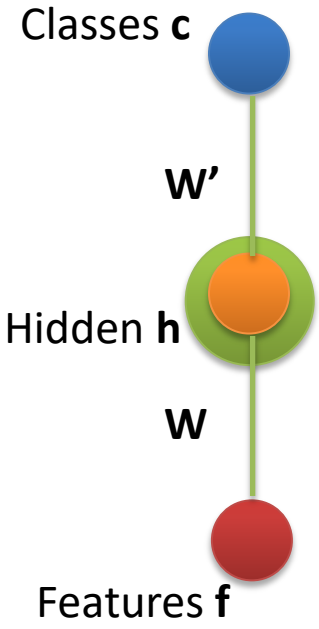
h = q(z) = softmax(z)

Squishes a set of real numbers into probabilities in the range (0,1)

$$\mathbf{q}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Softmax

Cross-Entropy



loss = $H(p,q) = -\sum_x p(x)\,\log q(x).$

If z is the correct category, p(x) = 0 for all x not equal to z

So the summation goes away and you get -p(z) log q(z)

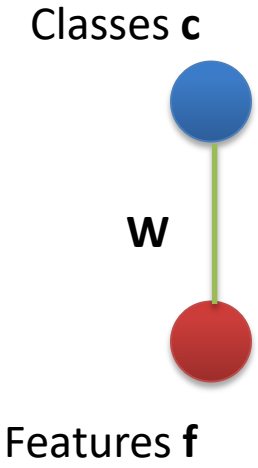
But, p(z) is 1.

So loss = - log q(z). But q is the softmax function … so …

loss = - log (softmax(z))

where z is the output of the correct output node.

Step 2: Create a loss function



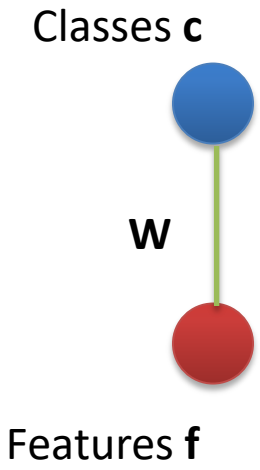
- The cross entropy loss for different data points …

These are more difficult to classify because f1 is only a little more than f2 + f3

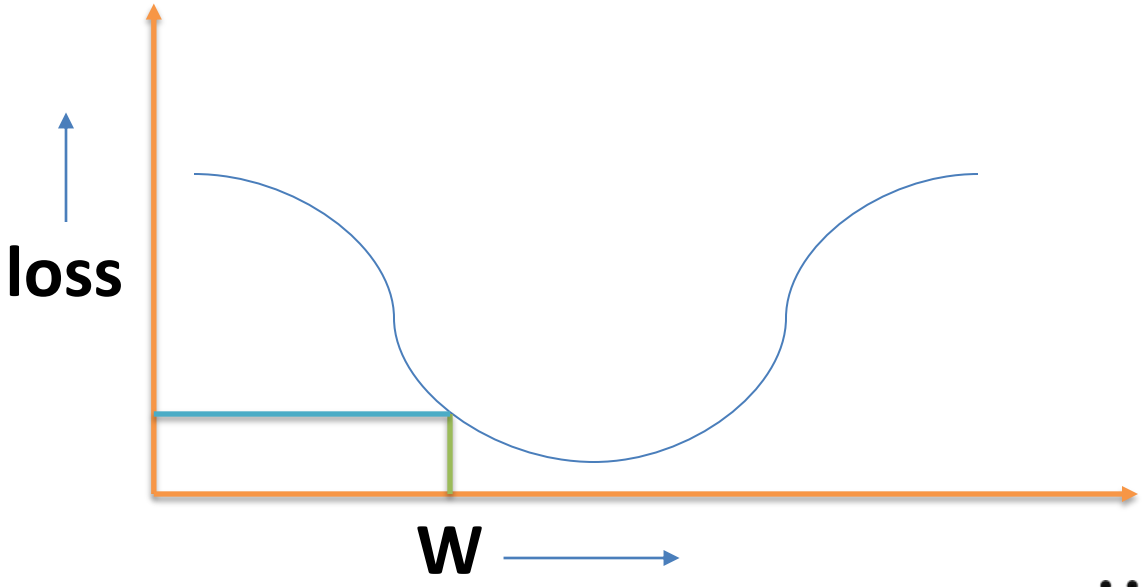
c = 0	$f_1 = 4$	$f_2 = 2$	$f_3 = 1.9$	cross entropy = 0.6444
c = 0	$f_1 = 5$	$f_2 = 2$	$f_3 = 1.9$	cross entropy = 0.2873
c = 0	$f_1 = 6$	$f_2 = 2$	$f_3 = 1.9$	cross entropy = 0.1155
c = 0	$f_1 = 10$	$f_2 = 2$	$f_3 = 1.9$	cross entropy = 0.0022

These are easier to classify because f1 is a lot more than f2 + f3, so the classifier fares better, hence the cross-entropy is lower.

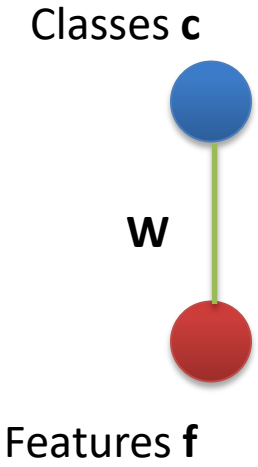
Step 3: Minimize the loss



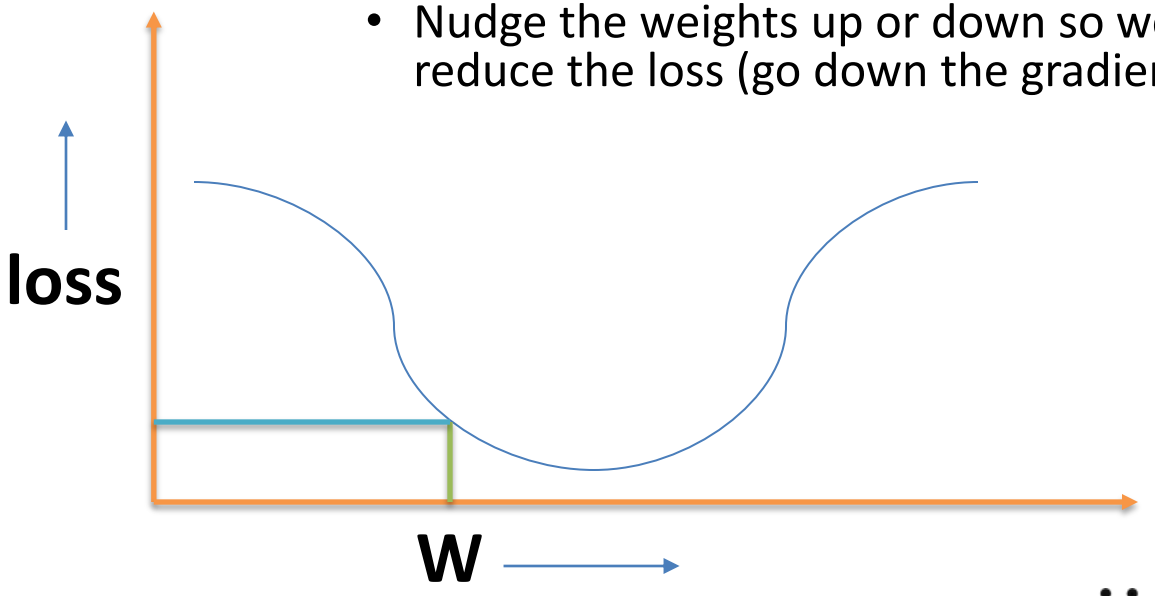
- Now we have a loss function that reflects the degree of incorrectness of a classifier.
- The best classifier is one whose **weights and bias values minimize the loss**.
- Training is nothing but **finding the weights and biases that minimize the loss**.



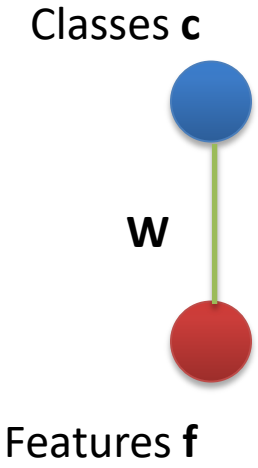
Step 3: Minimize the loss



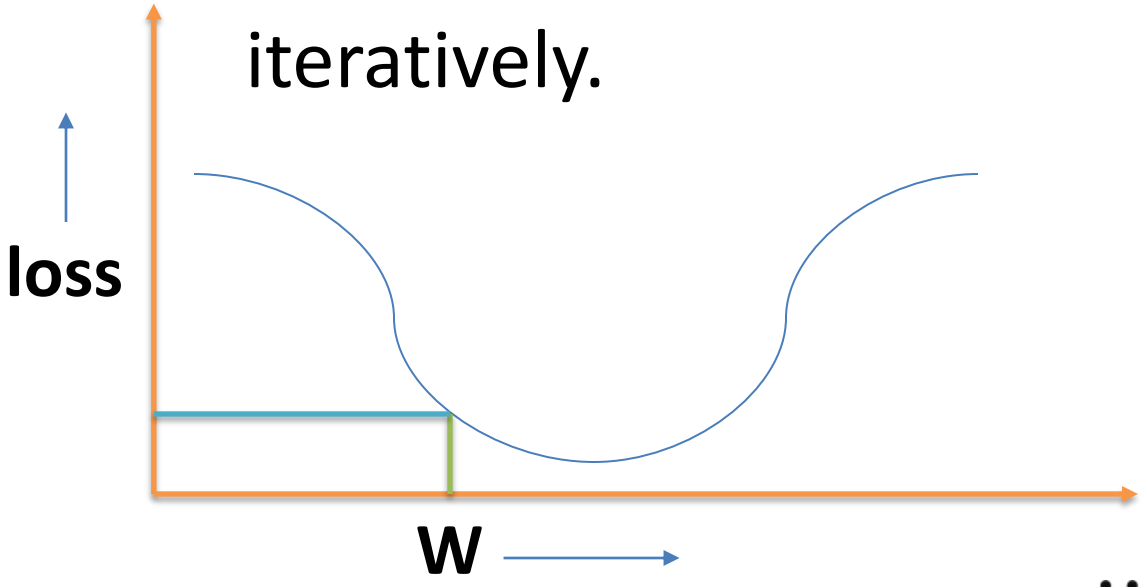
- You can iteratively train a classifier (find the weights and biases that minimize the loss):
  - Start with random values for weights and biases
  - Adjust the weights so the loss (computed by the loss function) decreases.
    - Compute the loss for the current weights.
    - Nudge the weights up or down so we reduce the loss (go down the gradient)!



Step 3: Minimize the loss

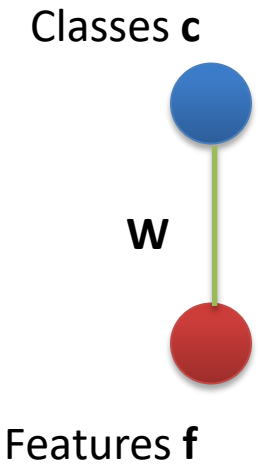


- Let's say the classifier has a scalar weight x … and assume the loss function x^2+3
- Train this classifier (find the x that minimizes the loss) iteratively.

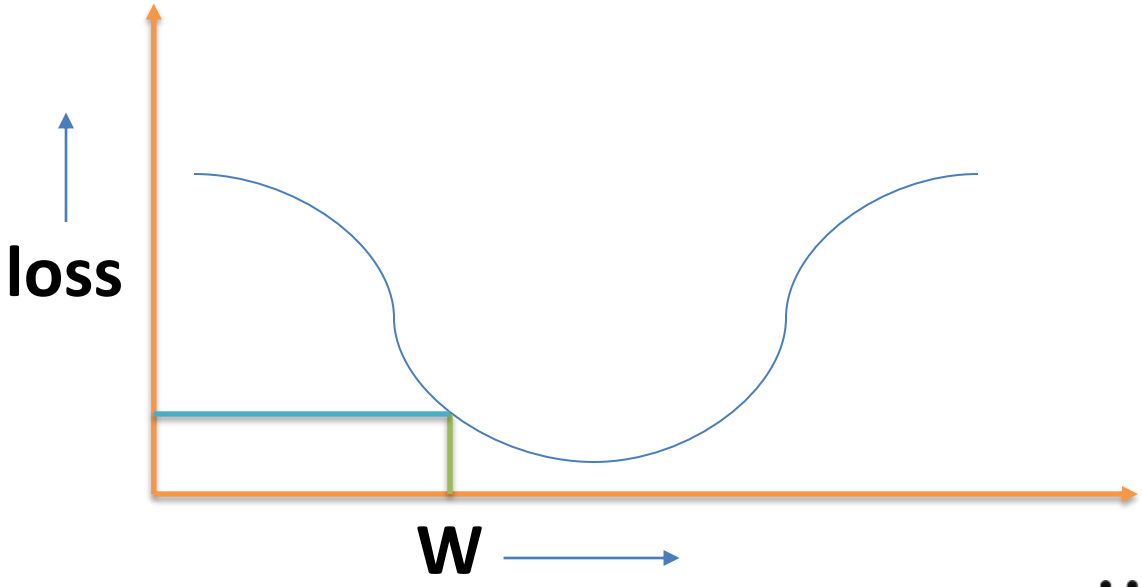


Step 3: Minimize the loss

Forward Pass

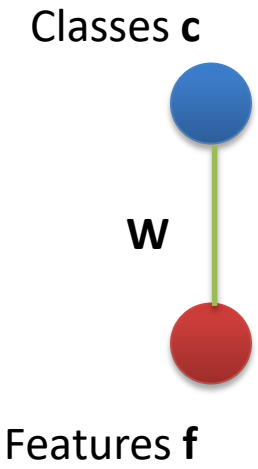


- Start with a random value for **x**
- Let's just start with **x** = **2**
- Compute the loss **x^2 +3** for **x** = **2**. That is **2^2 +3** which is **7**



Step 3: Minimize the loss

Backward Pass

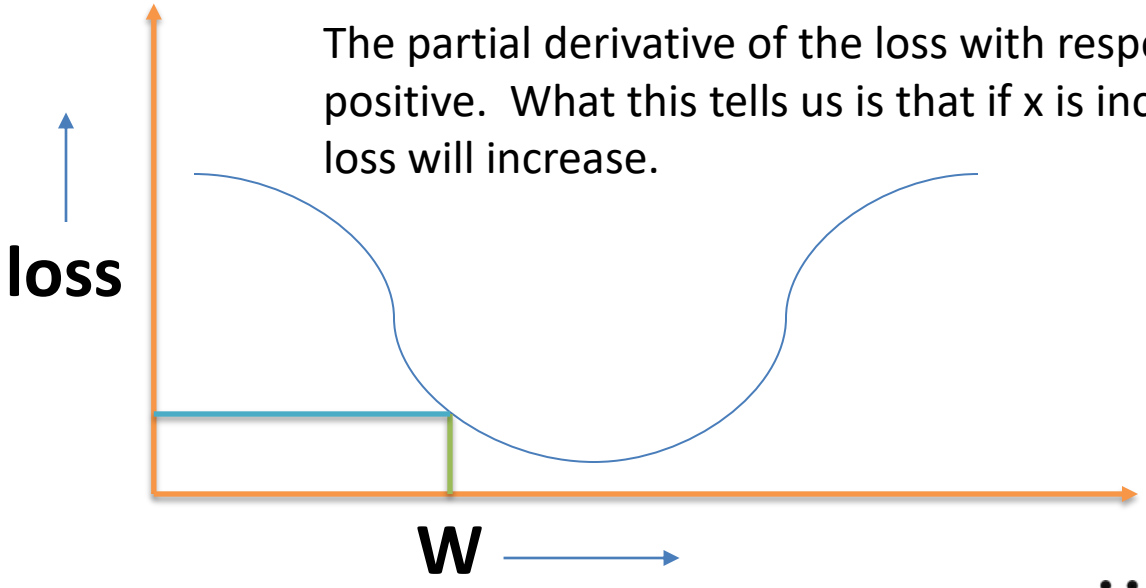


– The partial derivative of the loss with respect to the weight is …

– d loss / dx =

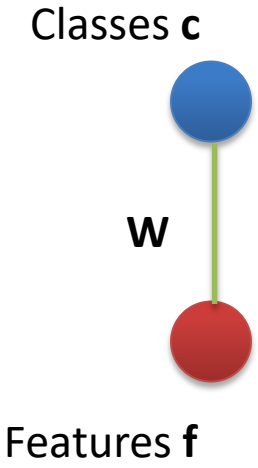
= **d (x^2 + 3) / dx**

= 2*x = **4**



The partial derivative of the loss with respect to x is positive. What this tells us is that if x is increased the loss will increase.

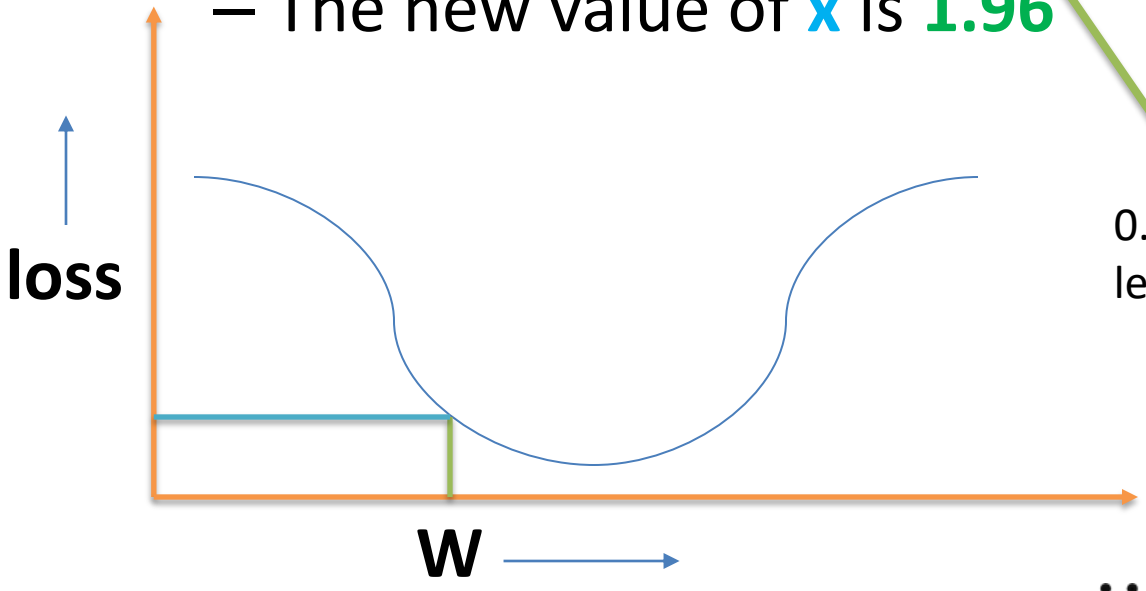
Step 3: Minimize the loss



– Since the loss will increase if x increases, instead decrease the weight x so that the loss decreases … **x** = **x** – ***0.01*** * **4**

= **2** – ***0.01*** * **4**

– The new value of **x** is **1.96**



0.01 is the learning rate

Step 3: Minimize the loss

- Calculate the loss to convince yourself that the loss has decreased.
- We started with x = 2 which gave a loss of 7
- After one iteration, x = 1.96
- The loss is now 6.8416 which is less than 7
- Repeat the forward and backward steps a few hundred times and x will reach (actually approach) 0.

Step 3: Minimize the loss

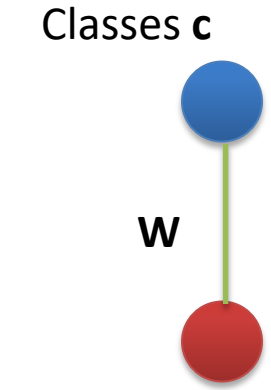
- We just trained a neural network using the backpropagation algorithm!
- But, we did not use any training data.
- How do we take training data into account during training?

Step 3: Minimize the loss

- How do we take training data into account during training?
- It's all in the loss function.
- To take training data into account, use a loss function that involves training data (the parameters to be learnt are the variables and the training data are the constants).

Step 3: Minimize the loss

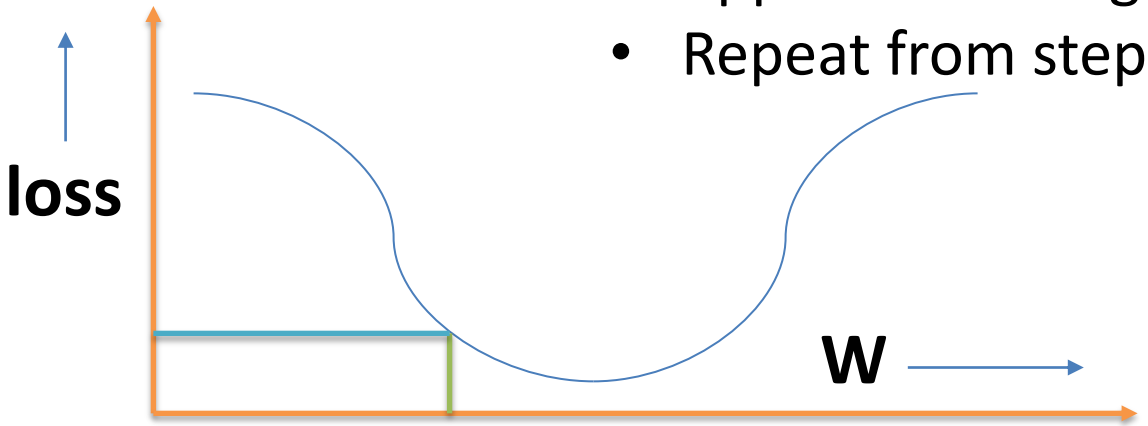
If you have training data ->



Forward Pass

Backward Pass

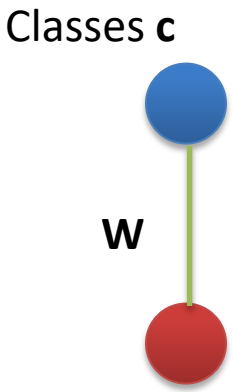
- Start with random values for the variables (weights and biases) to be minimized.
- Randomly select a subset (or one) of the training data as input.
- Compute the loss for that input.
- Calculate the gradient of the loss for each parameter.
- **Nudge the parameter** in a direction opposite to the gradient.
- Repeat from step 2!



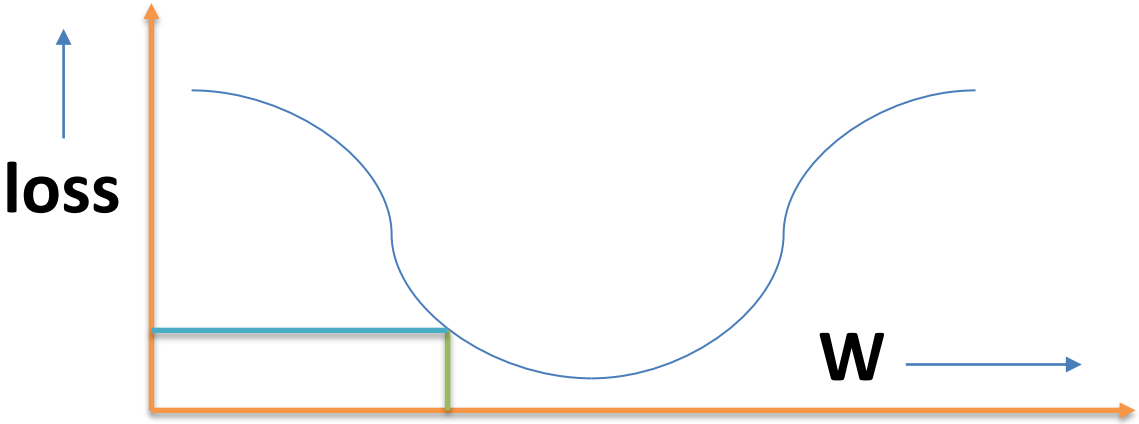
Back-Propagation = Forward Pass + Backward Pass

Nudging the parameters

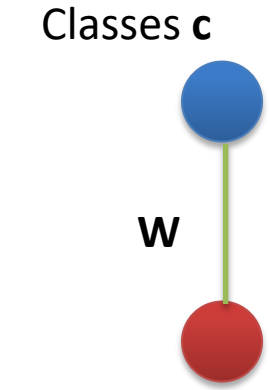
- Stochastic Gradient Descent (SGD)
  - ✓Calculate the gradient
  - ✓$W_{11} = W_{11} - 0.01 * \text{gradient}$



0.01 is the learning rate

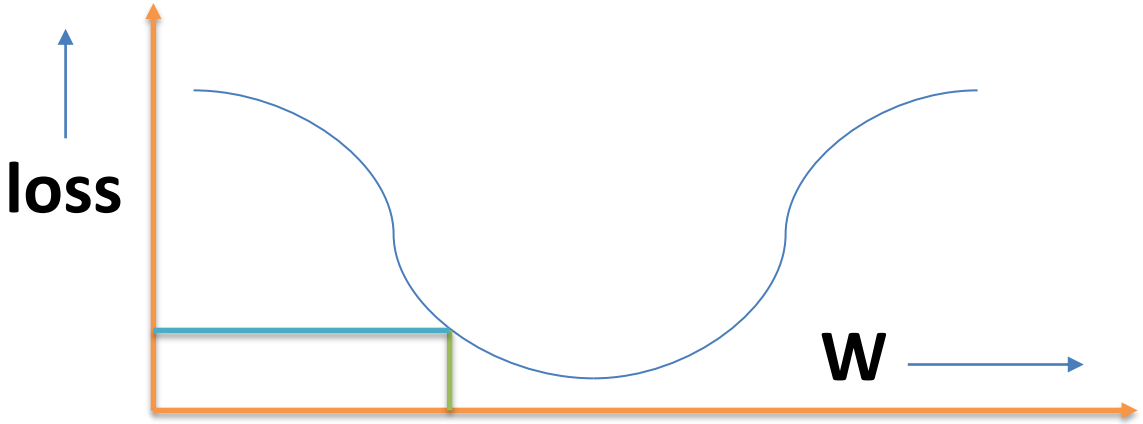


Nudging the parameters

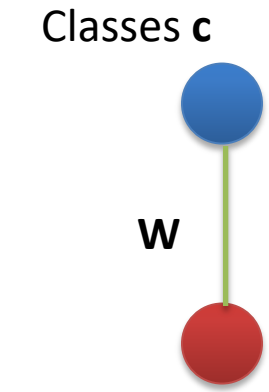


- Adaptive Gradient (AdaGrad)
  - ✓Calculate the gradient
  - ✓$W_{11} = W_{11} - 0.01 * \mathbf{k} *$ gradient

  (where **k** depends on the past gradient history of W_{11})

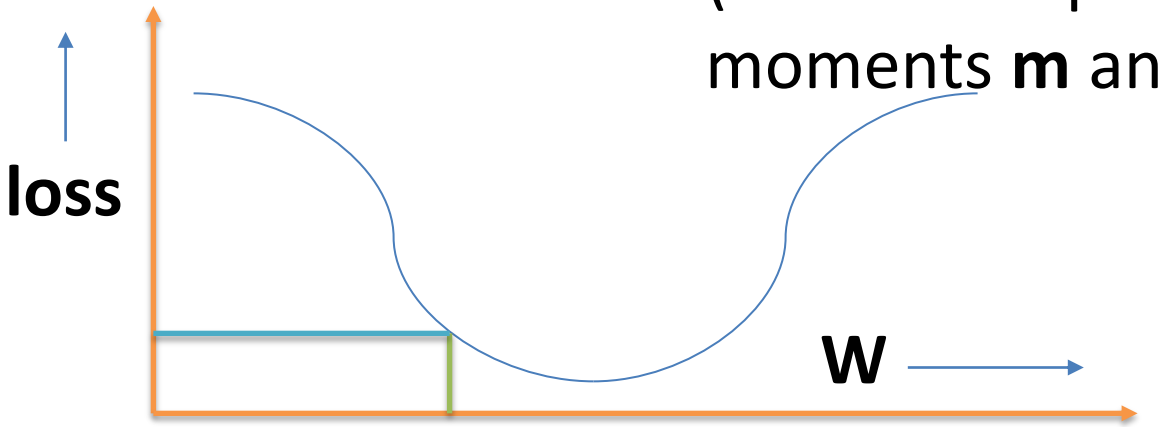


Nudging the parameters



- Adam
  - ✓Calculate the gradient
  - ✓Use it to calculate the first order moment **m** and the second order moment **g**
  - ✓$W_{11} = W_{11} - 0.01 * \mathbf{k}$

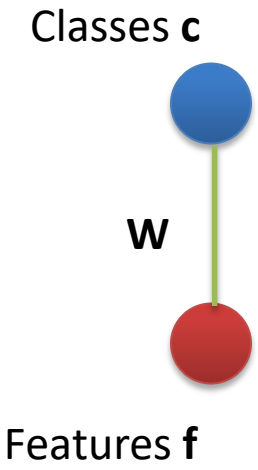
  (where **k** depends on the moments **m** and **g**)



Classification using Neural Networks

- Now, let's train a classifier (find the weights that minimize the loss) for toy datasets based on Toy Problem 1 and Toy Problem 2.
- Pytorch does backpropagation automatically for us, so you only have to construct your neural network, choose the loss function, and for batches of input data, compute the loss. The rest of it is handled automatically by Pytorch.

Neural Network Classifiers Problem 3

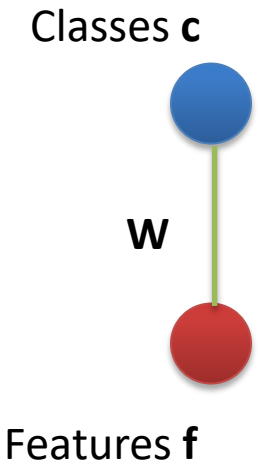


Come up with weights such that if $f_1 * f_2 > 0$ the classifier will select c_1 else c_2 ! Examples >>

$f_1 = 1$ $f_2 = -2$	$c = 2$
$f_1 = 1$ $f_2 = 2$	$c = 1$
$f_1 = -1$ $f_2 = -2$	$c = 1$
$f_1 = -1$ $f_2 = 2$	$c = 2$

… learn the weights by machine learning.

Neural Network Classifiers Problem 3



Using the training data …

$f_1 = 1 \;\; f_2 = -2$	$c = 2$
$f_1 = 1 \;\; f_2 = 2$	$c = 1$
$f_1 = -1 \;\; f_2 = -2$	$c = 1$
$f_1 = -1 \;\; f_2 = 2$	$c = 2$

… we want to learn the weights automatically.

$W_{11} = ?$ $W_{21} = ?$
$W_{12} = ?$ $W_{22} = ?$

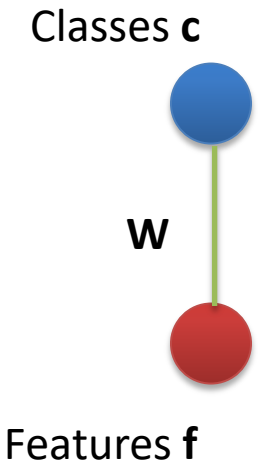
This is a classification task. So, train a classifier!

Neural Network Classifiers Problem 3

- Surprise!
- You will find that try as you might, you cannot train a classifier (with the single layer of neurons) that classifies this dataset correctly.
  - The classifier's loss will not go down
  - When tested on the test dataset, the classification accuracy will hover around 50%
- Why was it not possible to train the classifier?

Neural Network Classifiers Problem 3

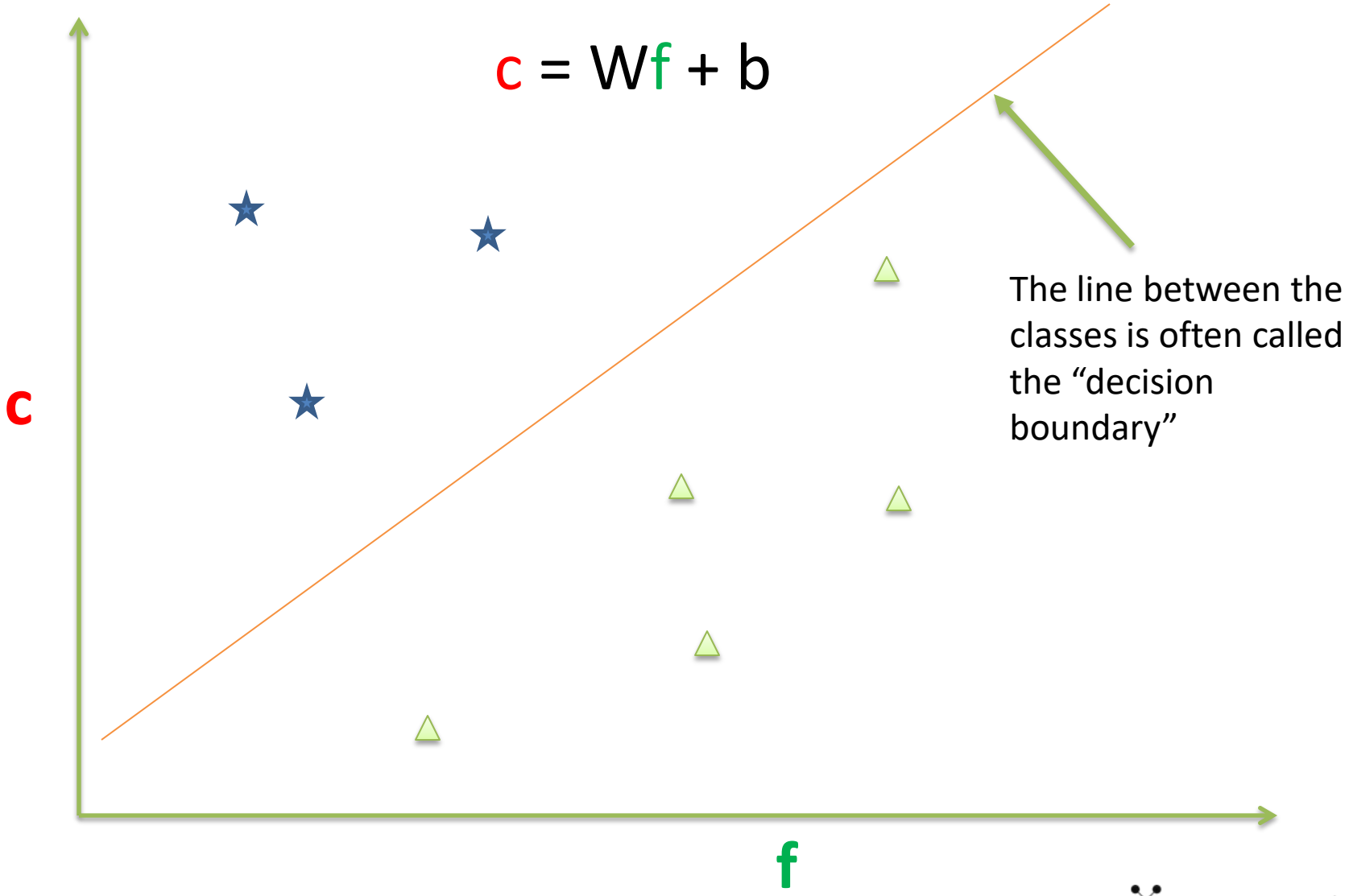
The trouble is that we're learning parameters **W** and **b** such that …



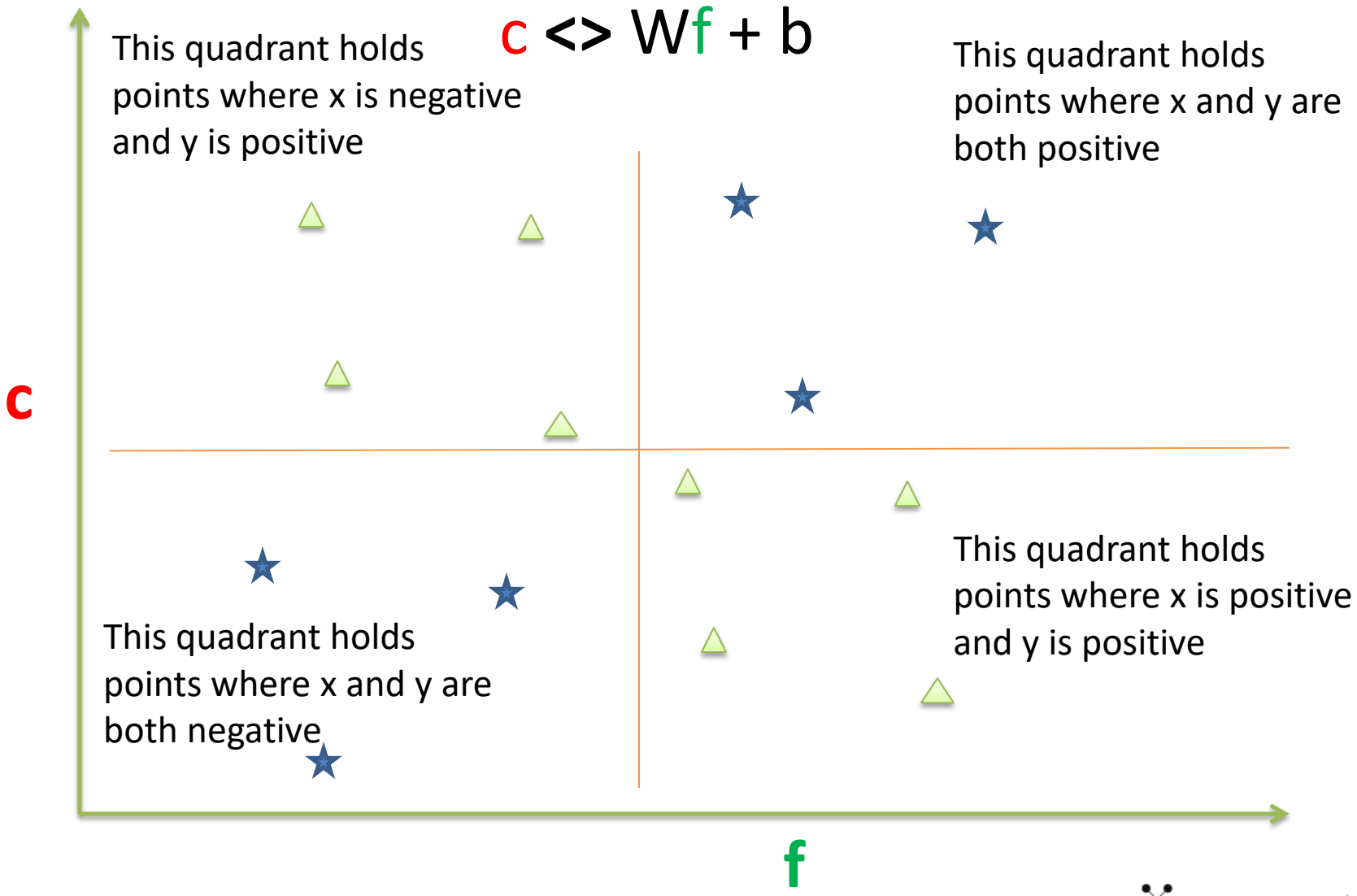
$$\begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} * \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \\ W_{13} & W_{23} & W_{33} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}$$

$$\mathbf{c} = \mathbf{W}\mathbf{f} + \mathbf{b}$$

Linear Decision Boundary

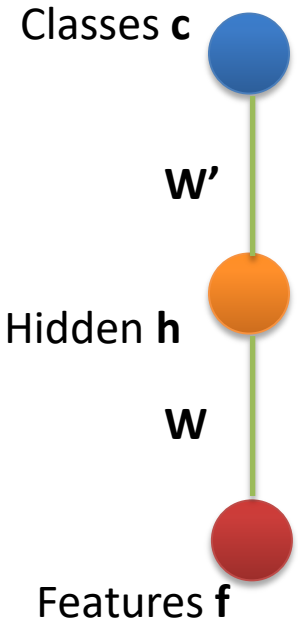


Decision Boundary for Problem 3



Deep Learning Algorithms

Can we learn a non-linear separator if we add one more layer of neurons **W'** ???



$$c_1 = W'_{11}h_1 + W'_{12}h_2$$

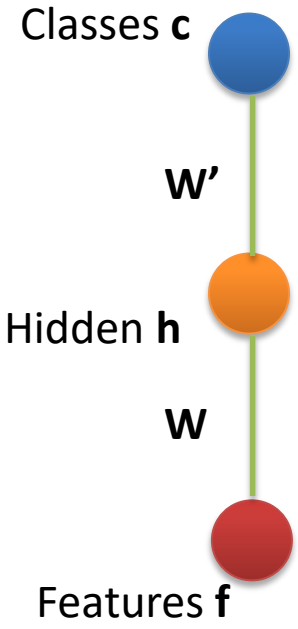
$$c_2 = W'_{21}h_1 + W'_{22}h_2$$

$$h_1 = W_{11}f_1 + W_{12}f_2$$

$$h_2 = W_{21}f_1 + W_{22}f_2$$

Deep Learning Algorithms

Can we learn a non-linear separator if we add one more layer of neurons **W'** ???

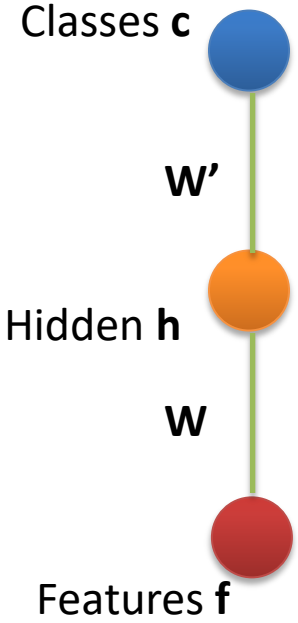


$$c_1 = W'_{11}(W_{11}f_1 + W_{12}f_2) + W'_{12}(W_{21}f_1 + W_{22}f_2)$$

$$c_2 = W'_{21}(W_{11}f_1 + W_{12}f_2) + W'_{22}(W_{21}f_1 + W_{22}f_2)$$

Deep Learning Algorithms

Can we learn a non-linear separator if we add one more layer of neurons **W'** ???



$$c_1 = (W'_{11}W_{11} + W'_{12}W_{21})\, f_1 + (W'_{11}W_{12} + W'_{12}W_{22})\, f_2$$

$$c_2 = (W'_{21}W_{11} + W'_{22}W_{21})\, f_1 + (W'_{21}W_{12} + W'_{22}W_{22})\, f_2$$

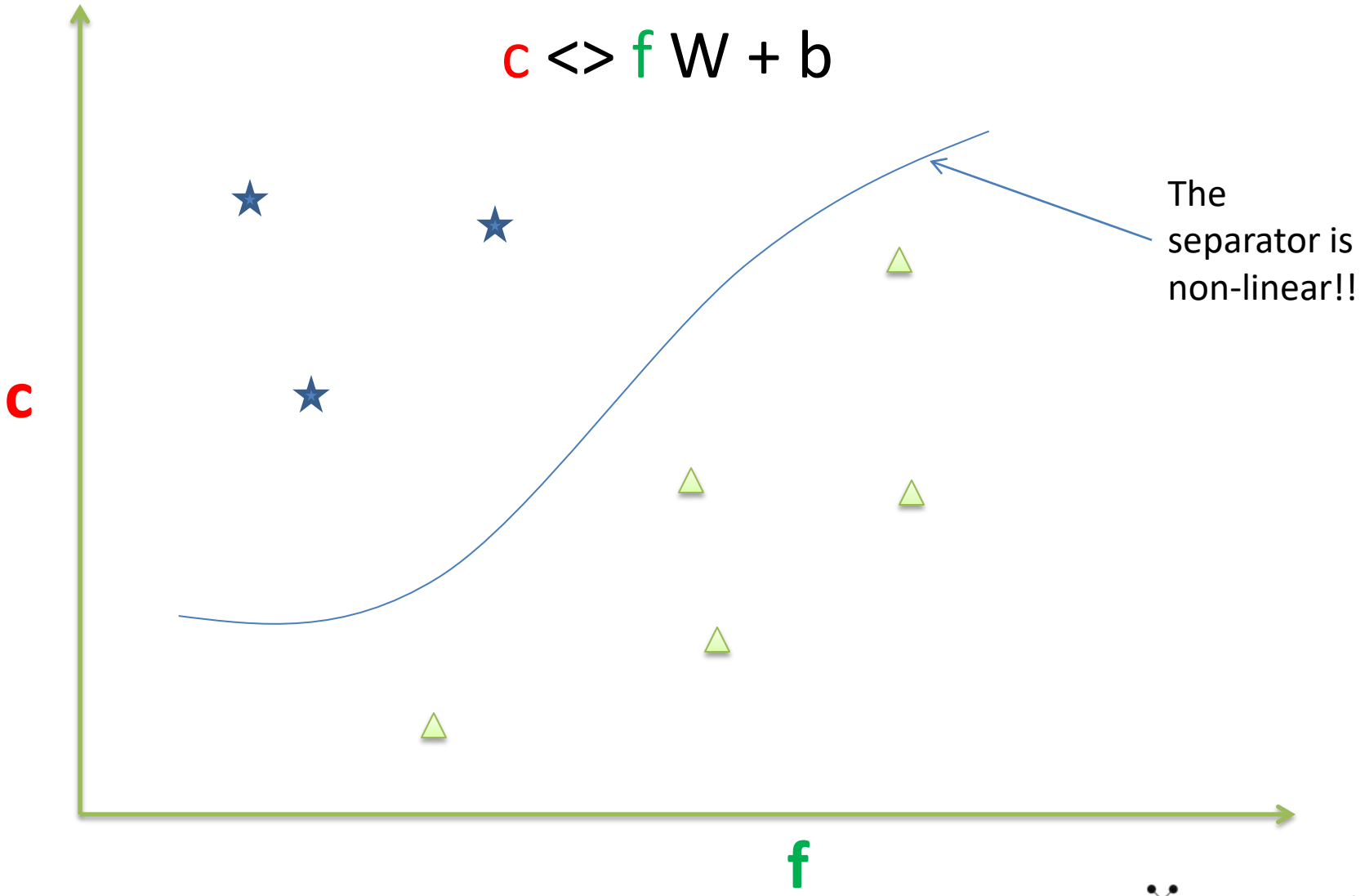
A linear combination of the features once again!

$$\mathbf{c = f\ W\ W'}$$

Neural Network Classifiers Problem 3

- As you can see, the weights factor out!
- So all you get is another linear decision boundary with a set of weights that is the product of the two weight matrices.
- Is there any way you can get an arbitrary decision boundary like this .

So, how can we learn a non-linear separator like this?

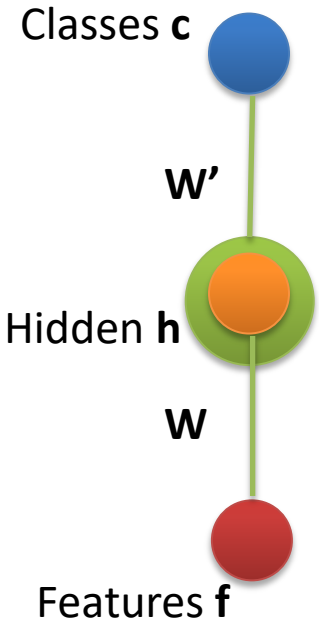


Neural Network Classifiers Problem 3

- You have to introduce a non-linearity between the two layers of neurons.
- You do that using a non-linear function called an activation function.

Introducing a non-linearity

We can learn a non-linear separator if we introduce a non-linear function **g**.



$$c_1 = W'_{11}h_1 + W'_{12}h_2$$
$$c_2 = W'_{21}h_1 + W'_{22}h_2$$

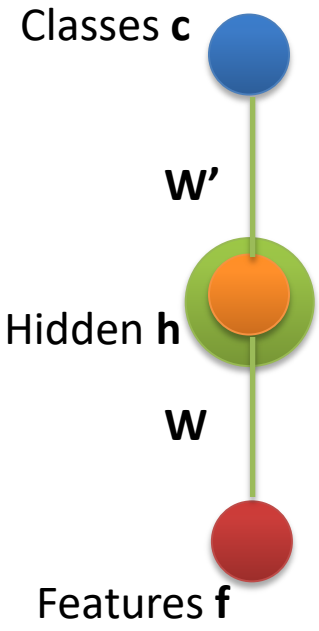
$$h_1 = g(a_1)$$
$$h_2 = g(a_2)$$

$$a_1 = W_{11}f_1 + W_{12}f_2$$
$$a_2 = W_{21}f_1 + W_{22}f_2$$

Neural Network Classifiers Problem 3

- The output of each layer of neurons (after the linear transform) is called the pre-activation.
- After the pre-activation passes through the non-linear activation function, the output is called the activation.
- I've represented the pre-activation by the orange circle and the activation by the green circle in these diagrams.

Activation Function



$$c_1 = W'_{11}h_1 + W'_{12}h_2$$
$$c_2 = W'_{21}h_1 + W'_{22}h_2$$

$$h_1 = g(a_1)$$
$$h_2 = g(a_2)$$

$$a_1 = W_{11}f_1 + W_{12}f_2$$
$$a_2 = W_{21}f_1 + W_{22}f_2$$

'g' is called the "activation function".
'a' is called the "pre-activation".
'g(a)' is called the "activation".

The activation becomes the input to the next higher layer.

Neural Network Classifiers Problem 3

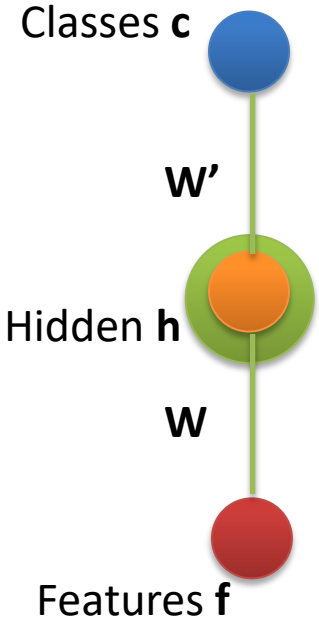
- There are a number of activation functions that are used.
- You have already seen one – the softmax – that is used only on the output layer.
- The oldest of the activation functions are the sigmoid and tanh.
- One of the more recent ones (2000) is the ReLU.

Do exercise 650s

Classification

Do exercise 650 to see how the different activation functions work.

Activation Function 1



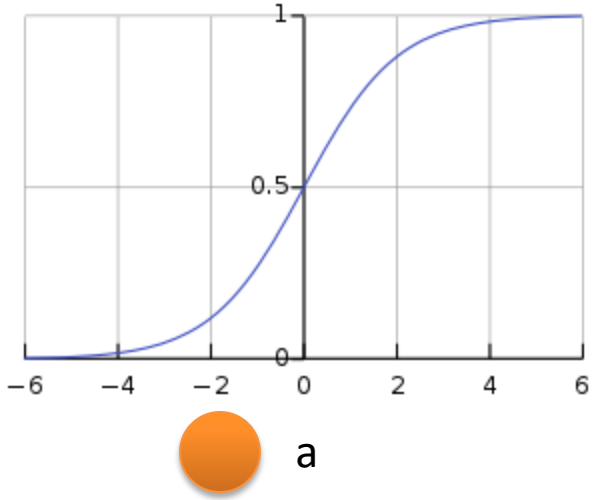
Sigmoid

h = g(a) = sigm(a)

Squishes a real number to (0,1)

$$\frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1}$$

h = sigm(a)



Converting From Real Numbers to 0 - 1

'a' is called the "pre-activation".

a ranges from –inf to +inf

This is the input to the sigmoid function.

Converting From Real Numbers to 0 - 1

'c' is the "pre-activation" of the final layer.

c ranges from –inf to +inf

This last pre-activation is passed as input to the final non-linearity which is typically the softmax function.

I will refer to 'c' as the "final pre-activation"

Sigmoid / Logistic Function

- In the following slides, we show how a sigmoid activation function transforms the input from the space of real numbers to that of 0 to 1.

Do exercise 650s

Converting From Real Numbers to 0 - 1

If a ranges from –inf to +inf.

e^{-a} ranges from 0 to +inf.

Converting From Real Numbers to 0 - 1

If e^{-a} ranges from 0 to +inf.

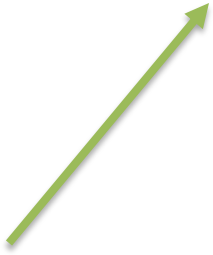
$1 + e^{-a}$ ranges from 1 to +inf.

Converting From Real Numbers to 0 - 1

If $1 + e^{-a}$ ranges from 1 to +inf.

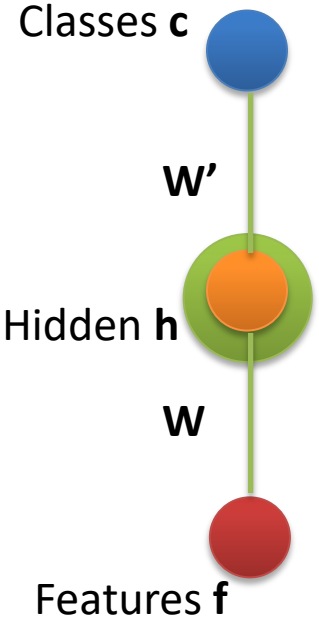
$1/1 + e^{-a}$

ranges from 0 to 1.



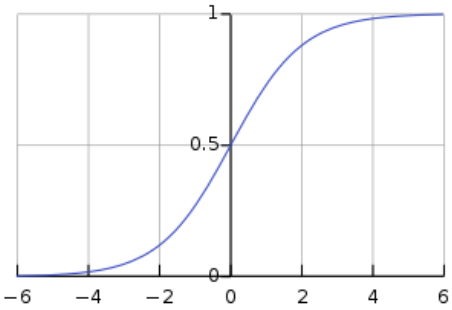
Sigmoid function

Activation Function 1



Sigmoid

h = g(a) = sigm(a)



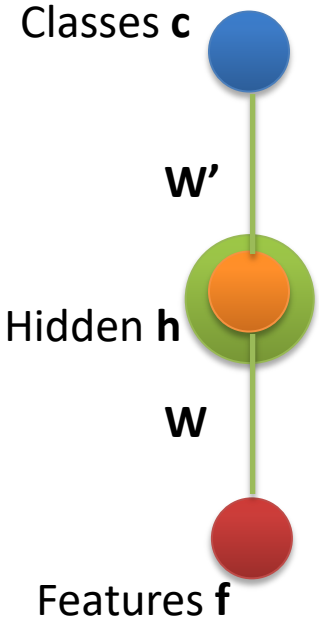
Squishes a real number to (0,1)

$$\frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1}$$

It has a derivative at every value of a.

d(sigm(a))/d(a) = sigm(a) * (1 – sigm(a))

Activation Function 2



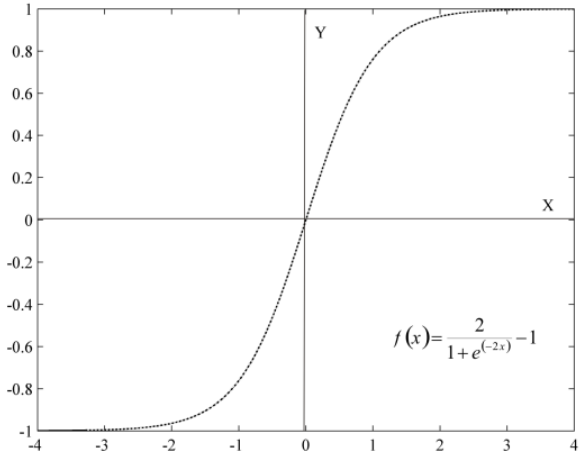
Hyperbolic Tangent

h = g(a) = tanh(a)

Squishes a real number to (-1,1)

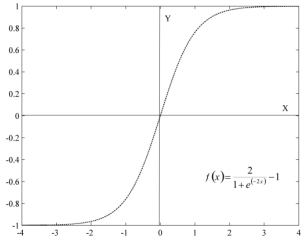
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

h = tanh(a)



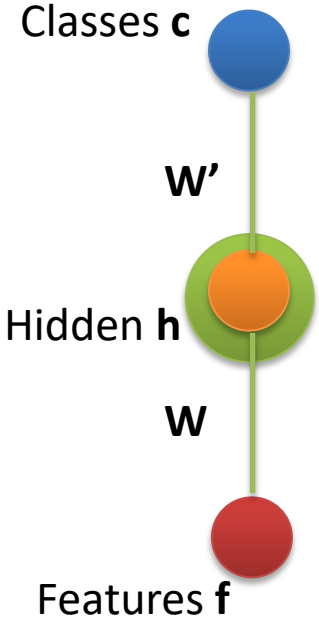
a

Activation Function 2



Hyperbolic Tangent

h = g(a) = tanh(a)



Squishes a real number to (-1,1)

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

It has a derivative at every value of a.

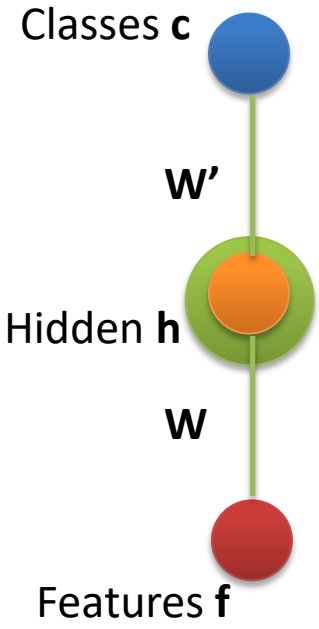
d(tanh(a))/d(a) = 1 – tanh(a)^2

Activation Function 3

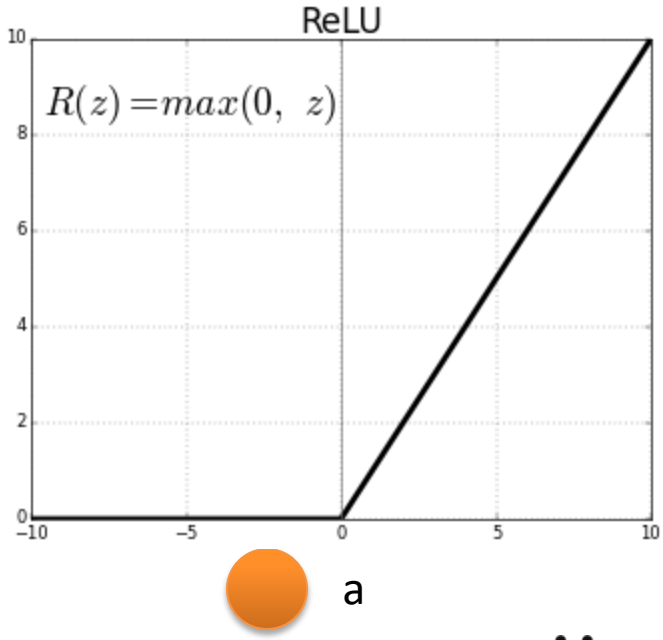
Rectified Linear Unit

h = g(a) = ReLU(a)

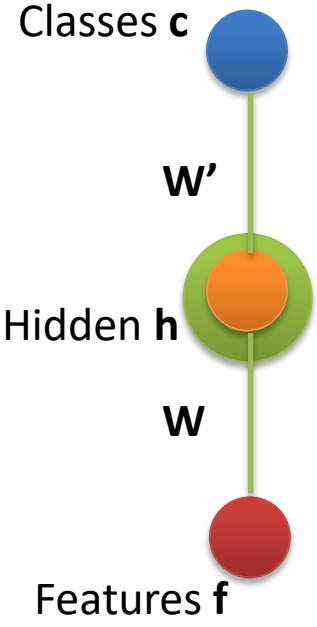
Squishes a real number to (0,inf)



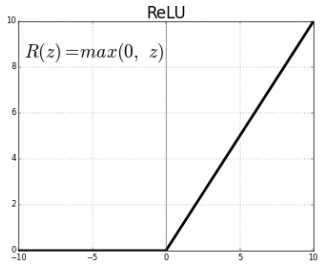
h = ReLU(a)



Activation Function 3



Rectified Linear Unit



h = g(a) = ReLU(a)

Squishes a real number to (0,inf)

If a <= 0, ReLU(a) = 0

If a > 0, ReLU(a) = a

It has a derivative at every value of a.

$$d(\text{ReLU}(a))/d(a) = 1_{a>0}$$

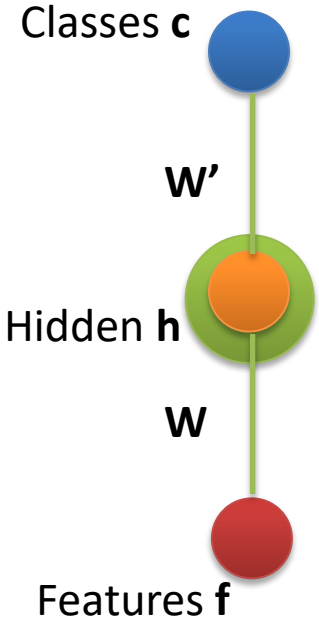
$1_{a>0}$ just means "1 if a > o else 0"

Activation Function 0

Softmax is also a non-linear activation function but it is never used between layers because it converts a dense representation of information into an approximation of the one-hot encoding which is very inefficient at carrying information through a system (so if you put the softmax between layers, you won't get much value from layers above the softmax activation function).

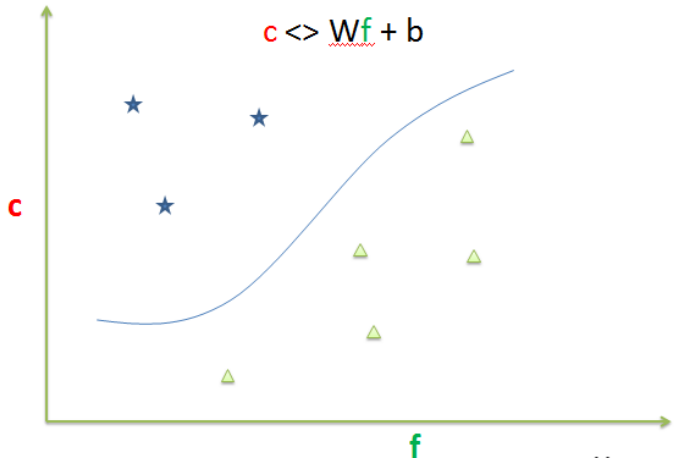
Deep Learning

$$c_1 = W'_{11}h_1 + W'_{12}h_2$$
$$c_2 = W'_{21}h_1 + W'_{22}h_2$$



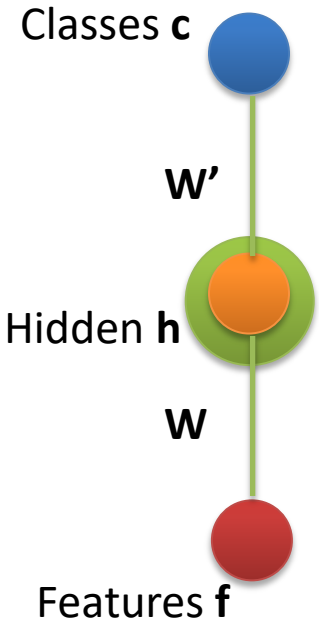
$$h_1 = g(a_1)$$
$$h_2 = g(a_2)$$

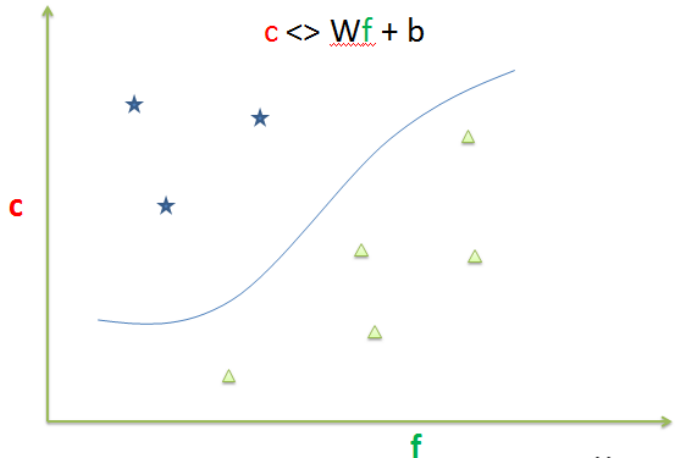
$$a_1 = W_{11}f_1 + W_{12}f_2$$
$$a_2 = W_{21}f_1 + W_{22}f_2$$



So <u>now</u> if you choose the weights W and W' suitably, you will get a non-linear separator between classes c_1 and c_2.

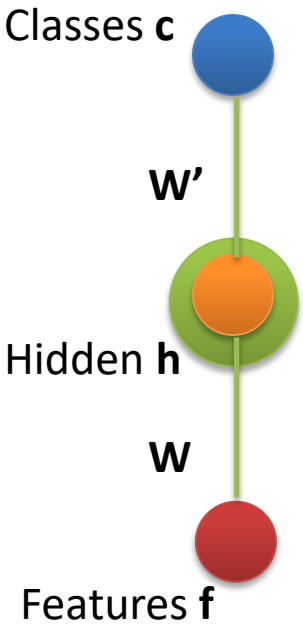
Deep Learning

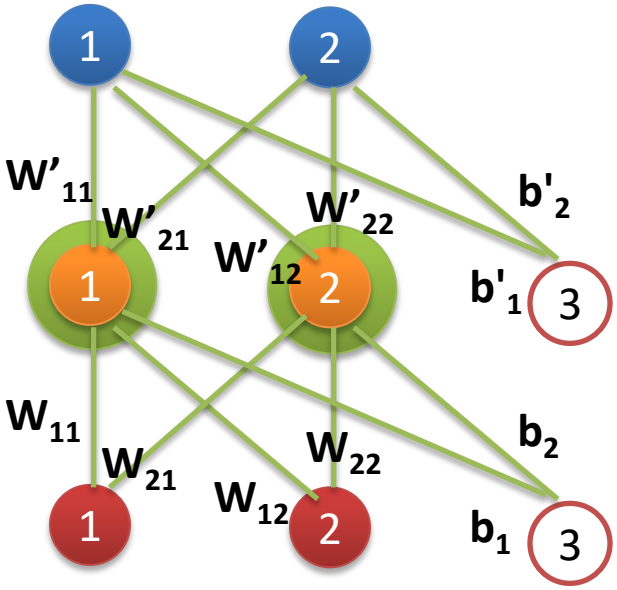


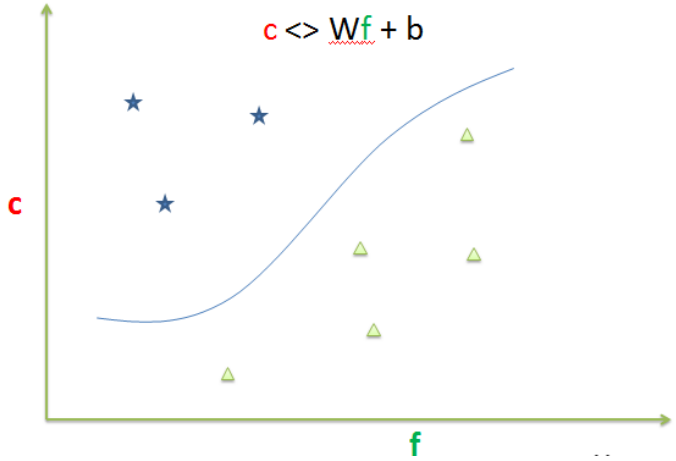


This is the essence of deep learning – using a multi-layered network with non-linearities at each stage.

Deep Learning

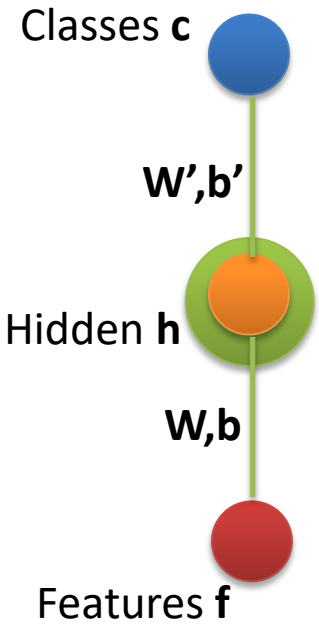






How do you make a deep learning model learn the weights W and b?

Training a Deep Neural Network



- How do you learn the weights automatically?
  - Step 1: Get some **training data**
  - Step 2: Create a **loss function** reflecting the badness of the neural net
  - Step 3: Adjust the weights and biases to **minimize the loss (error)** on the training data

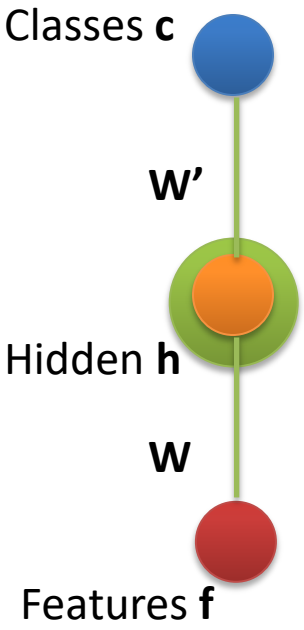
Let the **machine learn** weights and biases for each layer such that if $f_1 * f_2 > 0$ the classifier will select c_1 else c_2 !

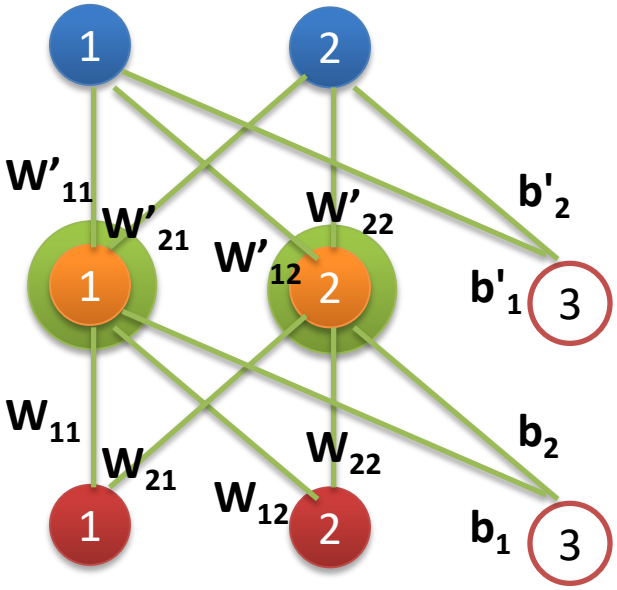
W = ? b = ? W' = ? b' = ?

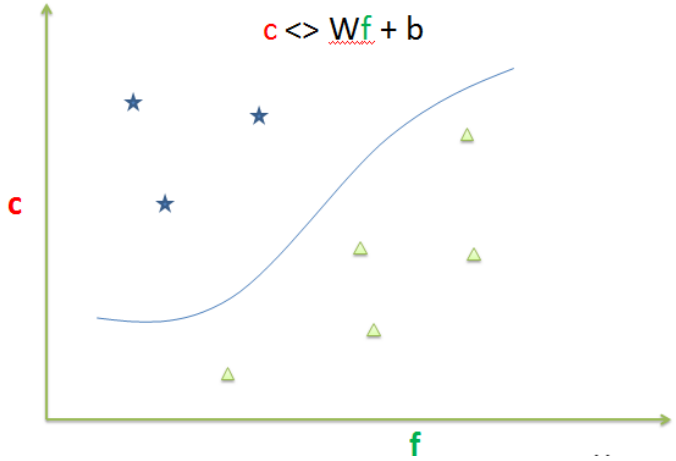
Deep Learning

- You can now train a classifier with more than one layer on the XOR dataset.
- Convince yourself that this classifier's loss decreases over multiple epochs.
- Try different activation functions.

Deep Learning Math

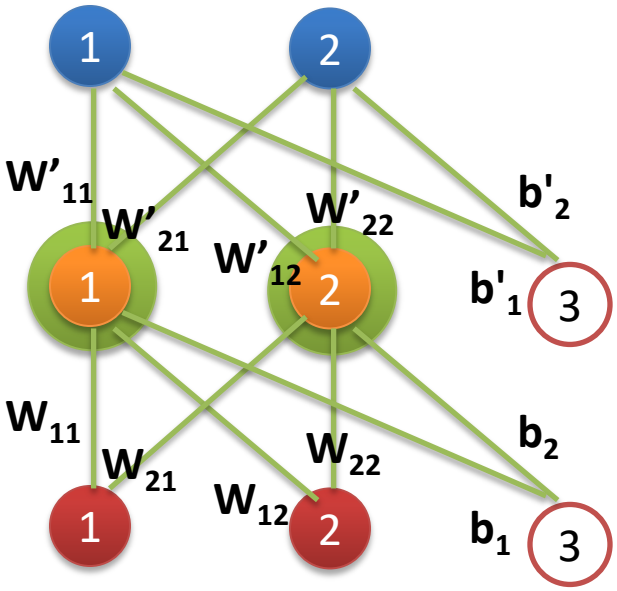






You've learnt how to train a deep learning model (make it learn the weights W and b). But do you know the math?

Deep Learning Math



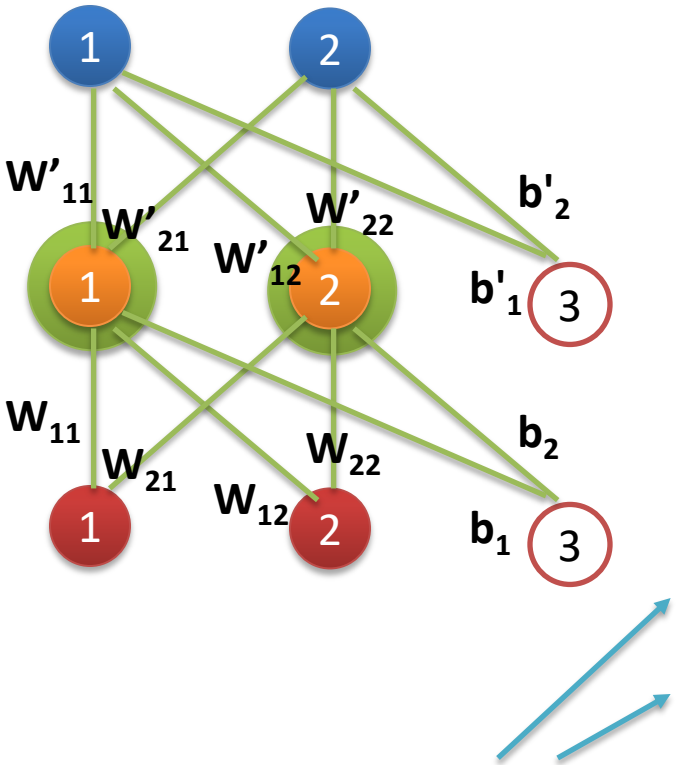
You're already familiar with the forward pass.

You compute the loss for a batch of training data and for the current parameters.

What about the backward pass?

You compute the derivates of the loss with respect to each of the parameters and then nudge the parameters so that the loss decreases.

Deep Learning Math



What are the derivatives needed for the backward pass?

Derivatives needed:

d(loss)/d(W')

d(loss)/d(b')

d(loss)/d(W)

d(loss)/d(b)

Backpropagation

d(loss)/d(W')

d(loss)/d(b')

d(loss)/d(W)

d(loss)/d(b)

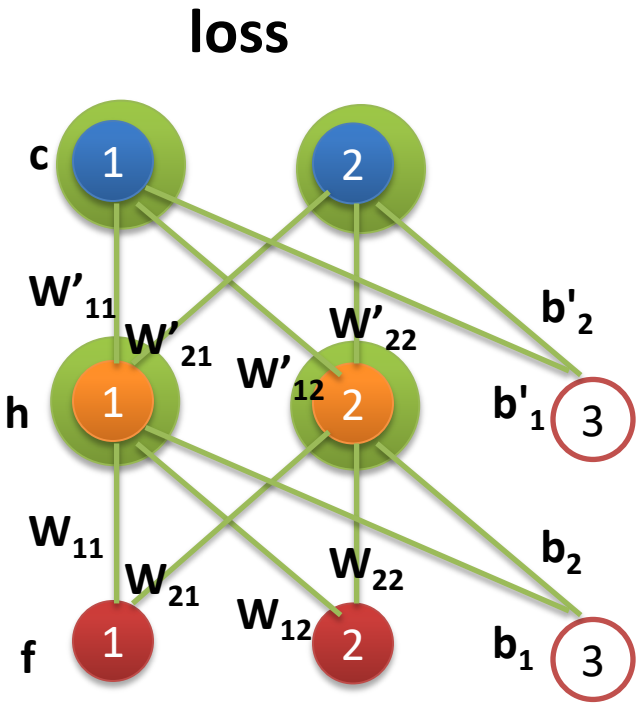
- These are the only derivatives you really need because they tell you which way to nudge the parameters.

Backpropagation

- Notice that all the derivatives are of the loss.
- So, to compute these derivatives, you have to walk back from the loss through the nodes nearest the loss.
- What's each node? It's a function.
- Layers of neurons are just nested functions
- Chain rule: *If* $f(x) = f(g(x))$

  then $df/dx = df/dg * dg/dx$
- Let us try this with **d(loss)/d(W')**

d(loss)/d(W')



loss = -log(softmax(c))

where c = W'h + b'

loss = -log(softmax(W'h + b'))

Chain:

Intermediate computations from W' to the loss

loss <- log <- softmax <- c <- W'

Backpropagation

Chain for **d(loss)/d(W') :**

Intermediate computations from W' to the loss

loss <- log <- softmax <- c <- W'

- **In the forward pass, you walk from W' to loss**
- **In the backward pass, you come back from loss to W'**

1

d(loss)/dW' = { d(loss)/dlog * dlog/dsoftmax } *

2 3

dsoftmax/dc * dc/dW'

Backpropagation

Chain for **d(loss)/d(W') :**

Intermediate computations from W' to the loss

loss <- log <- softmax <- c <- W'

- **In the backward pass, you come back from loss to W' chaining derivatives along the way**

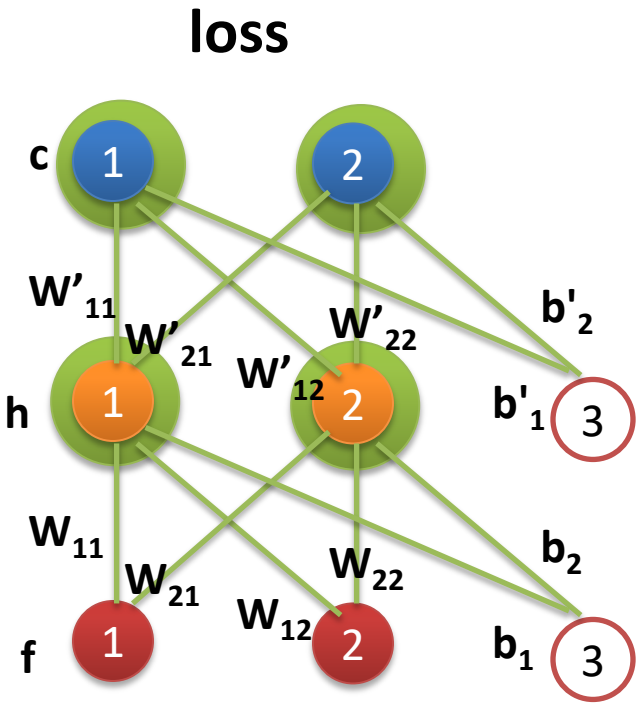
1 = dloss / dsoftmax(c)

d(loss)/dW' = [{ d(loss)/d(log) * d(log)/dsoftmax } *

2 = dloss / dc 3 = dloss/dW'

dsoftmax/dc] * dc/dW'

d(loss)/d(log)



loss = -1 * log(softmax(c))

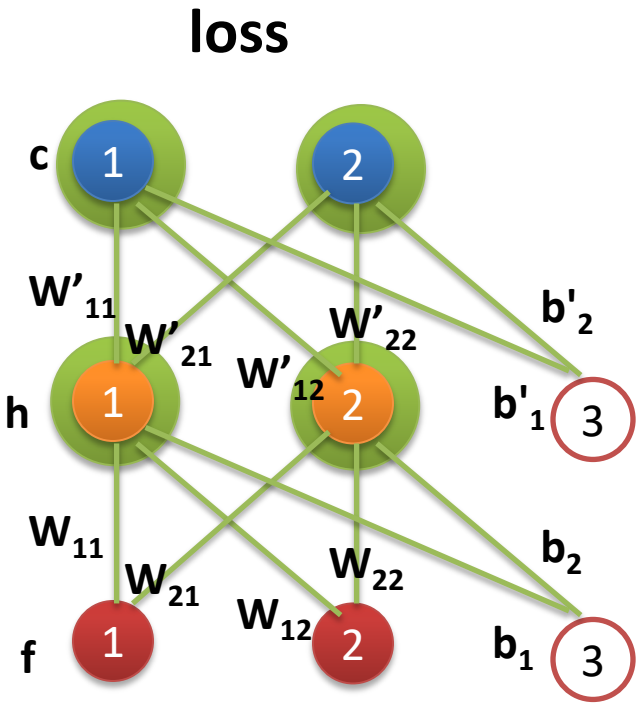
So,

d(loss) / d(log(softmax(c)))

Using d(-1 * x)/dx = -1

d(loss) / d(log(softmax(c))) = -1

d(loss)/d(softmax)



d(loss)/d(softmax(c))

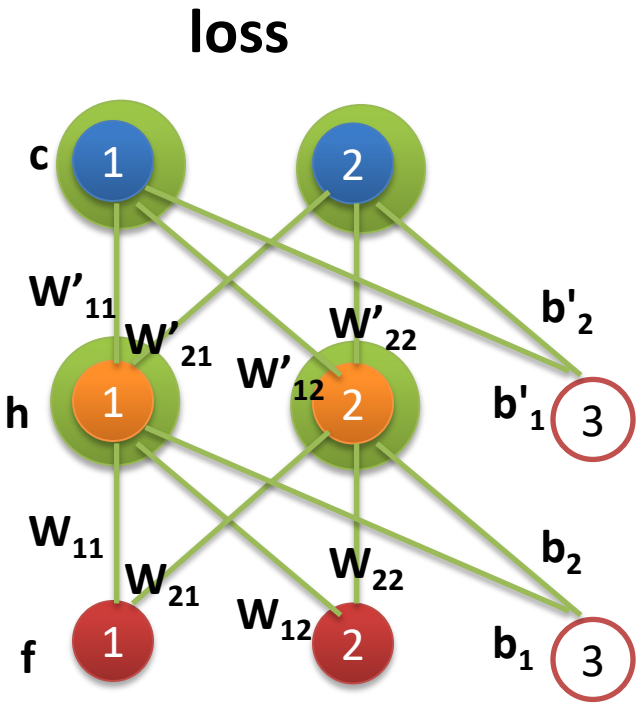
= d(loss)/d(log(softmax(c))) *
d(log(softmax(c)))/d(softmax(c))

… by the chain rule …

but we already have

d(loss)/d(log(softmax(c))) = -1

d(loss)/d(softmax)



So, d(loss)/d(softmax(c))

= d(loss)/d(log(softmax(c))) *
d(log(softmax(c)))/d(softmax(c))

… by the chain rule

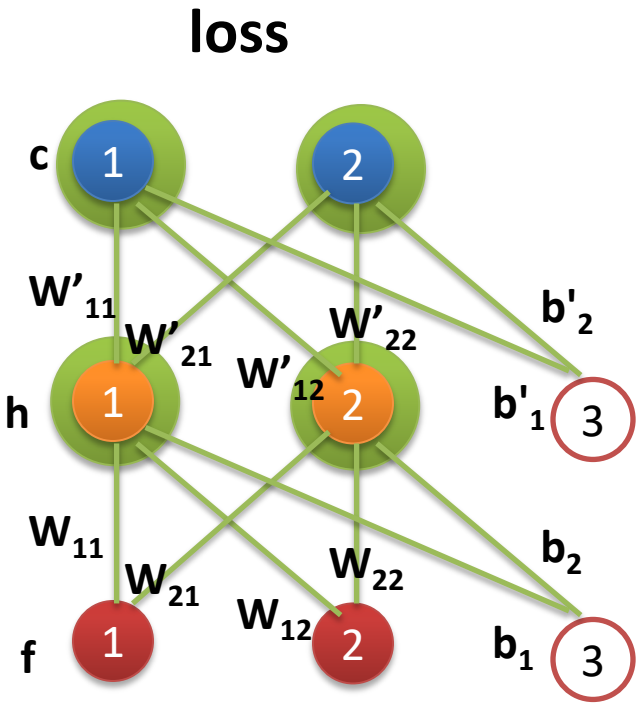
= -1 * d(log(softmax(c)))/d(softmax(c))

(substituting -1 for
d(loss)/d(log(softmax(c))))

Using d(log(x))/dx = 1/x in the above …

d(loss)/d(softmax(c)) = -1/softmax(c)

d(loss)/d(softmax)



d(loss)/d(softmax(c)) =

-1/softmax(c)

Backpropagation

loss <- log <- softmax <- c <- W'

1 = dloss / dsoftmax(c)

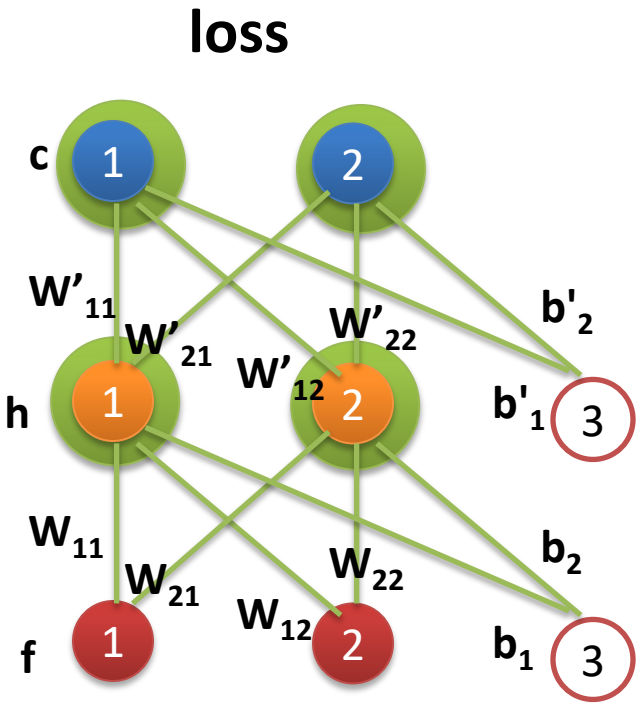
d(loss)/dW' = [{ d(loss)/dlog * dlog/dsoftmax } *

2 = dloss / dc 3 = dloss/dW'

dsoftmax/dc] * dc/dW'

We have completed step 1 and computed dloss/dsoftmax(c). So we now proceed to Step 2.

d(loss)/d(c)



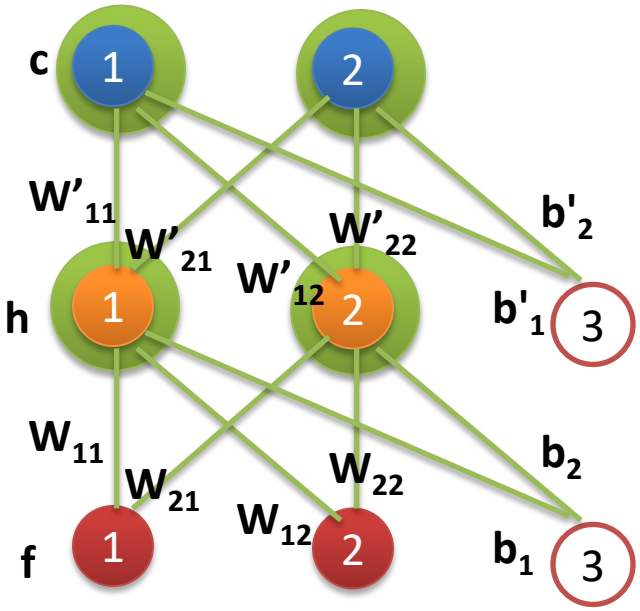
d(loss)/d(c) =

**{d(loss)/d(softmax(c)) *
d(softmax(c))/d(c)}**

We've already computed
d(loss)/d(softmax(c)

What is d(softmax(c))/d(c)?

d(softmax(c))/d(c)



What is d(softmax(c))/d(c)?

I'm going to tell you that …

d(softmax(c))/d(c) =

softmax(c) (t – softmax(c))

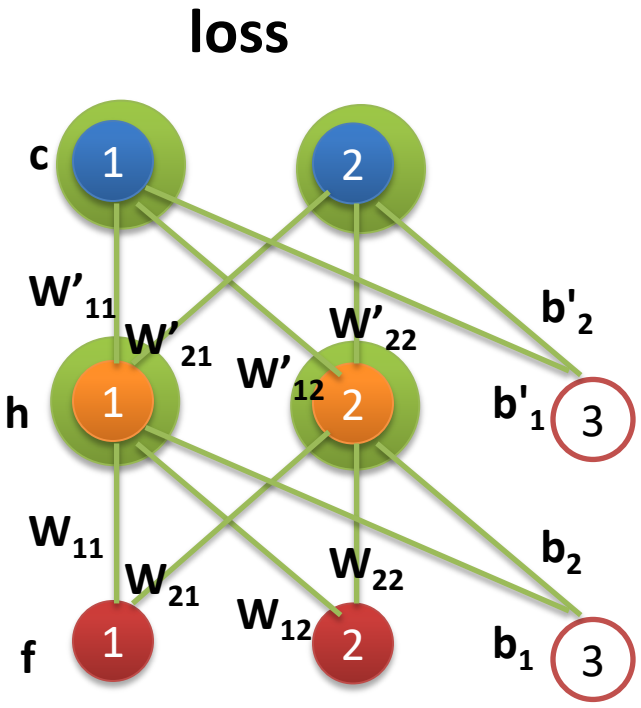
For a derivation of the above, visit the Youtube link …

https://www.youtube.com/watch?v=1N837i4s1T8

t is the one-hot vector* of the correct class.

***In a two-class classification problem, t is [1,0] if the correct class is 0 and [0,1] if not.**

d(loss)/d(c)



d(loss)/d(c) =

{d(loss)/d(softmax(c)) *

d(softmax(c))/d(c)}

But, d(loss)/d(softmax(c))

= -1/softmax(c)

and d(softmax(c))/d(c)

= softmax(c) (t – softmax(c))

So, d(loss)/d(c) =

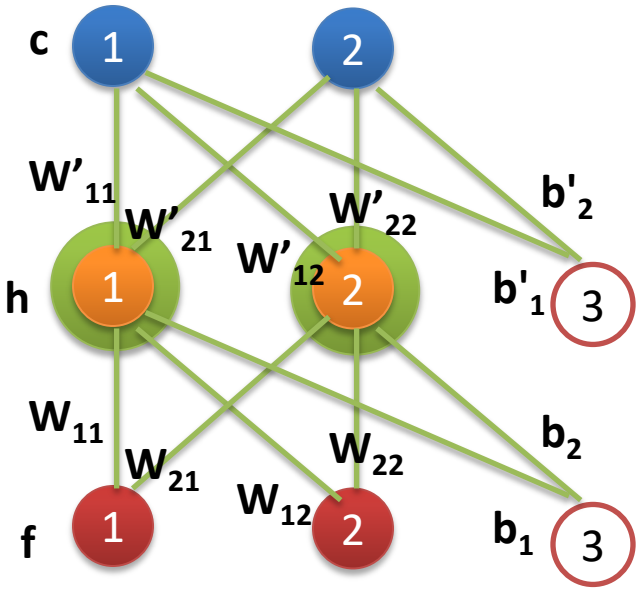
{ -1/softmax(c) *

softmax(c) (t – softmax(c))}

= -1 * (t – softmax(c))

d(loss)/d(c) = (softmax(c) – t)

d(loss)/d(c)



Chain rule:

d(loss)/dc

=

{d(loss)/d(softmax(c)) * d(softmax(c))/d(c)}

=

(softmax(c) – t)

Backpropagation

It looks simple because I'm working with tensors.

All these derivations are explained very rigorously without the use of tensors in this Youtube video by Hugo Larochelle

https://www.youtube.com/watch?v=1N837i4s1T8

Backpropagation

Chain for **d(loss)/d(W'):**

Intermediate computations from W' to the loss

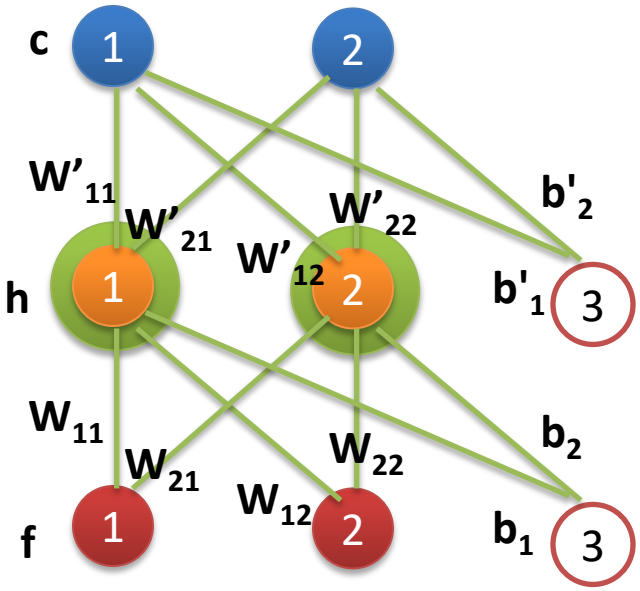
loss <- log <- softmax <- c <- W'

We've walked back from loss to c.

The difficult part is over!

From here, the math is easy (in tensor space).

d(loss)/d(W')



Chain rule:

d(loss)/dW'

=

{d(loss)/d(c)} * d(c)/d(W')

But c = hW'+b'

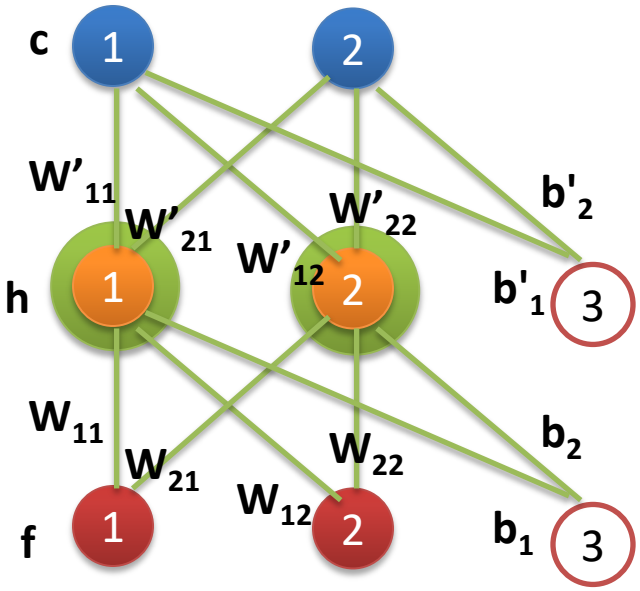
So, d(c)/dW'

= d(hW'+b')/dW'

= h^T

= h^T * (softmax(c) – t)

d(loss)/d(b’)



Chain rule:

d(loss)/d(b’)

=

{d(loss)/d(c)} * d(c)/d(b’)

But c = hW’+b’

So d(c)/db’

= d(hW’+b’)/db’

= 1

= (softmax(c) – t)

Backpropagation

We've just thrown a bunch of equations at the students.

They're not going to have digested this yet.

So, let's take a specific input and output and work through the backpropagation algorithm.

This way, they'll see what those equations mean, and just how **easy** it all is!

Backpropagation

d(loss)/d(W') ★

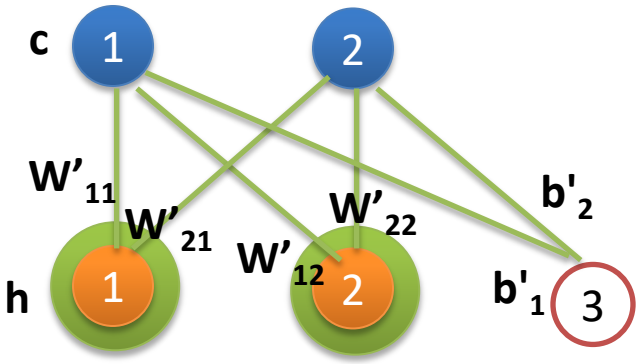
d(loss)/d(b') ★

d(loss)/d(W)

d(loss)/d(b)

- Yay! We got two of them!
- Let's do an exercise to compute d(loss)/d(W') and d(loss)/d(b') manually just so we really get it.

Exercise on d(loss)/d(W') and d(loss)/d(b')



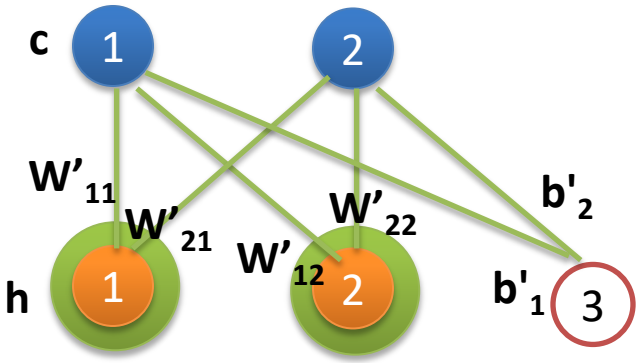
Let's say our training data is:

h1 = -10, h2 = 20 correct class=1

Let's start with randomly picked weight and bias matrices.

$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \quad b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Our training data is:

h1 = -10, h2 = 20 correct class=1

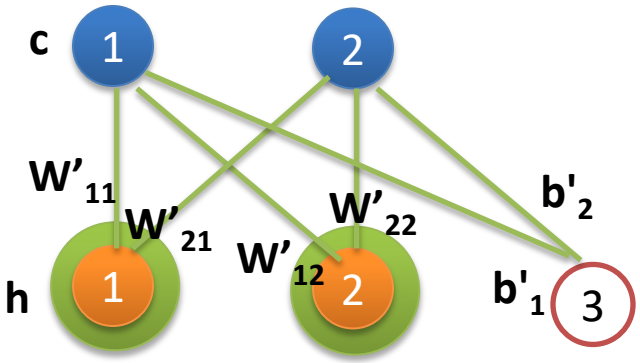
The input vector is therefore.

$$h = \begin{bmatrix} -10 & 20 \end{bmatrix}$$

The correct class is 1, so the target one hot vector is

$$t = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Let's start with the forward pass. We have assumed W' and b' and the input vector is h

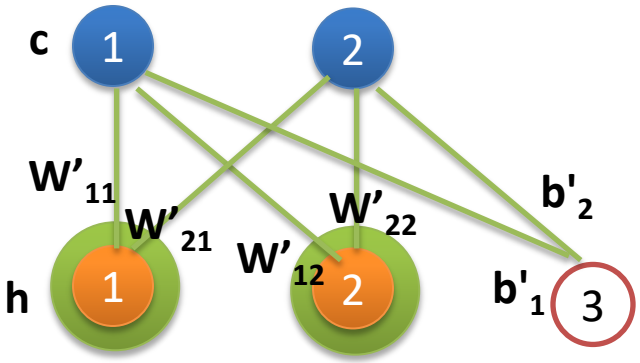
$$h = \begin{bmatrix} -10 & 20 \end{bmatrix}$$

So, we have everything we need to calculate the outputs c

c = hW'+b'

What is c?

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$h = \begin{bmatrix} -10 & 20 \end{bmatrix}$$

$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

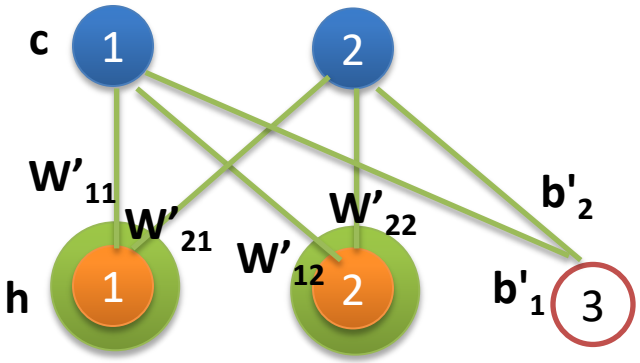
$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

c = hW'+b'

$$\begin{bmatrix} -10 & 20 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} 30 & 0 \end{bmatrix}$$

So, c $= \begin{bmatrix} 30 & 0 \end{bmatrix}$

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Now that we have the final pre-activation c, we need to calculate the final activation, which is softmax(c).

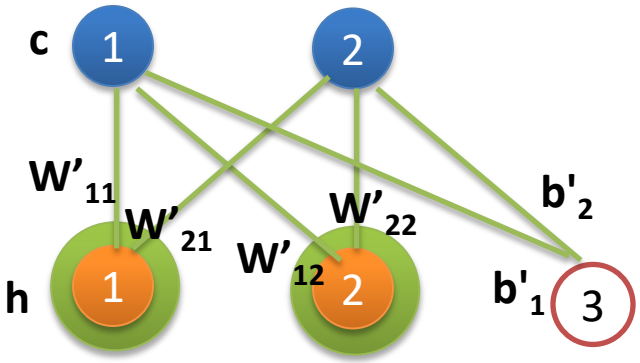
$$c = \begin{bmatrix} 30 & 0 \end{bmatrix}$$

The softmax essentially squishes the ouputs into extreme probabilities.

So what is softmax(c)?

Exercise on d(loss)/d(W') and d(loss)/d(b')

$$c = \begin{bmatrix} 30 & 0 \end{bmatrix}$$



The softmax essentially squishes the ouputs into extreme probabilities.

$$\text{softmax}(c) = \begin{bmatrix} 1 & 9.3*10^{-14} \end{bmatrix}$$

$$\text{softmax}(c) \sim= \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$\log(\text{softmax}(c)) \sim= \begin{bmatrix} 0 & -30 \end{bmatrix}$$

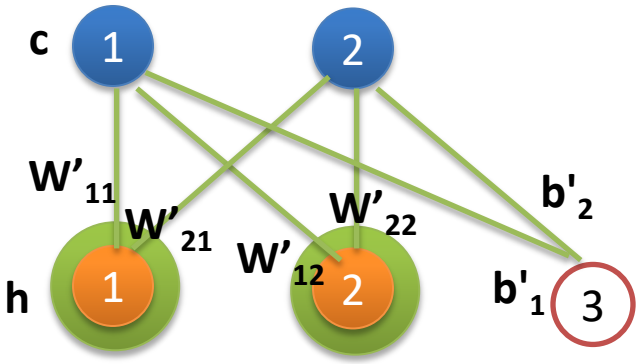
$$-\log(\text{softmax}(c)) \sim= \begin{bmatrix} 0 & 30 \end{bmatrix}$$

$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

30 is the loss, since the right class is 1

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

The forward pass is completed.

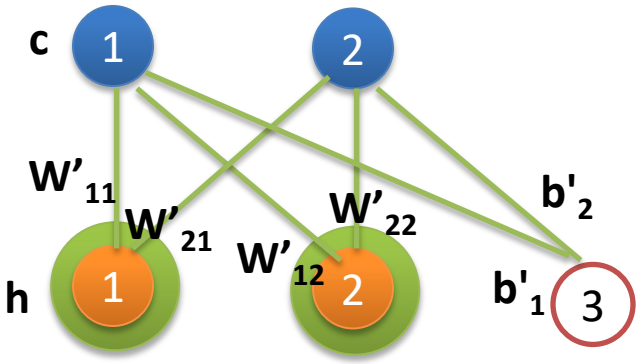
We have loss = 30

Now to compute d(loss)/d(W')

and d(loss)/d(b').

This is the backward pass.

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

To get to either d(loss)/d(W')

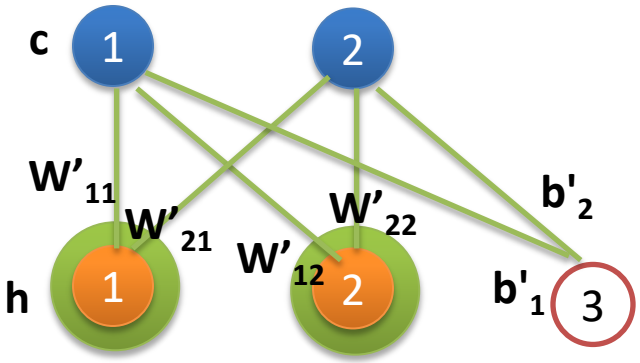
or d(loss)/d(b') …

We need to first compute …

d(loss)/d(c) =

(softmax(c) – t)

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

d(loss)/d(c) = (softmax(c) – t)

But from the forward pass,

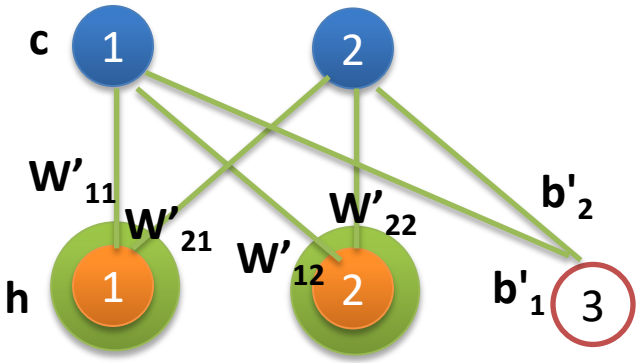
$$\text{softmax}(c) \sim= \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The correct class is 1, so the target one hot vector is

$$t = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

So (softmax(c) – t) = $\begin{bmatrix} 1 & -1 \end{bmatrix}$

Exercise on d(loss)/d(W') and d(loss)/d(b')



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

d(loss)/d(c) = softmax(c) – t

So, d(loss)/d(c)

$$\begin{bmatrix} 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -1 \end{bmatrix}$$

So, d(loss)/d(c) = $\begin{bmatrix} 1 & -1 \end{bmatrix}$

Backpropagation

Once you have d(loss)/d(c), the rest is easy!

Chain rule:

d(loss)/dW' = {d(loss)/d(c)} * d(c)/d(W')

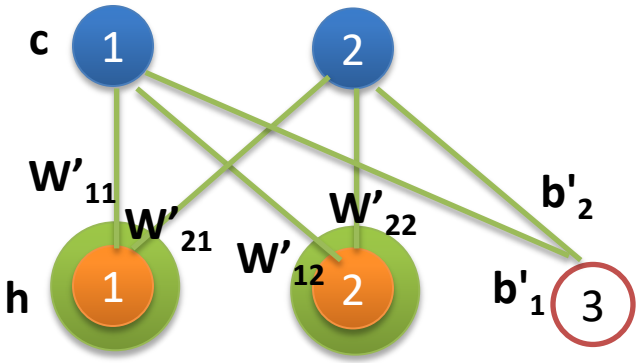
= h^T * (softmax(c) – t)

d(loss)/db' = {d(loss)/d(c)} * d(c)/d(b')

= 1 * (softmax(c) – t)

Exercise on d(loss)/d(W') and d(loss)/d(b')

$$h = \begin{bmatrix} -10 & 20 \end{bmatrix}$$



$$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

d(loss)/dW' =

{d(loss)/d(c)} * d(c)/d(W')

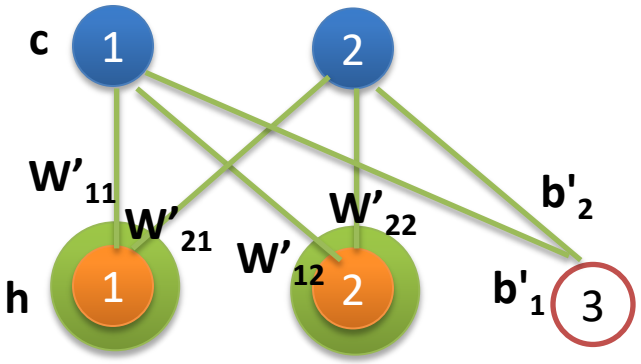
$$= h^T * (\text{softmax}(c) - t)$$

$$= \begin{bmatrix} -10 \\ 20 \end{bmatrix} * \begin{bmatrix} 1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -10 & 10 \\ 20 & -20 \end{bmatrix}$$

Exercise on d(loss)/d(W') and d(loss)/d(b')

$h = \begin{bmatrix} -10 & 20 \end{bmatrix}$



d(loss)/db' =

{d(loss)/d(c)} * d(c)/d(b')

= 1 * (softmax(c) – t)

$W' = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$

$= \begin{bmatrix} 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 \end{bmatrix}$

$b' = \begin{bmatrix} 0 & 0 \end{bmatrix}$

$= \begin{bmatrix} 1 & -1 \end{bmatrix}$

Backpropagation

d(loss)/d(W') 

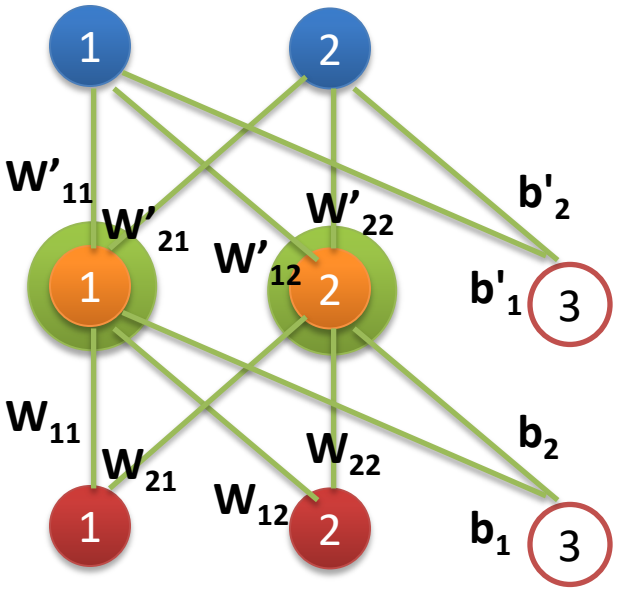
d(loss)/d(b') 

d(loss)/d(W)

d(loss)/d(b)

- Yay! We got two of them! Two to go.

Deep Learning Math



Let's get the remaining two

Derivatives needed:

d(loss)/d(W')

d(loss)/d(b')

Those were easy!

d(loss)/d(W)

d(loss)/d(b)

Now let's try these two!

Backpropagation

- To compute the derivatives we need, we have to walk back from the loss through the nodes nearest the loss.

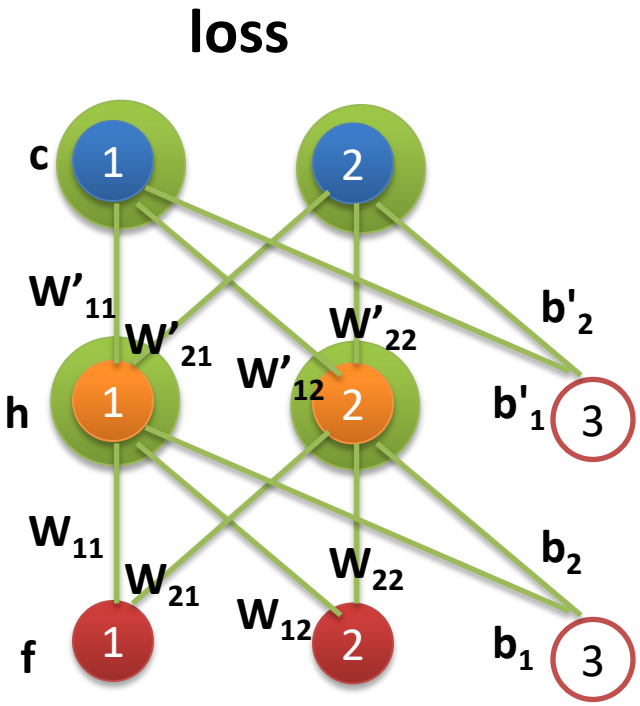
Chain for **d(loss)/d(W)**

= d(loss)/d(c) * d(c)/d(W)

Intermediate computations from W to the loss

loss <- log <- softmax <- c <- h <- a <- W

d(loss)/d(W)



loss = -log(softmax(c))

where c = W'h + b'

But h = relu(a)

And a = Wx + b

So

loss = -log(

softmax(W' * relu(Wx + b) + b'))

Chain:

Intermediate computations from W' to the loss

loss <- softmax <- c <- h <- a <- W

Backpropagation

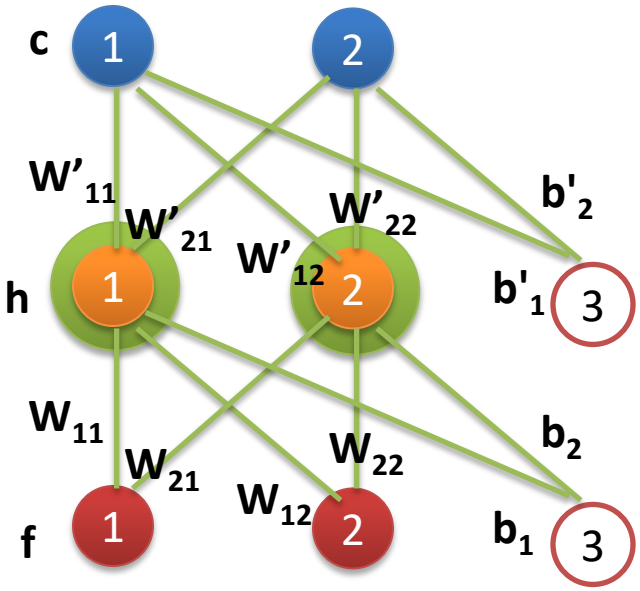
loss <- log <- softmax <- c <- h <- a <- W

We've already computed the chain up to c

d(loss)/d(c) = (softmax(c) – t)

Let's proceed from there by getting d(loss)/dh

d(loss)/d(h)



Chain rule:

d(loss)/d(h)

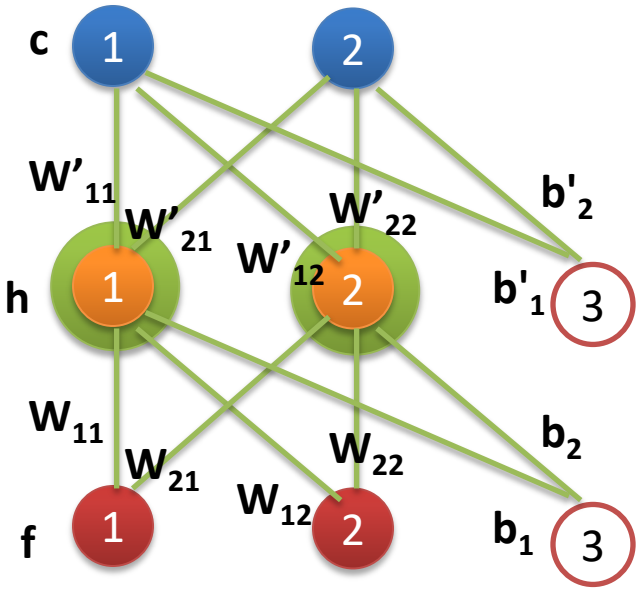
=

{d(loss)/d(softmax(c)) * d(softmax(c))/d(c)} * d(c)/d(h)

=

(softmax(c) – t) * W^T

d(loss)/d(a)



Chain rule:

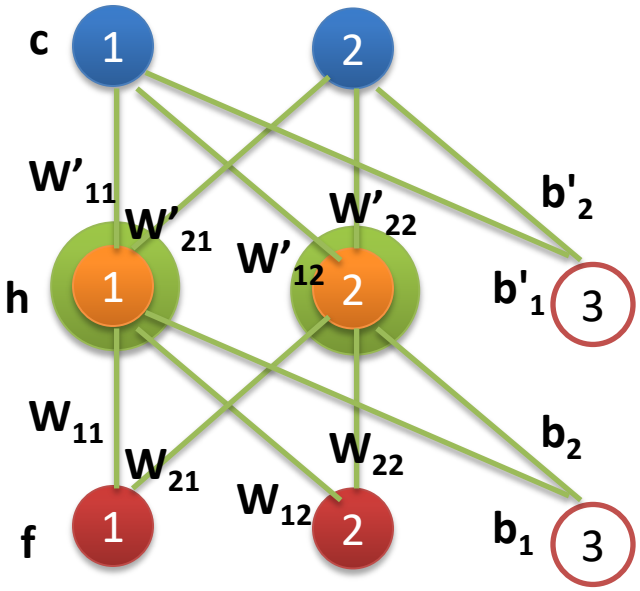
d(loss)/da

= {d(loss)/d(softmax(c)) * d(softmax(c))/d(c)} * d(c)/d(h) * d(h)/d(a)

but h = relu(a), so dh/da = $1_{a>0}$

= $(softmax(c) - t) * W^T * (1_{a>0})$

d(loss)/d(W)



Chain rule: d(loss)/dW

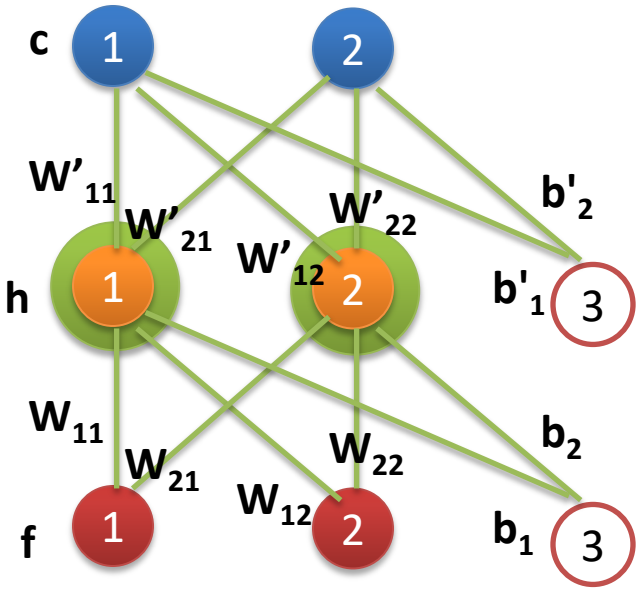
=

{d(loss)/d(softmax(c)) * d(softmax(c))/d(c)} * d(c)/d(h) * d(h)/da * da/dW

=

(softmax(c) – t) * W^T * $1_{a>0}$ * x

d(loss)/d(b)



Chain rule: d(loss)/db

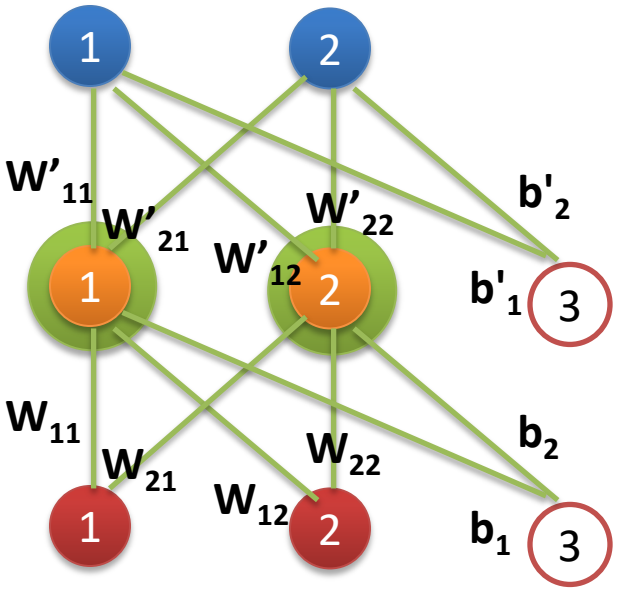
=

{d(loss)/d(softmax(c)) * d(softmax(c))/d(c)} * d(c)/d(h) * d(h)/da * da/db

=

(softmax(c) – t) * W^T * $1_{a>0}$ * 1

Deep Learning Math



Now we have all the derivatives needed for the backward pass.

Derivatives needed:

d(loss)/d(W')

d(loss)/d(b')

d(loss)/d(W)

d(loss)/d(b)

Those were easy!

These were easy too!

Image Processing

We've seen how deep learning classifiers work.

Their inputs & outputs are real numbers.

We've encoded documents as real numbers.

$\mathbf{f}_1$ = count of "a" $\mathbf{f}_2$ = count of "b" $\mathbf{f}_3$ = count of "c"

$\mathbf{c}_1$ = Sports $\mathbf{c}_2$ = Politics

- a b a c a b c a c => Politics
- a b a a c => Sports

Can we do something like this with images?

Image Processing

Let's start with the MNIST dataset.

It contains 70,000 images of handwritten digits.

Each of these 70,000 images is a digit (0 to 9).

Each image is labelled as 0,1,2 …,9.

This is a classification problem (deciding between a finite set of choices, labelling with a finite set of labels, etc.)

What are the features?

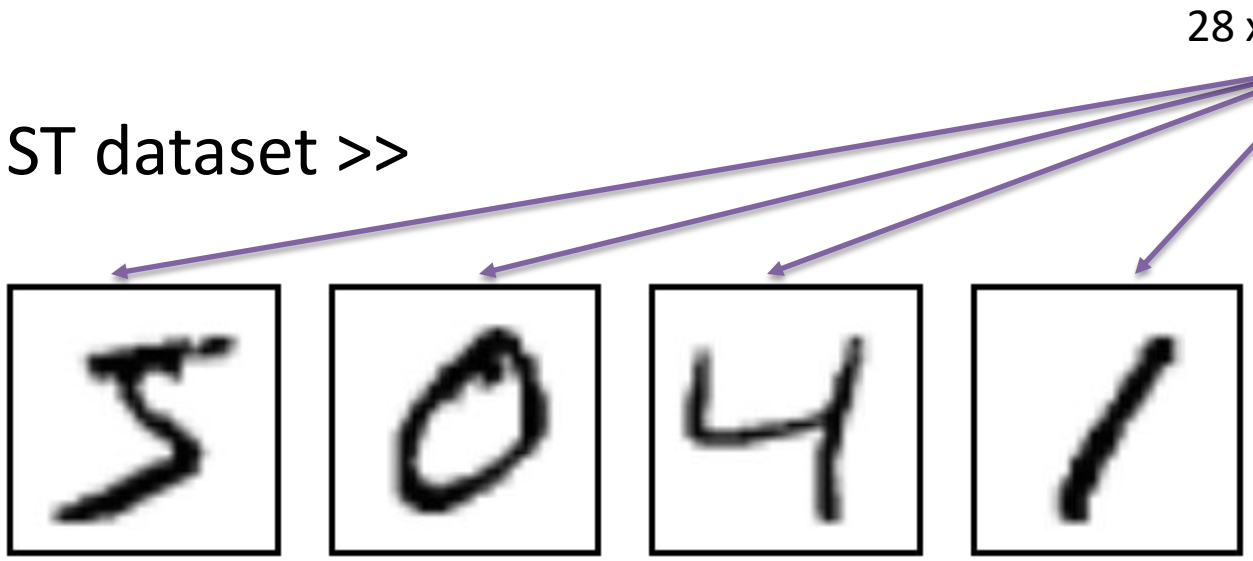
Each image is a greyscale 28 x 28 image (8-bits per pixel).

So one could just read the pixels out into a vector of integers of length 784 and use them as the input.

Applications to Image Processing

Use deep learning for image classification …

The MNIST dataset >>



The input to an image classification task is the image's pixels

Outputs: **5** **0** **4** **1**

The output of the MNIST image classification task is the digit (there are 10 classes)

Applications to Image Processing

Each image is represented by a 2D grid of pixels (a matrix of integers) for greyscale images (and 3 or 4 matrices for colour).

0 3 2 3 1	0 0 2 1 0	0 0 0 3 0	0 0 0 0 0
0 2 0 0 0	0 2 0 0 2	3 0 0 2 0	0 0 0 2 0
0 0 1 2 0	3 0 0 0 3	3 2 3 3 0	0 0 3 0 0
0 0 3 0 0	2 0 0 2 0	0 0 0 3 0	0 2 0 0 0
3 2 0 0 0	0 2 3 0 0	0 0 0 3 0	0 0 0 0 0

Applications to Image Processing

Traditionally you would flatten these numbers out …

5 = 0 3 2 3 1 0 2 0 0 0 0 1 2 0 0 0 3 0 0 3 2 0 0 0

0 = 0 0 2 1 0 0 2 0 0 2 3 0 0 0 3 2 0 0 2 0 0 2 3 0 0

0 3 2 3 1	0 0 2 1 0	0 0 0 3 0	0 0 0 0 0
0 2 0 0 0	0 2 0 0 2	3 0 0 2 0	0 0 0 2 0
0 0 1 2 0	3 0 0 0 3	3 2 3 3 0	0 0 3 0 0
0 0 3 0 0	2 0 0 2 0	0 0 0 3 0	0 2 0 0 0
3 2 0 0 0	0 2 3 0 0	0 0 0 3 0	0 0 0 0 0

… and then pass the vector to a classifier.

Image Classification

Each image is a 28 x 28 (8-bit **greyscale**) image.

So one could just read the pixels out into a vector of integers of length 784 and use them as the input.

But there's a better way - grouping together pixels that are close to each other (in 2D) in an image.

This is because each small area (in 2D) in an MNIST image contains local clues to the digit contained.

For example, a horizontal segment ending in the upper left area strongly suggests the number 7*.

* LeCun et al "Gradient-Based Learning Applied to Document Recognition"

LeNet5 Image Classification

In 1998* a paper described a deep neural network architecture called LeNet5 for image classification that used three types of layers:

F layers = Fully connected layers
C layers = Convolutional Layers
S layers = Subsampling Layers

Both C and S worked on local areas of the image.

Only F took as input a vectorized array.

* LeCun et al "Gradient-Based Learning Applied to Document Recognition"

LeNet 5

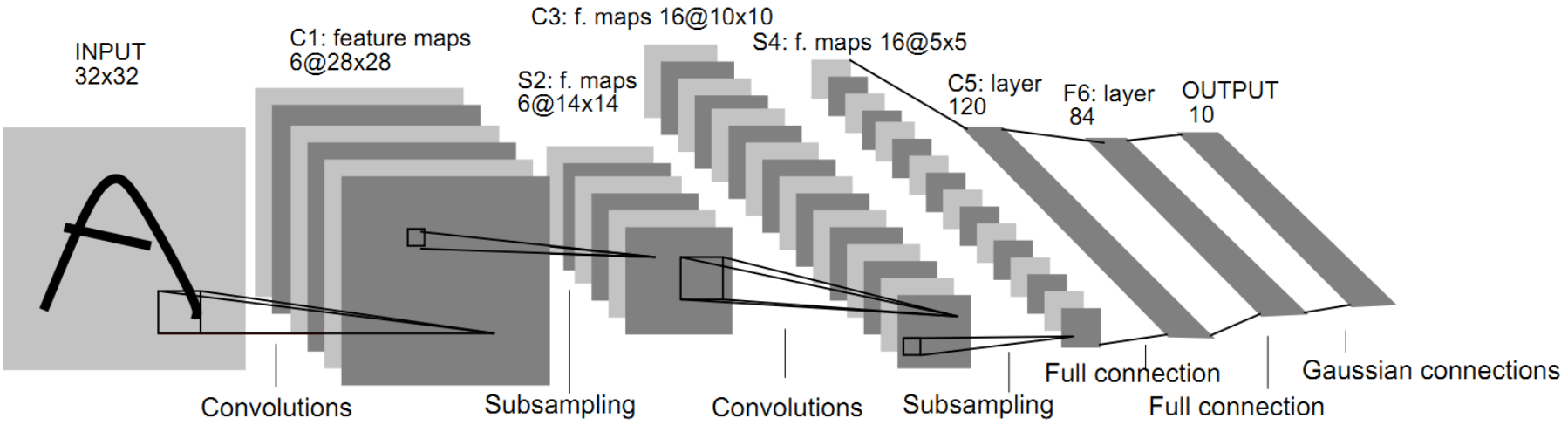


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Used **3 types of neural network layers** on the MNIST task

F layers = Fully connected layers
C layers = Convolutional Layers
S layers = Subsampling (nowadays usually called “pooling”)

… and avoided flattening out pixels till the very last layer.

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

LeNet5 Image Classification

Let's look at the 3 types of layers:

F layers = Fully connected layers

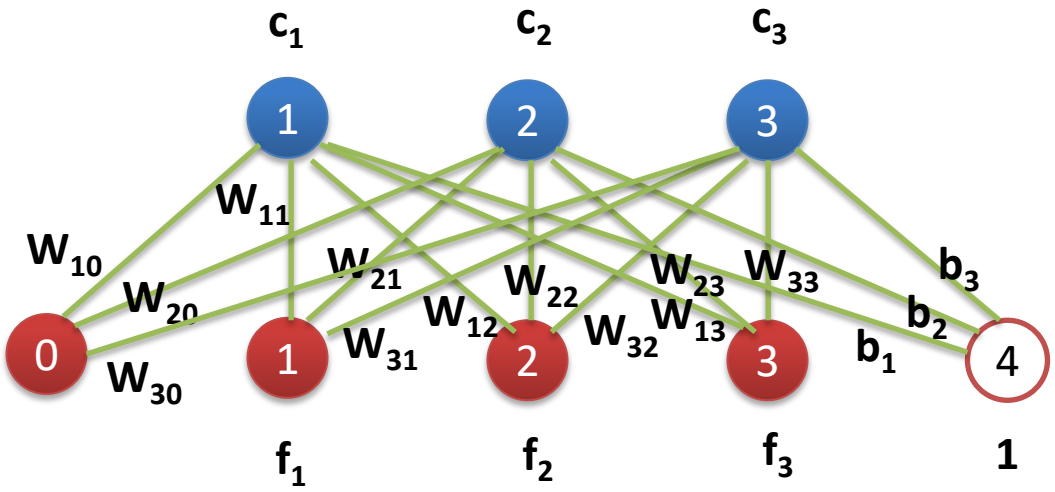
C layers = Convolutional Layers

S layers = Subsampling Layers

* LeCun et al "Gradient-Based Learning Applied to Document Recognition"

Fully Connected Layer

We've seen this already …

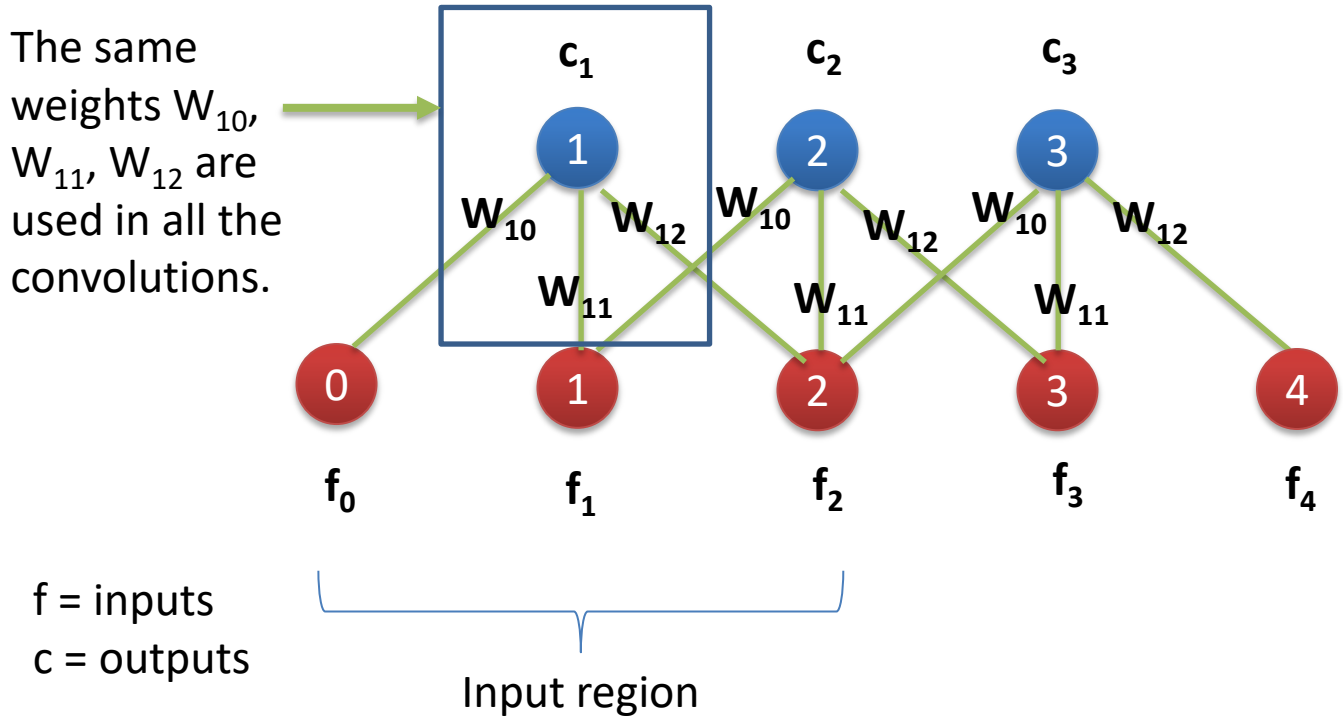


f = inputs
c = outputs

Every input is connected to every output by an interconnection (neuron)

Convolutional Layer

Only some of the inputs are connected to some of the outputs, and the interconnections share weights.

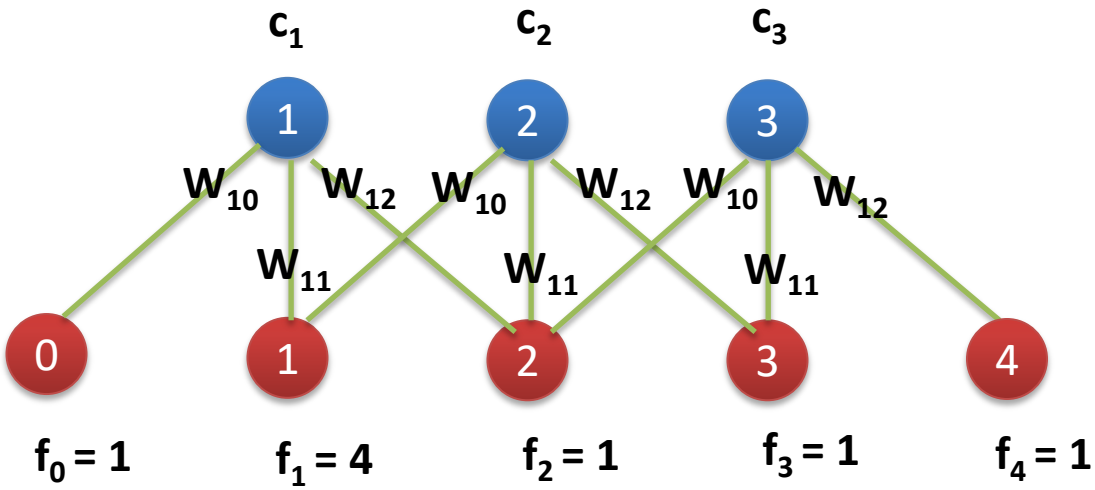


A finite (usually small) number of inputs is connected to one output.

Convolutional Layer Example

Supposing the weights were as follows ...

$W_{10} = -1$
$W_{11} = +2$
$W_{12} = -1$

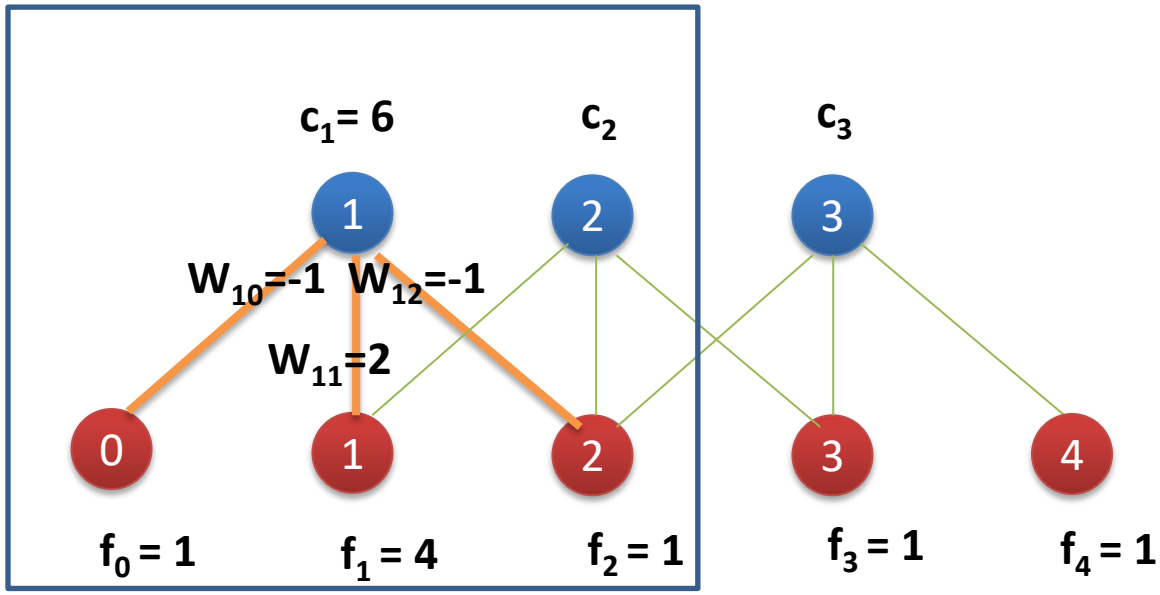


What are $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$?

Convolutional Layer Example

Supposing the weights were as follows ...

$W_{10} = -1$
$W_{11} = +2$
$W_{12} = -1$

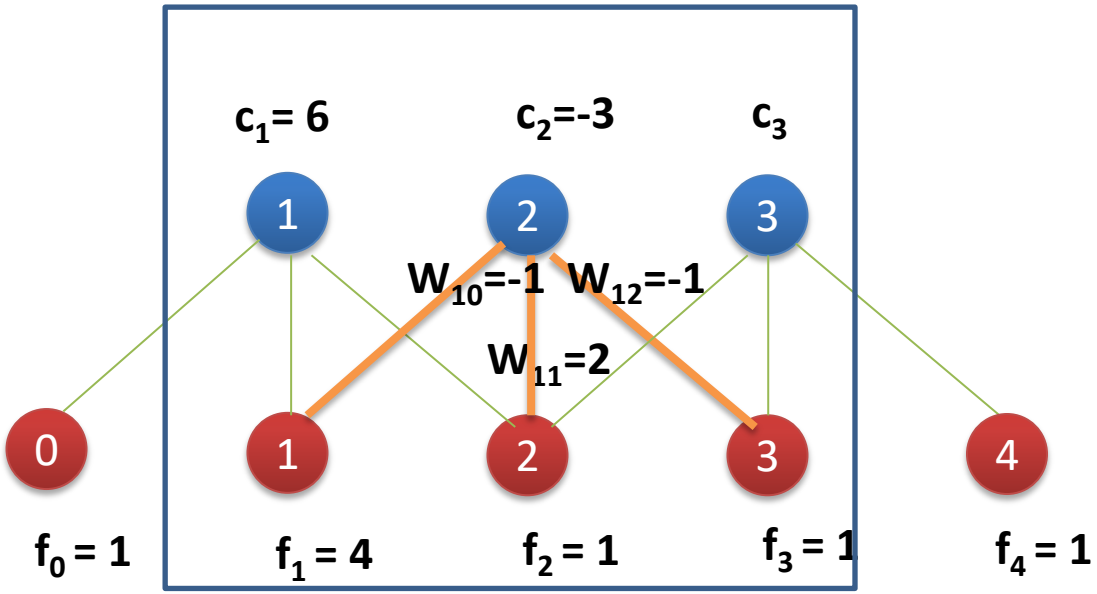


c_1=6 (by applying the weights)
What are **c_2** and **c_3**?

Convolutional Layer Example

Supposing the weights were as follows ...

$W_{10} = -1$
$W_{11} = +2$
$W_{12} = -1$

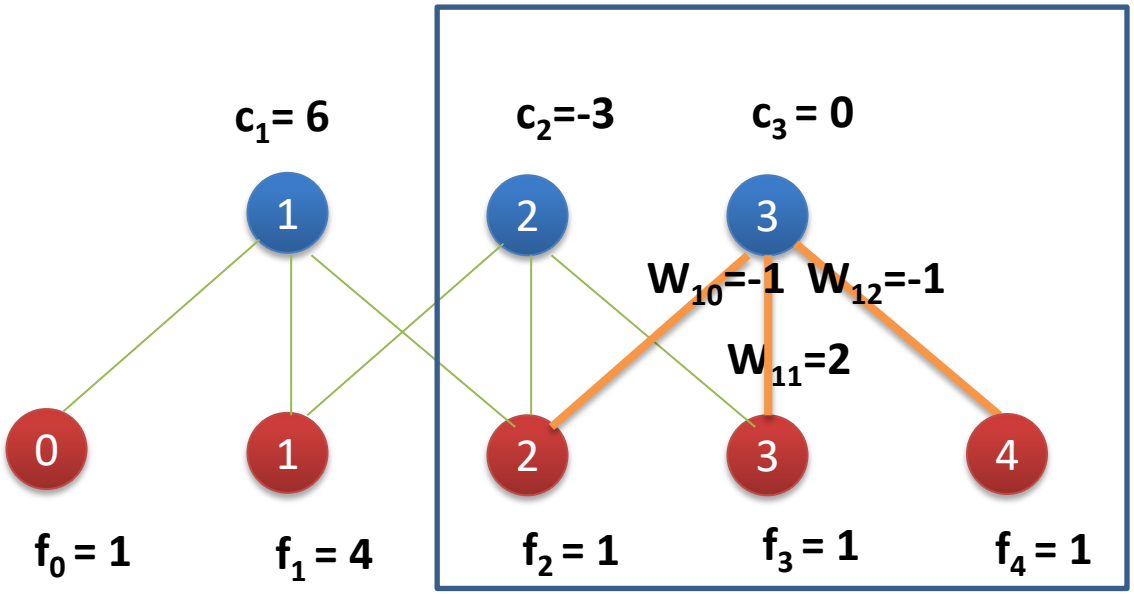


c_2=-3 (by taking another step/stride)
What is **c_3**?

Convolutional Layer Example

Supposing the weights were as follows ...

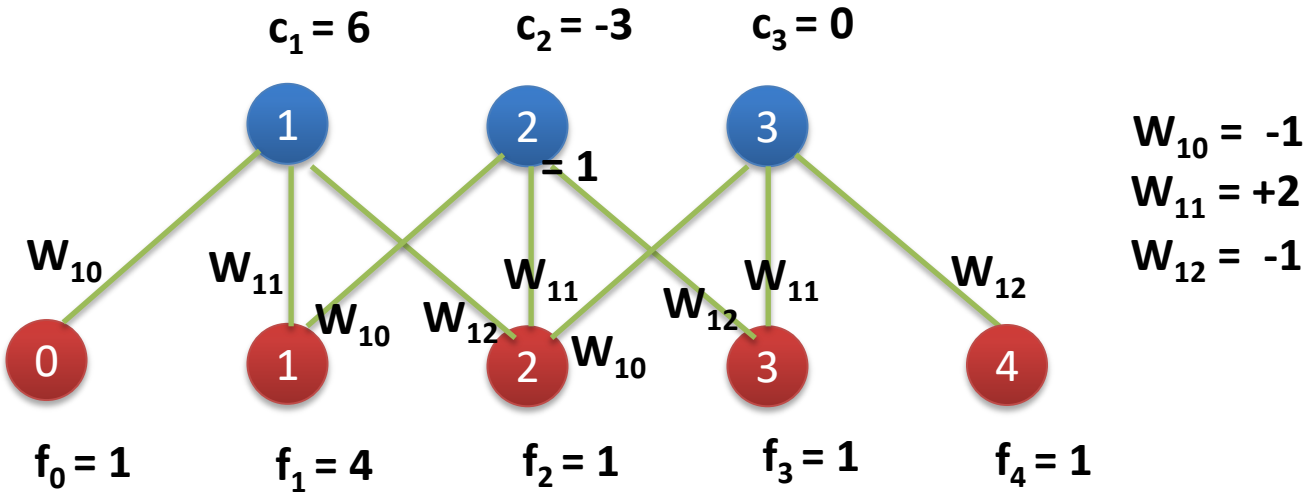
$W_{10} = -1$
$W_{11} = +2$
$W_{12} = -1$



c_3=0 (by taking another step/stride)
Easy!

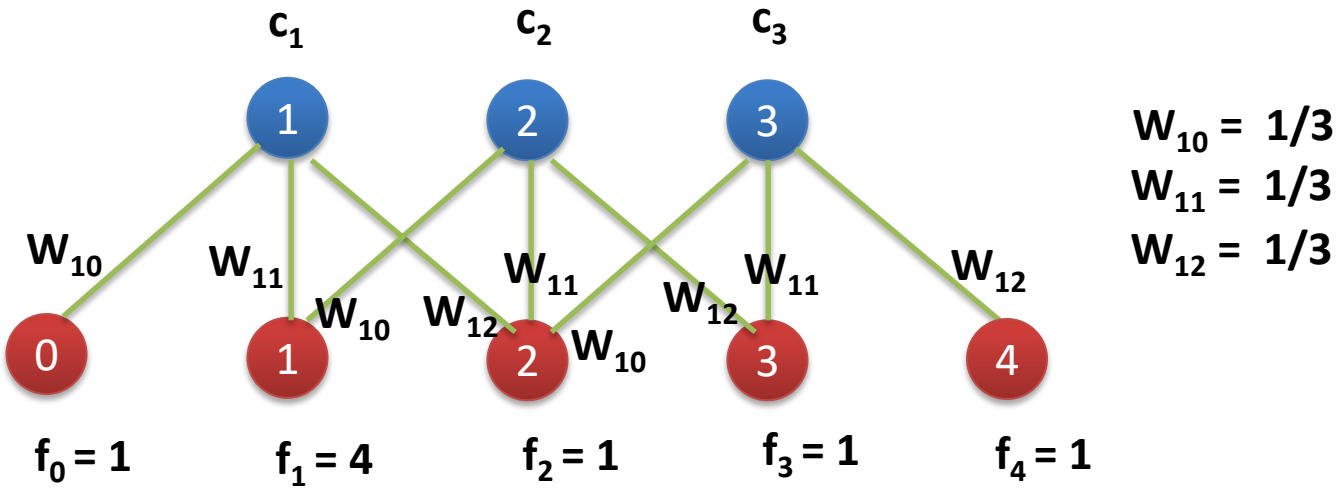
Convolutional Layer Example

The outputs are calculated from local regions of the inputs. Note that we used the same weights for every local region.



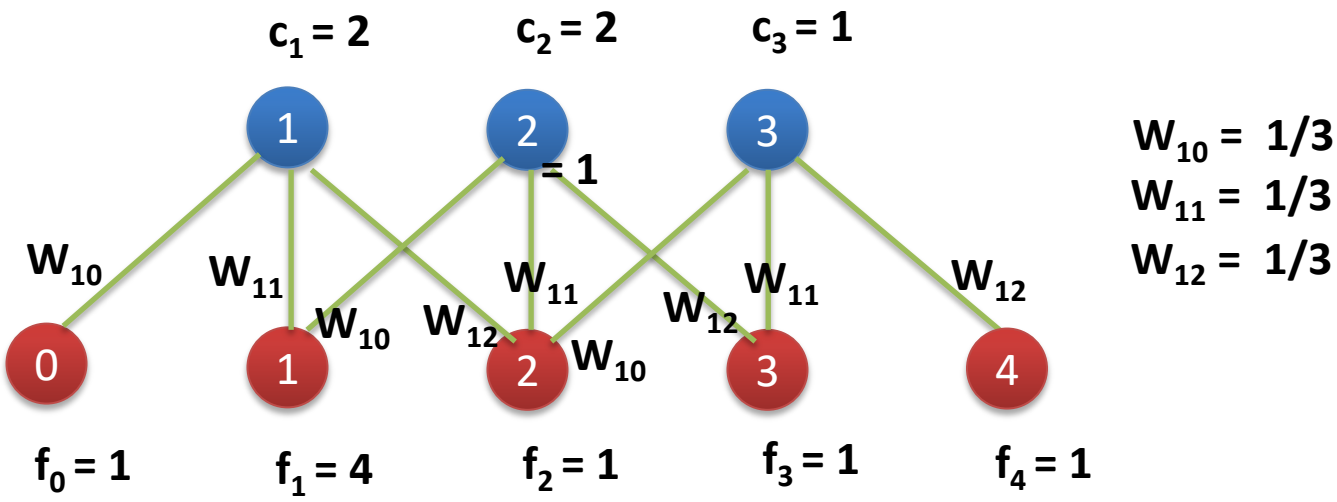
Convolutional Layer Example

Try it again for this set of weights!



What are c_1 , c_2 and c_3?

Convolutional Layer Example



The outputs are smoother … these weights act as low-pass filters and smooth the output (blur the image). (They average all the pixels in the local region).

2D Convolutional Layer

With images, your convolutions are in 2 dimensions …

Let's say we have a 5x5 image with 1 bit pixels like this.

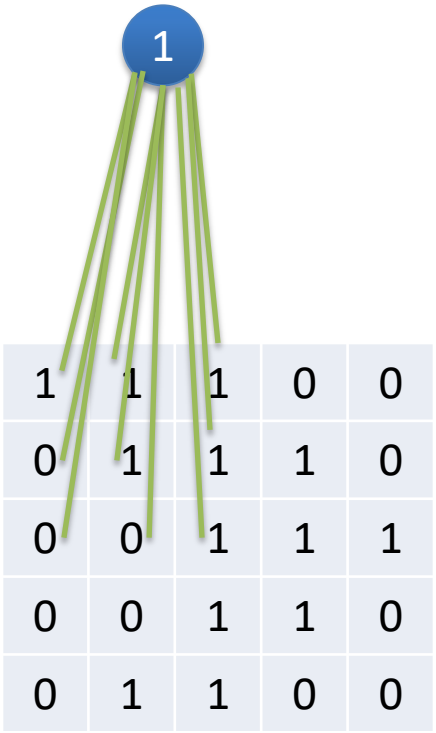
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

Let's say the convolutional layer covers a 3x3 region of the image …

It takes as input a 3x3 region of the image and produces one output … which is the sum of the weighted inputs.



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

Let's say the convolutional layer covers a 3x3 region of the image … its weights can be represented by a grid …

weights

1	0	1
0	1	0
1	0	1

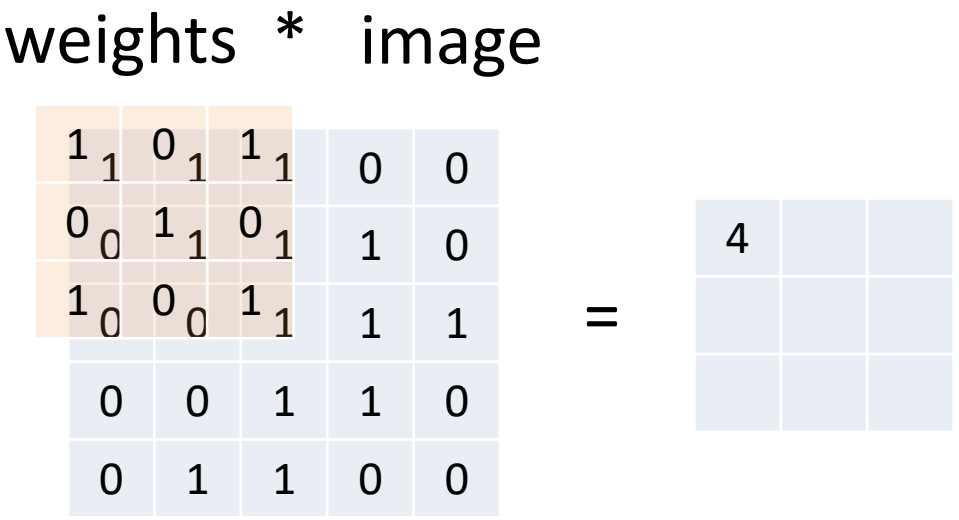
image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

We can apply the convolution just like we did in 1D.

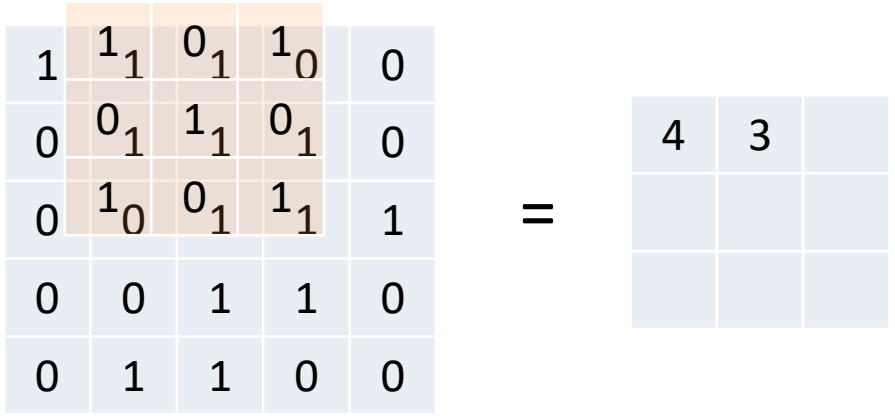


Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

Now we move around the image … stride by stride.
A stride can be one pixel or more, but there's usually an overlap of local regions before and after a stride.

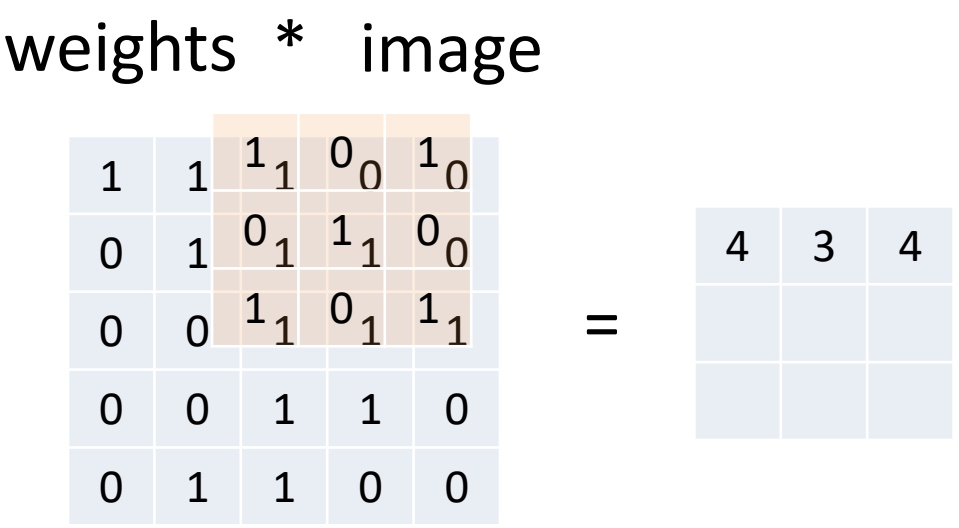
weights * image



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

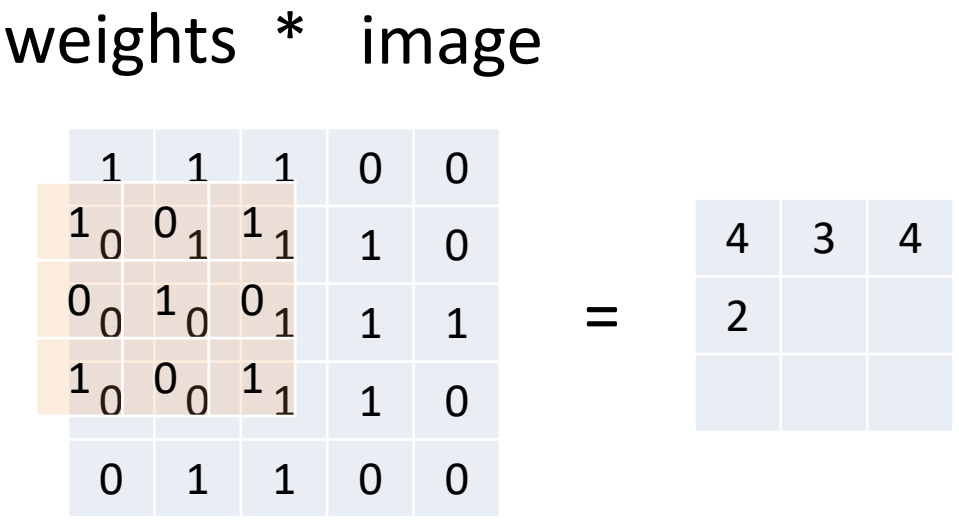
Since we are moving 1 pixel in each step, the stride here is considered as 1.



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

Since we are moving 1 pixel in each step, the stride here is considered as 1 in both x and y axes.



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

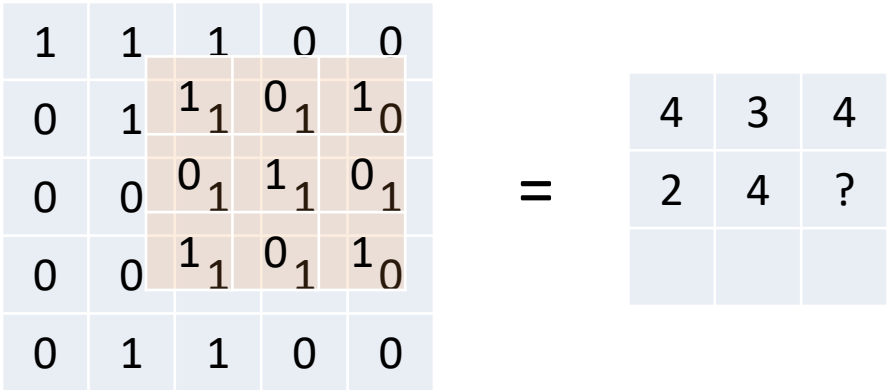
Since we are moving 1 pixel in each step, the stride here is considered as 1 in both x and y axes.

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

Now you do it! What's the next one?

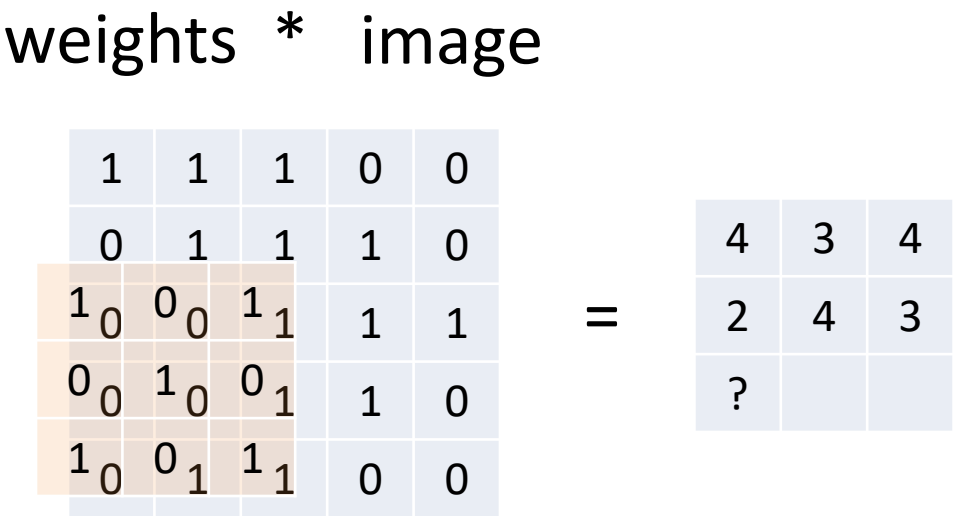
weights * image



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

What's the next one?



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

What's the next one?

weights * image

1	1	1	0	0
0	1	1	1	0
0	1_0	0_1	1_1	1
0	0_0	1_1	0_1	0
0	1_1	0_1	1_0	0

=

4	3	4
2	4	3
2	?	

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

What's the next one?

weights * image

1	1	1	0	0
0	1	1	1	0
0	0	1	0	1
0	0	0	1	0
0	1	1	0	1

=

4	3	4
2	4	3
2	3	?

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

There you have it!

This is sometimes called a feature map.

weights * image

1	0	1
0	1	0
1	0	1

*

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

=

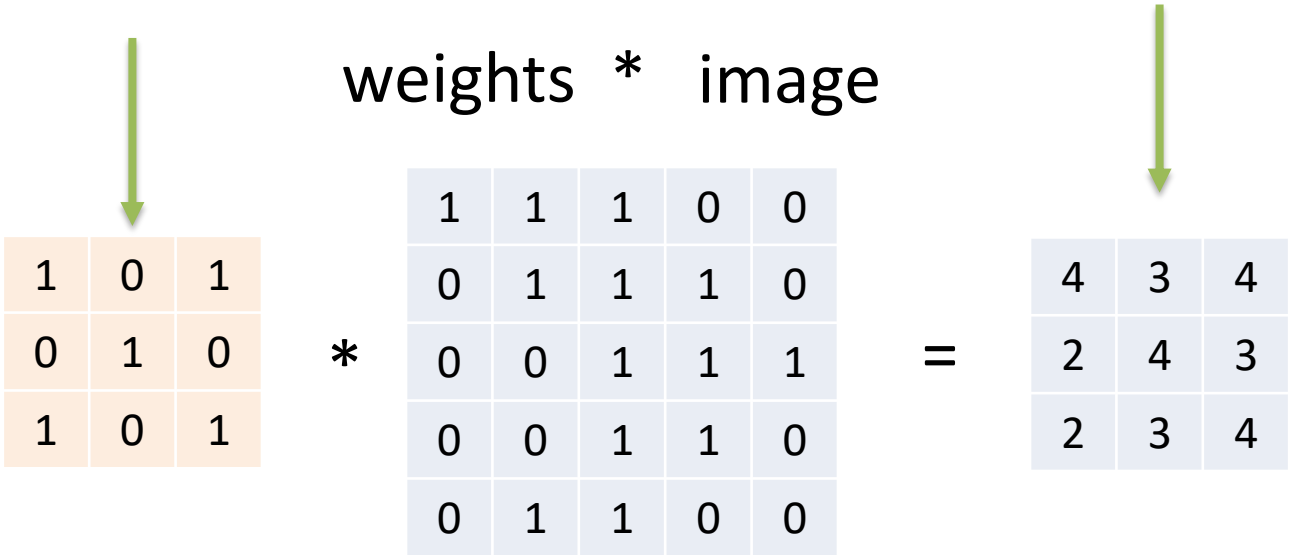
4	3	4
2	4	3
2	3	4

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

And the weights are called a kernel.

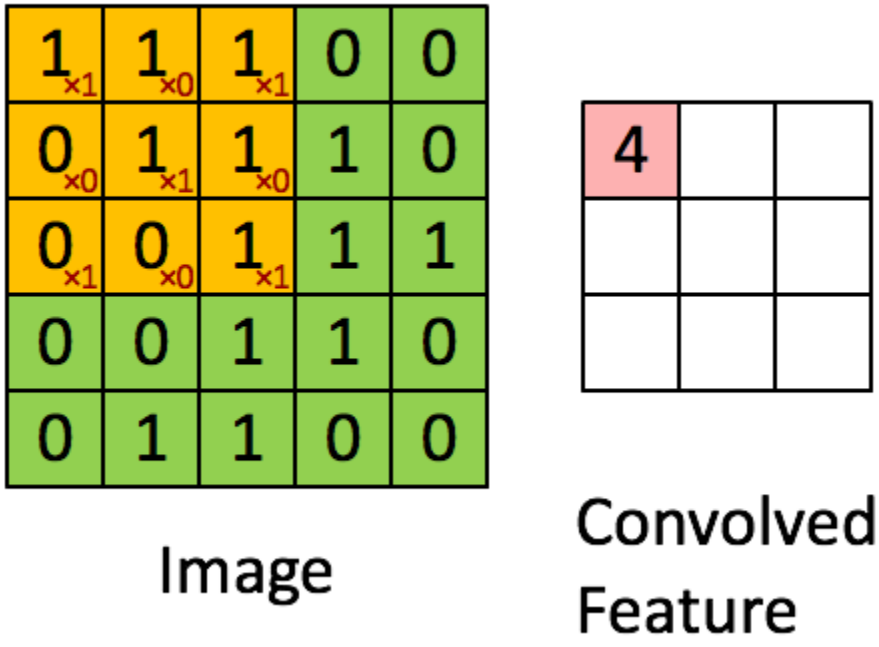
This is sometimes called a feature map.



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

You can watch that as an animation here …



Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

2D Convolutional Layer

In some convolutional layers you can use **multiple kernels** to produce **multiple feature maps**.

weights * image

1	0	1
0	1	0
1	0	1

0	1	0
1	-4	1
0	1	0

*

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

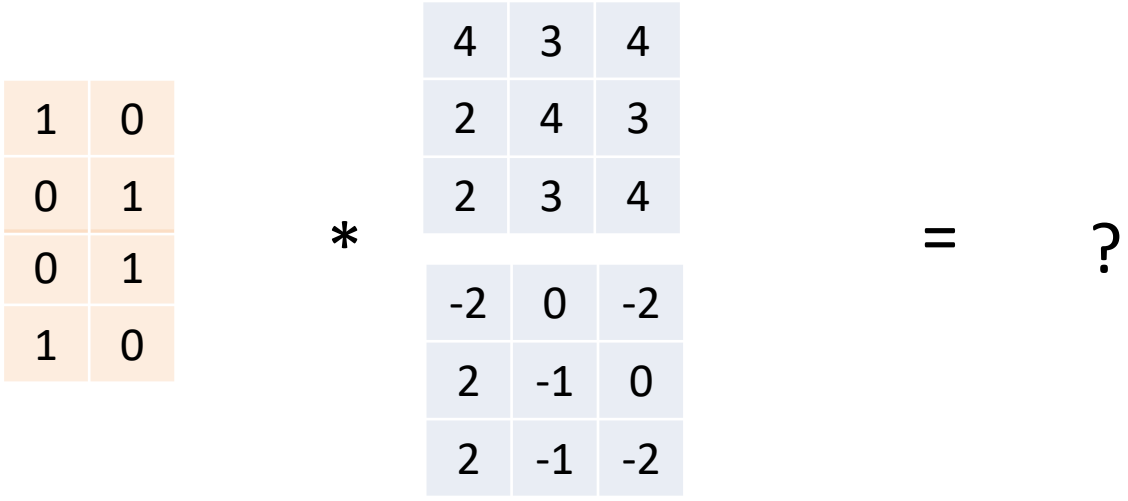
=

4	3	4
2	4	3
2	3	4

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

2D Convolutional Layer

You can also use the **feature maps** as inputs to higher convolutional layers. The kernels can take inputs from multiple feature maps.



weights * image

2D Convolutional Layer

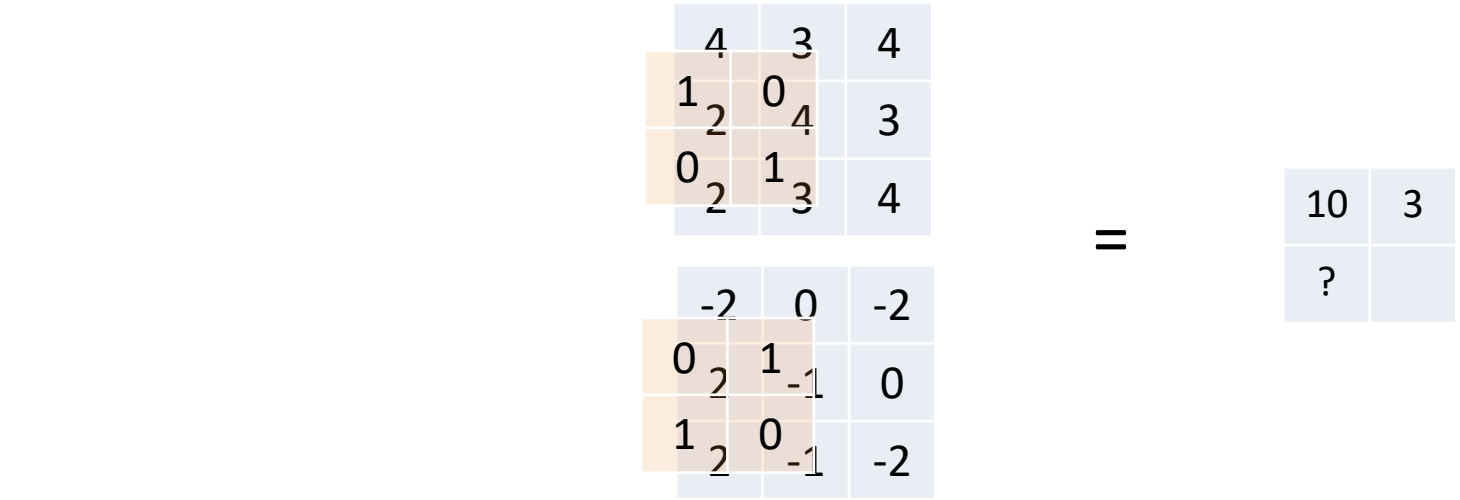
You can also use the **feature maps** as inputs to higher convolutional layers.

2D Convolutional Layer

You can also use the **feature maps** as inputs to higher convolutional layers.

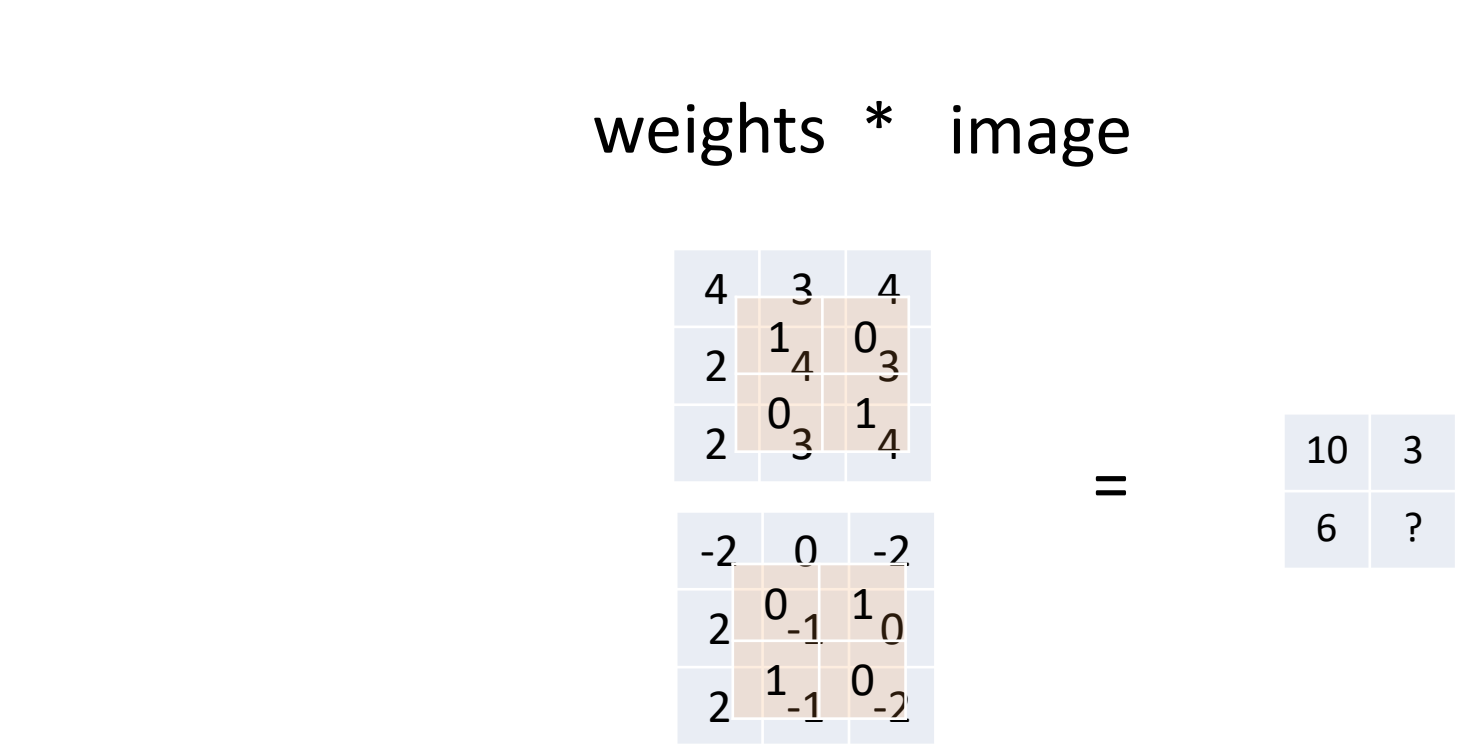
2D Convolutional Layer

Your turn. What's the next output?



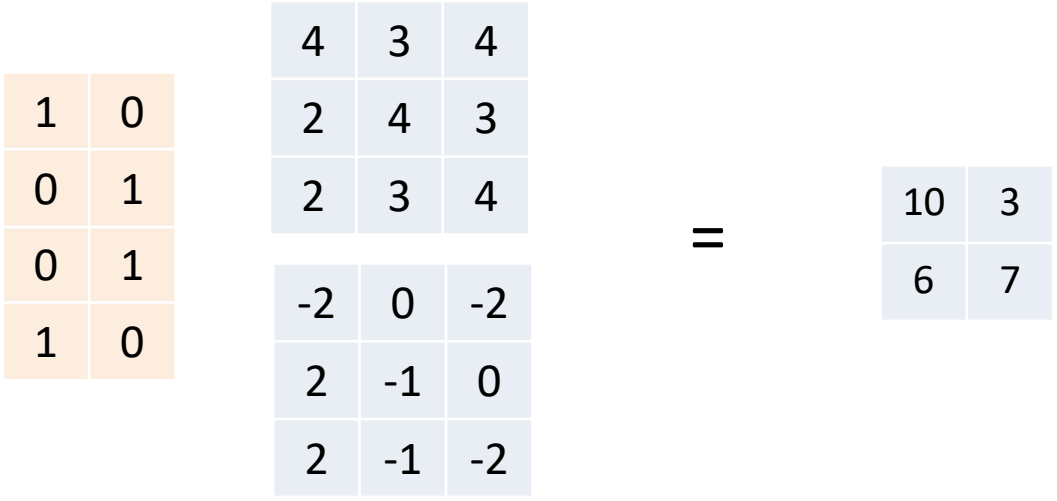
2D Convolutional Layer

Your turn. What's the next output?



2D Convolutional Layer

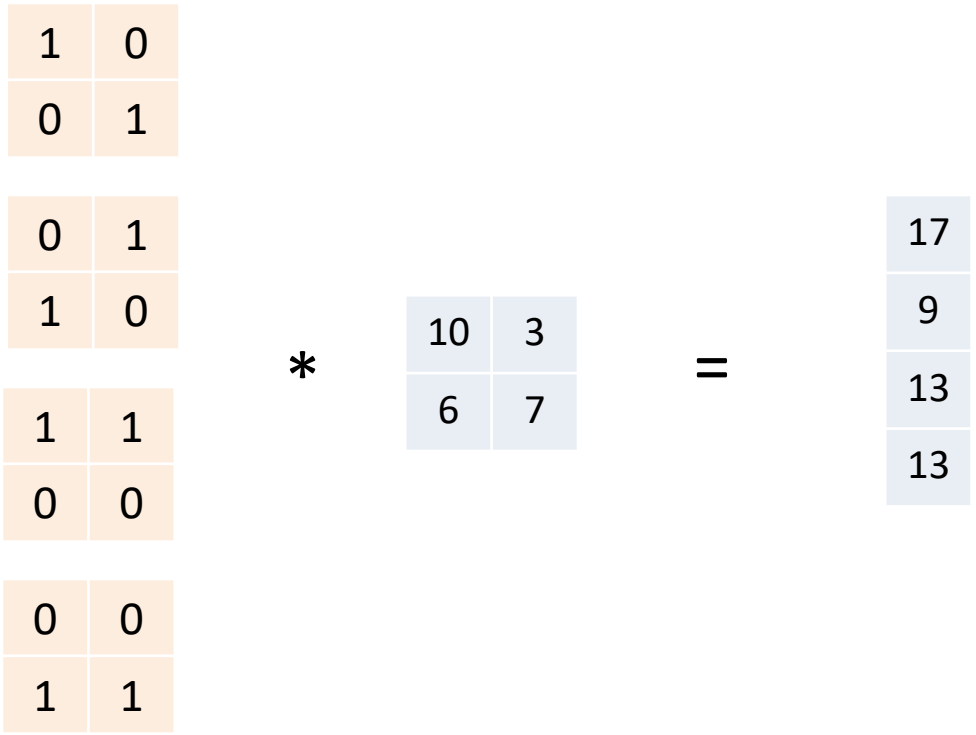
The number of output feature maps depends on the number of kernels.



weights * image

2D Convolutional Layer

You can flatten a 2D image by using kernels of the same size as the image (the number of kernels equals the output vector).



LeNet5 Image Classification

We're now familiar with 2 of the 3 types of layers:

F layers = Fully connected layers
C layers = Convolutional Layers
S layers = Subsampling Layers

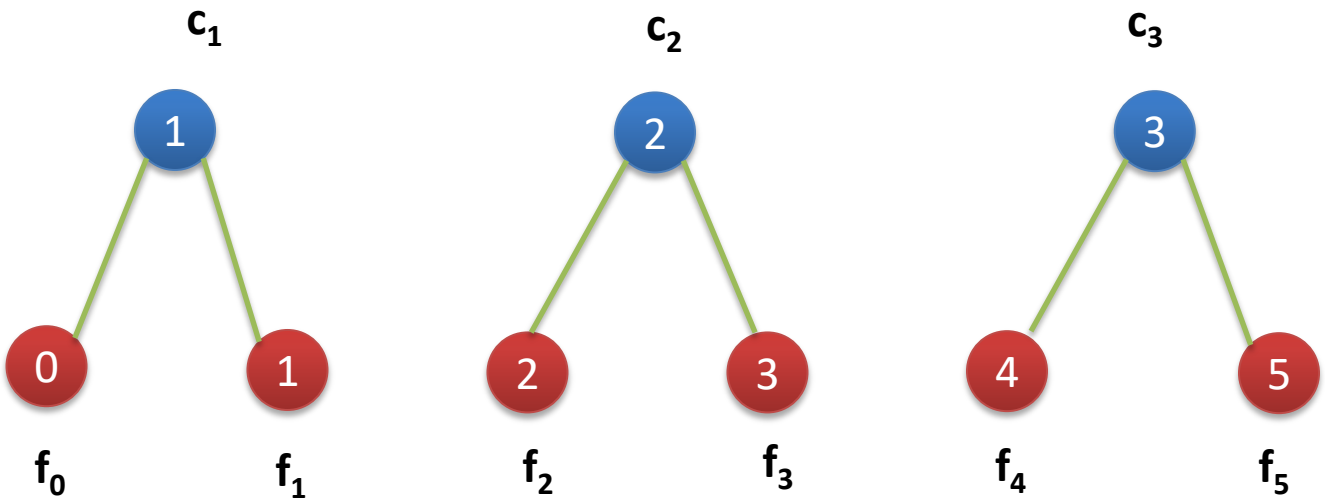
Let's now see how subsampling layers work.

Their purpose, according to LeCun et al, is to reduce the sensitivity of the output to shifts and distortions by reducing the resolution of the feature map.

* LeCun et al "Gradient-Based Learning Applied to Document Recognition"

Pooling (Subsampling) Layer

No overlap in connections to inputs, and resolution is reduced (in order to increase translational invariance).
There are no weights involved.
There are two common types: max and average.

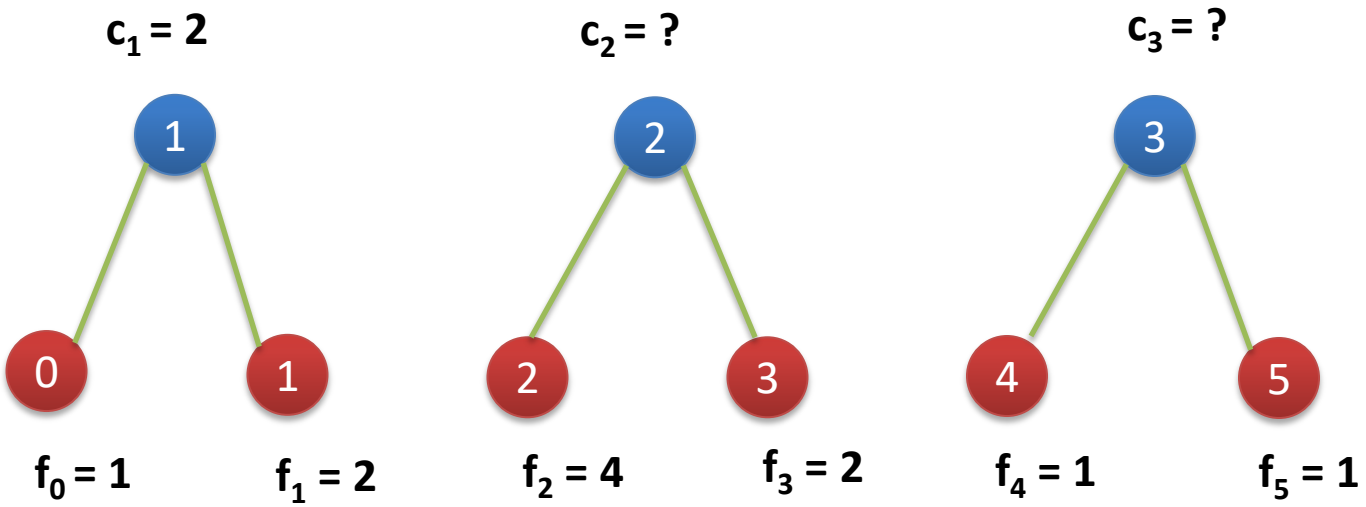


Max pooling: $c_{i/2} = \max(f_i, f_{i+1})$

Average pooling: $c_{i/2} = (f_i + f_{i+1})/2$

Pooling Example

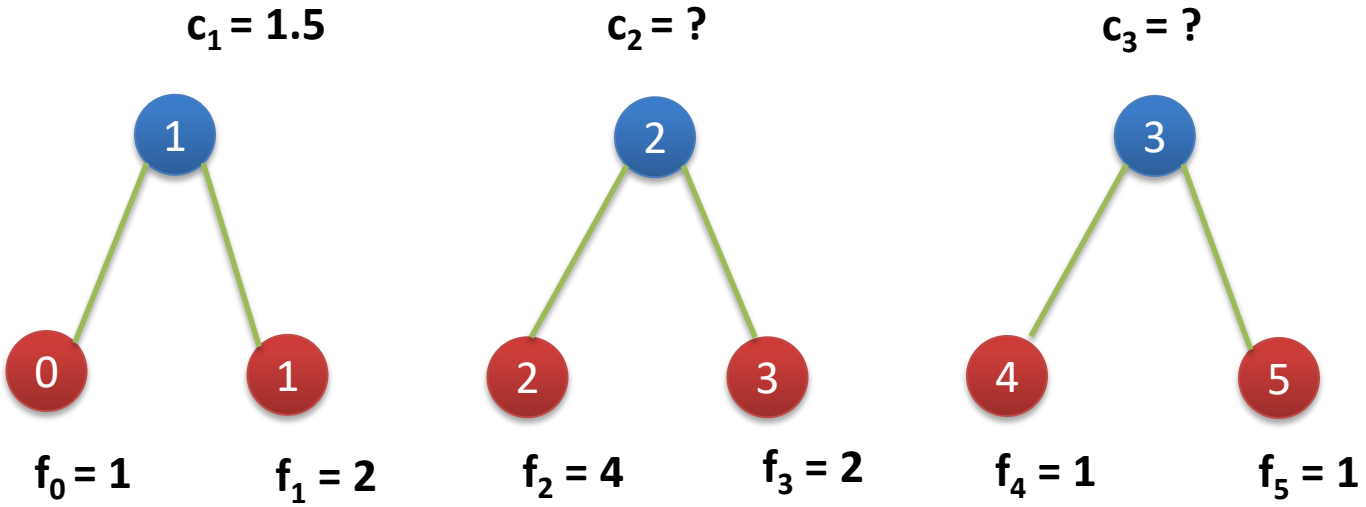
Try it for these inputs. Let's try max pooling.



Max pooling: $c_{i/2} = \max(f_i, f_{i+1})$

Pooling Example

Try it for these inputs. Let's try average pooling.



Average pooling: $c_{i/2} = (f_i + f_{i+1})/2$

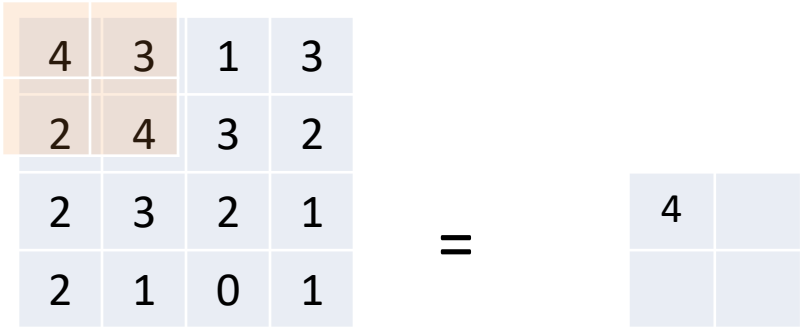
Subsampling Layers

Let's max pooling in 2 dimensions.

Subsampling Layer

It works similarly in 2 dimensions.

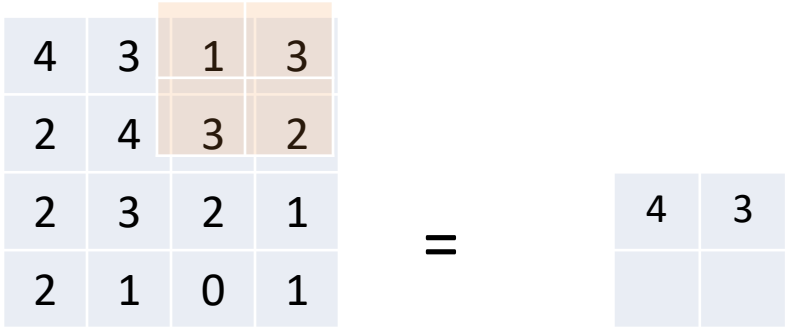
Max pooling



Subsampling Layer

It works similarly in 2 dimensions.

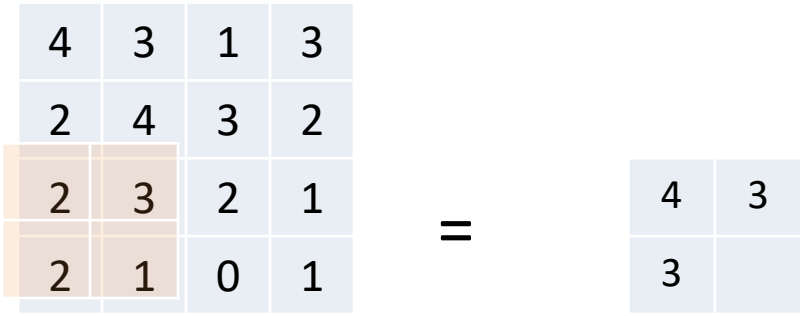
Max pooling



Subsampling Layer

It works similarly in 2 dimensions.

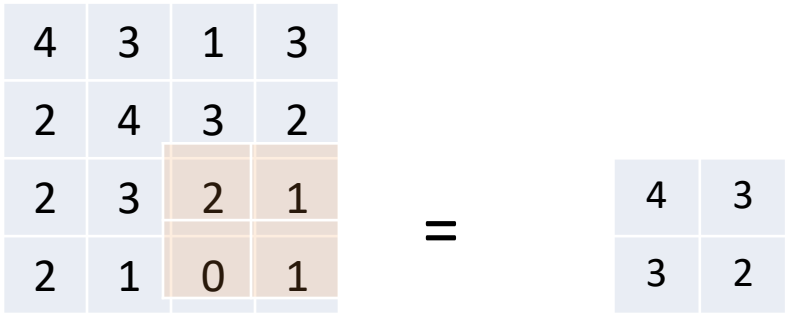
Max pooling



Subsampling Layer

It works similarly in 2 dimensions.

Max pooling



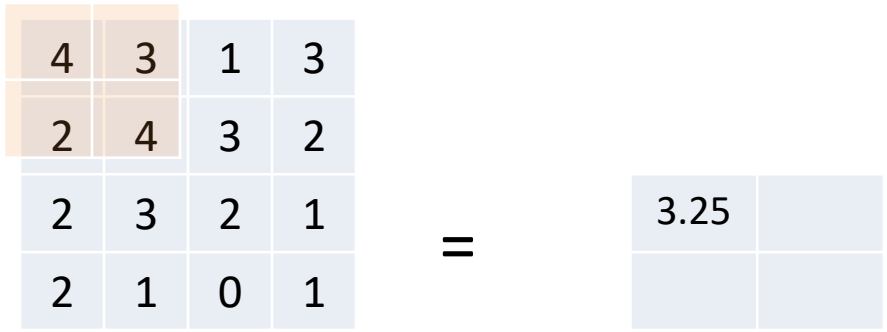
Subsampling Layers

Let's try average pooling in 2 dimensions.

Subsampling Layer

It works similarly in 2 dimensions.

Average pooling



Subsampling Layer

It works similarly in 2 dimensions.

Average pooling

4	3	1	3
2	4	3	2
2	3	2	1
2	1	0	1

=

3.25	2.25

Subsampling Layer

It works similarly in 2 dimensions.

Average pooling

4	3	1	3
2	4	3	2
2	3	2	1
2	1	0	1

=

3.25	2.25
2	

Subsampling Layer

It works similarly in 2 dimensions.

Average pooling

4	3	1	3
2	4	3	2
2	3	2	1
2	1	0	1

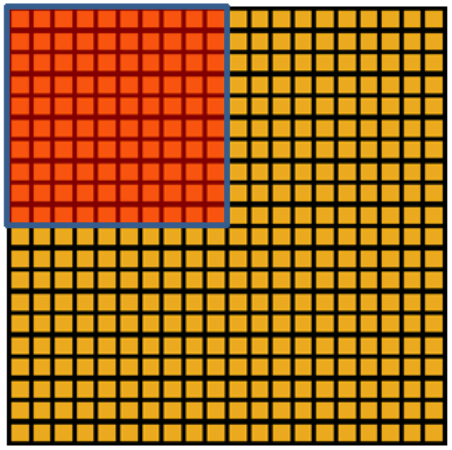
=

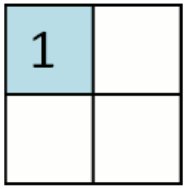
3.25	2.25
2	1

2D Subsampling Layer

In 2 dimensions …

You're basically shrinking the image.





Convolved feature

Pooled feature

Animated image from: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

LeNet 5

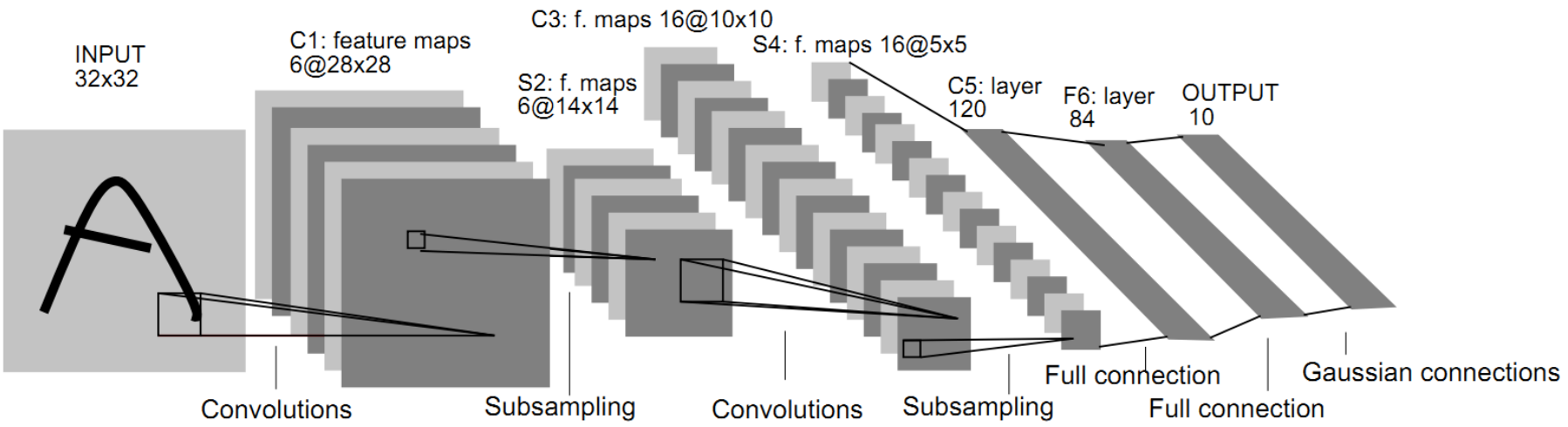
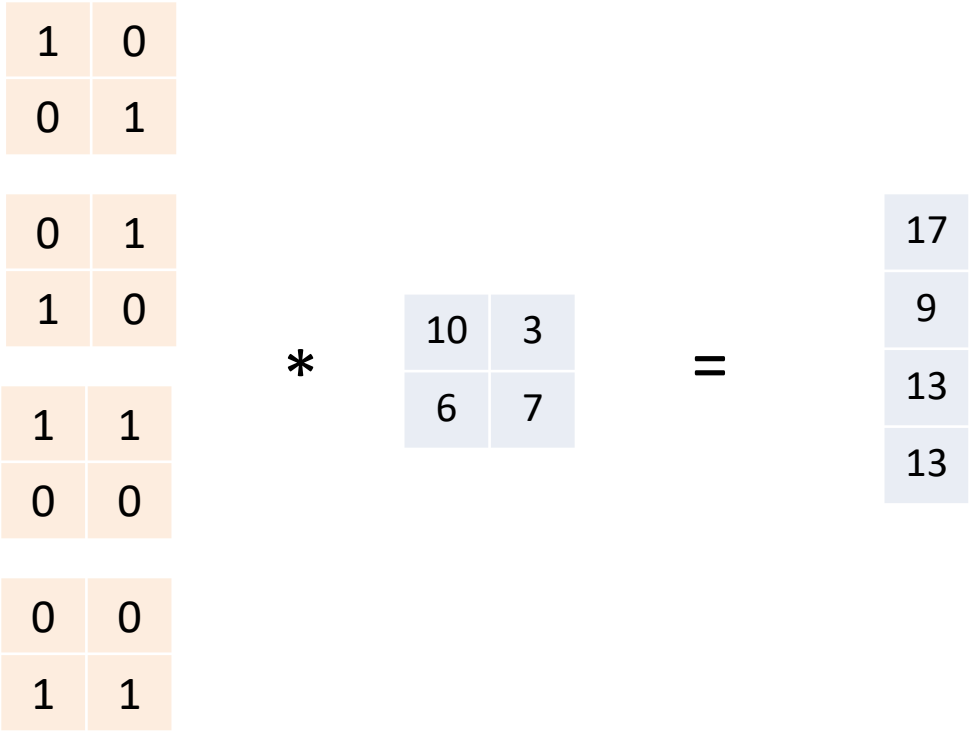


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

C1 layer = Convolutional layer mapping 1 channel to 6
S2 layer = Subsampling layer mapping 6 channels to 6
C3 layer = Convolutional layer mapping 6 channels to 16
S4 layer = Subsampling layer mapping 16 channels to 16
C5 layer = Convolutional layer mapping 16 channels to 120
F6 layer = Maps input vector of length 120 to an output vector of length 84
F7 layer = Maps input vector of length 84 to an output vector of length 10

2D Convolutional Layer

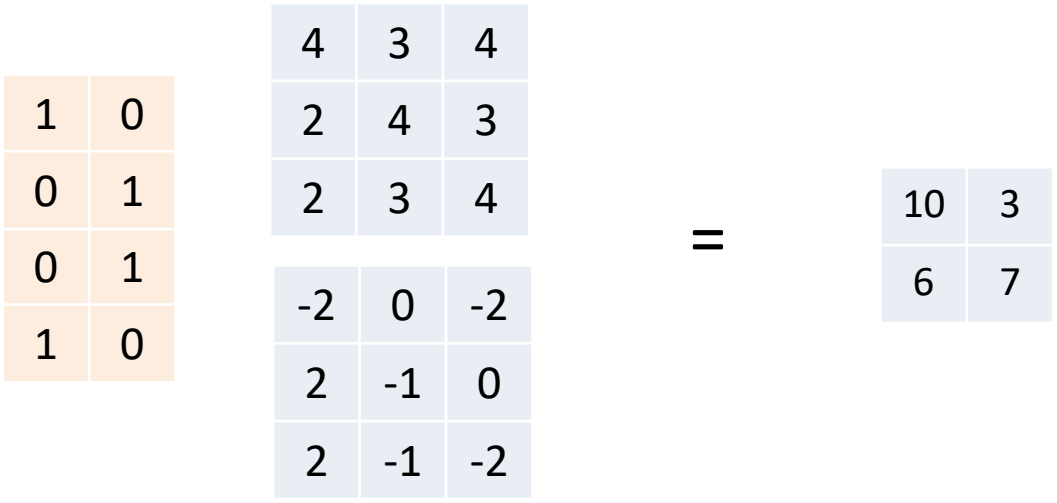
This convolutional layer has 4 kernels (each taking inputs from 1 feature map). It can be thought of as a mapping from 1 channel (feature map) to 4 channels.



weights * image

2D Convolutional Layer

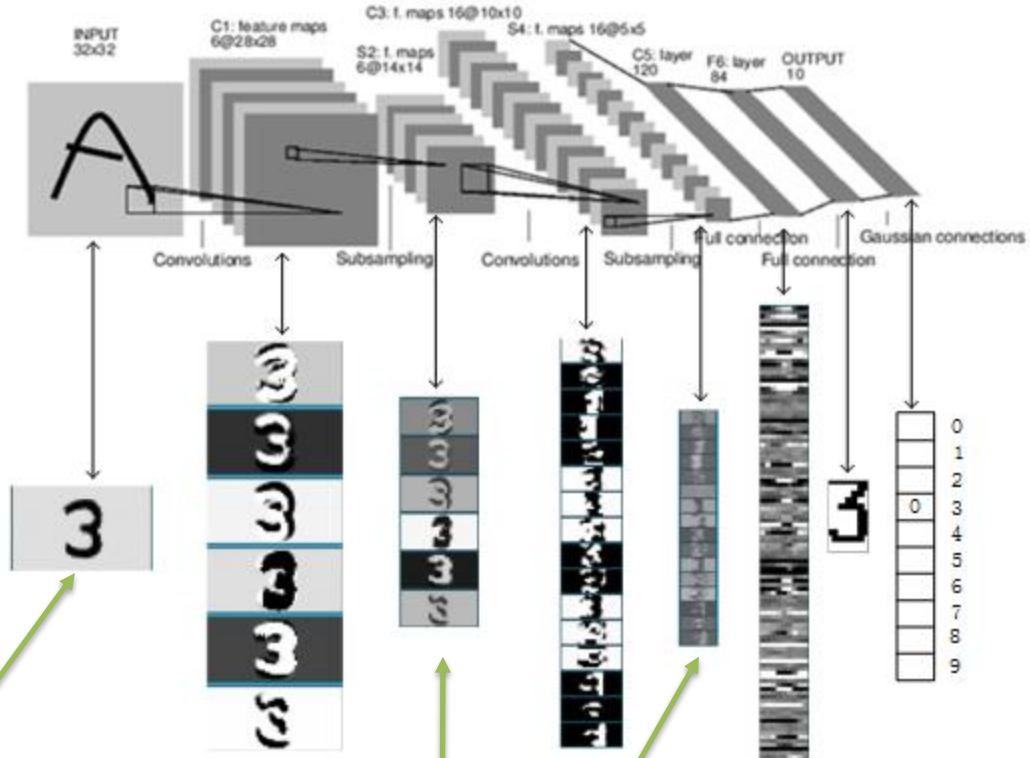
This convolutional layer has 1 kernel (taking inputs from 2 feature maps). It can be thought of as a mapping from 2 channels (feature maps) to 1 channel.



weights * image

LeNet 5

LeNet 5 - what each layer does



205

aiaioo labs

More Powerful Models

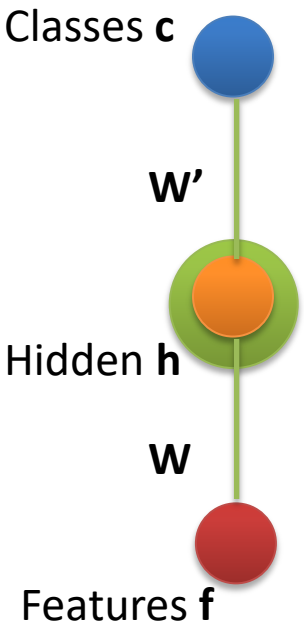
More powerful classification models have been developed in the years since and have achieved superhuman performance on the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) using 1.4 million images categorized into 1000 categories.

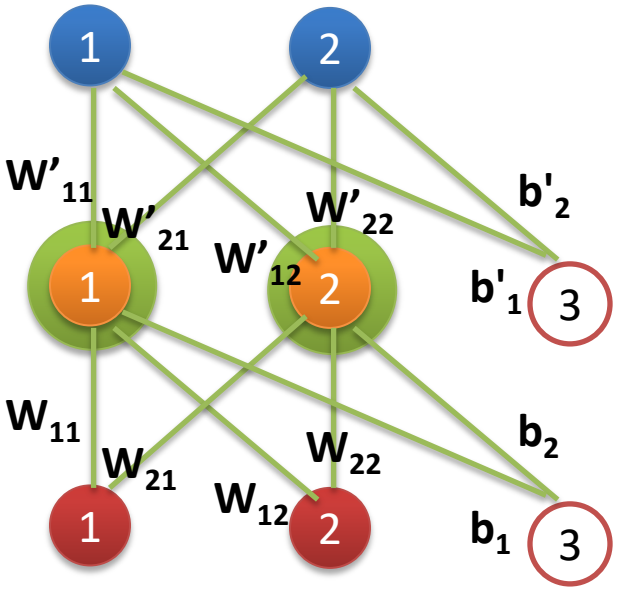
These include:

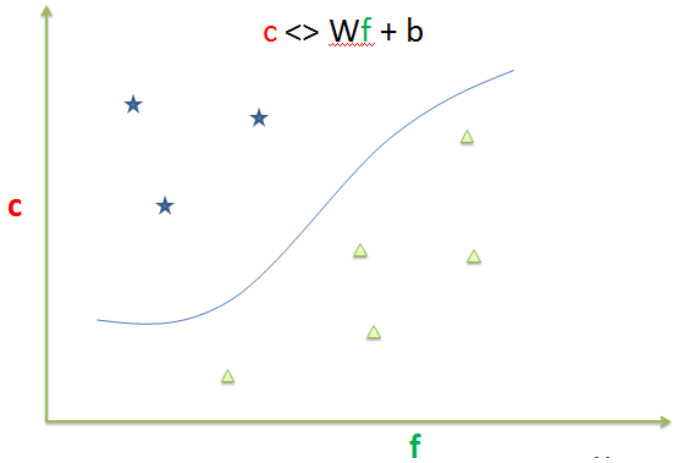
AlexNet (ILSVRC 2012 winner), Clarifai (ILSVRC 2013 winner), VGGNet (ILSVRC 2014 2nd Place), GoogLeNet (ILSVRC 2014 winner), ResNet (ILSVRC 2015 winner).

ResNet had 152 layers. It beat humans (5.1% error rates) on ILSVRC. It had an error rate of just 3.57%.

Deep Learning Models







The deep learning models we have seen are limited to:

a) a fixed number of features (f_1 & f_2)

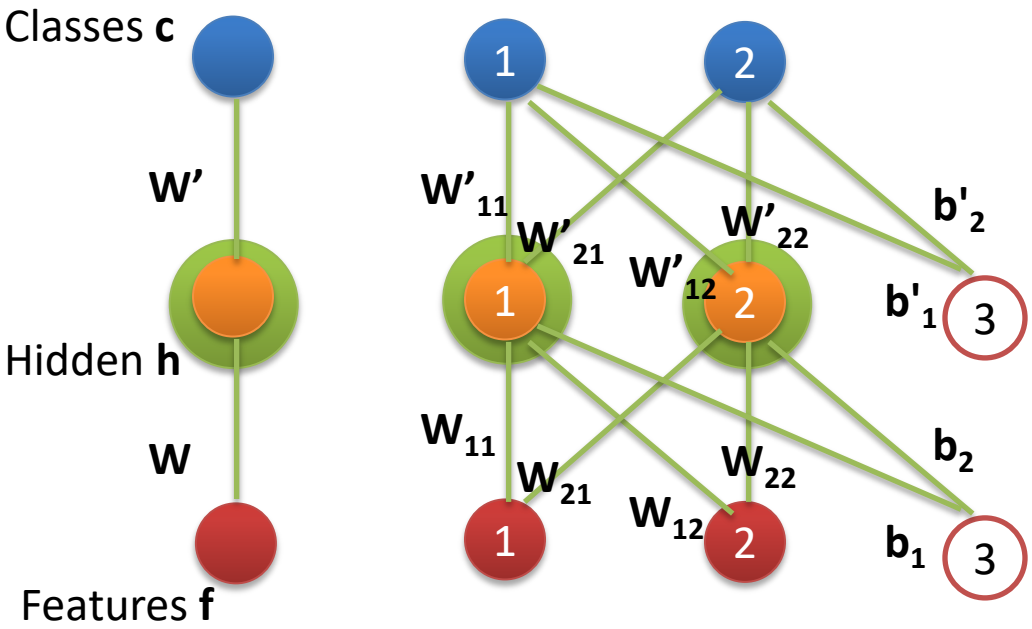
b) all features being read at one shot

Sequential Deep Learning Models

But there are lots of real world problems where the features form long sequences (that is, they have an ordering):

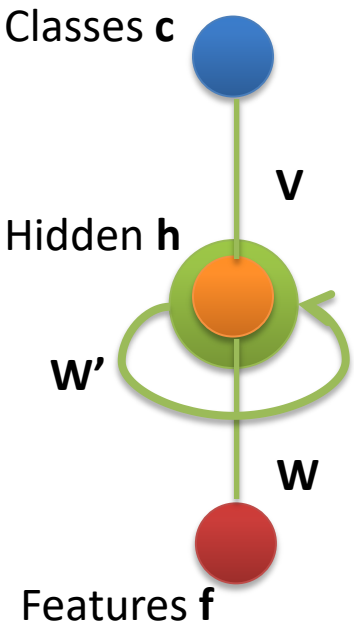
a) **Speech recognition**
b) **Machine translation**
c) **Handwriting recognition**
d) **DNA sequencing**
e) **Self-driving car sensor inputs**
f) **Sensor inputs for robot localization**

Sequential Deep Learning Models



- Is there a deep learning model that can be presented with features sequentially?

Sequential Deep Learning Models

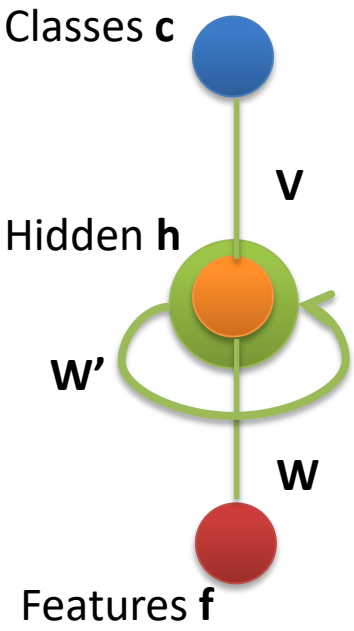


Yes.

They're called …

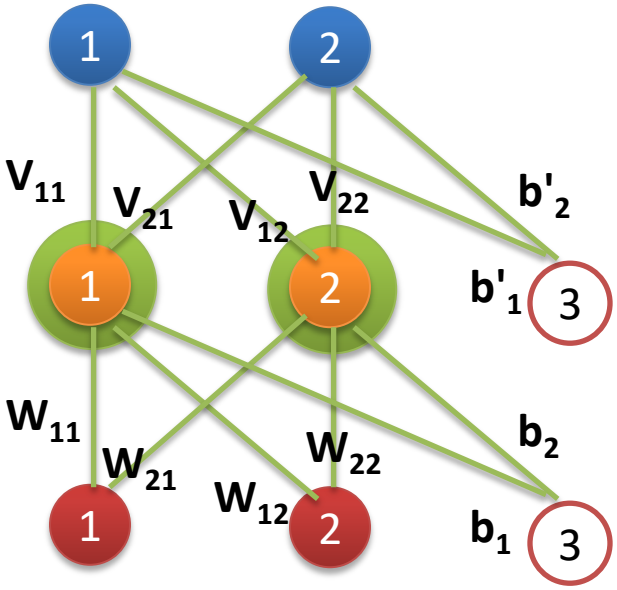
Recurrent Neural Networks (RNNs):

Sequential Deep Learning Models



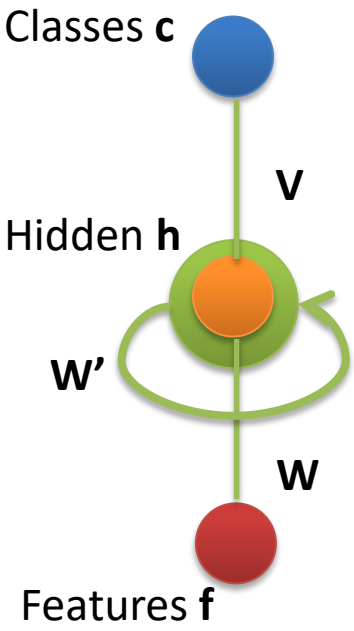
Recurrent Neural Networks (RNNs):

At time t = 0



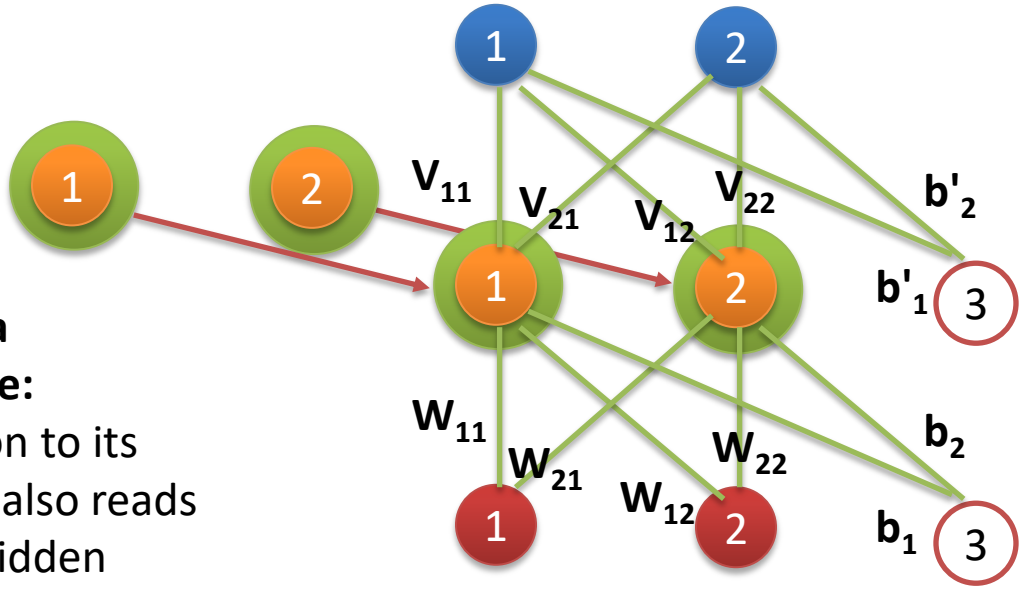
At any point in time, an RNN looks almost like a regular multilayer neural network …
Almost!

Sequential Deep Learning Models



Recurrent Neural Networks (RNNs):

At time t = 0

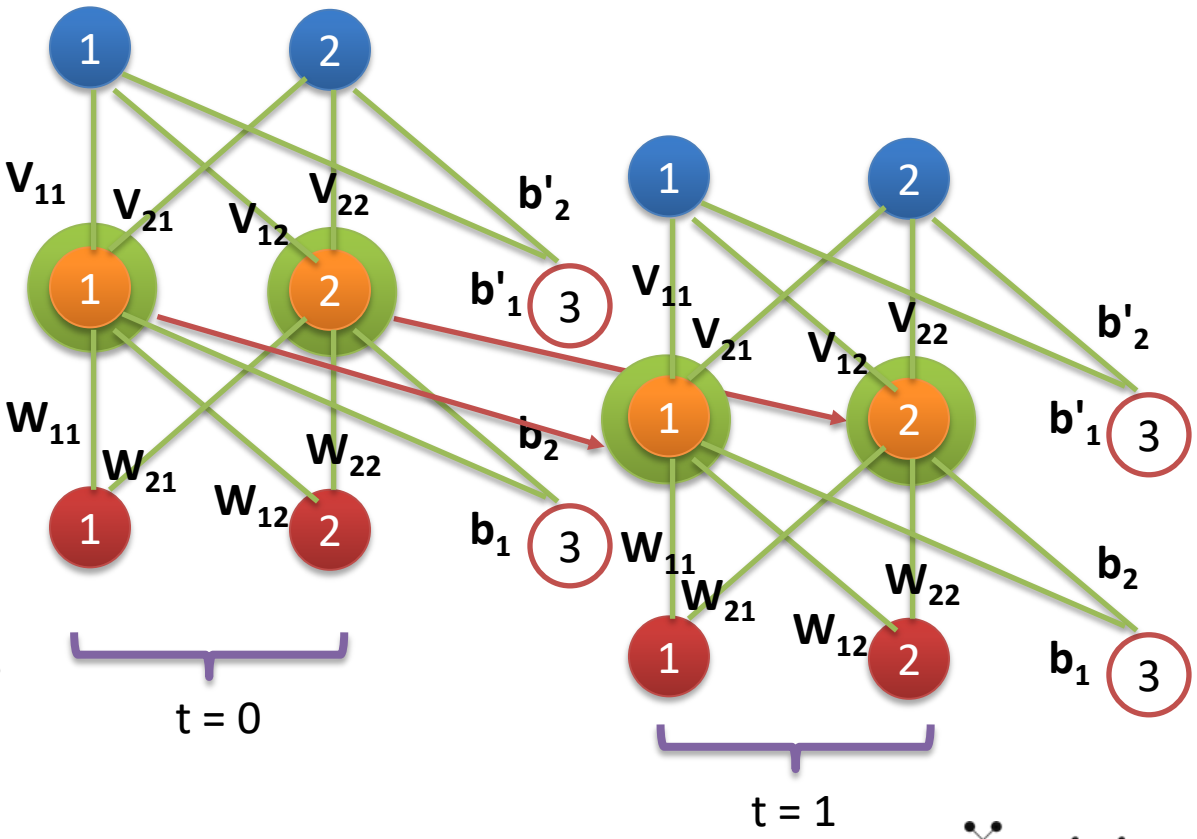


There is a difference:
In addition to its inputs, it also reads its own hidden "state" … from the previous time step.

Sequential Deep Learning Models

Recurrent Neural Networks (RNNs):

At time t = 1

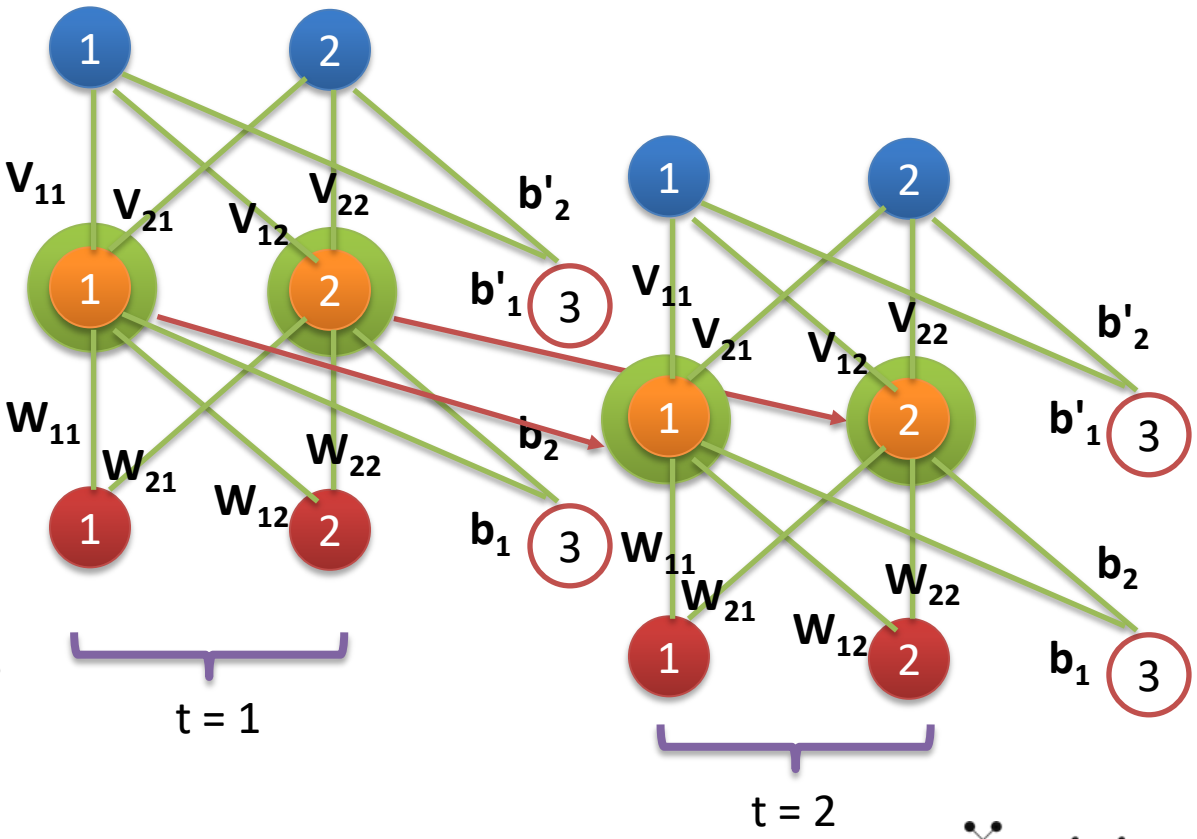


Now when it reads the previous hidden "state" … the vector of previous hidden state values contains the values from t = 0

Sequential Deep Learning Models

Recurrent Neural Networks (RNNs):

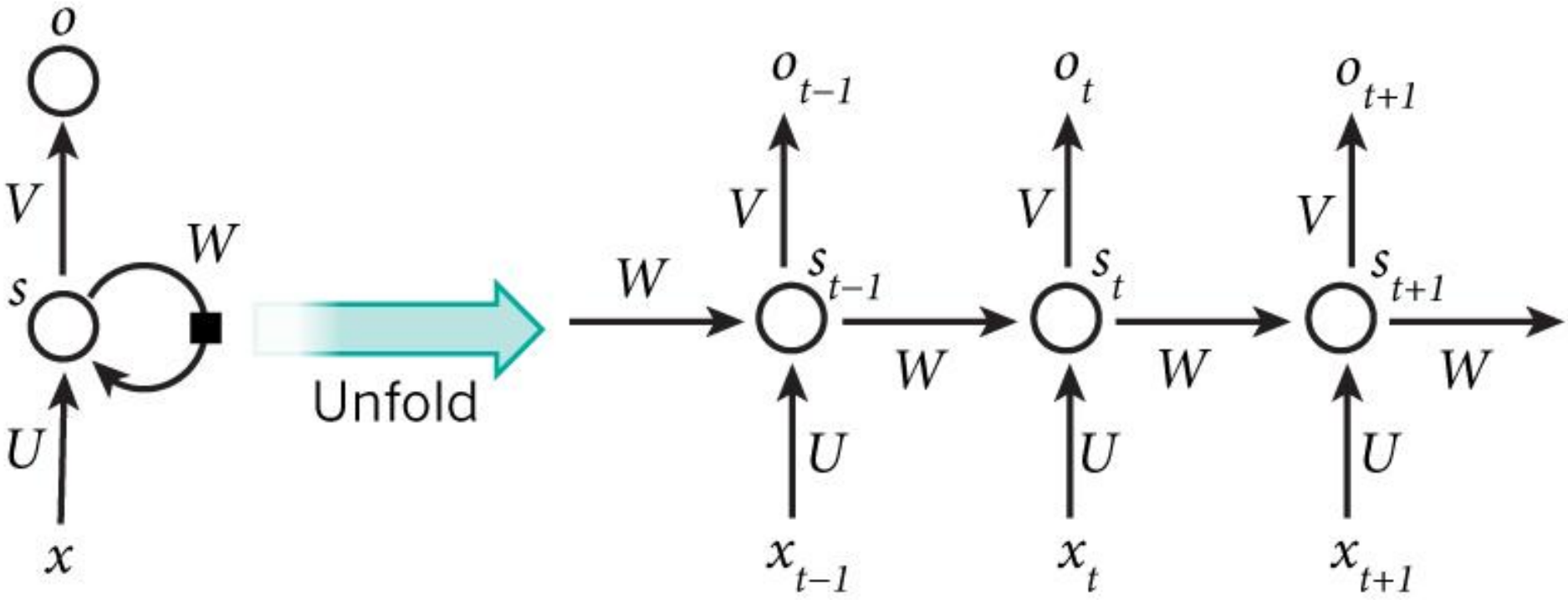
At time t = 2



Now when it reads the previous hidden "state" … the vector of previous hidden state values contains the values from t = 1

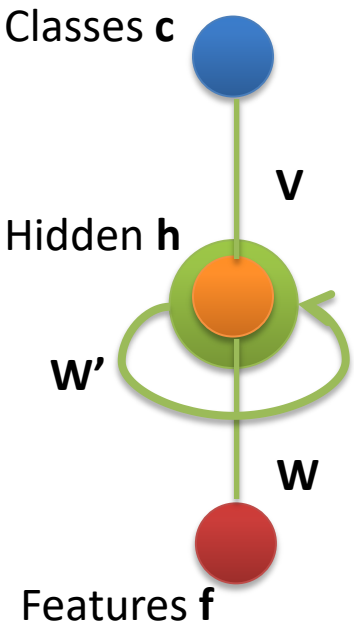
Sequential Deep Learning Models

Illustration of RNNs from the WildML blog.



http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Sequential Deep Learning Models



We're going to explore RNNs using a toy problem - adding up the bits in a binary sequence.

Generated data:

2	0 1 0 0 1
3	1 0 0 1 1
5	1 1 1 1 1
0	0 0 0 0 0

http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/

Sequential Deep Learning Models

Recurrent Neural Networks (RNNs):

Let's say the state is managed by some machinery in a box.

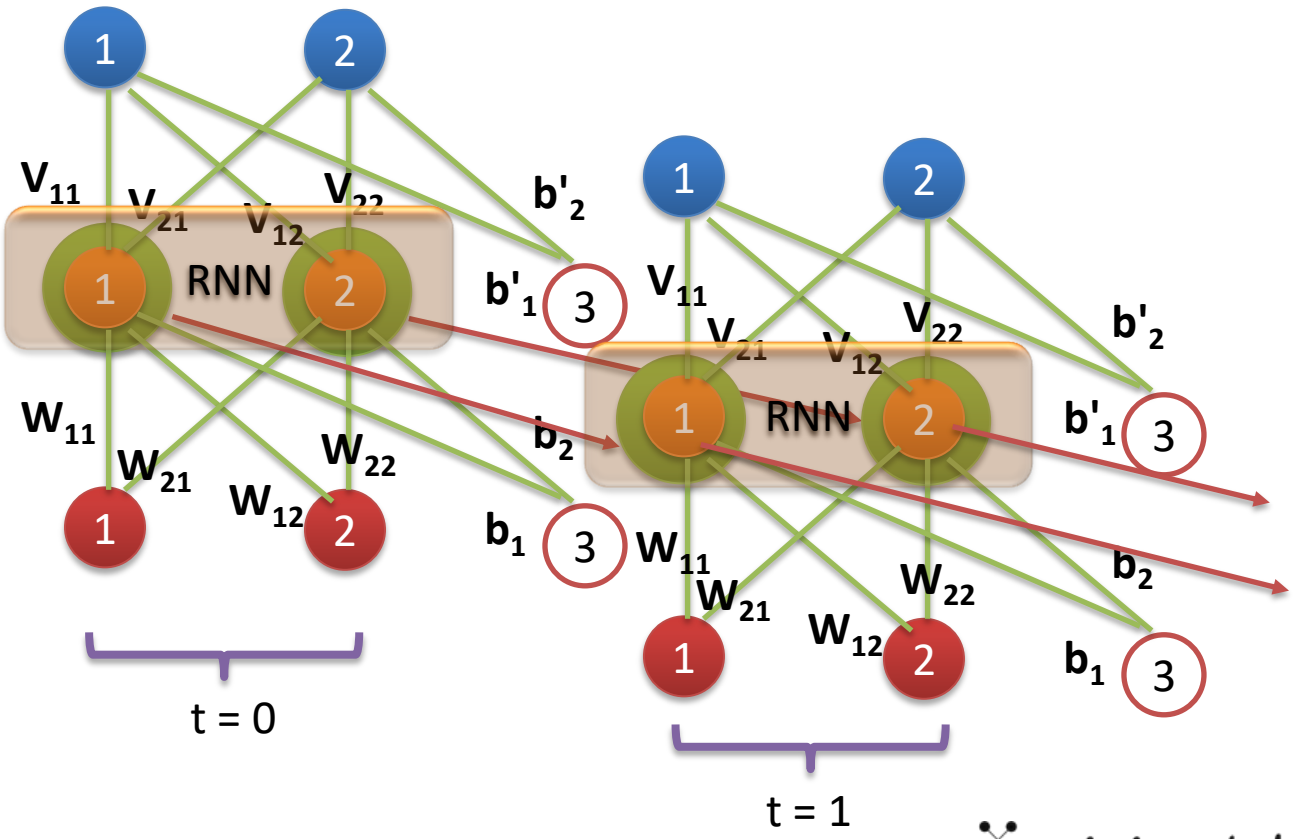
The box is something that takes …

a) an input
b) the previous state

… and returns …

a) an output
b) the new state

… at each time-step.



Sequential Deep Learning Models

Long Short-Term Memory (LSTMs):

At time t = 1

In LSTMs, the box is more complex.

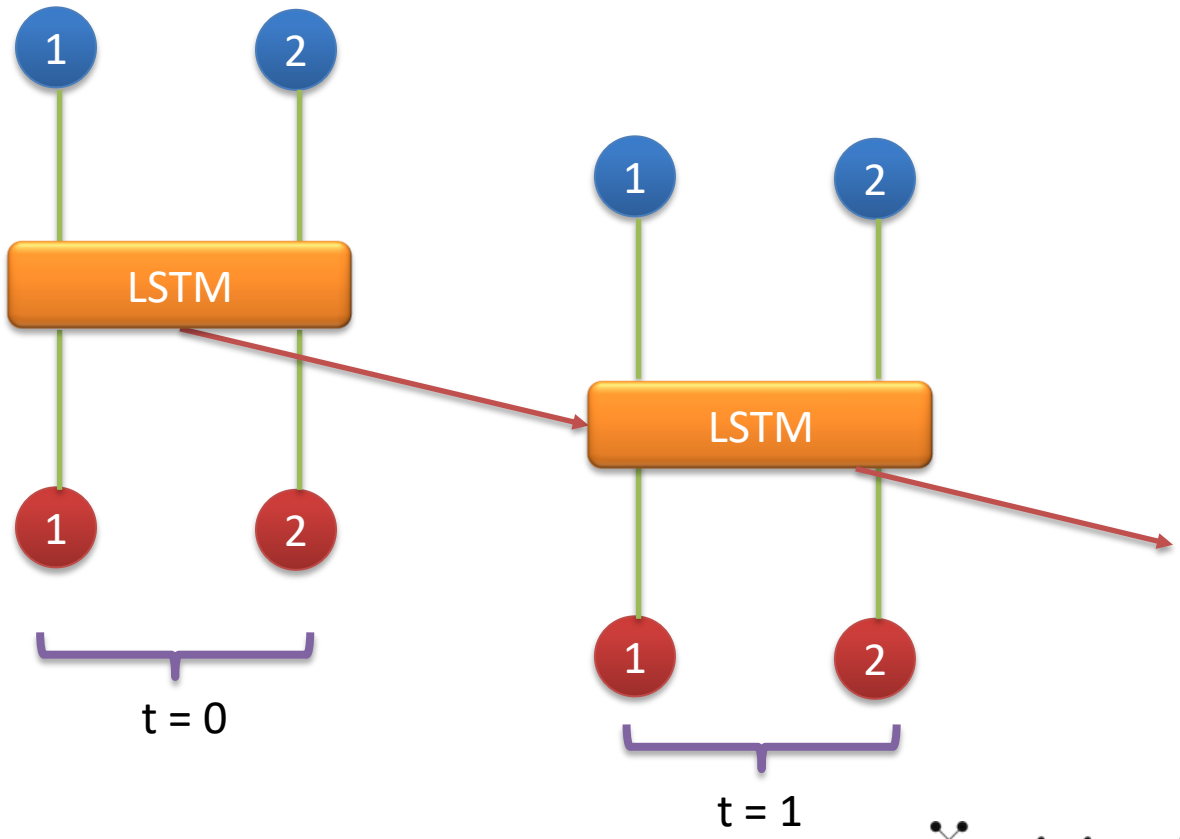
But it still takes …

a) an input
b) the previous state

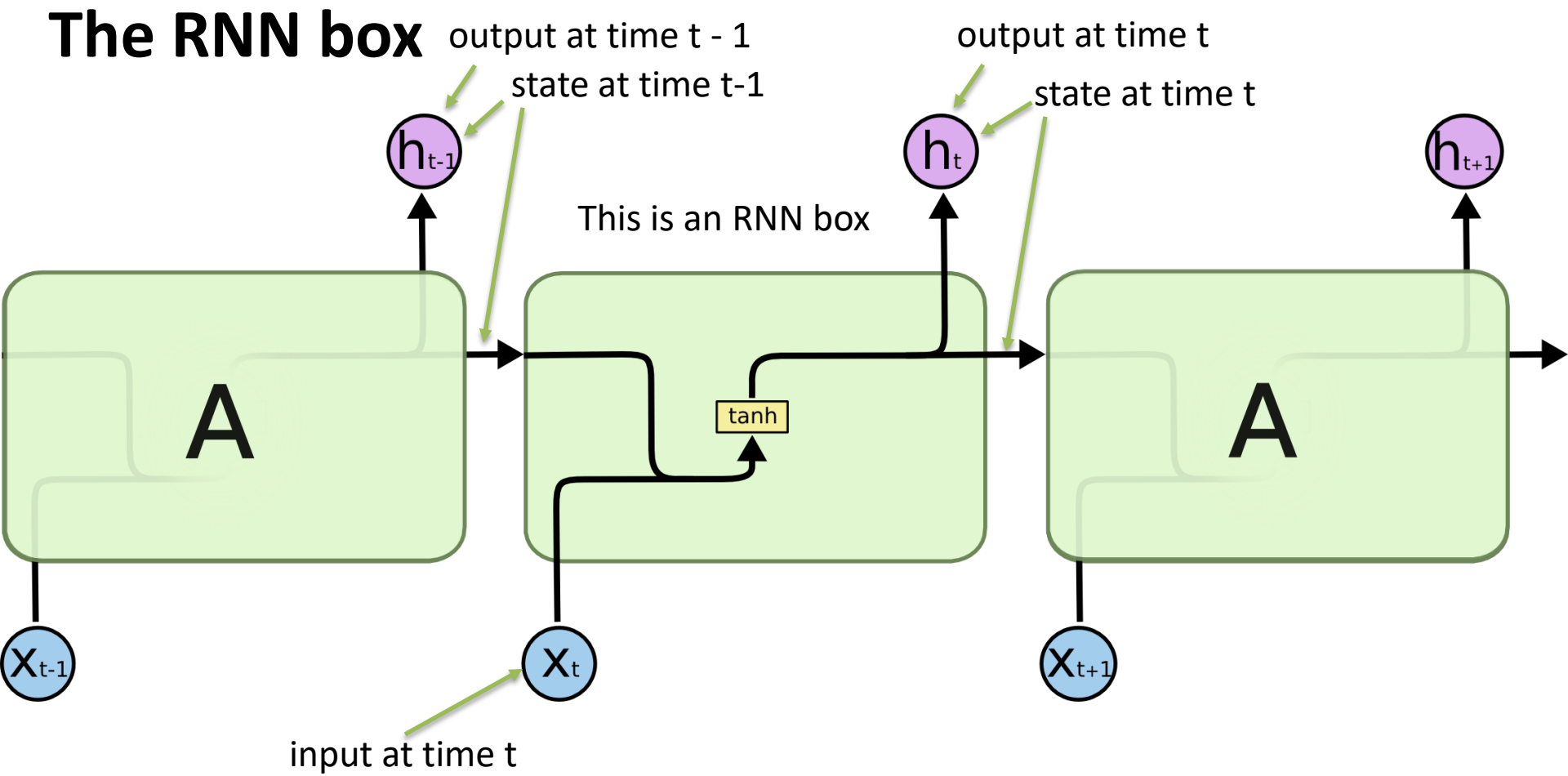
… and returns …

a) an output
b) the new state

… at each time-step.



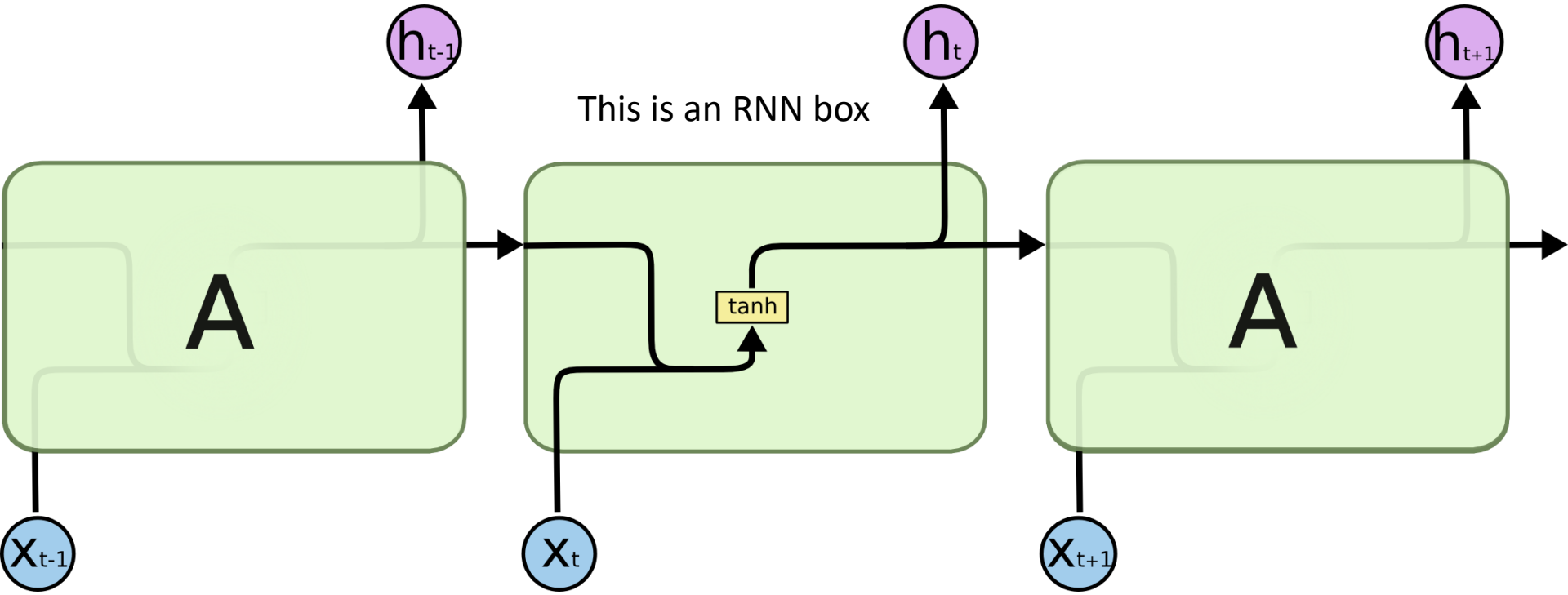
Sequential Deep Learning Models



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential Deep Learning Models

The RNN box

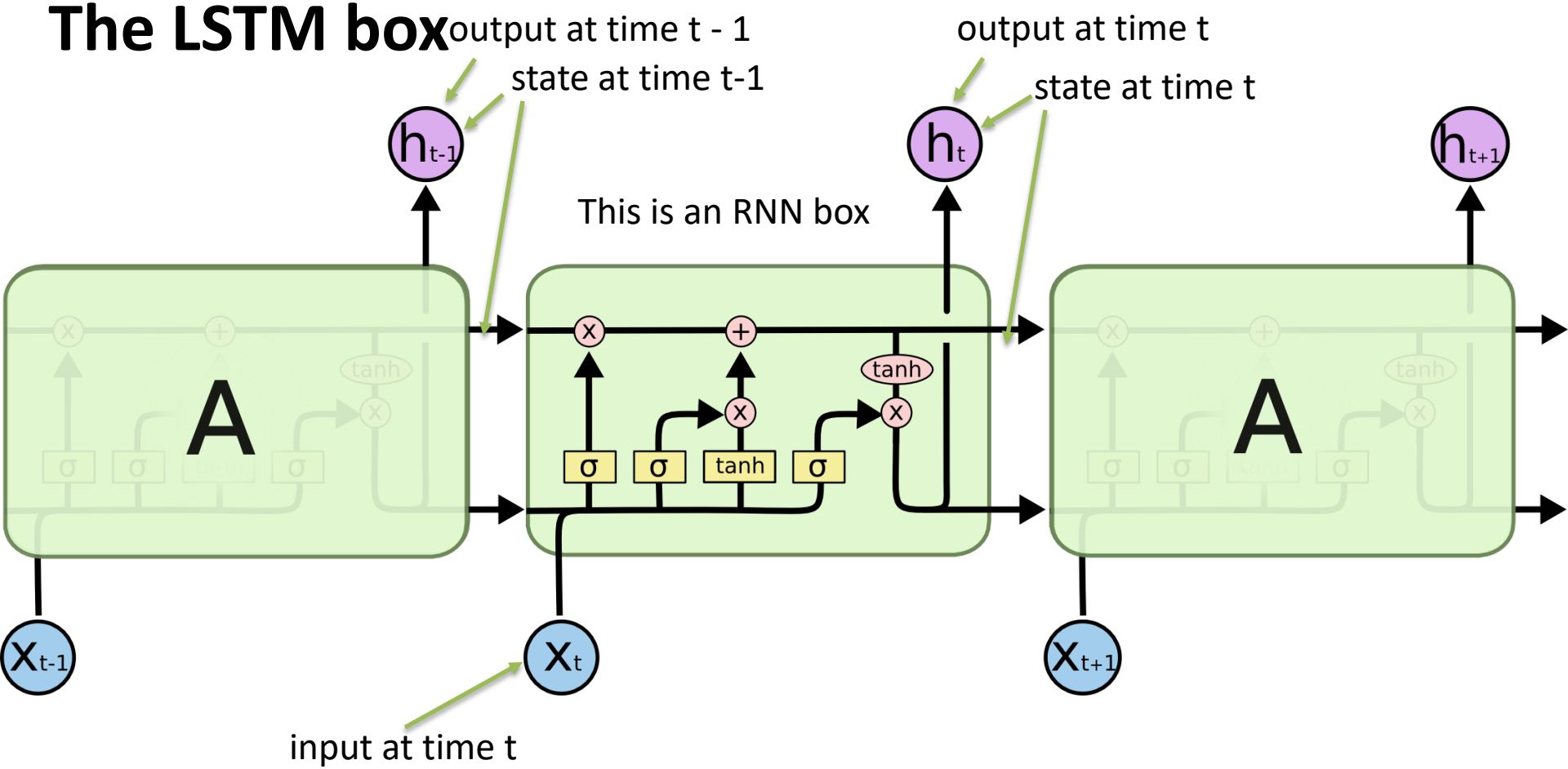


The code for this in pytorch is:

```
x = torch.cat((state, features_at_current_step), 1)
state = output = F.tanh(x.mm(W) + b)
```

Sequential Deep Learning Models

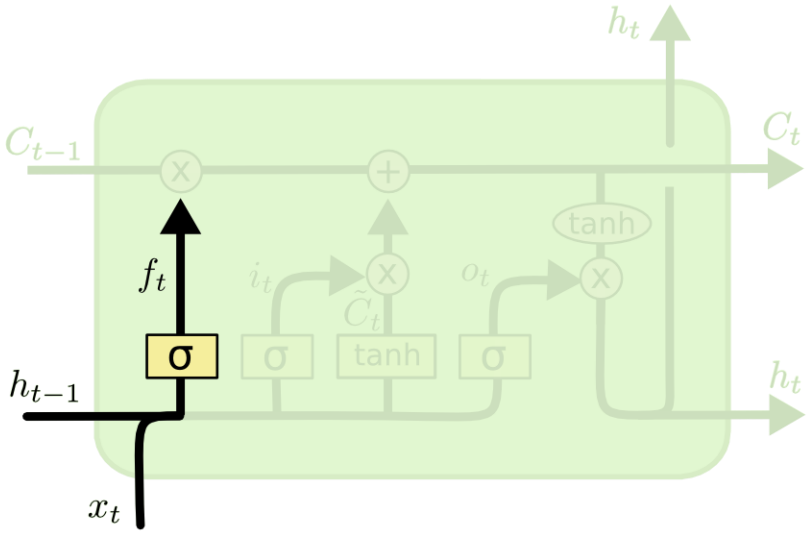
The LSTM box



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential Deep Learning Models

How the LSTM works (step-by-step walk-through)

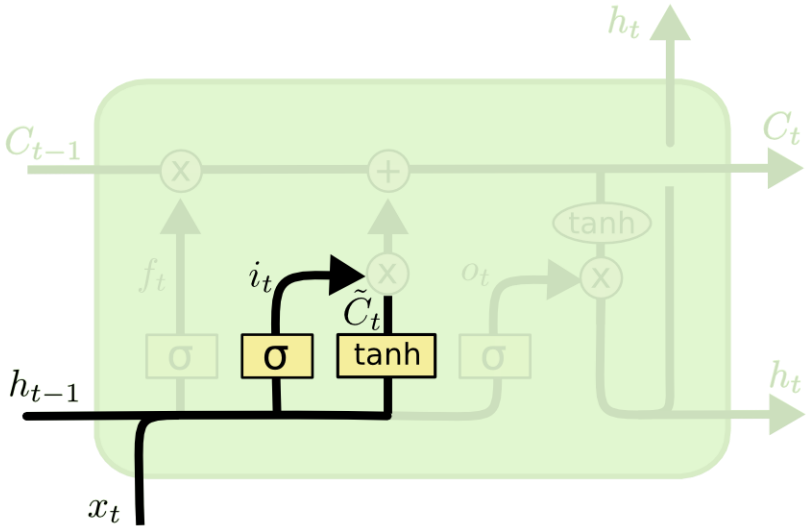


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Just copied and pasted from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential Deep Learning Models

How the LSTM works (step-by-step walk-through)



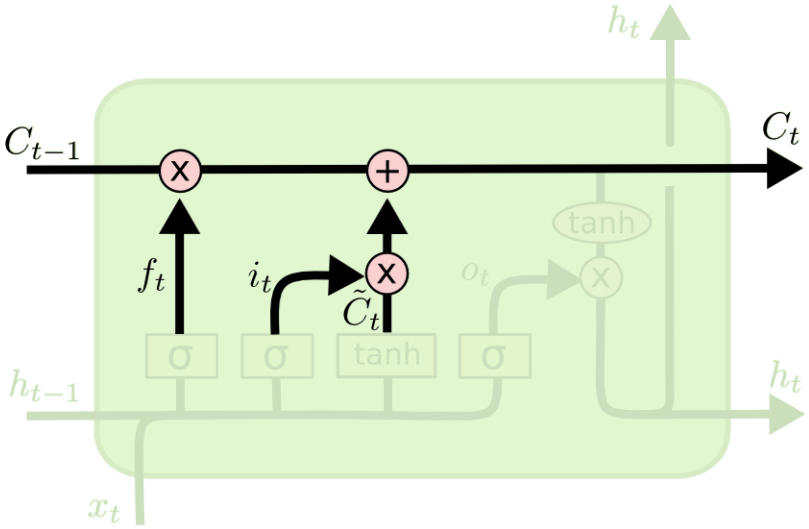
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Just copied and pasted from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential Deep Learning Models

How the LSTM works (step-by-step walk-through)

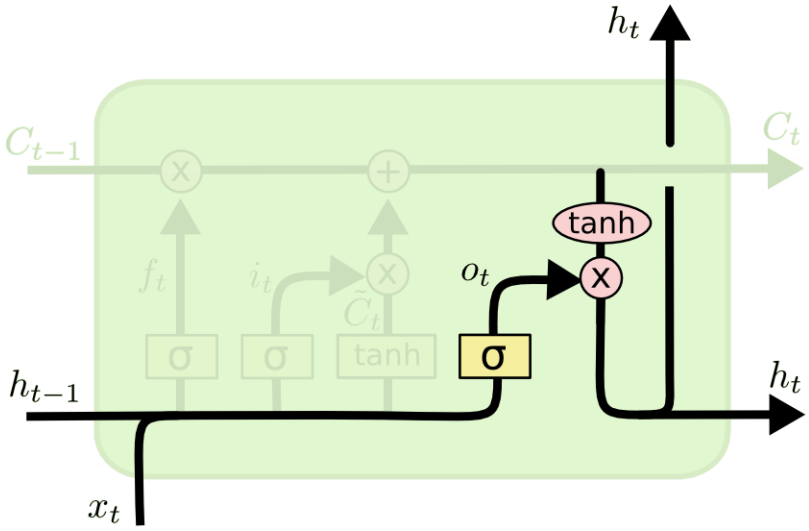


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Just copied and pasted from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential Deep Learning Models

How the LSTM works (step-by-step walk-through)



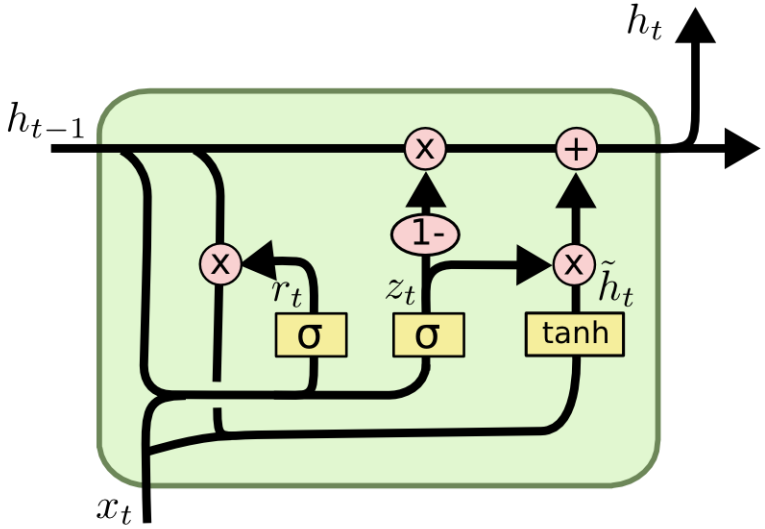
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Just copied and pasted from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential Deep Learning Models

GRUs



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Just copied and pasted from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sequential to Sequence Models

Sequence to Sequence Models:

These can understand and generate sequences.

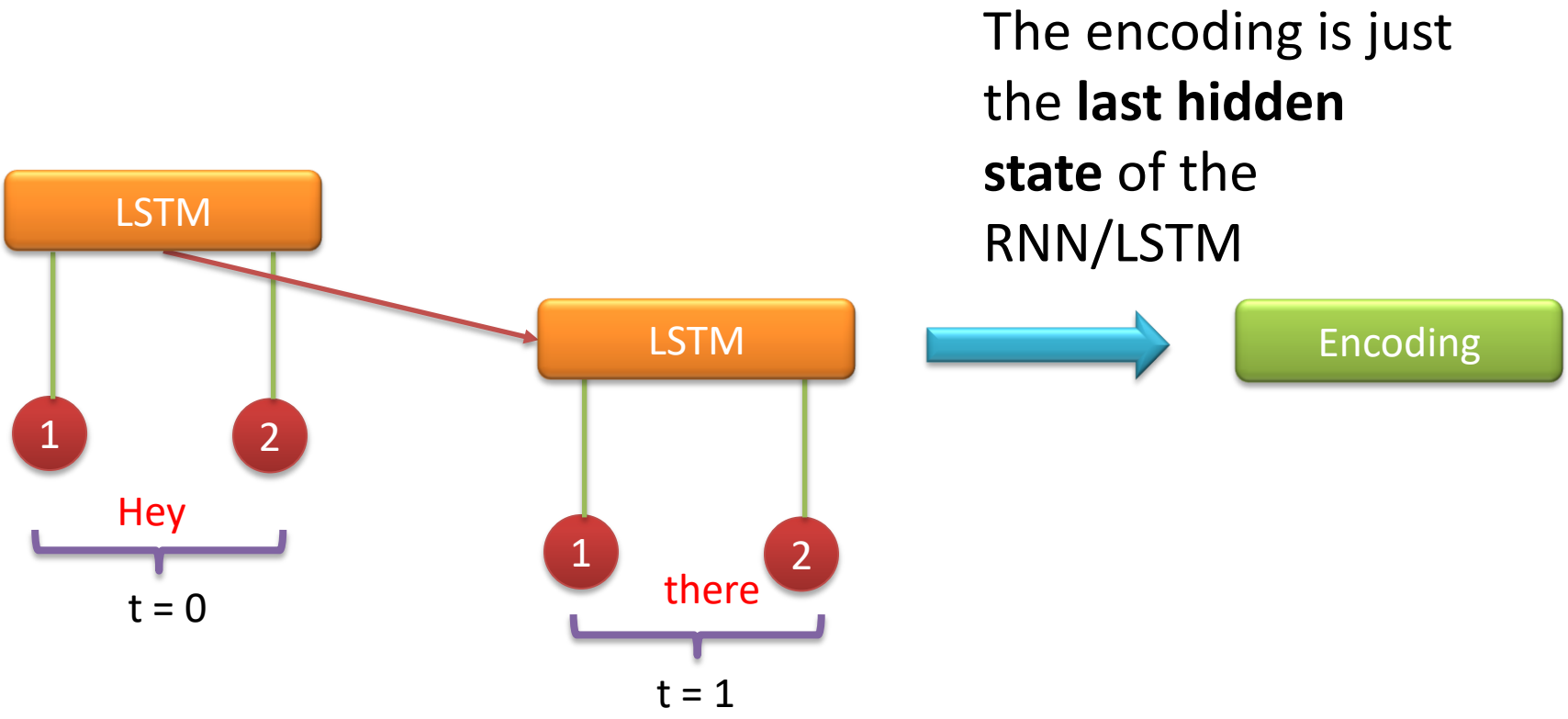
Parts:

1) Encoder - the part that encodes sequences
2) Decoder - the part that generates sequences

Sequential to Sequence Models

The Encoder:

Just an RNN (maybe LSTM) without the output.

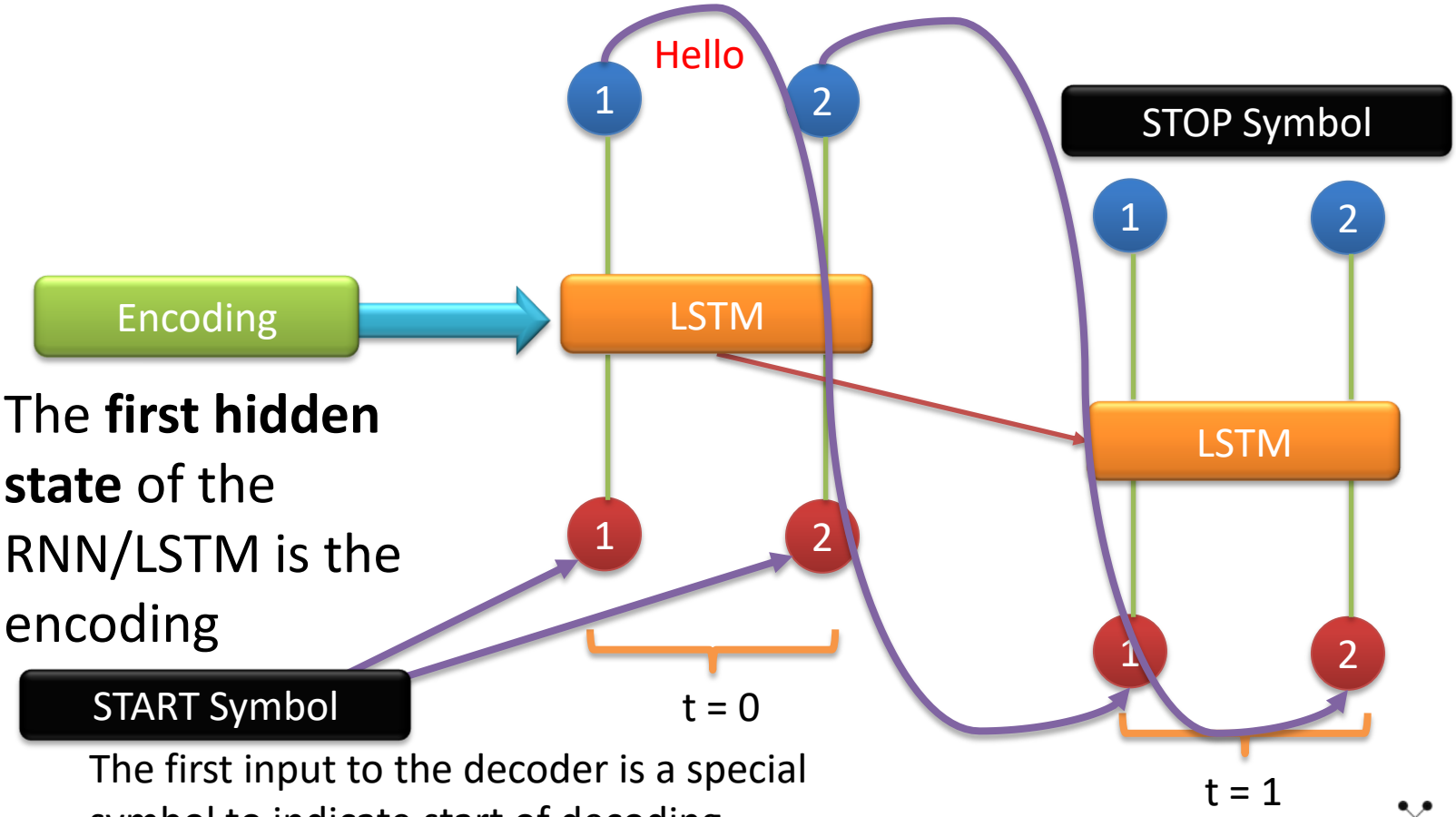


The encoding is just the **last hidden state** of the RNN/LSTM

Sequential Deep Learning Models

The Decoder:

Just another RNN (maybe LSTM) with output.



The **first hidden state** of the RNN/LSTM is the encoding

The first input to the decoder is a special symbol to indicate start of decoding

Concepts for Further Reading

General:

1. LSTMs and GRUs (related to RNNs)
2. Dropout
3. Vanishing Gradient Problem
4. Deep Neural Networks as Universal Approximators

Important for NLP:

1. Sequence to Sequence Models
2. Attention Models
3. Embeddings (Word2Vec, CBOW, Glove)

Important for CV:

1. Convolutional Neural Networks (CNNs)
2. Pooling Layers
3. Attention Models
4. Multidimensional RNNs

Links for Further Learning

Hugo Larochelle's course:

https://www.youtube.com/watch?v=SGZ6BttHMPw&list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH

Tutorials:

https://jasdeep06.github.io/posts/towards-backpropagation/

http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/

https://github.com/jcjohnson/pytorch-examples

http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/

https://distill.pub/2016/augmented-rnns/

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

https://theneuralperspective.com/2016/11/20/recurrent-neural-networks-rnn-part-3-encoder-decoder/

https://theneuralperspective.com/2016/11/20/recurrent-neural-network-rnn-part-4-attentional-interfaces/

Siraj Rawal's videos:

https://www.youtube.com/watch?v=h3l4qz76JhQ&list=PL2-dafEMk2A5BoX3KyKu6ti5_Pytp91sk

https://www.youtube.com/watch?v=cdLUzrjnlr4

Advanced Concepts for Further Reading

1. Zero-Shot, One-Shot and Few-Shot Learning
2. Meta Learning in Neural Networks
3. Transfer learning
4. Neural Turning Machines
5. Key-Value Memory
6. Pointer Networks
7. Highway Networks
8. Memory Networks
9. Attention Models
10. Generative Adversarial Networks
11. Relation Networks (for reasoning) and FiLM
12. BIDAF model for question answering

Links for Further Learning

Papers

Awesome Deep Learning Papers – someone's made a iist! - https://github.com/terryum/awesome-deep-learning-papers

Backpropagation: http://www.nature.com/nature/journal/v323/n6088/abs/323533a0.html
LeCun on backpropagation: http://yann.lecun.com/exdb/publis/index.html#lecun-88
Efficient Backprop: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
ReLU: http://www.nature.com/nature/journal/v405/n6789/full/405947a0.html

LSTM: ftp://ftp.idsia.ch/pub/juergen/lstm.pdf
GRU: https://arxiv.org/abs/1412.3555
LSTM a search space odyssey: https://arxiv.org/pdf/1503.04069.pdf

Schmidhuber Deep Learning Overview: http://www.idsia.ch/~juergen/deep-learning-overview.html
Handwriting recognition: https://www.researchgate.net/publication/24213728_A_Novel_Connectionist_System_for_Unconstrained_Handwriting_Recognition
Generating sequences with RNNs: https://arxiv.org/pdf/1308.0850v5.pdf
Speech Recognition with RNNs: https://arxiv.org/pdf/1303.5778.pdf
LeCun's papers: http://yann.lecun.com/exdb/publis/index.html#selected

Natural Language Processing
Sequence to Sequence: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
Neural Machine Translation: https://arxiv.org/pdf/1409.0473.pdf
BIDAF: https://arxiv.org/abs/1611.01603
Pointer Networks: https://arxiv.org/abs/1506.03134
Neural Turing Machine: https://arxiv.org/abs/1410.5401
Dynamic Coattention Networks for Question Answering: https://arxiv.org/pdf/1611.01604.pdf
Machine Comprehension: https://arxiv.org/abs/1608.07905v2
GNMT: https://research.googleblog.com/2016/09/a-neural-network-for-machine.html
Stanford Glove: https://nlp.stanford.edu/pubs/glove.pdf

Computer Vision
Digit Recognition: https://arxiv.org/pdf/1003.0358.pdf
OverFeat: https://arxiv.org/abs/1312.6229
Feature hierarchies: https://arxiv.org/abs/1311.2524
Spatial Pyramid Pooling: https://arxiv.org/abs/1406.4729
Performance on ImageNet 2012: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks
Surpassing Human-Level Performance on ImageNet: https://arxiv.org/abs/1502.01852
Surpassing Human-Level Face Recognition Performance: https://arxiv.org/abs/1404.3840

Reinforcement Learning
Deep Q-Networks: https://deepmind.com/research/dqn/
Hybrid Reward Architecture https://blogs.microsoft.com/ai/2017/06/14/divide-conquer-microsoft-researchers-used-ai-master-ms-pac-man/
AlphaGo Paper: https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf
AlphaGo Zero Paper: https://deepmind.com/research/alphago/

Speech:
https://www.techrepublic.com/article/why-ibms-speech-recognition-breakthrough-matters-for-ai-and-iot/
https://www.technologyreview.com/s/602714/first-computer-to-match-humans-in-conversational-speech-recognition/

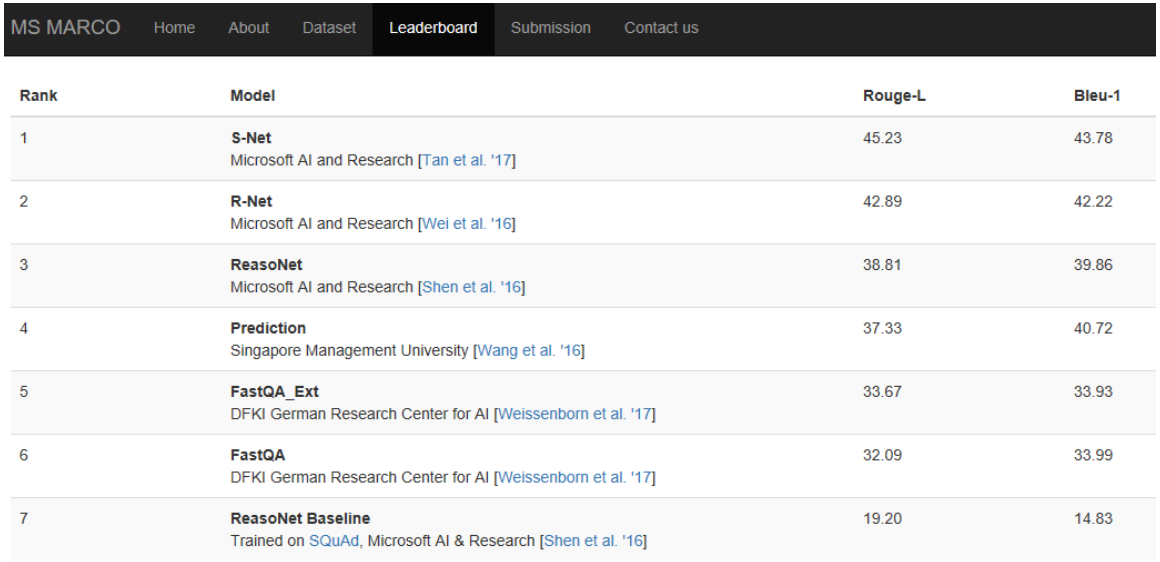
Links to competitions you can participate in

Squad

https://rajpurkar.github.io/SQuAD-explorer/

MSMarco

http://www.msmarco.org/leaders.aspx



MS MARCO Home About Dataset Leaderboard Submission Contact us

Rank	Model	Rouge-L	Bleu-1
1	**S-Net** Microsoft AI and Research [Tan et al. '17]	45.23	43.78
2	**R-Net** Microsoft AI and Research [Wei et al. '16]	42.89	42.22
3	**ReasoNet** Microsoft AI and Research [Shen et al. '16]	38.81	39.86
4	**Prediction** Singapore Management University [Wang et al. '16]	37.33	40.72
5	**FastQA_Ext** DFKI German Research Center for AI [Weissenborn et al. '17]	33.67	33.93
6	**FastQA** DFKI German Research Center for AI [Weissenborn et al. '17]	32.09	33.99
7	**ReasoNet Baseline** Trained on SQuAd, Microsoft AI & Research [Shen et al. '16]	19.20	14.83

ConvAI

http://convai.io/1_round/

History of Deep Learning

- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is "good enough", why deep?
- 2006: RBM initialization
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition
- 2015.2: Image recognition surpassing human-level performance
- 2016.3: Alpha GO beats Lee Sedol
- 2016.10: Speech recognition system as good as humans

Source: http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/DL.pdf

Recent History of Deep Learning

2017.07: Relation Networks beats humans at relational reasoning on a toy dataset.
2017.10: AlphaGo Zero teaches itself Go and beats AlphaGo (which beat Lee Sidol).
2017.12: AlphaZero teaches itself chess and Go and beats Stockfish 8 and AlphaGo Zero.
2017.12: Text to speech system that sounds convincingly human (Tacotron 2).

Source: http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/DL.pdf

THE END

Deep Learning Basics

Cohan Sujay Carlos
Aiaioo Labs
Benson Town
Bangalore
India