Trend de la Trend

Benefits and Pitfalls for Deep Learning with Functional Programming

HELLO!

I am Brad Johns

I should write something here but I’m lazy.

Here’s my github: https://github.com/bradjohns94

A Quick Intro to Deep Learning

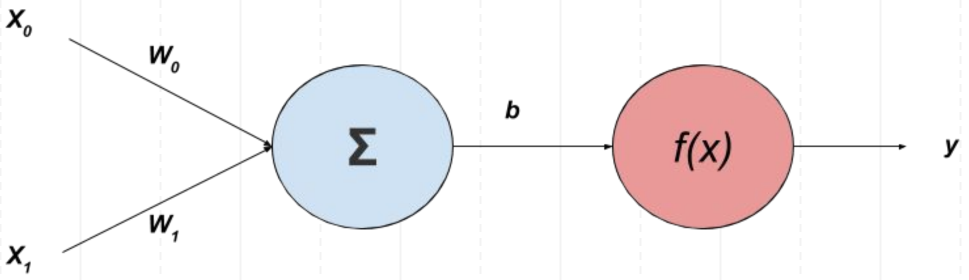
I’ll try to make this quick and painless

Perceptrons

4 Variables

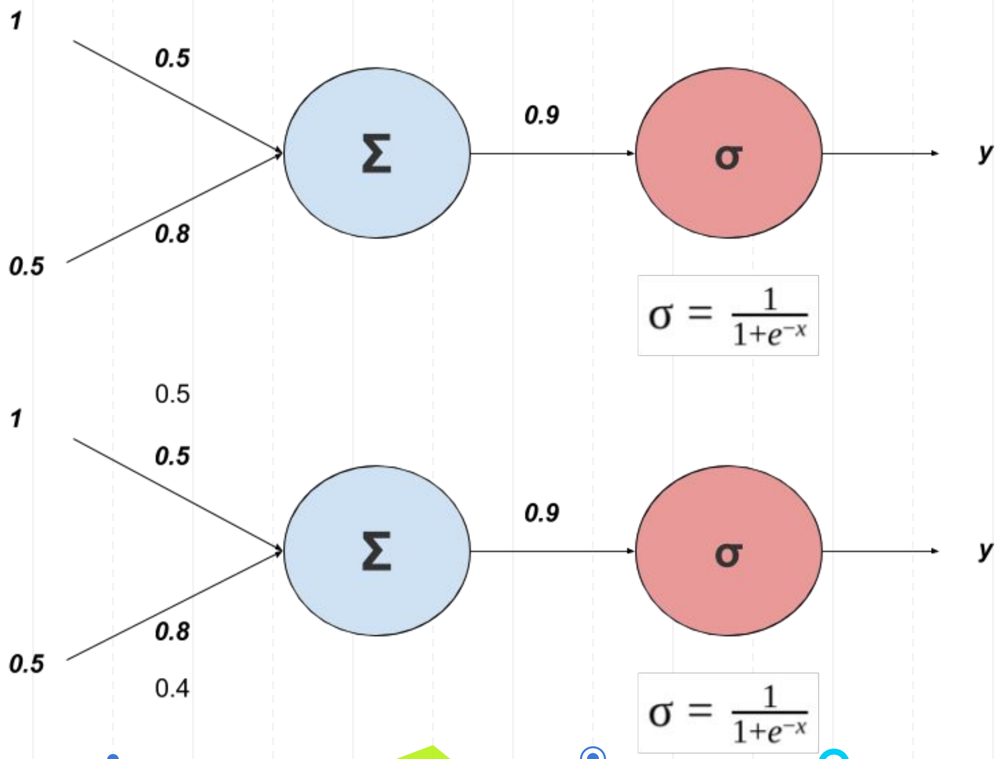
- Inputs
- Weights
- Bias
- Activation Function

Multiply the inputs by weights, add a bias, run it through the function. Easy.

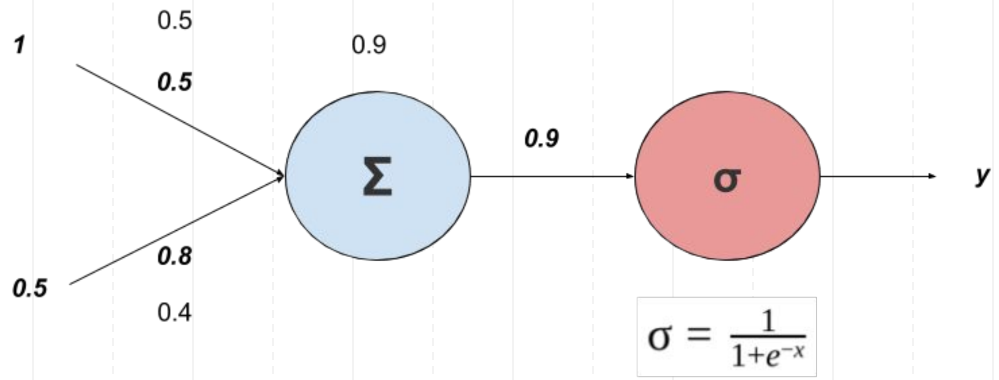


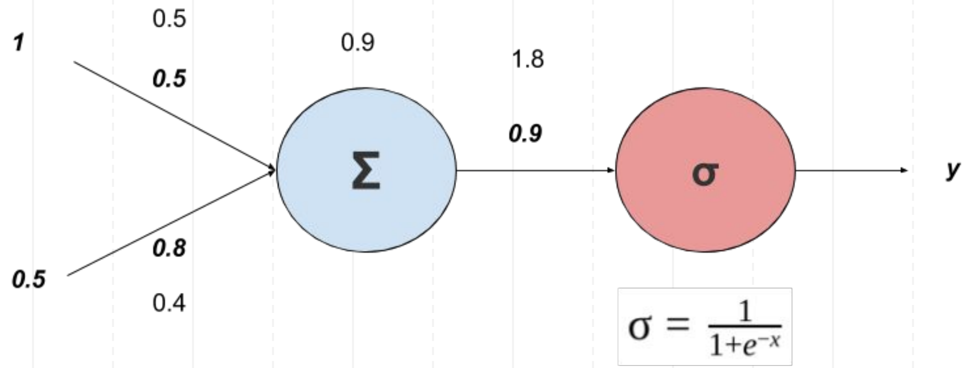
$$y = f\left(\sum_{i=0}^{||x||} (w_i \times x_i) + b\right)$$

Perceptrons - Example

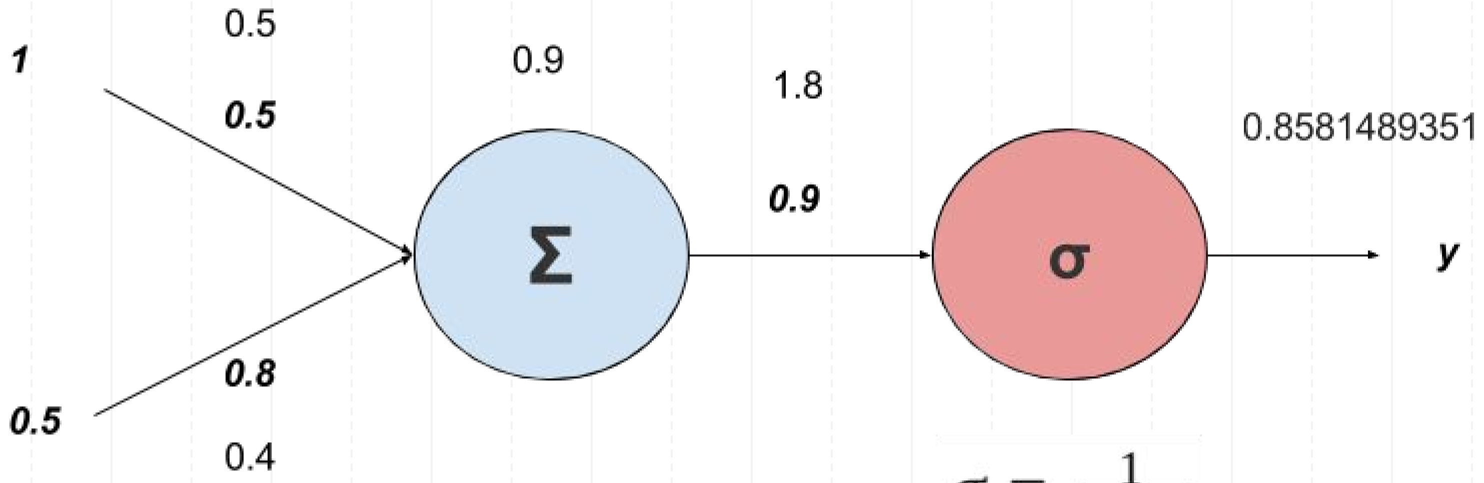


Perceptrons - Example





Perceptrons - Example



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

Neural Networks

Step 1:

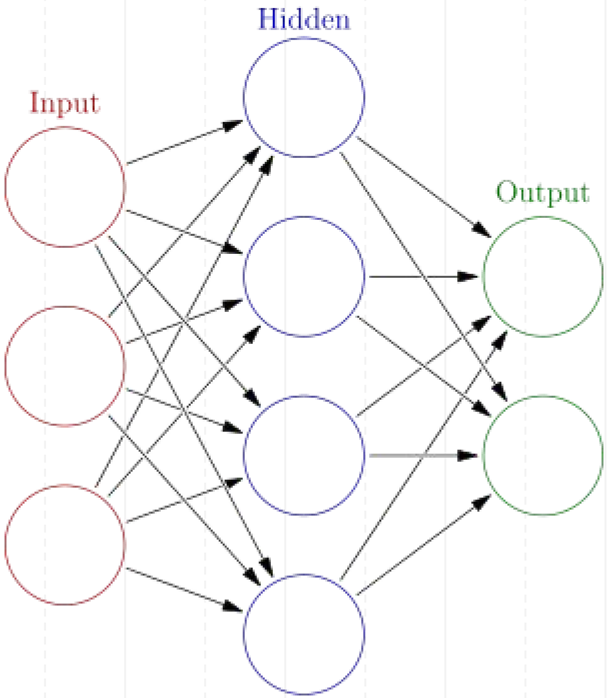
Take Some Perceptrons

Step 2:

Glue them together in layers

Step 3:

Profit



The “Learning” Part

I’m sorry in advance

Backpropagation and Gradient Descent

1. Calculate the “Cost” (C) of the network by comparing actual vs. expected outputs
2. Determine the impact or “error” (δ^L) of each output neuron/perceptron on the cost
3. Find out how much each hidden layer contributes to the next layers error (δ^l)
4. Calculate how much each weight/bias contributes to the error of the associated neuron/perceptron (∂w and ∂b)
5. Adjust weights and biases according to their effect on the error

Wow. Code. (Scala)

```scala
/* Calculate the pre-activation output value Z */
private def calcZ(inputs: Matrix): Matrix = {
  if (inputs.getShape != (1, inputSize))
    throw new IllegalArgumentException(s"Input matrix must be of shape (1x${inputSize}) (got ${inputs.getShape})")
  (inputs * weights) + biases
}

def run(inputs: Matrix): Matrix = {
  activate(calcZ(inputs))
}

/* Reweight the weights and biases of the layer and return the weighted error
 * for the previous layer */
def fit(inputs: Matrix, weightedError: Matrix): Matrix = {
  /* *:* -> component multiplication */
  val error = weightedError *:* activationFn.derivative(calcZ(inputs))
  val res = error * this.weights.transpose
  this.biases -= error * learningRate
  this.weights -= (inputs.transpose * error) * learningRate
  res
}
```

Wow. Math.

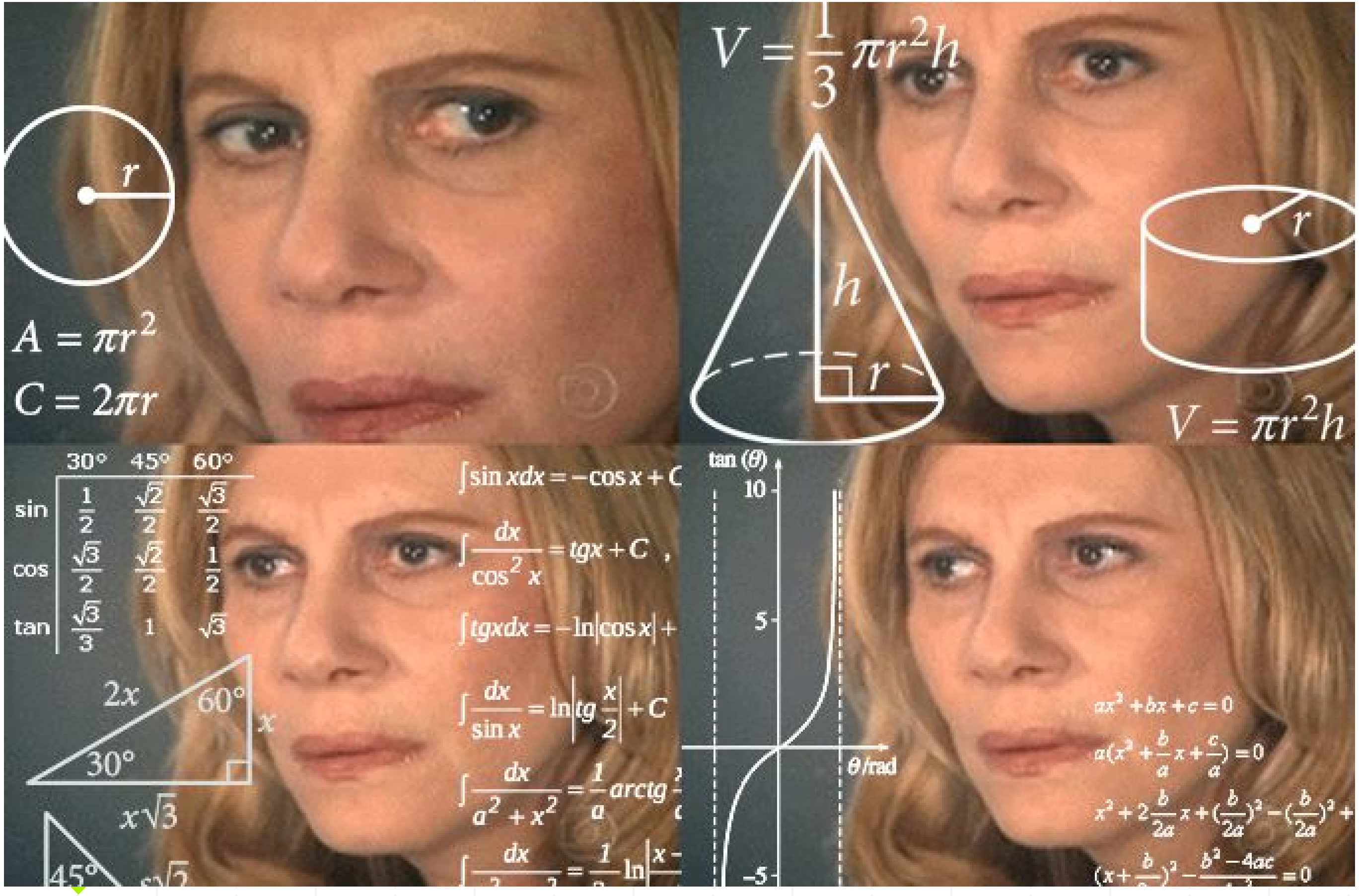
Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$


r
$A = \pi r^2$
$C = 2\pi r$
$V = \frac{1}{3}\pi r^2 h$
h
r
r
$V = \pi r^2 h$
30° 45° 60°
sin $\frac{1}{2}$ $\frac{\sqrt{2}}{2}$ $\frac{\sqrt{3}}{2}$
cos $\frac{\sqrt{3}}{2}$ $\frac{\sqrt{2}}{2}$ $\frac{1}{2}$
tan $\frac{\sqrt{3}}{3}$ 1 $\sqrt{3}$
$2x$
60°
x
30°
$x\sqrt{3}$
45°
$\int \frac{dx}{\cos^2 x} = tgx + C$
$\int \frac{dx}{\sin x} = \ln\left|tg\frac{x}{2}\right| + C$
tan (θ)
10
5
-5
θ/rad
$ax^2 + bx + c = 0$
$a(x^2 + \frac{b}{a}x + \frac{c}{a}) = 0$

The Important Bits

We’ll get to the functional part soon - I promise

The Vanishing/Exploding Gradient Problem

- Errors for earlier layers are dependent on errors from later layers
- Errors tend to increase or decrease exponentially as you backpropagate
- When your errors are too big or too small you learn effectively nothing
- Deep Learning: a pile of hacks to fix this problem

Example Hack 1: Convolution

- Primarily used for image processing/other 2D data sets
- Don't fully connect your way to the next layer
- Connect small squares of neurons to the next layer
- Pool outputs to estimate relative position

Example Hack 2: LSTMs

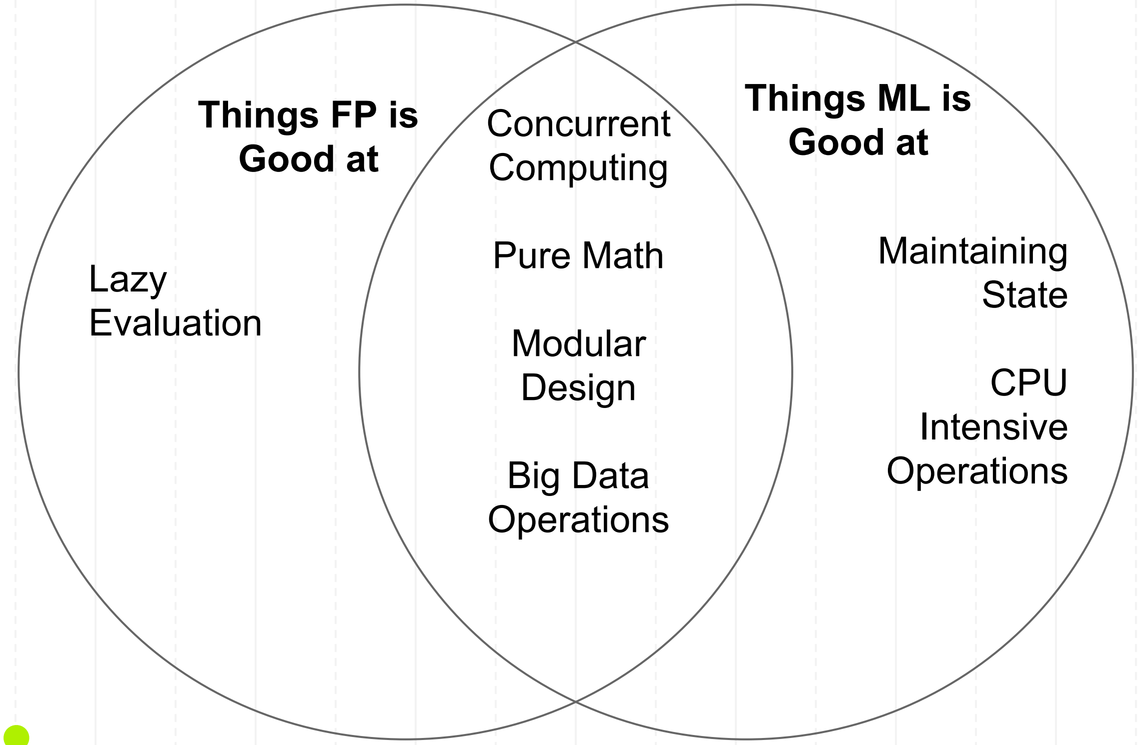
- Primarily used for NLP/other sequential data
- Feed the network back into itself recurrently
- Use special “forget” neurons to determine what data is important to remember
- Mostly magic

Getting on With it

Functional Programming and Deep Learning

So What Was the Point of All That?

- Understanding the structure of Neural Nets and the Modularity of Deep Learning modules lets us see how functional programming plays well with it
- Understanding the structure of Neural Nets lets us look at how we can develop them functionally
- Knowing different implementations lets us explore what various technologies are good at
- You have some idea of how deep learning works now, so that’s cool I guess, right?


Things FP is Good at
Lazy Evaluation
Concurrent Computing
Pure Math
Modular Design
Big Data Operations
Things ML is Good at
Maintaining State
CPU Intensive Operations

So How Easy *is* Learning With FP?

```scala
class NeuralNetwork(numInputs: Int, layers: List[Layer], costFn: Cost) {
  /* Verify the size of each layer */
  layers.foldLeft(numInputs) { (lastOutputs, layer) =>
    val (inputSize, layerSize) = layer.getShape
    if (lastOutputs != inputSize)
      throw new IllegalArgumentException(s"Cannot connect layer with ${lastOutputs} outputs to layer with ${inputSize} inputs'
    layerSize
  }

  private def runStack(inputs: Matrix): (Stack[Matrix], Matrix) = {
    layers.foldLeft( (new Stack[Matrix](), inputs) ) { case ((inputStack, layerInputs), layer) =>
      inputStack.push(layerInputs)
      (inputStack, layer.run(layerInputs))
    }
  }

  /* Feed a value into the network and get the result */
  def run(inputs: Matrix): Matrix = runStack(inputs)._2

  /* Train the network based on the expected output and return what it output */
  def fit(inputs: Matrix, expected: Matrix): Matrix = {
    val (inputStack, outputs) = runStack(inputs)
    layers.foldRight(costFn.derivative(expected, outputs)) { (layer, weightedError) =>
      layer.fit(inputStack.pop, weightedError)
    }
    outputs
  }
}
```

And We're Not Even *Deep* Yet!

(Phrasing)

- The amazing thing about deep learning - every kind of layer is a new module
- Tiny tweaks to existing layers create new possibilities with the same formulas
- Deep learning modules like convolutional layers and LSTM modules rely on the same perceptron logic we've already designed
- Just add tweaks to activation functions and throw them together!

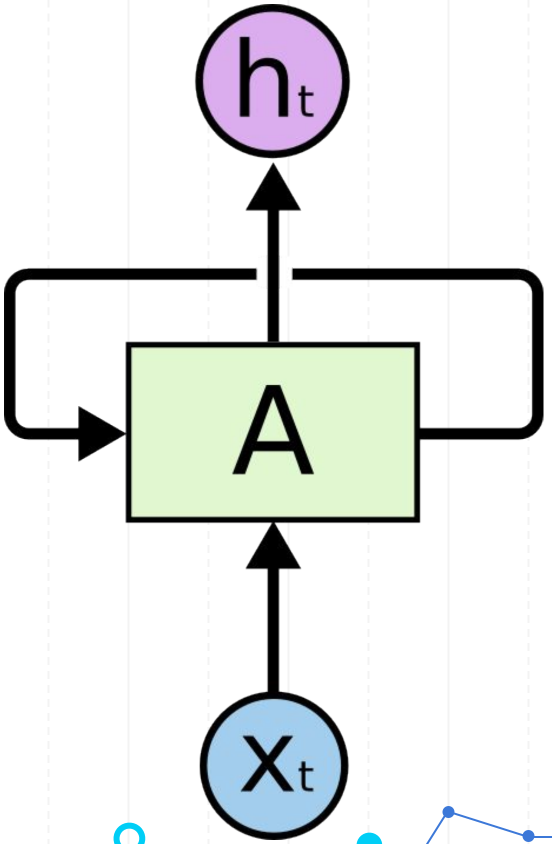
Case Study: Recurrent Neural Networks

Because saying “Hey look, this is functional!” is boring

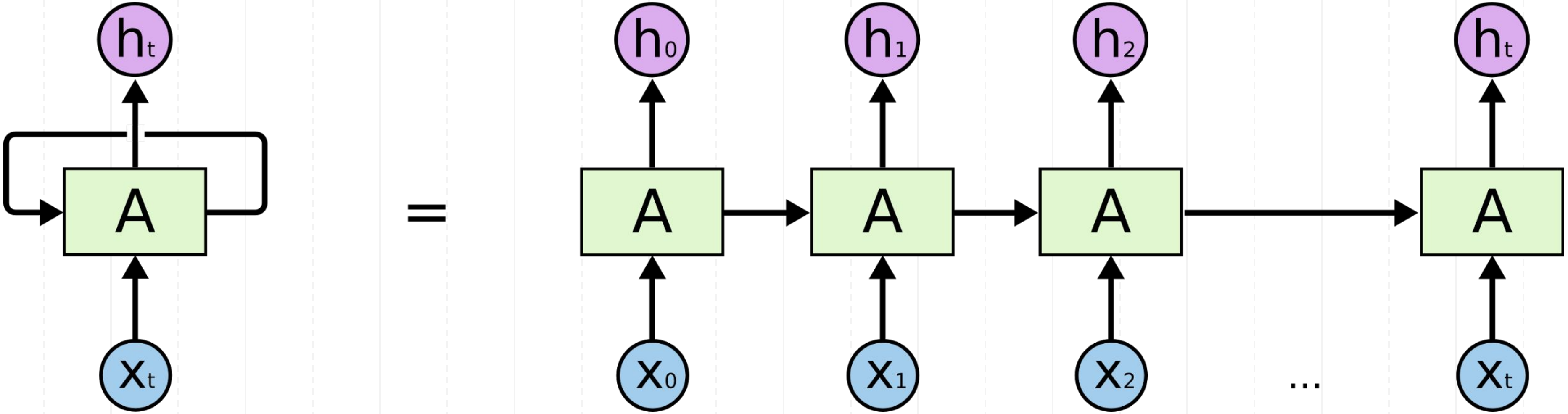
Yet Another Introduction

Take a Neural Network, add a twist (literally)

- Same input/hidden/output layer structure as before
- Hidden layers have an additional set of inputs: their last outputs
- RNNs can keep track of state unlike traditional NN’s, but get deep *fast*
- Let’s make this a little less scary...



Recurrent NNs: Now With Less Magic!



Encoding: Something Old, Something New

Live Demo of Encodes with RNNs

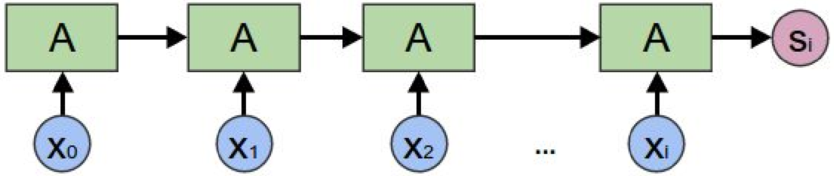
Encoding: Take a variable length input, encode it to one value

Operationally:

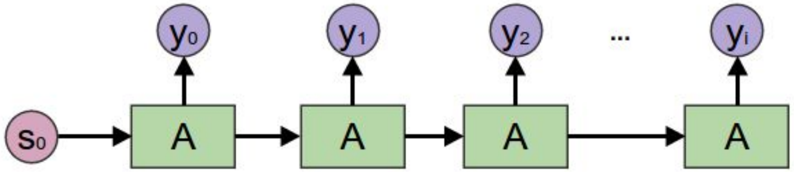
- Input a value
- Pass hidden layer output to next iteration
- Take the last output the hidden layer

So… you know… a fold

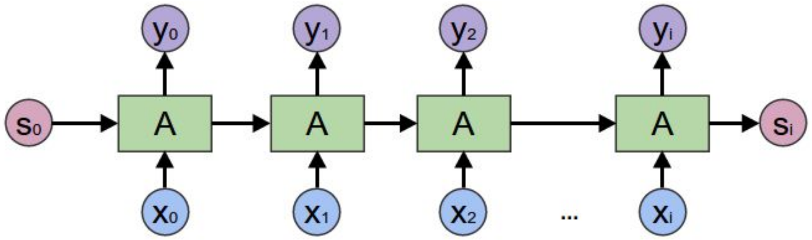
And it Keeps Going!



Encoding: Equivalent to Haskell foldl, Scala foldLeft



Generation: Equivalent to Haskell unfoldr, Scala unfoldRight



Standard RNN: Equivalent to Haskell mapAccumR, Scala no easy parallel

Finishing Up

Some Things to Know and Some Credit to Give

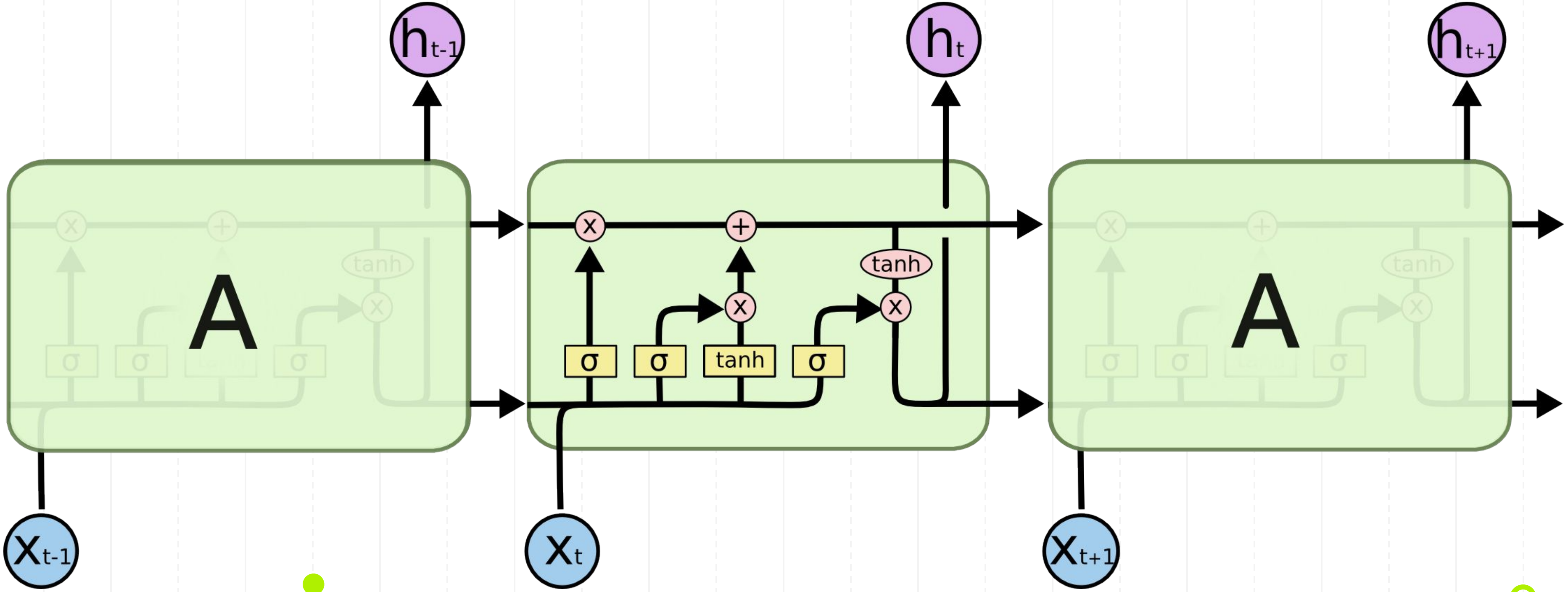
The “Pitfalls” Part

“Purer” functional languages have limited libraries:

- Scala - DeepLearning4Java (DL4J)
  - Not technically scala - but java libraries are scala libraries
  - Scala-specific port in alpha
  - Not great and the support community isn’t much better
  - https://deeplearning4j.org/
- Haskell - Grenade
  - Still in early development (0.1.0)
  - Very young - basically no community
  - https://github.com/HuwCampbell/grenade

Nobody Uses Pure RNNs - Use LSTMs

(Out of nowhere, I know. But really)



Resources

Deep Learning in General:

- http://neuralnetworksanddeeplearning.com/
- Main source for the intro to DL stuff

Deep Learning + Functional Programming and RNNs:

- http://colah.github.io/
- Main source for RNN and FP stuff

These guys are the best, please check them out

THANKS!

Any questions?

Code samples:
https://github.com/bradjohns94/MNIST-From-Scratch

Helpful Images But Mostly Memes

https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/300px-Colored_neural_network.svg.png

http://neuralnetworksanddeeplearning.com/images/tikz21.png

http://media.boingboing.net/wp-content/uploads/2016/11/bcf.png

https://media.tenor.com/images/191e856ae3d5ed9c280ff64c93164f55/tenor.gif

https://media.tenor.com/images/08c127d137e22d56677a6b0deb321887/tenor.gif

http://i.imgur.com/5JPp8NQ.gif

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-rolled.png

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

https://media.giphy.com/media/YIdO82g8zZfDa/giphy.gif

A Little More Because Stealing is Wrong

http://colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-encoding.png

http://colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-generating.png

http://colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-general.png

http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png