XIAYUN SUN | “JOY” | DISNEY STREAMING SERVICES

NEURAL NETWORK FROM SCRATCH

ME

- Distributed systems at day time
- Fantasising about research at night time

Disney streaming services

OUTLINE

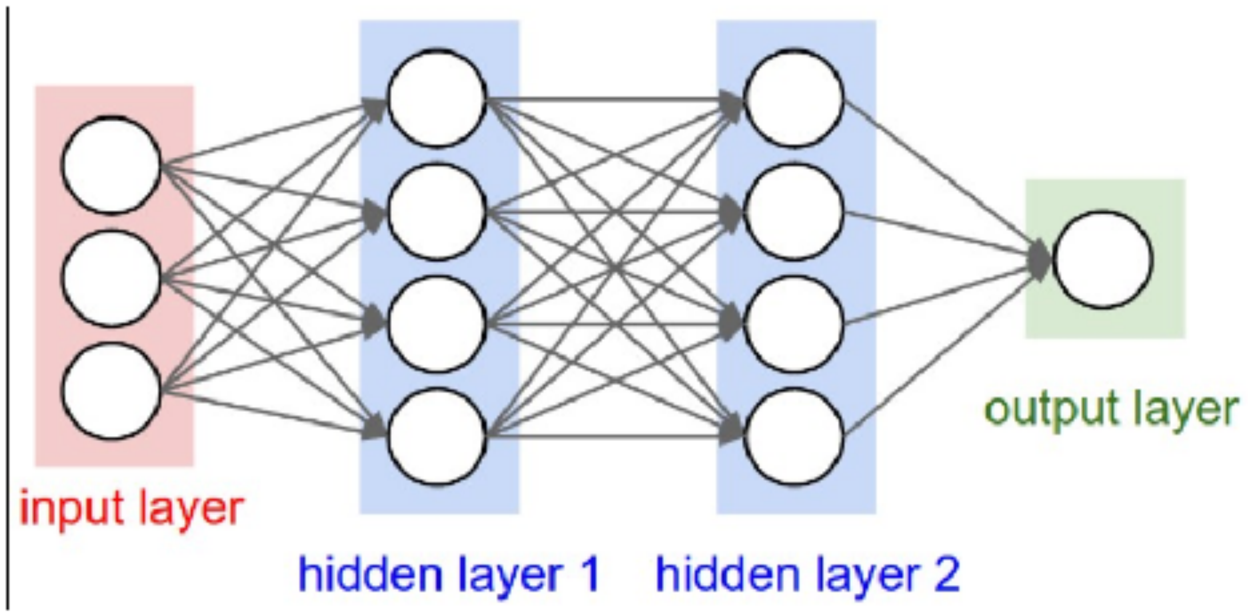
- What is a neural network?
- First neural network
- Wait…is that it?
- Graph computation
- Second neural network
- Third neural network
- But why not just TensorFlow

FIRST…TO PROVE I'M NOT CHEATING

```scala
object core extends ScalaModule with ScalafmtModule{
  def scalaVersion = "2.12.4"

  def scalacOptions = Seq("-Ypartial-unification")

  override def ivyDeps = Agg(
    ivy"co.fs2::fs2-core:1.0.0",
    ivy"co.fs2::fs2-io:1.0.0",
    ivy"org.typelevel::cats-core:1.4.0"
  )
}
```

WHAT IS A NEURAL NETWORK(1)

- A function approximator that can approximate anything (almost)
- List[(Weights, **Activation Function**)]
    - input :: hidden layers :: output

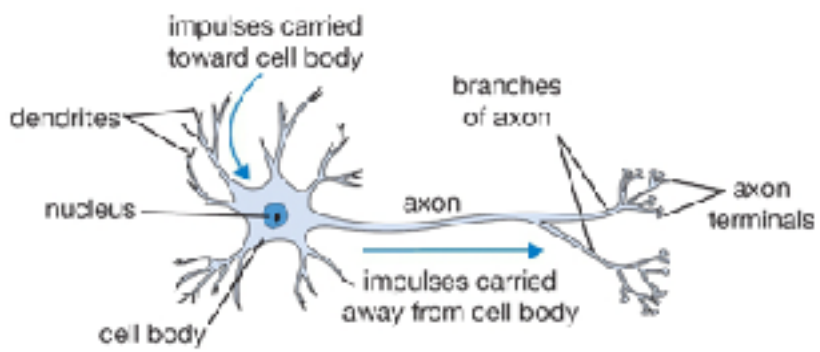


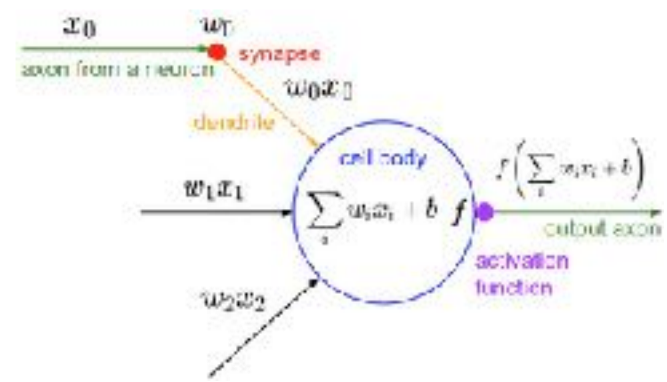
(1) to be precise, a feedforward neural network

http://cs231n.github.io/neural-networks-1/

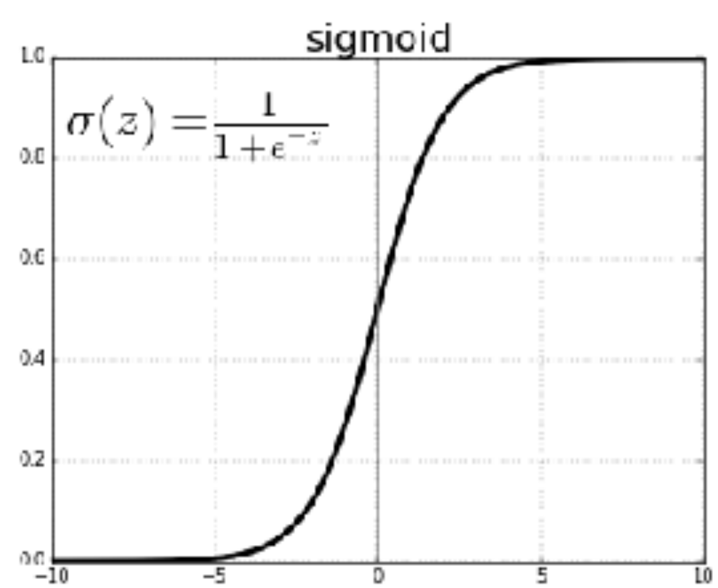
WHAT IS A NEURAL NETWORK — ACTIVATION FUNCTIONS

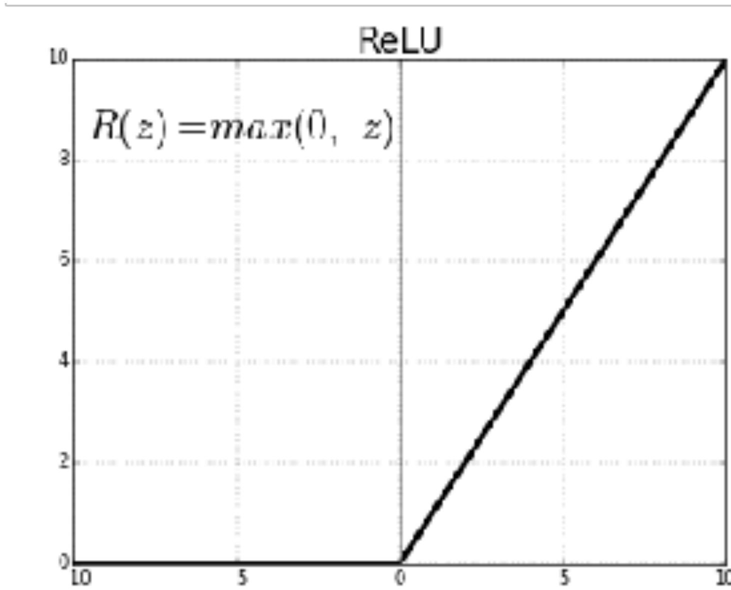
- Activation functions are like neurons, they “activate”





- They bring non-linearity
- Popular ones: Sigmoid, ReLU





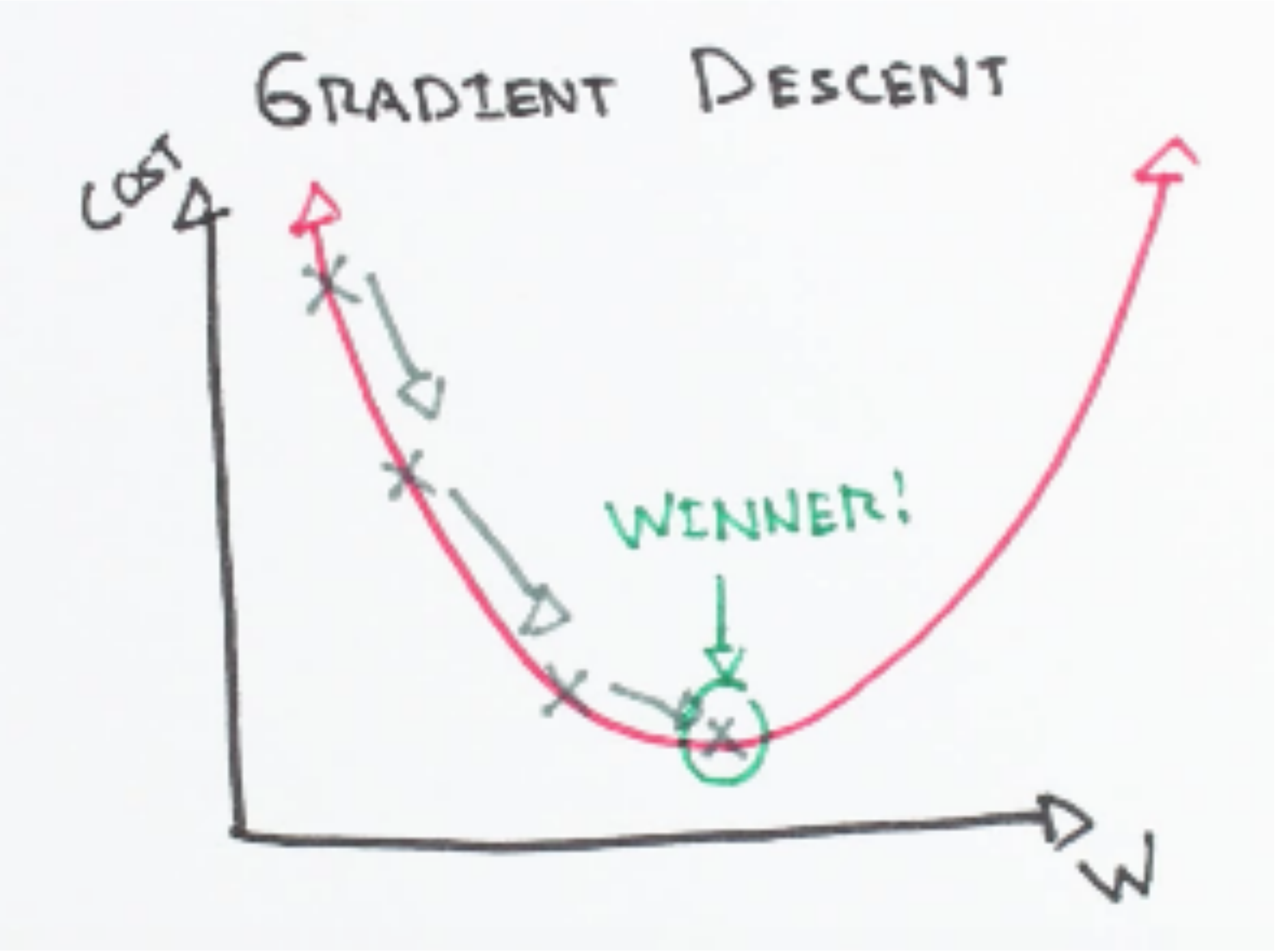
http://cs231n.github.io/neural-networks-1/

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

HOW TO TRAIN A NEURAL NETWORK

- Your model defines a function mapping inputs to output estimates
- Distance between output and target is your loss function
- We train the model by minimising the loss function
- Popular loss functions:
  - Mean Squared Error $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$
  - Cross entropy loss (Maximum Likelihood) $-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}\left[\log p_{\text{model}}(\boldsymbol{x})\right]$
- How to minimise loss function:
  - Gradient descent (next slide)

HOW TO TRAIN A NEURAL NETWORK

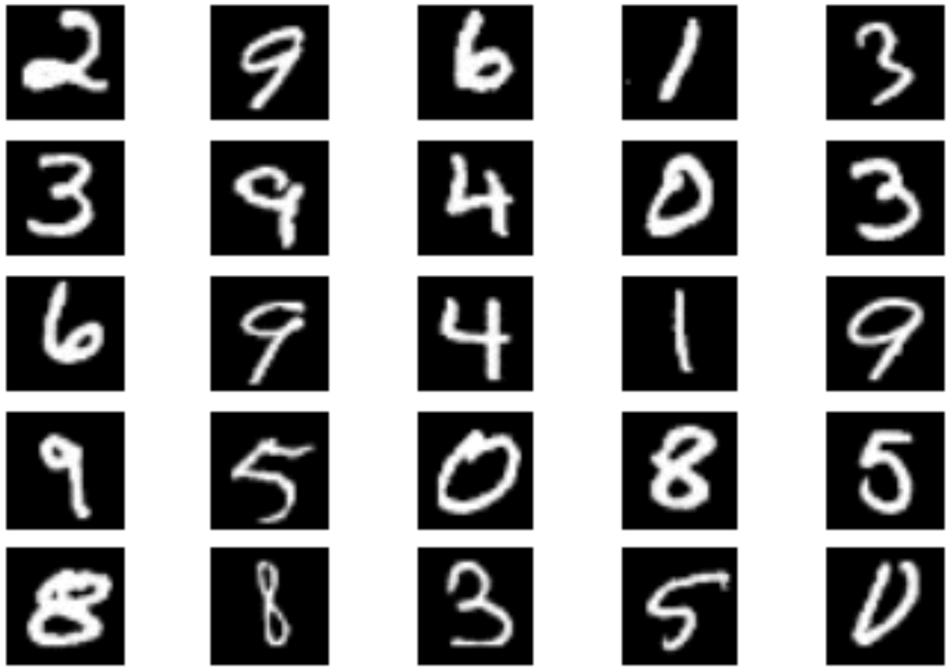


https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

FIRST NEURAL NETWORK ON MNIST

MNIST DATASET

- A benchmark dataset for classifying handwritten digits
- 60k train images, 10k test images –> manageable on a laptop



PREPARING MNIST DATA WITH FS2

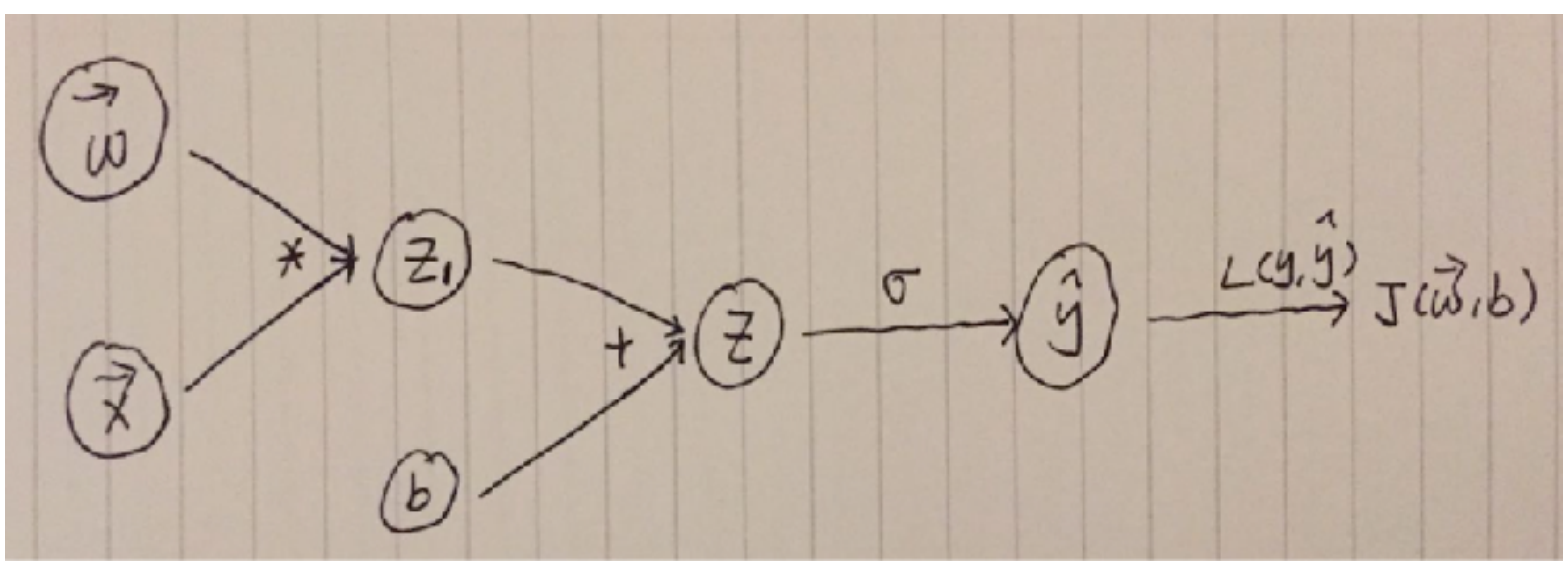
TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel

```scala
val images: fs2.Stream[IO, Matrix] =
  io.file
    .readAll[IO](path = Paths.get(imgFileName), global, chunkSize = 1024)
    .drop(16) // 16 bytes for magic number and meta data
    .map(java.lang.Byte.toUnsignedInt)
    .chunkN(imgDimension * imgDimension, allowFewer = false)
    .map(_.toVector)
    .map(v => Matrix.fromVector(v.map(_.toDouble), imgDimension, imgDimension))

// preprocess: x /= 255. y: binary classifier on digit 0
val imagesPreprocessed: fs2.Stream[IO, Matrix] =
  images.map(matrix => Matrix(matrix.m.map(_.map(_ / 255.0))))

val labelsPreprocessed: fs2.Stream[IO, Int] = labels.map(i => if (i == 0) 1 else 0)
```

FIRST NEURAL NETWORK

- Architecture
  - Weights => linear weights
  - Activation function => sigmoid
  - Loss function => cross entropy



FIRST NEURAL NETWORK — GRADIENT

$$z = \vec{w}^T \cdot \vec{x} + b$$

$$\hat{y} = \sigma(z)$$

$$J = \ln(1 + e^{(1-2y)z}) \quad \text{// trust me}$$

$$\frac{\partial J}{\partial \vec{w}} = \frac{\partial J}{\partial z} \cdot \frac{\partial z}{\partial w} = \frac{\partial J}{\partial z} \cdot \vec{x}$$

$$\frac{\partial J}{\partial z} = \frac{1}{1 + e^{(1-2y)z}} \cdot e^{(1-2y)z} \cdot (1-2y)$$

FIRST NEURAL NETWORK

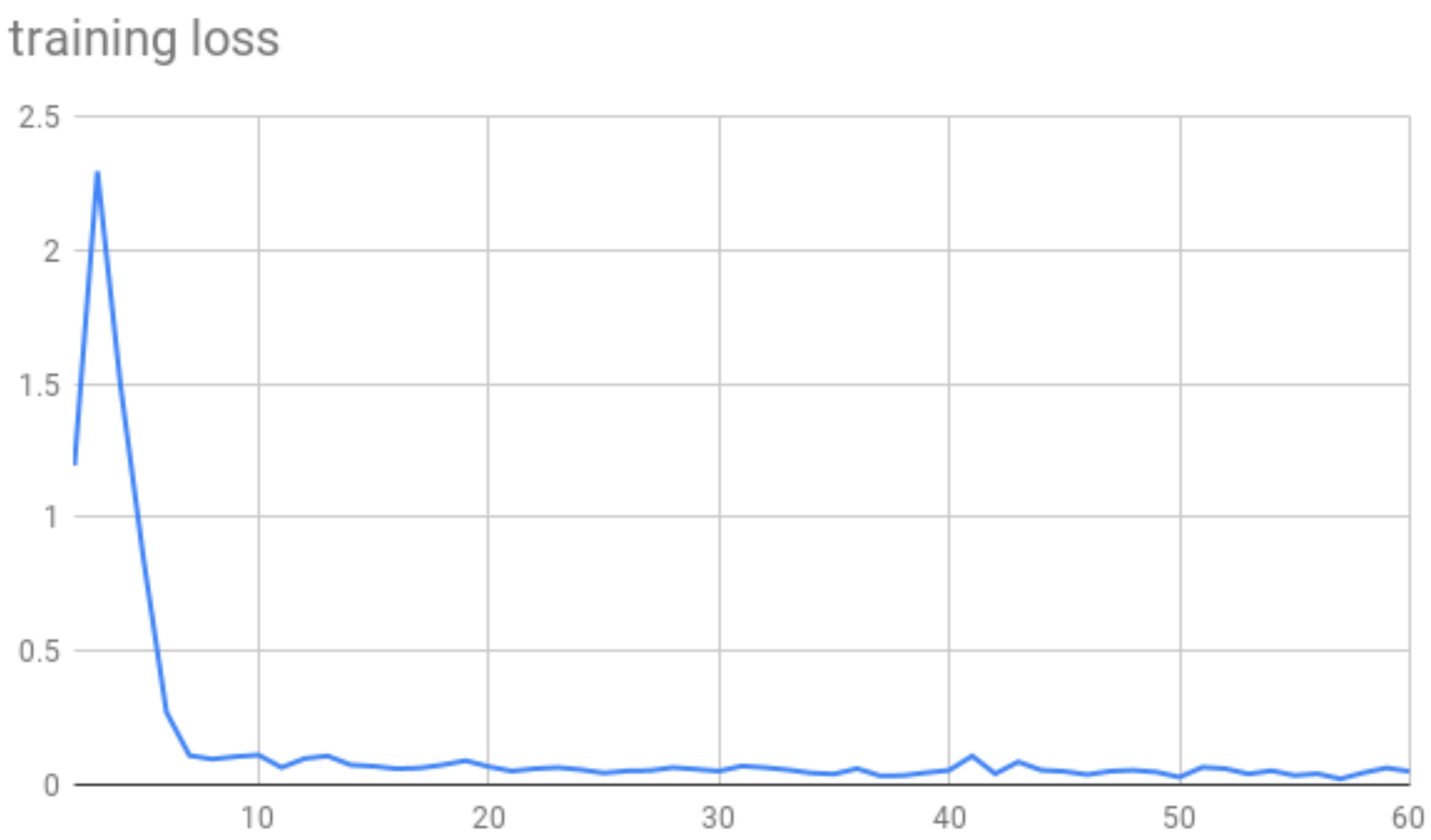
- Compute loss

```
val z = weights.zip(image.m.flatten).map { case (w, x) => w * x }.sum + bias
val yHat = 1 / (1 + pow(E, -z)) // sigmoid
val loss = log(1 + pow(E, (1 - 2*y)* z)) // cross entropy loss
```

- Learn from loss

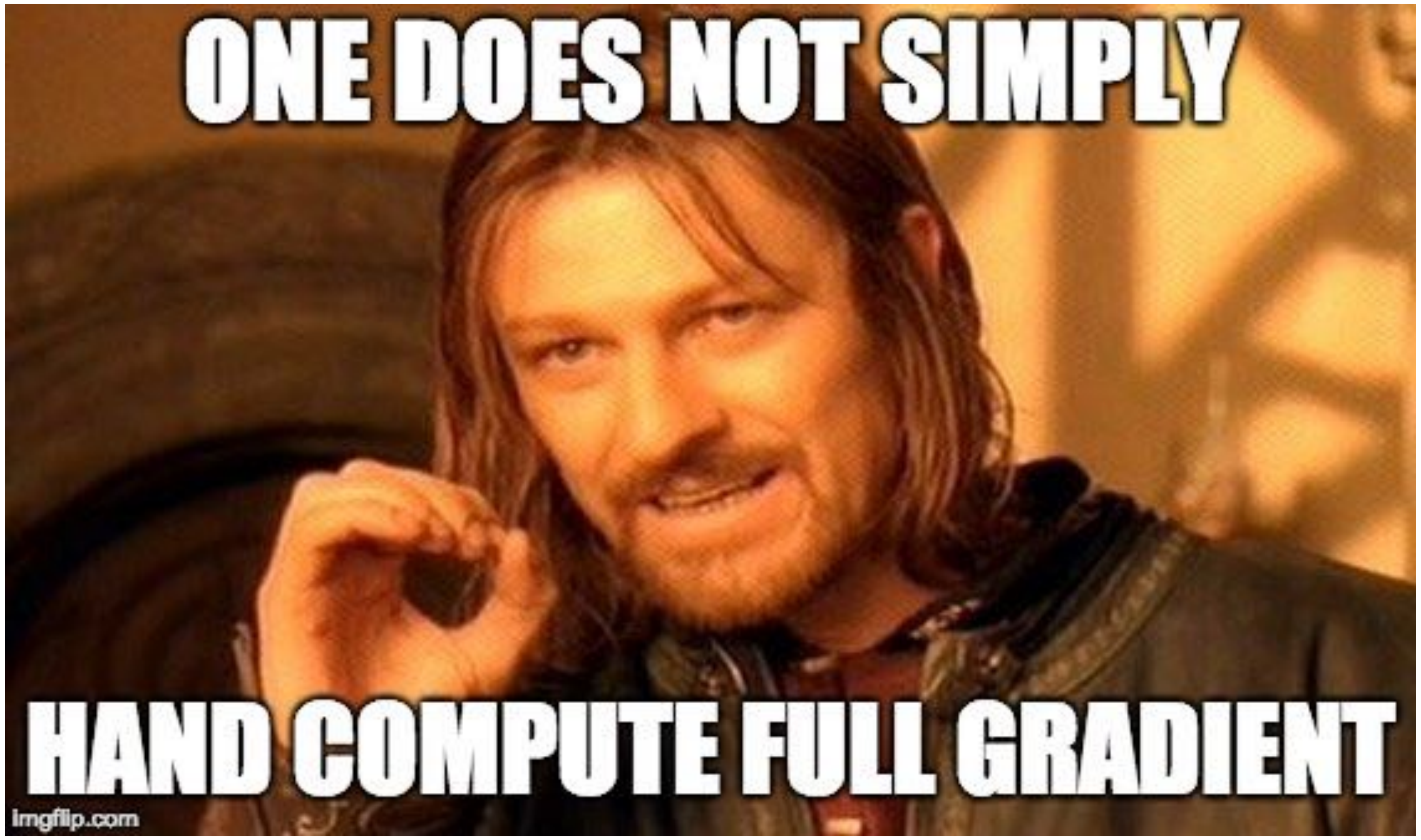
```
// dLoss/dZ
val a  = (1 - 2 * y) * z
val d = 1 / (1 + pow(E, a)) * pow(E, a) * (1 - 2 * y)

val weightsGradient = image.m.flatten.map(x => d * x)
val biasGradient = d

val newWeights = weights - learningRate * weightsGradient
val newBias = bias - learningRate * biasGradient
```

DEMO

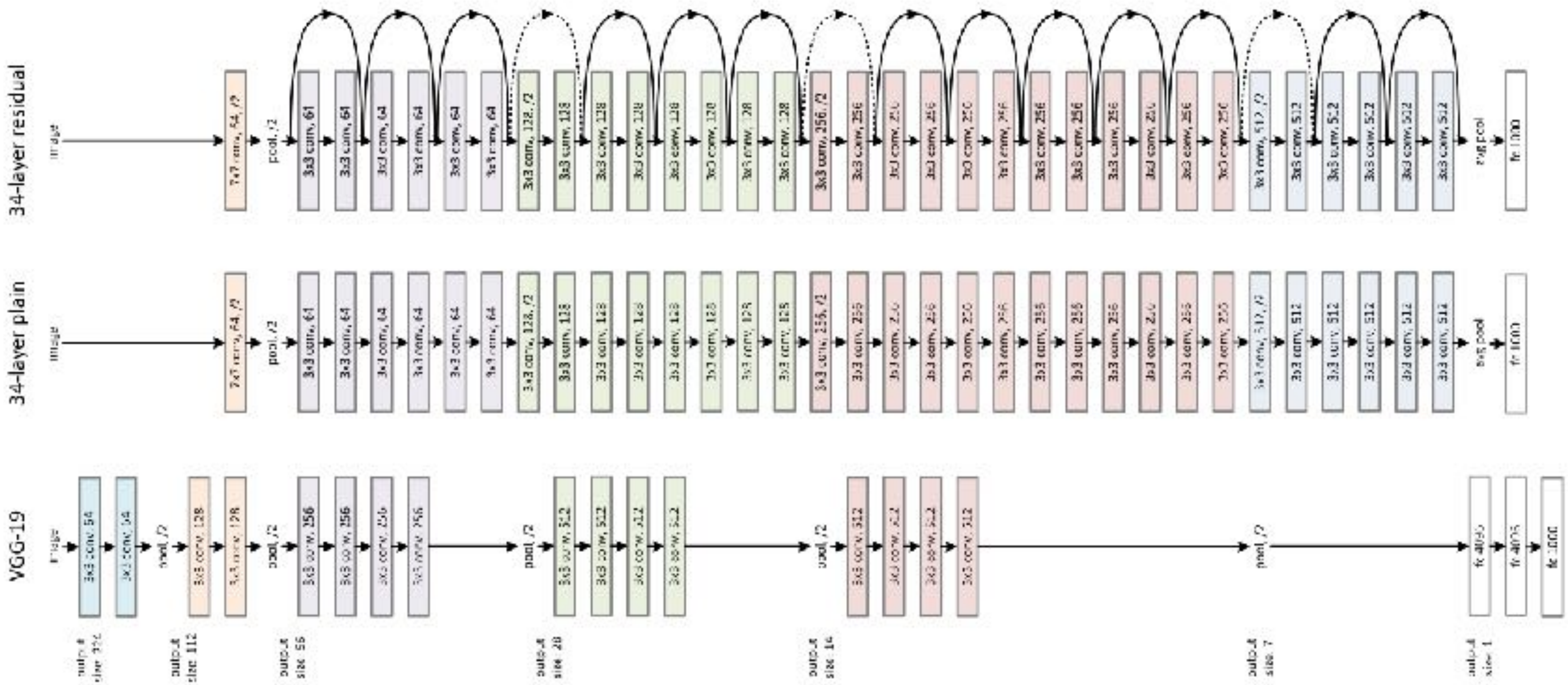


WAIT… IS THAT IT?

- I mean, it's just logistic regression…
- Also =>



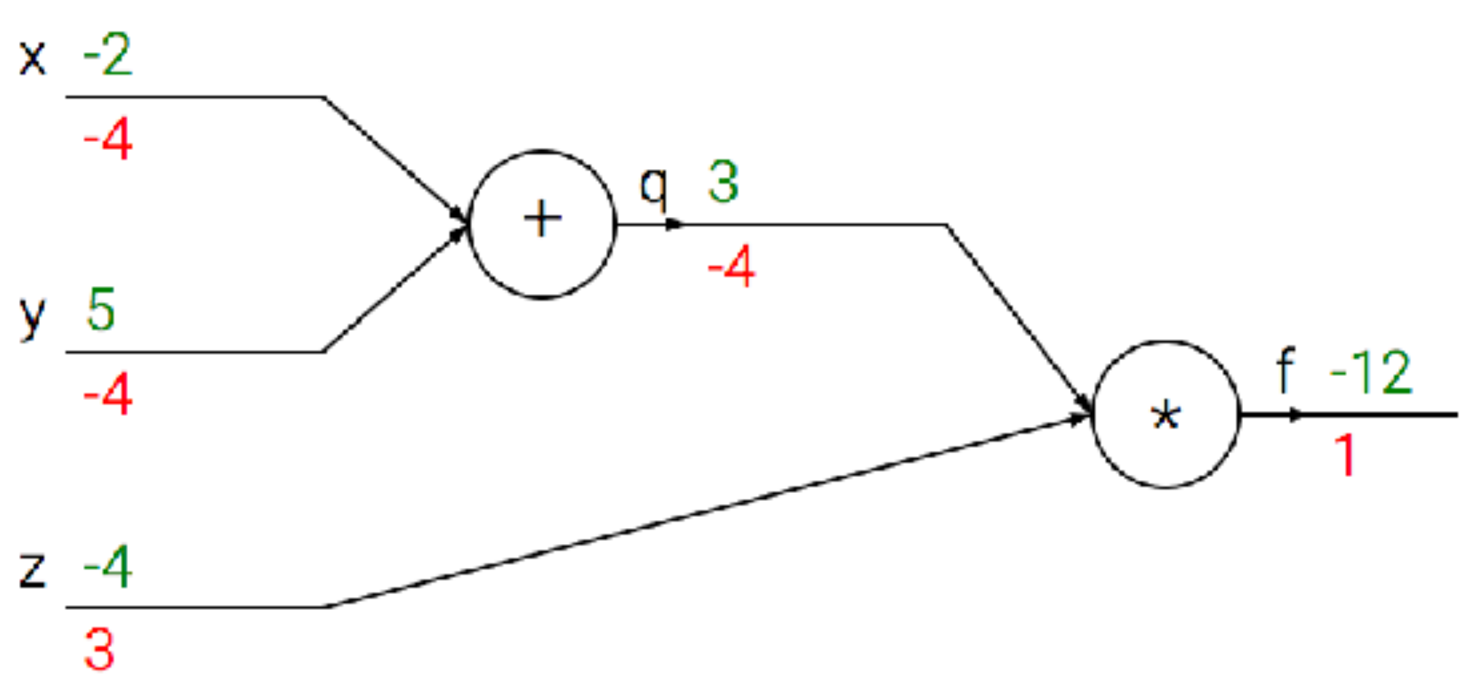
TRY COMPUTE THE GRADIENT OF THE BELOW



ResNet https://pythonmachinelearning.pro/understanding-advanced-convolutional-neural-networks/

BACKPROPAGATION

- A general method to calculate gradients.



http://cs231n.github.io/optimization-2/

BACKPROPAGATION IS GRAPH COMPUTATION

- Forward prop: build the graph
  - For each node, record its inputs and its consumers
- Backward prop: traverse the graph

```
def buildGrad(
  v: Node,
  gradTable: Map[String, Tensor]): Map[String, Tensor] =

  if (gradTable contains v.name) gradTable
    else {
      val g = consumersMap(v.name)
        .map(c => {
          val d = buildGrad(c, gradTable)(c.name)
          c.op.bprop(inputsMap(c.name), v, d)
        })
        .reduce((t1, t2) => Tensor.add(t1, t2))

      gradTable + ((v.name, g))
  }
```

GRAPH KEY COMPONENTS

- Nodes
- Operations
- Graph

GRAPH — NODES

- A node stores:
  - current computed value
  - associated operation

```scala
case class Node(
    name: String,
    v: Tensor,
    op: Op
)
```

GRAPH — OPERATIONS

- An "Op" can:
  - compute: eg. matrix multiplication, sigmoid, etc.
  - backprop: carry the gradient backwards to its inputs

```
sealed trait Op {
  def bprop(
   inputs: List[Node],
   x: Node,
   g: Tensor): Tensor
}

trait BinaryOp extends Op {
  def f(n1: Node, n2: Node): Tensor
}

trait SingleOp extends Op {
  def f(n: Node): Tensor
}
```

GRAPH — GRAPH ITSELF

- A graph remembers each node's inputs and consumers
- A graph carries out the complete backprop

```
trait Graph {
  val nodes: List[Node]
  val consumersMap: Map[String, List[Node]]
  val inputsMap: Map[String, List[Node]]

  def backProp(targets: List[Node], z: Node) = {

    def buildGrad(v: Node, gradTable: Map[String, Tensor]): Map[String, Tensor]
      // see previous slides

    targets
      .foldRight(Map[String, Tensor](z.name -> Scalar(1))) {
        case (t, gradTable) => buildGrad(t, gradTable)
      }
      .filterKeys(targets.map(_.name) contains _)
  }

}
```

GRAPH — PUTTING THINGS TOGETHER

- Use state monad to build the graph
  - unlock for comprehensions!

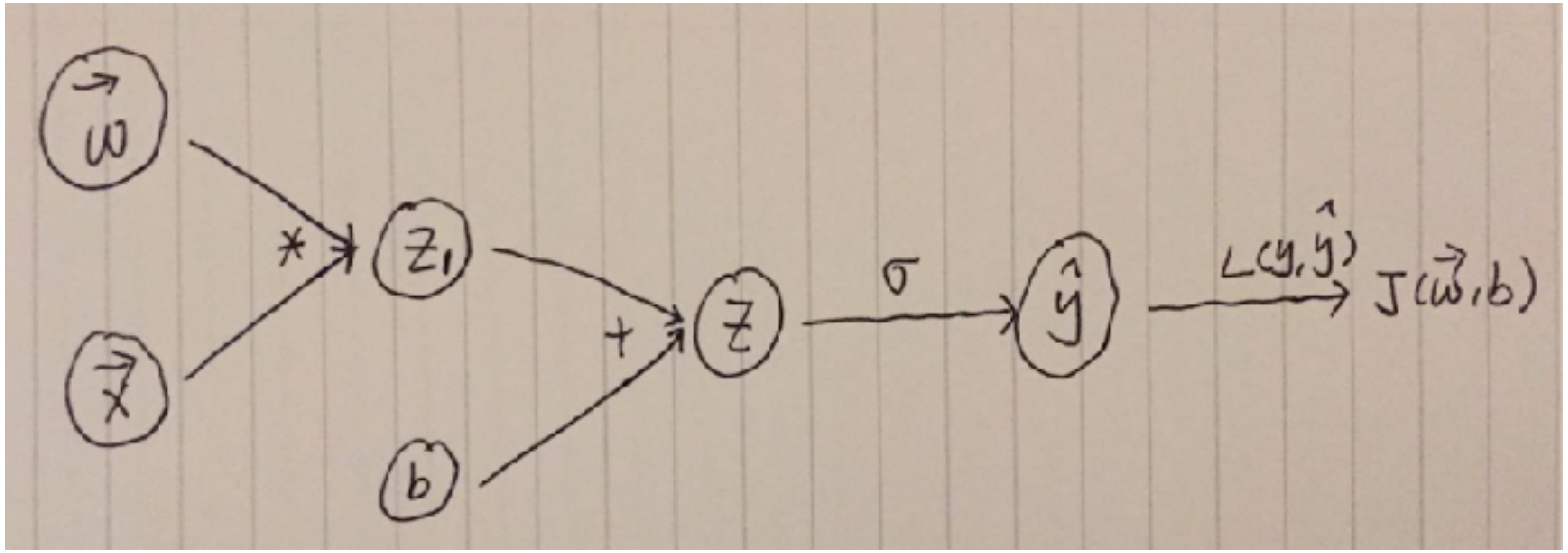
```
def BinaryStep(op: BinaryOp) = State[(List[Node], Graph), Tensor] {
  case (n1 :: n2 :: tail, g) => {
    val ans      = op.f(n1, n2)
    val nodeName = UUID.randomUUID().toString
    val newNode  = Node(nodeName, ans, op)

    val newGraph = new Graph {
      // update nodes, consumersMap, inputsMap
      ...
    }

    ((newNode :: tail, newGraph), ans)
  }
}

val graph = for {
  _   <- BinaryStep(MatMul)
  _   <- BinaryStep(Add)
  ans <- BinaryStep(CrossEntropy)
} yield ans
```

Disney streaming services

GRAPH — TRAIN

```
val w = Node("w", Matrix(Vector(weights)), Ident)
val b = Node("b", Scalar(bias), Ident)
val x = Node("x", Matrix(img.m.flatten.map(Vector(_))), Ident)
val y = Node("y", Scalar(_y), Ident)

val args = List(w, x, b, y)
val init = (args, emptyGraph(args))

val ((nodes, g), loss) = graph.run(init).value
val gradients = g.backProp(List(w, b), nodes.head)

// then do gradient descent
```

SECOND NEURAL NETWORK ON MNIST

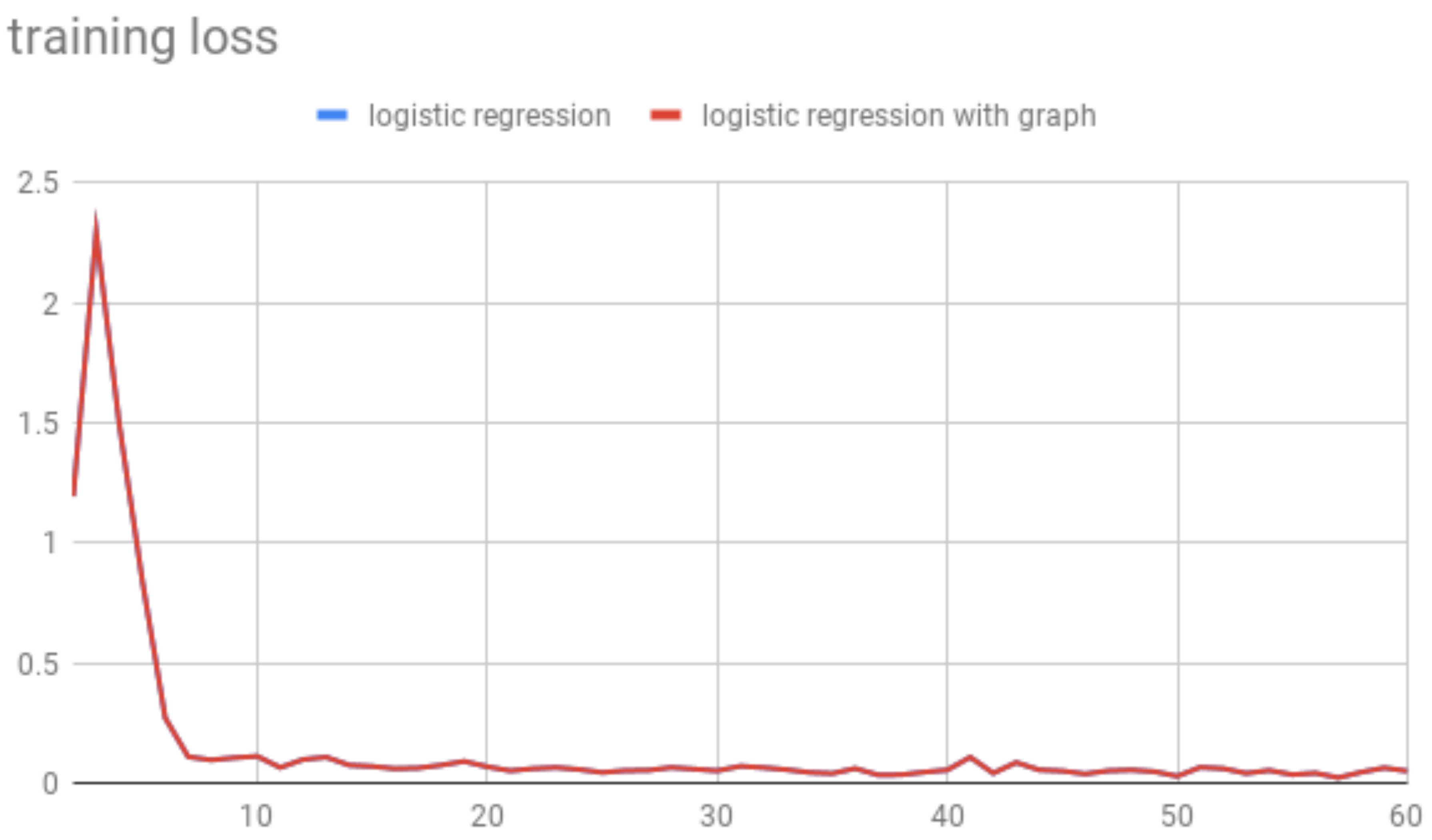


Disney streaming services

SECOND NEURAL NETWORK — GRAPH

```
private val graph = for {
  _   <- BinaryStep(MatMul)
  _   <- BinaryStep(Add)
  ans <- BinaryStep(CrossEntropy)
} yield ans
```

DEMO



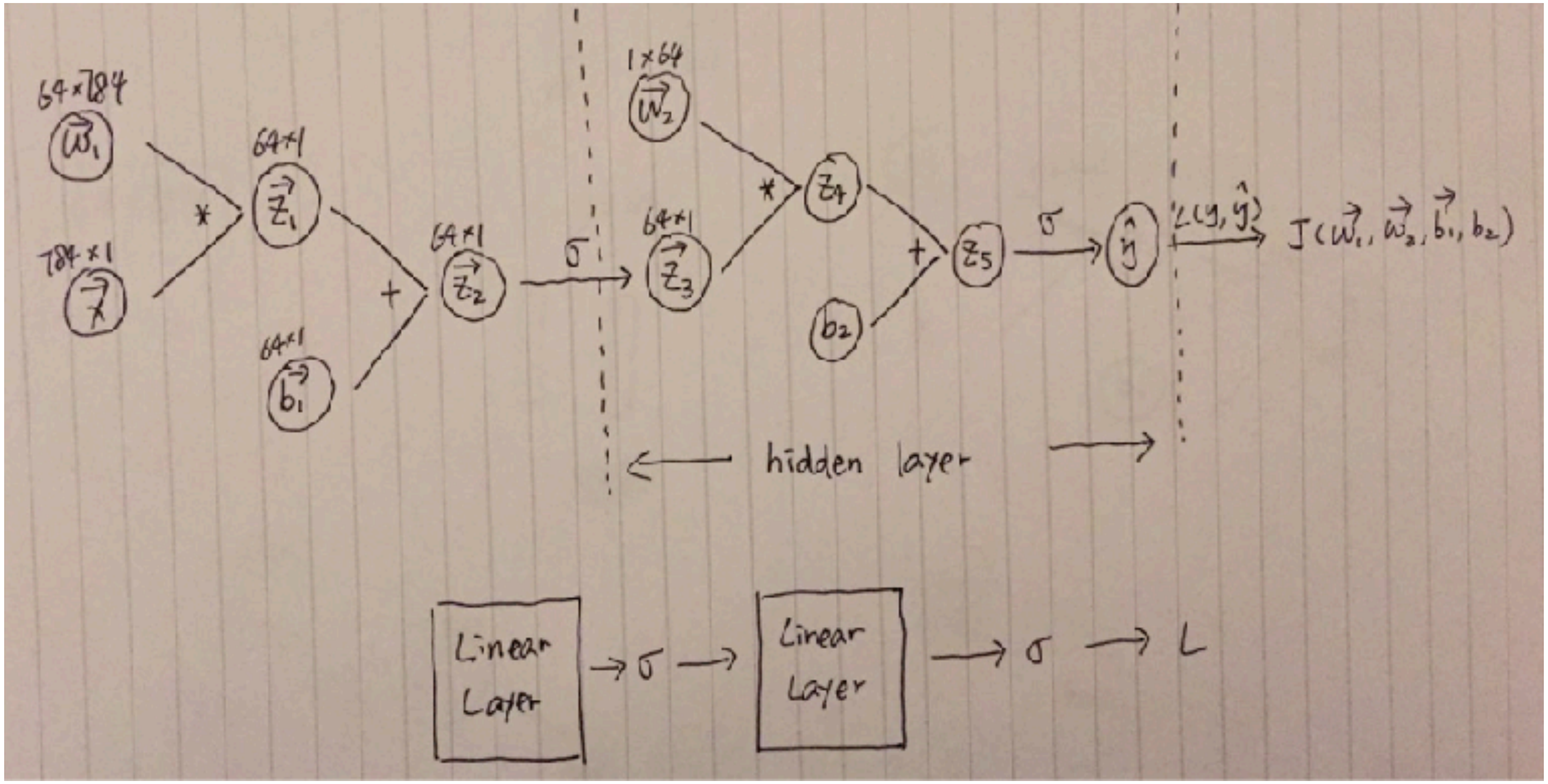
Disney streaming services

WAIT…IS THAT IT?

- I mean… that's still logistic regression, just trained in a different way.
- Me wants fancier networks!

THIRD NEURAL NETWORK

- "Hidden layer" is the black magic(1) of neural networks
- Let's add a hidden layer!



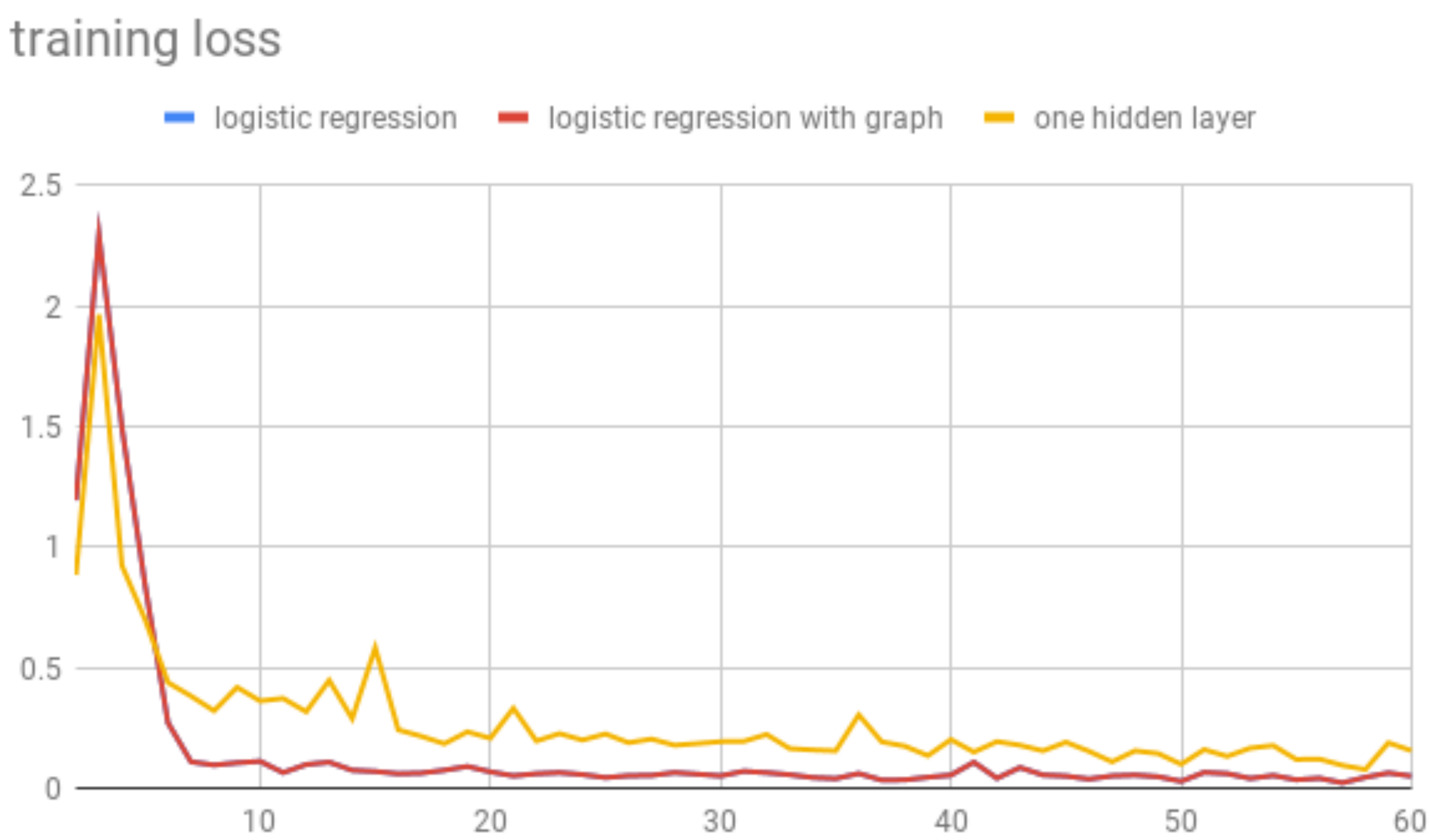
ok, **one** of the black magics

Disney streaming services

THIRD NEURAL NETWORK — GRAPH

```scala
private val trainGraph = for {
  _   <- BinaryStep(MatMul)
  _   <- BinaryStep(Add)
  _   <- SingleStep(Sigmoid)
  _   <- BinaryStep(MatMul) // hidden layer!
  _   <- BinaryStep(Add) // hidden layer!
  ans <- BinaryStep(CrossEntropy)
} yield ans
```

DEMO


training loss
logistic regression
logistic regression with graph
one hidden layer
2.5
2
1.5
1
0.5
0
10
20
30
40
50
60

WHY NOT JUST TENSORFLOW?

- In search for a robust, yet still efficient research language
  - It's difficult enough to debug neural networks already
  - What can go wrong? ↓

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

- Bugs are still a big problem

> a predictable range. We also noticed **significant improvements in performance of RND every time we discovered and fixed a bug** (our

> It ended up taking me 6 weeks to reproduce results, thanks to several software bugs. The question is, why did it take so long to find these bugs?

https://www.alexirpan.com/2018/02/14/rl-hard.html

https://blog.openai.com/reinforcement-learning-with-prediction-based-rewards/

Disney streaming services

WHY NOT JUST TENSORFLOW?

- My ideal research language
  - Typed, possibly higher kind support (Python is still limited)
  - Simple, clear syntax (Scala not quite there)
  - Native vector & parallelism (JVM is behind)
  - Version-control-friendly Notebook environment + visualisation toolbox
- More ideas/discussions! =]

Disney streaming services

THAT'S IT! I HOPE YOU WALK AWAY KNOWING:

- What a neural network is: stacked layers of weights + activation functions
- How to train a neural network: the idea of backpropagation
- [Stretch goal] Thoughts on how to better support robust and efficient ML research

CODE

- https://github.com/xysun/neural-network-scalax
- GitHub: @xysun

THANKS!