Lecture Notes for

Machine Learning in Python

Professor Eric Larson

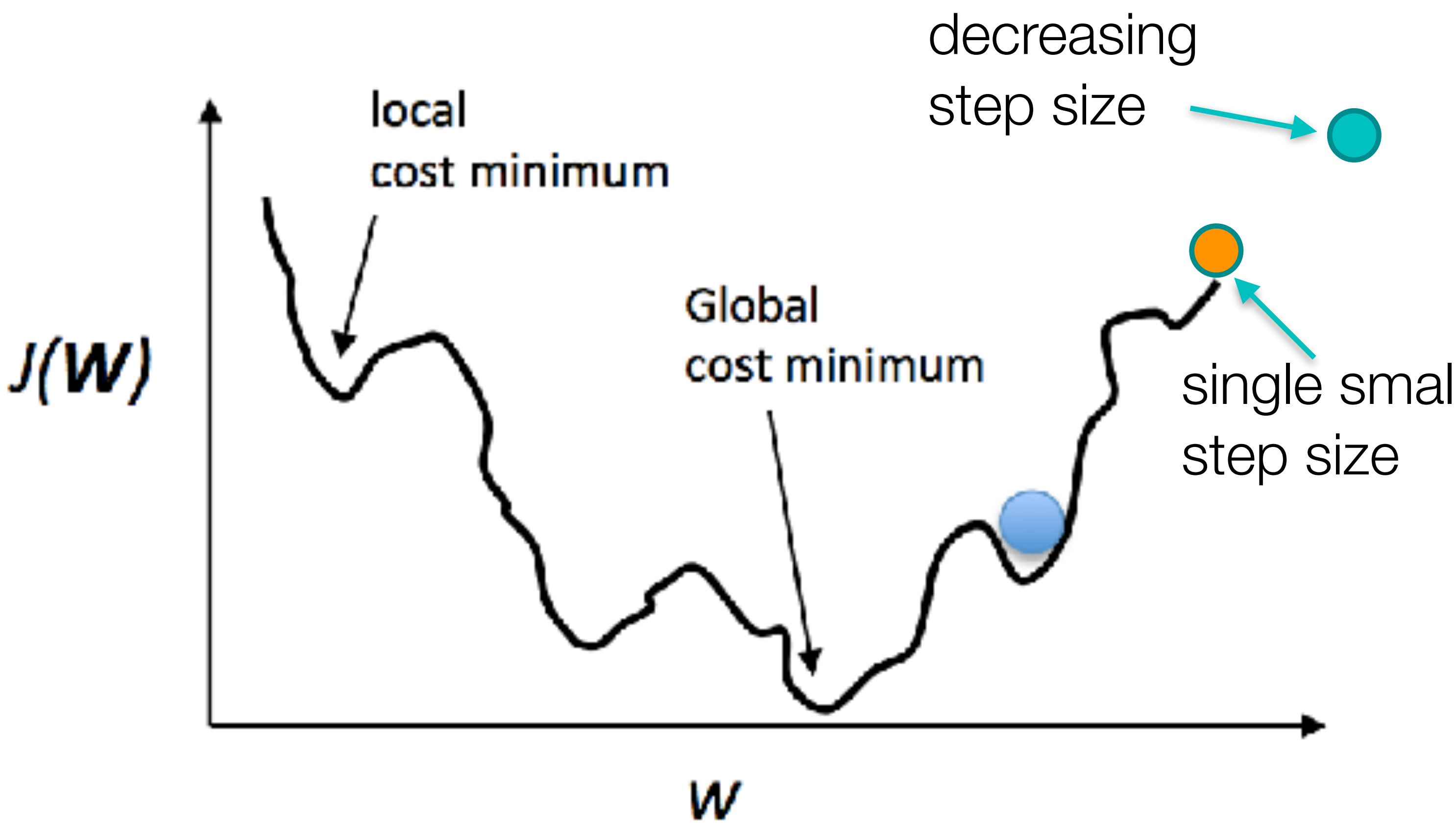
Week Eight B

Class Logistics and Agenda

- Welcome back from fall break!
- Grades Coming Soon, but slowly
- A2 posted, schedule revised
- Two Week Agenda:
    - *SVM Review*
    - *Neural Networks History*
    - *Multi-layer Architectures*
    - Programming Multi-layer training
- **Next Time**: end of NN, start ensemble classifiers

Last Time: Problems with Advanced Architectures

- Space is no longer convex
  - **Two solutions**:
    - “cool down” by decreasing step size for higher iterations
    - keep going in direction of previous gradient (to some degree)



Demo

Two Layer Perceptron

comparison:

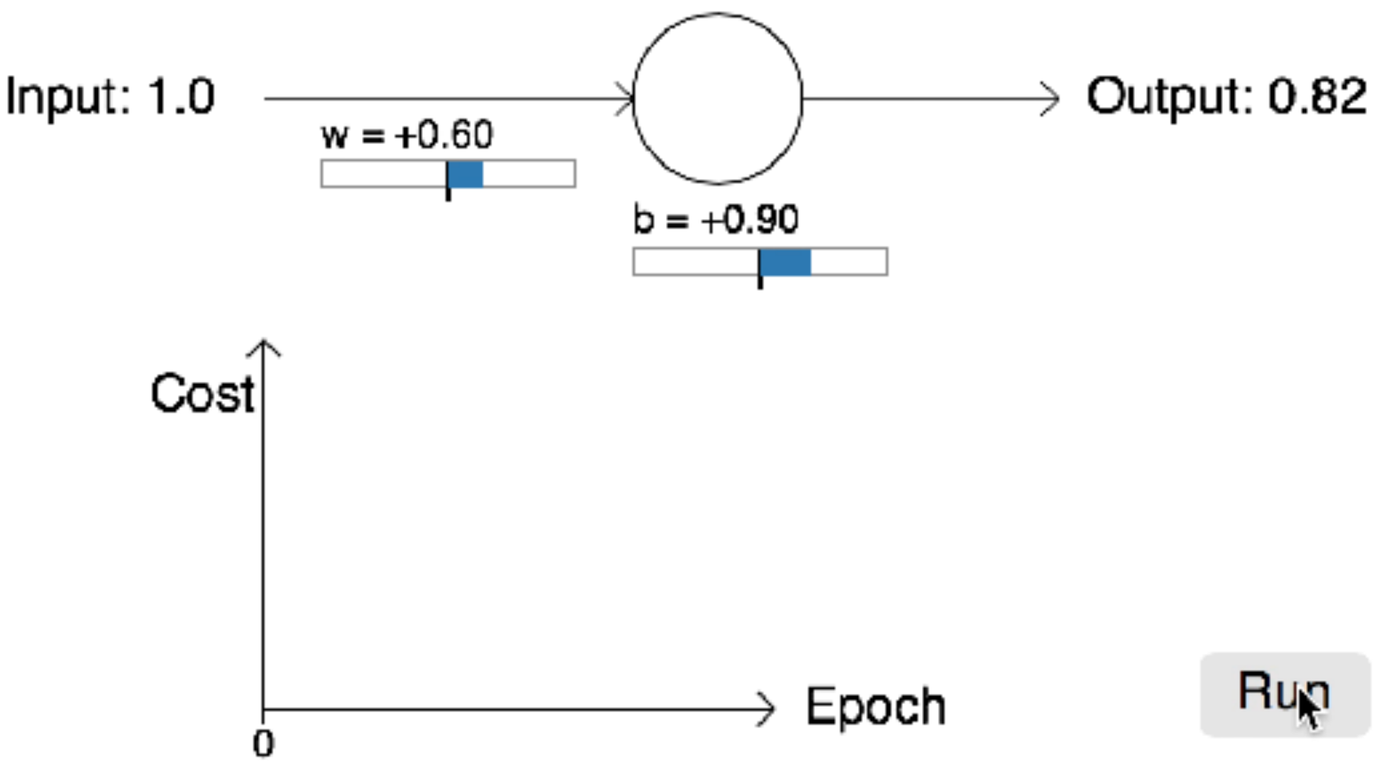
- mini-batch
  - momentum
  - decreased learning
- L-BFGS

Practical Implementation of Architectures

- A new cost function: **Cross entropy** and softmax

$$J(\mathbf{W}) = \sum_{k}^{M} (\mathbf{y}^{(k)} - \mathbf{a}^{(L)})^2$$

least squares objective,
tends to slow training initially



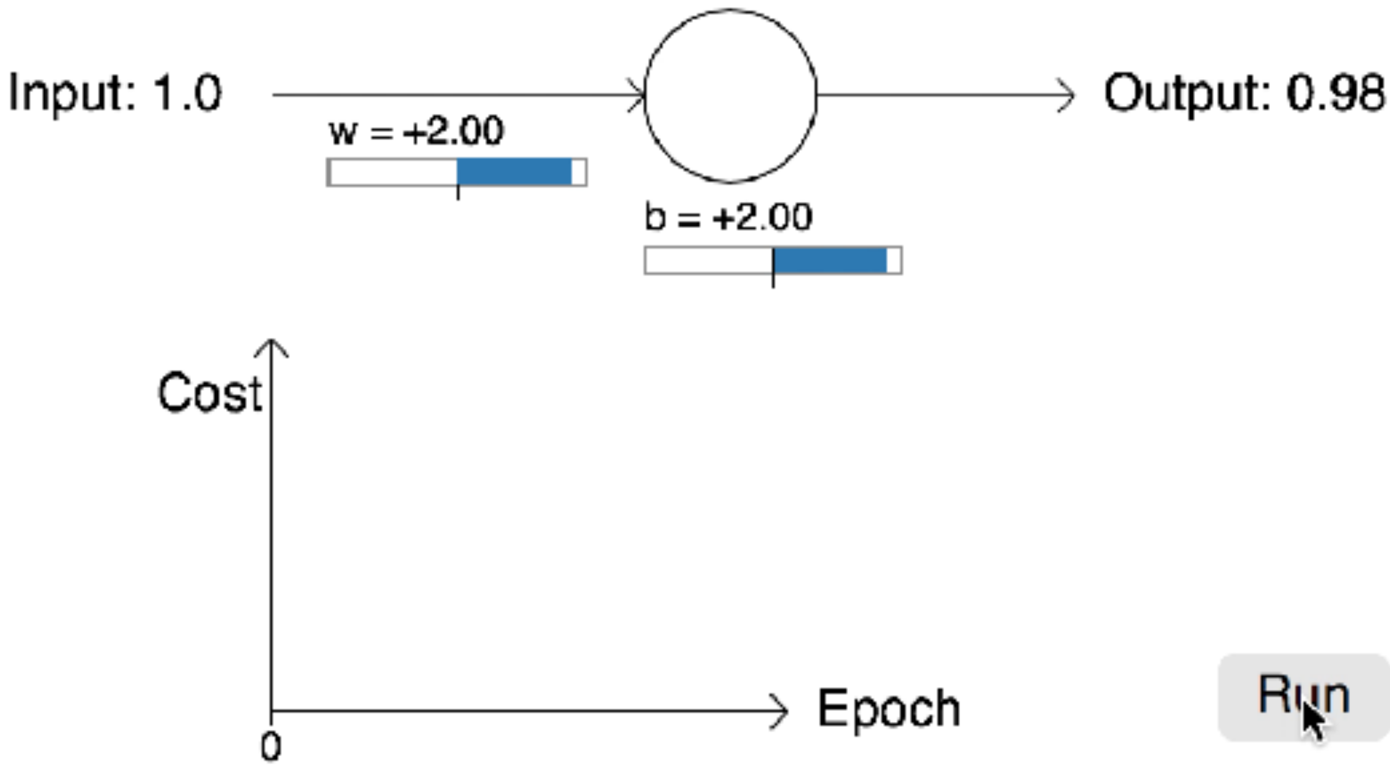
Neural Networks and Deep Learning, Michael Nielson, 2015

Practical Implementation of Architectures

- A new cost function: **Cross entropy** and softmax

$$J(\mathbf{W}) = \sum_{k}^{M} (\mathbf{y}^{(k)} - \mathbf{a}^{(L)})^2$$

least squares objective, tends to slow training initially

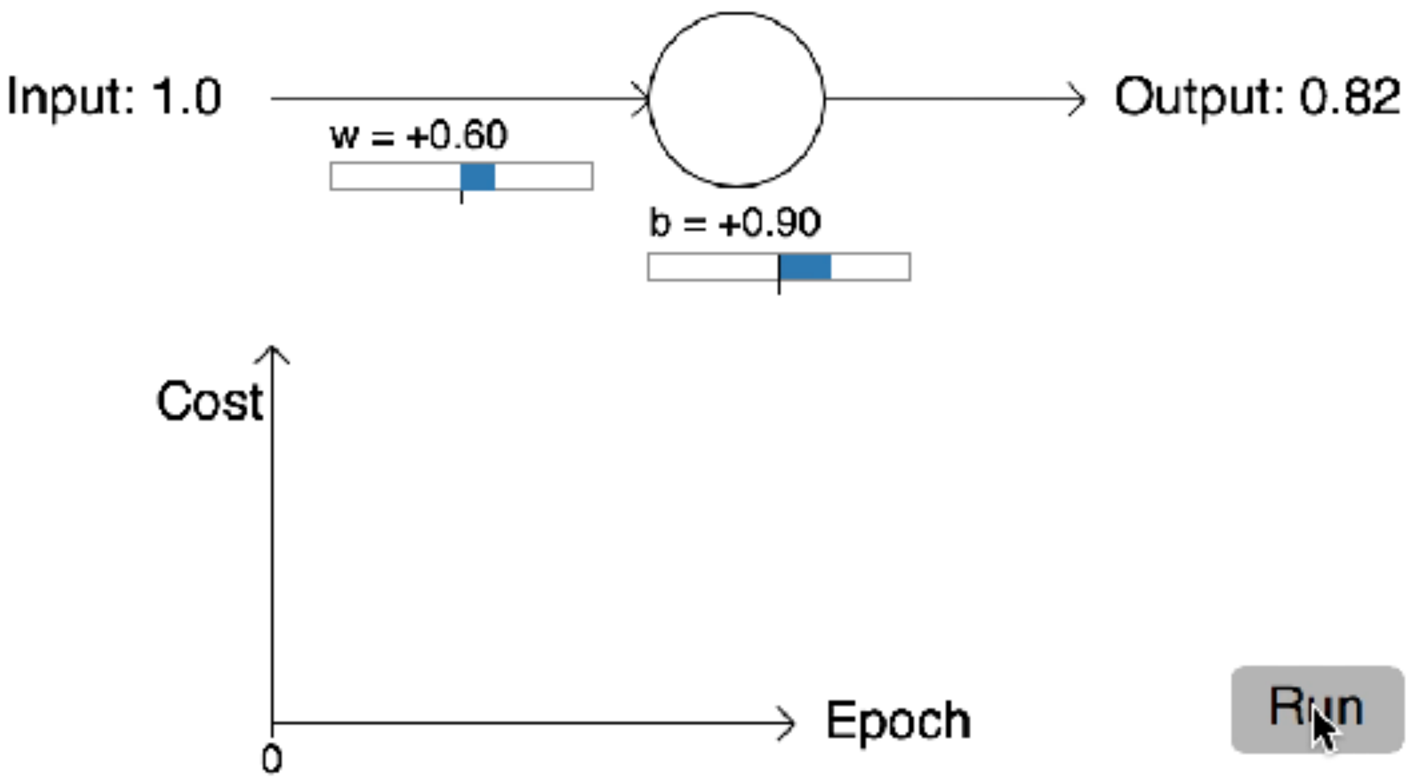


Practical Implementation of Architectures

- A new cost function: **Cross entropy** and softmax

$$J(\mathbf{W}) = -[\mathbf{y}^{(i)} \ln \mathbf{a}^{(L)} + (1 - \mathbf{y}^{(i)}) \ln(1 - \mathbf{a}^{(L)})]$$

speeds up initial training



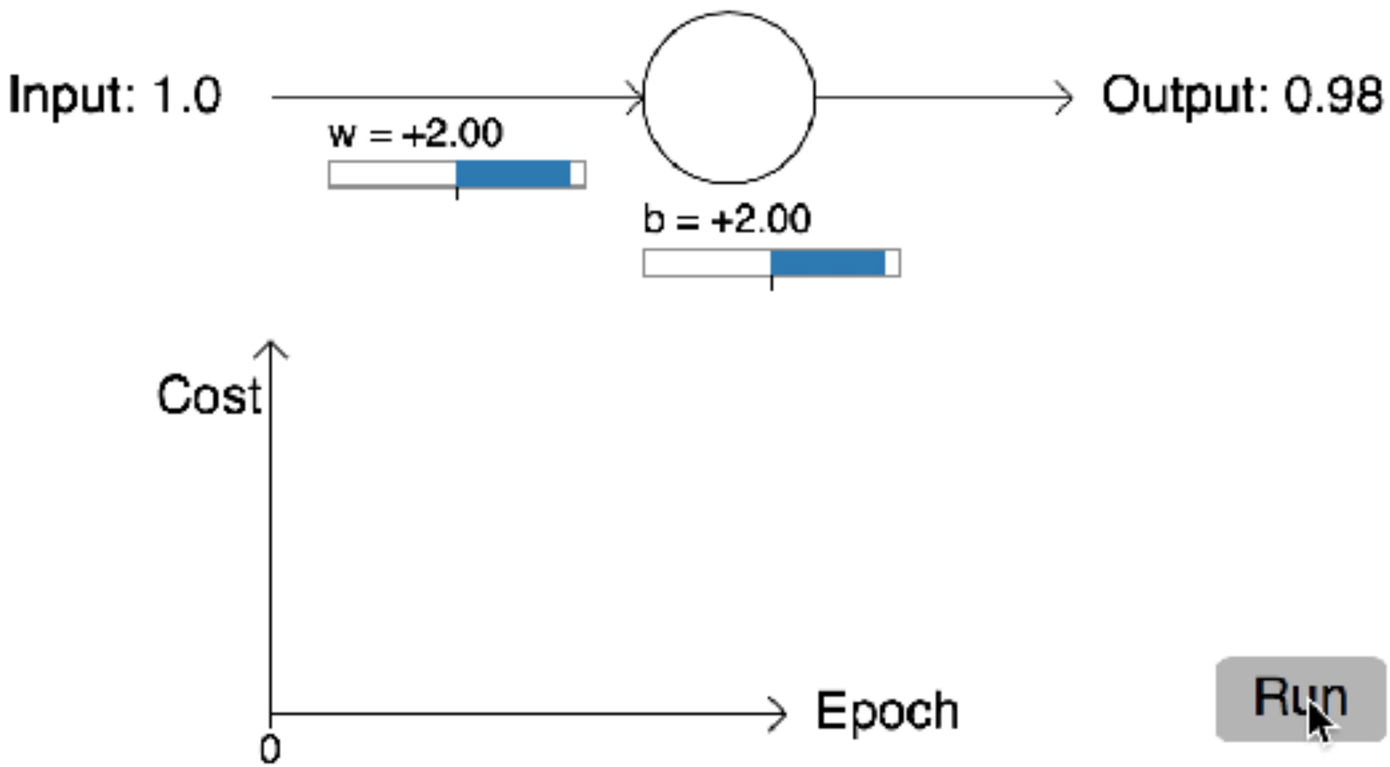
Neural Networks and Deep Learning, Michael Nielson, 2015

Practical Implementation of Architectures

- A new cost function: **Cross entropy** and softmax

$$J(\mathbf{W}) = -[\mathbf{y}^{(i)} \ln \mathbf{a}^{(L)} + (1 - \mathbf{y}^{(i)}) \ln(1 - \mathbf{a}^{(L)})]$$

speeds up initial training



Practical Implementation of Architectures

- A new cost function: **Cross entropy** and softmax

$$J(\mathbf{W}) = -[\mathbf{y}^{(i)} \ln \mathbf{a}^{(L)} + (1 - \mathbf{y}^{(i)}) \ln(1 - \mathbf{a}^{(L)})]$$

speeds up initial training

bp-5

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(k)} - \mathbf{a}^{(3)}) * \mathbf{a}^{(3)} * (1 - \mathbf{a}^{(3)})$$ old update

Neural Networks and Deep Learning, Michael Nielson, 2015

Practical Implementation of Architectures

- A new cost function: **Cross entropy** and softmax

$$J(\mathbf{W}) = -[\mathbf{y}^{(i)} \ln \mathbf{a}^{(L)} + (1 - \mathbf{y}^{(i)}) \ln(1 - \mathbf{a}^{(L)})]$$

speeds up initial training

$$\frac{\partial J(\mathbf{W})}{\mathbf{z}^{(L)}} = ({}_i\mathbf{a}^{(L+1)} - \mathbf{y}^{(i)})$$

```
# vectorized backpropagation
sigma3 = (A3-Y_enc) # <- this is only line
sigma2 = (W2.T @ sigma3)*A2*(1-A2)

grad1 = sigma2[1:,:] @ A1
grad2 = sigma3 @ A2.T
```

$$\frac{\partial J(\mathbf{W})}{\mathbf{z}^{(2)}} = ({}_i\mathbf{a}^{(3)} - \mathbf{y}^{(i)})$$

new update

```
# vectorized backpropagation
sigma3 = -2*(Y_enc-A3)*A3*(1-A3)
sigma2 = (W2.T @ sigma3)*A2*(1-A2)

grad1 = sigma2[1:,:] @ A1
grad2 = sigma3 @ A2.T
```

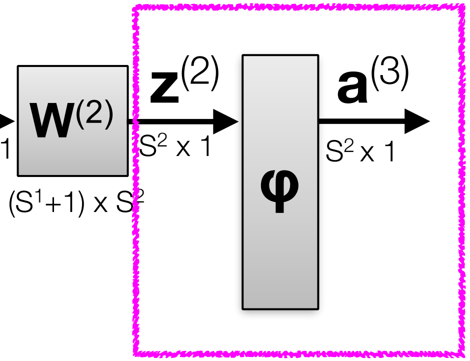
$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(2)}} = -2(\mathbf{y}^{(k)} - \mathbf{a}^{(3)}) * \mathbf{a}^{(3)} * (1 - \mathbf{a}^{(3)})$$ old update

bp-5

Neural Networks and Deep Learning, Michael Nielson, 2015

Practical Implementation of Architectures

- A new cost function: Cross entropy and **softmax**



$$a_j^{(L+1)} = \frac{\exp(z_j^{(L)})}{\sum_i \exp(z_i^{(L)})}$$

instead of final layer sigmoid!

it has many of the same properties as Cross Entropy
but also the advantage of **interpretation as a probability**

if $J(\mathbf{W}) = -\Sigma\, y_j \ln(a_j^{(L)})$, then update equations are **identical** to that of Cross Entropy.

Neural Networks and Deep Learning, Michael Nielson, 2015

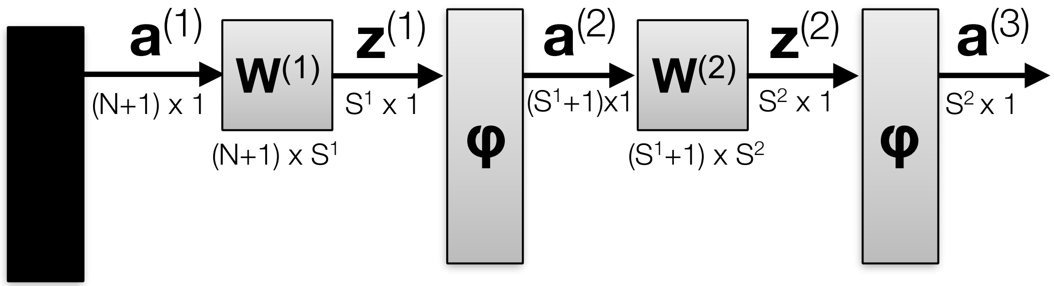
Practical Implementation of Architectures

- L1 regularization:
  - there is an error in the cod I gave you? *Can you find it?*
- Beyond L1 and L2 regularization
  - dropout
  - expansion

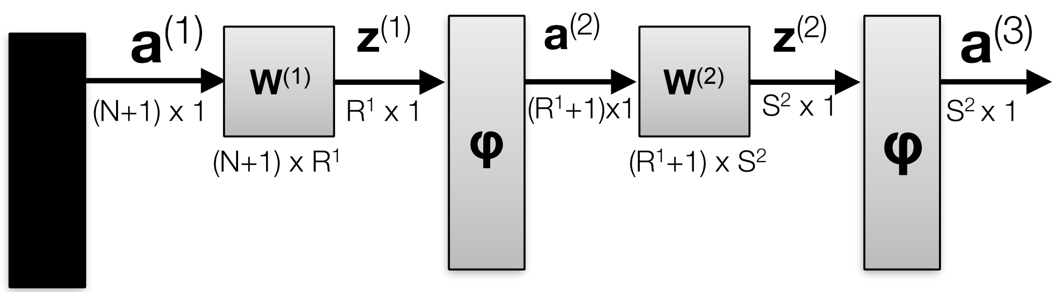
yeah, I misspelled that as a joke!

Practical Implementation of Architectures

- Dropout
  - don’t train all hidden neurons at the same time



For each mini-batch,
R^1 is subset of S^1

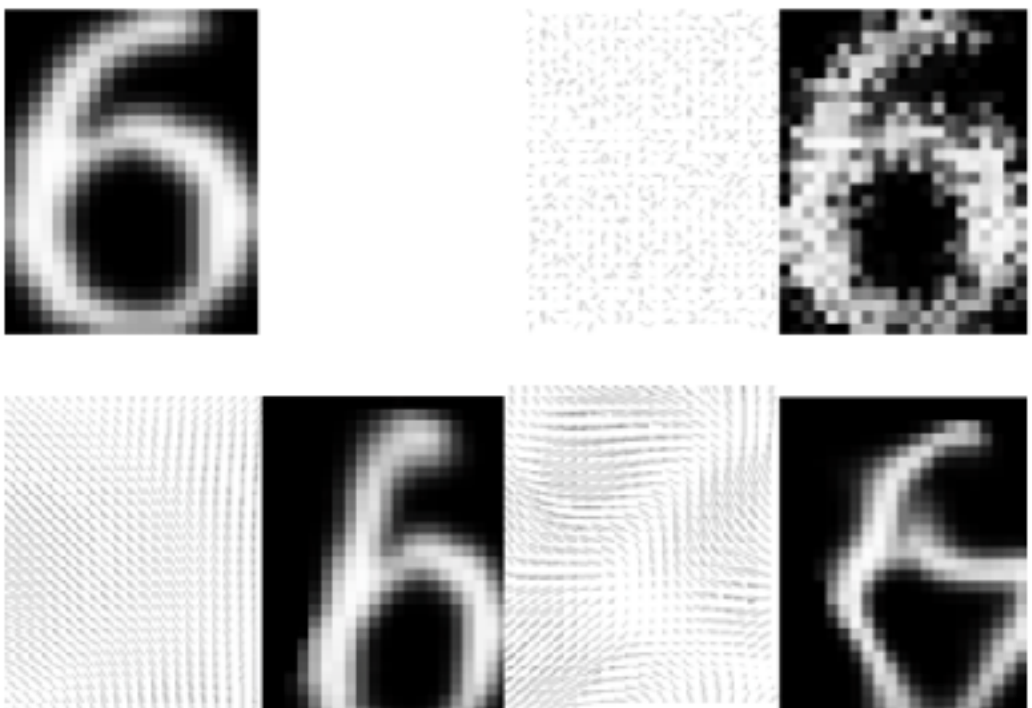


Why does this work?

Practical Implementation of Architectures

- Expansion
  - get more data
  - perturb your data

Neural Networks and Deep Learning, Michael Nielson, 2015



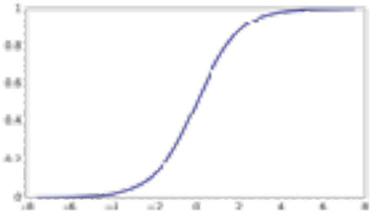
Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis, by Patrice Simard, Dave Steinkraus, and John Platt (2003)

98.4% → 99.3%

Practical Implementation of Architectures

- Weight initialization
  - uniform distribution makes weights unrealistic
  - try not to **saturate** your neurons right away!

$\mathbf{a}^{(L+1)} = \boldsymbol{\phi}(\mathbf{W}^{(L)}\mathbf{a}^{(L)})$



each row is sum of layer inputs

want sum to be between $\varepsilon < \Sigma < 1-\varepsilon$ for no saturation
solution: squash initial weights sizes

- a nice choice: each element of **W** is selected from a Gaussian with zero mean
- for adding Gaussian distributions, variances add together:
  - make each variance $1/\mathbf{W}_{\text{num\_elements\_in\_row}}$
  - which is the same as standard deviation = $1/\mathit{sqrt}(\mathbf{W}_{\text{num\_elements\_in\_row}})$

Practical Details

- Neural networks can separate any data through multiple layers. The true realization of Rosenblatt:

> "Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time…"

- **Universality**: No matter what function we want to compute, we know that there is a neural network which can do the job.
- One hidden layer can solve any problem with enough data
  - but… it might be better to have even more layers for decreased computation and generalizability

Demo

Two Layer Perceptron

Practical implementations