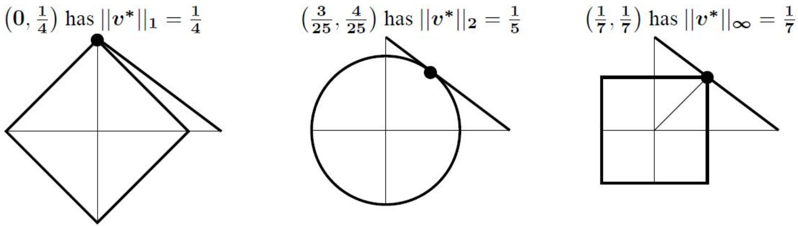
Teaching About Learning

Gilbert Strang

MIT

Minimize $\|\mathbf{v}\|_p$ among vectors (v_1, v_2) on the line $3v_1 + 4v_2 = 1$


$(0, \frac{1}{4})$ has $||v^*||_1 = \frac{1}{4}$
$(\frac{3}{25}, \frac{4}{25})$ has $||v^*||_2 = \frac{1}{5}$
$(\frac{1}{7}, \frac{1}{7})$ has $||v^*||_\infty = \frac{1}{7}$

The solutions v^* to the ℓ^1 and ℓ^2 and ℓ^∞ minimizations.

The ℓ^1 solution $(0, \frac{1}{4})$ is **sparse**.

Orthonormal $v_1 \dots v_r$

$Av_i = \sigma_i u_i$ Orthonormal $u_1 \dots u_r$

Principal components in the SVD $A = U\Sigma V^T$

$$A \begin{bmatrix} \mathbf{v}_1 & \dots & \mathbf{v}_r & \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \dots & \mathbf{u}_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}$$

Construction $A^T A v_i = \sigma_i^2 v_i$ eigenvectors of $A^T A$

$u_i = Av_i / \sigma_i$ also orthornormal

Eckart-Young property of $U\Sigma V^T = \sigma_1 u_1 v_1^T + \cdots + \sigma_r u_r v_r^T$

$A_k =$ sum of first k of those rank 1 pieces has rank k

Best approximation $\|A - A_k\| \leq \|A - B\|$ if B has rank k

$\|A\|_2 = \sigma_{\max}$ or $\|A\|_F^2 = \sigma_1^2 + \cdots + \sigma_r^2$ or $\|A\|_N = \sigma_1 + \cdots + \sigma_r$

Alternating algorithm to minimize $\|\mathbf{A} - \mathbf{C}\mathbf{R}^\mathbf{T}\|$

Find best R with C fixed, then C with R fixed

Unsupervised learning: The only instructor is linear algebra

Three bases for the column space of A

1. Orthonormal columns in Q $\qquad A = QR$ is Gram-Schmidt
2. Orthonormal columns in U $\qquad A = U\Sigma V^T$ is the SVD
3. **Independent columns in C taken directly from A**

   $\mathbf{A = CMR^T = }$ **(columns from A) M (rows from A)**

   $\mathbf{M}$ = mixing matrix often written as U

Goal of deep learning

Create a function F that learns how to classify data vectors v

You told it the correct classification for the training data

A giant optimization finds weights in F
to reproduce those correct classifications $F(v)$

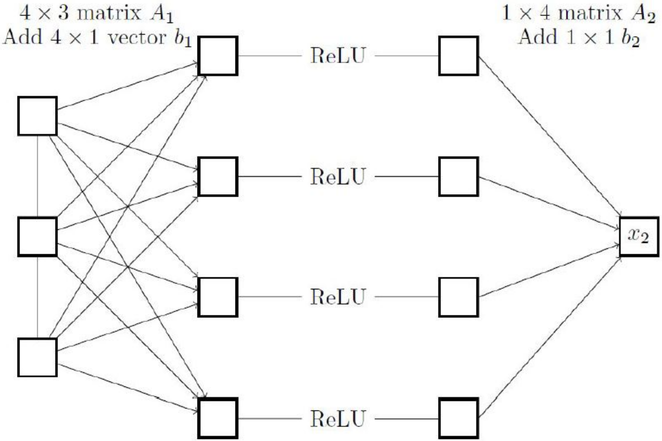
Construction of Deep Neural Networks

1	Key operation	Composition $F = F_3(F_2(F_1(\mathbf{x}, \mathbf{v}_0)))$
2	Key rule	Chain rule for x-derivatives of F
3	Key algorithm	Stochastic gradient descent to find $\mathbf{x}$
4	Key subroutine	Backpropagation to compute grad F
5	Key nonlinearity	ReLU $(y) = \max(y, 0) =$ ramp function

$$\textbf{Layer k} \quad \mathbf{v_k} = \mathbf{F_k}(\mathbf{v_{k-1}}) = \mathbf{ReLU}(\mathbf{A_k}\mathbf{v_{k-1}} + \mathbf{b_k})$$

Weights $\mathbf{x}$ for layer k $\mathbf{A_k}$ = matrix and $\mathbf{b_k}$ = offset vector

$\mathbf{v}_0$ = training data / $\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}$ hidden layers / $\mathbf{v}_\ell$ = output



4×3 matrix A_1
Add 4×1 vector b_1

ReLU

ReLU

ReLU

ReLU

1×4 matrix A_2
Add 1×1 b_2

x_2

Three components
of $\mathbf{v}_0$ for each
training sample

$\mathbf{y}_1$ at layer 1
$\mathbf{y}_1 = A_1\mathbf{v}_0 + b_1$

$\mathbf{v}_1$ at layer 1
$\mathbf{v}_1 = \text{ReLU}(y_1)$

Output $\mathbf{v}_2 = \mathbf{v}_0\ell$
$A_2\mathbf{v}_1 + b_2$

Figure from `math.mit.edu/learningfromdata`

Key computation: Weights $\mathbf{x}$ minimize overall loss $L(\mathbf{x})$

$$L(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} \text{ loss } \ell(\mathbf{x}, \mathbf{v}_0^j) \text{ on sample } j$$

"Square loss" $=$ error $\ell(\mathbf{x}, \mathbf{v}_0^j) = \|F(\mathbf{x}, \mathbf{v}_0^j) - \text{ true } \|^2$

Cross-entropy loss, hinge loss, ...

Classification problem: true $= 1$ or -1

Regression problem: true $=$ vector

Gradient descent $\mathbf{x}_{k+1} = \text{ arg min } \|\mathbf{x}_k - s_k \nabla L(\mathbf{x}_k, \mathbf{v})\|$

Stochastic descent $\mathbf{x}_{k+1} = \text{ arg min } \|\mathbf{x}_k - s_k \nabla \ell(\mathbf{x}_k, \mathbf{v})\|$

Mathematical questions

1. Convergence rate of descent and accelerated descent (when $\mathbf{x}_{k+1}$ depends on $\mathbf{x}_k$ and $\mathbf{x}_{k-1}$: momentum added)
2. Do the weights $A_1, b_1, \ldots$ generalize to unseen test data? (Early stopping / Do not overfit the data)
3. Replace samples v or don't replace in stochastic descent? (Theory versus practice)

1. Stochastic gradient descent optimizes the weights A_k, b_k
2. Backpropagation in the computational graph computes derivatives with respect to weights $\mathbf{x} = A_1, \mathbf{b}_1, \ldots, A_\ell$
3. The learning function $F(\mathbf{x}, \mathbf{v}_0) = \ldots F_3(F_2(F_1(\mathbf{x}, \mathbf{v})))$

$$F_1(\mathbf{v}_0) = \max(A_1 \mathbf{v}_0 + b_1, 0) = \text{ReLU} \circ \text{ affine map}$$

$F(\mathbf{v})$ is continuous piecewise linear: how many pieces?

This measures the “expressivity” of the network

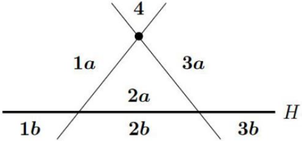
Assume 1 hidden layer with N neurons

v_0 has m components / v_1 has N components / N ReLU's

The number of flat regions bounded by the N hyperplanes is:

$$r(N,m) = \sum_{i=0}^{m} \binom{N}{i} = \binom{N}{0} + \binom{N}{1} + \cdots + \binom{N}{m}$$

$N = 3$ folds in a plane will produce $1+3+3=7$ pieces



Start with 2 folds
$\leftarrow r(2,2) = 4$
Add new fold
$\leftarrow r(2,1) = 3$
Polya's Cake Problem $r(5,3)$

Recursion $r(N,m) = r(N-1,m) + r(N-1,m-1)$

$\mathbf{F(x) = F_2(F_1(x))}$ **is continuous piecewise linear**

One hidden layer of neurons: deep networks have many more

Overfitting is not desirable! Gradient descent stops early!

"Generalization" measured by success on unseen test data

Big problems often **underdetermined** # weights > # samples

Stochastic Gradient Descent finds weights that generalize well

Backpropagation = Automatic Differentiation: Reverse Mode

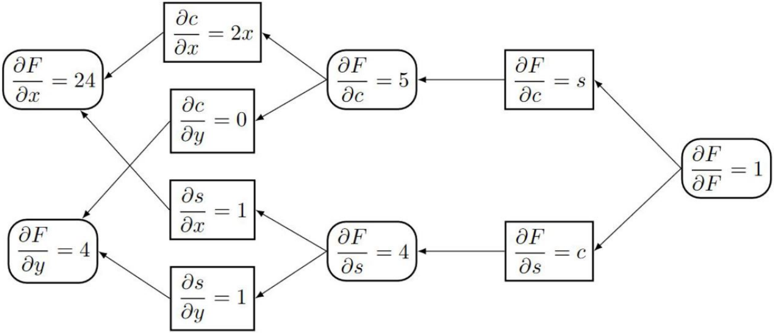
Nick Higham referred to AD in the Dec 2017 SIAM News

The derivatives of F are computed in parallel with F itself

Differentiate every step in the computational graph

This produces the chain rule for dF/dx

Backpropagation: Derivatives of $x^2(x+y)$ at $x=2, y=3$



$$\frac{\partial F}{\partial x} = 24$$

$$\frac{\partial F}{\partial y} = 4$$

$$\frac{\partial c}{\partial x} = 2x$$

$$\frac{\partial c}{\partial y} = 0$$

$$\frac{\partial s}{\partial x} = 1$$

$$\frac{\partial s}{\partial y} = 1$$

$$\frac{\partial F}{\partial c} = 5$$

$$\frac{\partial F}{\partial s} = 4$$

$$\frac{\partial F}{\partial c} = s$$

$$\frac{\partial F}{\partial s} = c$$

$$\frac{\partial F}{\partial F} = 1$$

Stochastic Gradient Descent

Update $\mathbf{x}$ using one random sample $\mathbf{v}$ (or a minibatch)

Simple methods start well (**semi-convergence**)

Just stop them early: Noise is not a disaster to correct

Kaczmarz chooses $\mathbf{x}_{k+1}$ to solve equation $i(k)$ in $Ax = b$

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{\mathbf{b}_i - \mathbf{a}_i^T \mathbf{x}_k}{\|\mathbf{a}_i\|^2}\,\mathbf{a}_i}$$

Norm-squared sampling   Linear algebra + probability

Choose equation i with probability proportional to $\|\mathbf{a}_i\|^2$

Randomized multiplication AB of very large matrices

Choose column of A / row of B with probability $\approx \|\mathbf{a}_i\| \, \|\mathbf{b}_i\|$

Columns/rows stay sparse/positive/meaningful

A revolution in linear algebra for large matrices

Matrix approximation $A \approx CMR$ $\mathbf{M} =$ **mixing matrix**

Randomized Numerical Linear Algebra

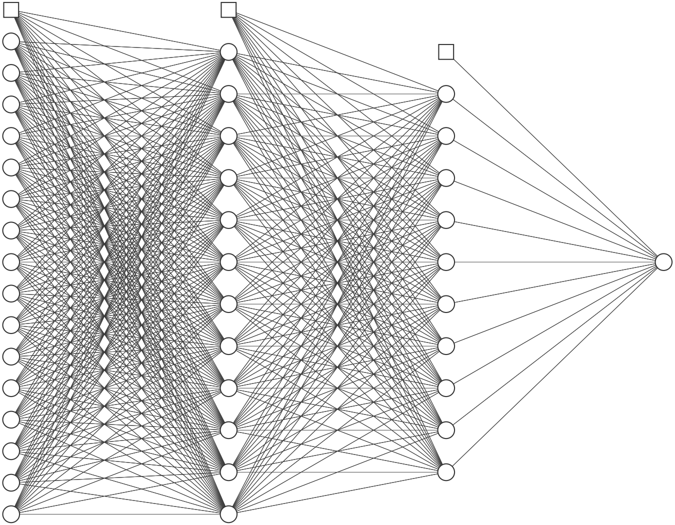
For very large matrices, randomization has brought a revolution

Example: Multiply AB with column-row sampling $(AS)(S^TB)$

$$AS = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \begin{bmatrix} s_{11} & 0 \\ 0 & 0 \\ 0 & s_{32} \end{bmatrix} = \begin{bmatrix} s_{11}\mathbf{a}_1 & s_{32}\mathbf{a}_3 \end{bmatrix}$$

Norm-squared sampling Choose columns of A rows of B with probabilities proportional to $\|a_i\| \|b_i^T\|$

This choice minimizes the **sampling variance**



Input Layer ∈ $\mathbb{R}^{16}$

Hidden Layer ∈ $\mathbb{R}^{12}$

Hidden Layer ∈ $\mathbb{R}^{10}$

Output Layer ∈ $\mathbb{R}^{1}$