Lecture 2 Neural Networks

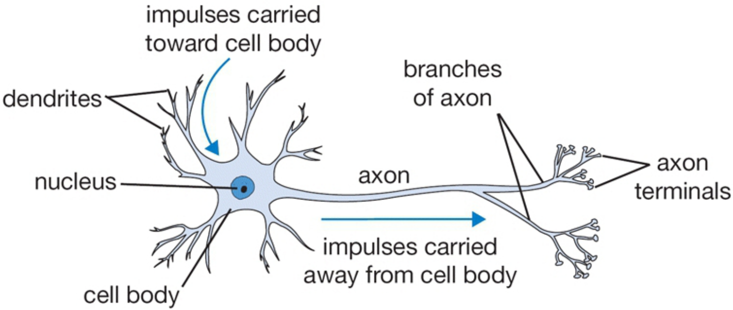
Fei Tan

Department of Economics
Chaifetz School of Business
Saint Louis University

E6930 Introduction to Neural Networks

July 1, 2024

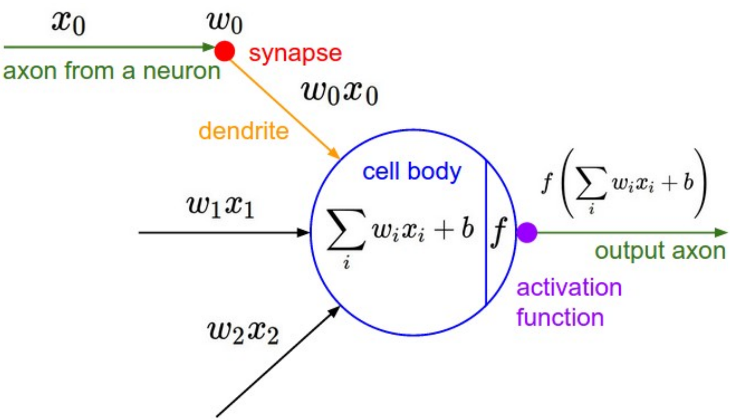
Biological Neuron



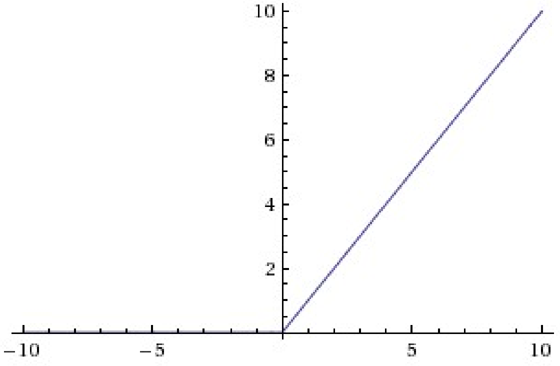
The Road Ahead...

1. Basic Architecture
2. Pre-training Processing
3. Model Training

Mathematical Neuron

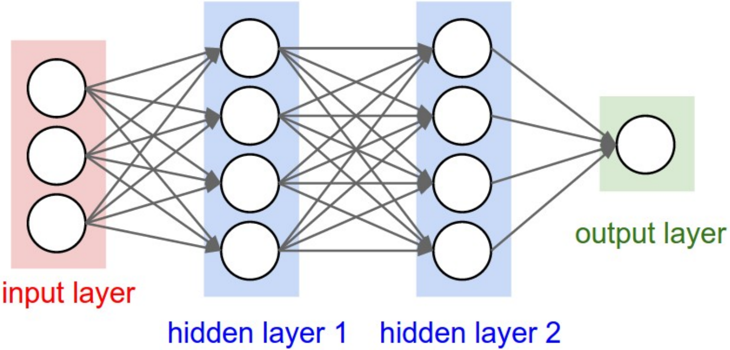


Activation Functions



- Rectified linear unit (ReLU): $f(x) = \max(0, x)$
- Leaky ReLU: $f(x) = \max(0.01x, x)$ (avoid dying ReLU)
- Historically, sigmoid or tanh (vanishing/exploding gradient)

Multilayer Perceptron

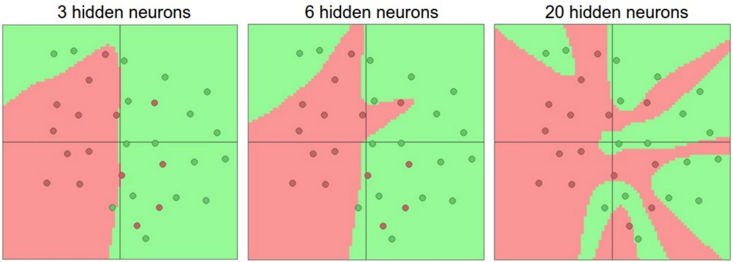


- 3-layer fully connected neural network
- not counting input; no activation in output layer
- Why go 'deep'? (representational power)

Forward Pass

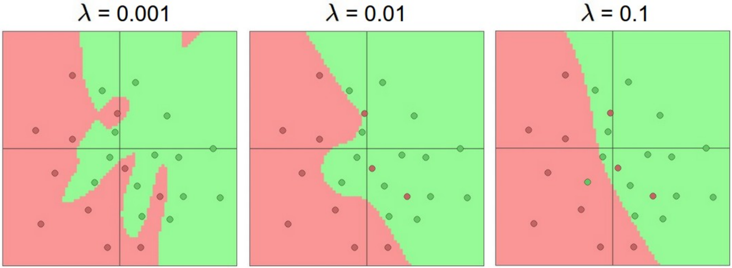
```
import numpy as np

# Forward pass
f = lambda x: 1.0/(1.0 + np.exp(-x)) # sigmoid
    activation
x = np.random.randn(3, 1)   # random input vector
    (3x1)
h1 = f(np.dot(W1, x) + b1)  # first hidden layer
    activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # second hidden layer
    activations (4x1)
out = np.dot(W3, h2) + b3   # output neuron (1x1)
```

Overfitting



- In-sample fitting vs. out-of-sample generalization
- How to control overfitting?

Regularization

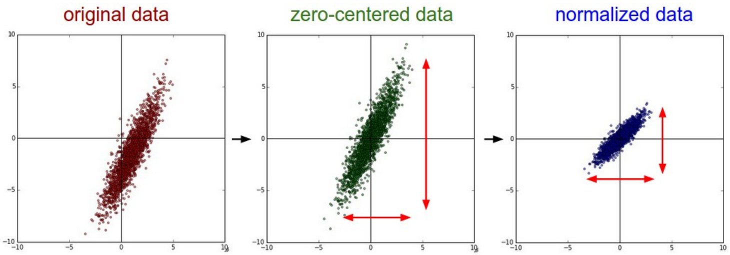


- L^1, L^2, max-norm, and dropout to control overfitting
- Regularization/prior from structural models

The Road Ahead...

1. Basic Architecture
2. Pre-training Processing
3. Model Training

Data Standardization



- Mean subtraction + normalization by s.d. (or principal component, whitening)
- *Only* on training set, then on validation/test set

Weight Initialization

- Avoid all-zero initialization, use small random numbers to break symmetry

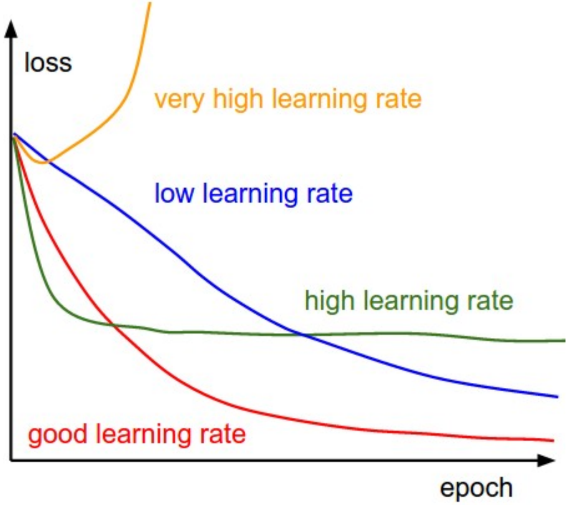
```
W = 0.01 * np.random.randn(D,H)         # randn
    samples from standard Gaussian
b = np.zeros((1,H))                     # bias
w = np.random.randn(n) / sqrt(n)        #
    normalization; n = number of inputs
w = np.random.randn(n) * sqrt(2.0/n)    # ReLU
    neurons
```

- Batch normalization to stabilize learning

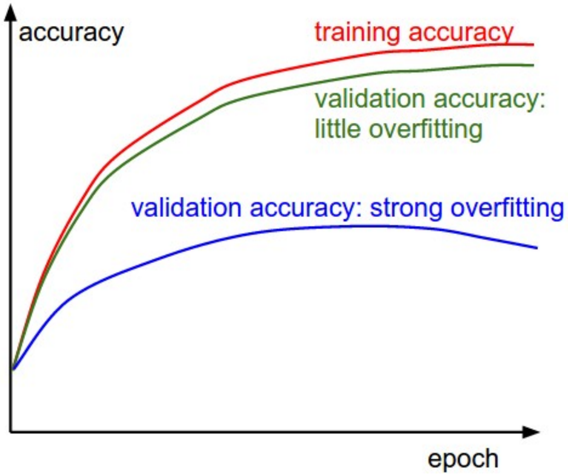
The Road Ahead...

1. Basic Architecture
2. Pre-training Processing
3. Model Training

Diagnosis: Loss Function



Diagnosis: Accuracy



Other Diagnoses

- Ratio of update to weight magnitudes ($\sim 10^{-3}$)

```
# Parameter vector W and gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW       # SGD update
update_scale = np.linalg.norm(update.ravel())
W += update
print update_scale / param_scale # about 1e-3
```

- Activation/gradient distribution per layer

Non-adaptive Learning

- Nesterov momentum update

```
v_prev = v                              # back up
v = mu * v - learning_rate * dx   # velocity
    update; mu = 0.9
x += -mu * v_prev + (1 + mu) * v # position
    update
```

- Anneal learning rate over time

- Newton's second-order method

$$x_{k+1} = x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

  - expensive to compute and inverse Hessian $\nabla^2 f(x)$
  - Quasi-Newton method, e.g. L-BFGS

Adaptive Learning

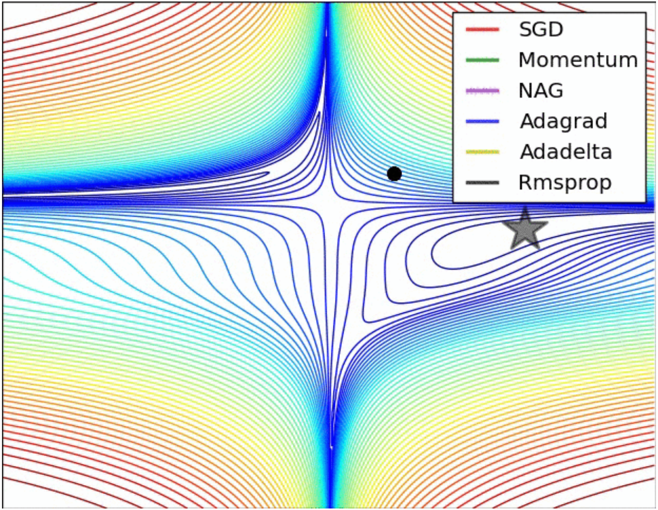
- Adagrad/RMSprop

```
cache += dx**2  # Adagrad
cache = decay_rate * cache + (1 - decay_rate)
    * dx**2      # RMSprop; decay_rate = 0.99
x += - learning_rate * dx / (np.sqrt(cache) +
    eps)
```

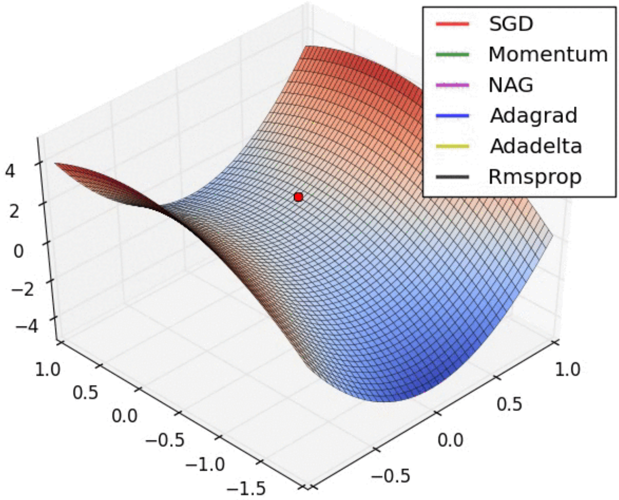
- Adam (recommended)

```
m = beta1*m + (1-beta1)*dx  # beta1 = 0.9
mt = m / (1-beta1**t)       # bias correction;
     t = iter counter
v = beta2*v + (1-beta2)*(dx**2) # beta2 =
    0.999
vt = v / (1-beta2**t)
x += -learning_rate * mt / (np.sqrt(vt) + eps)
```

Learning Dynamics



Learning Dynamics (Cont'd)



Miscellaneous Pointers

- Gradient check with small batch of data
- Prefer larger networks with proper regularization
- For continuous and multivariate outcome space, *discretize* and train *independently* for each attribute
- Random search for good hyperparameters
- Form model ensemble for extra performance

References

- cs231n.stanford.edu – CS231n: Deep Learning for Computer Vision, by Stanford University
- Goodfellow, Bengio & Courville (2016), "Deep Learning", MIT Press
- Ioffe & Szegedy (2015), "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", arXiv:1502.03167
- Goldberg (1991), "What Every Computer Scientist Should Know About Floating-Point Arithmetic", *Computer Surveys*