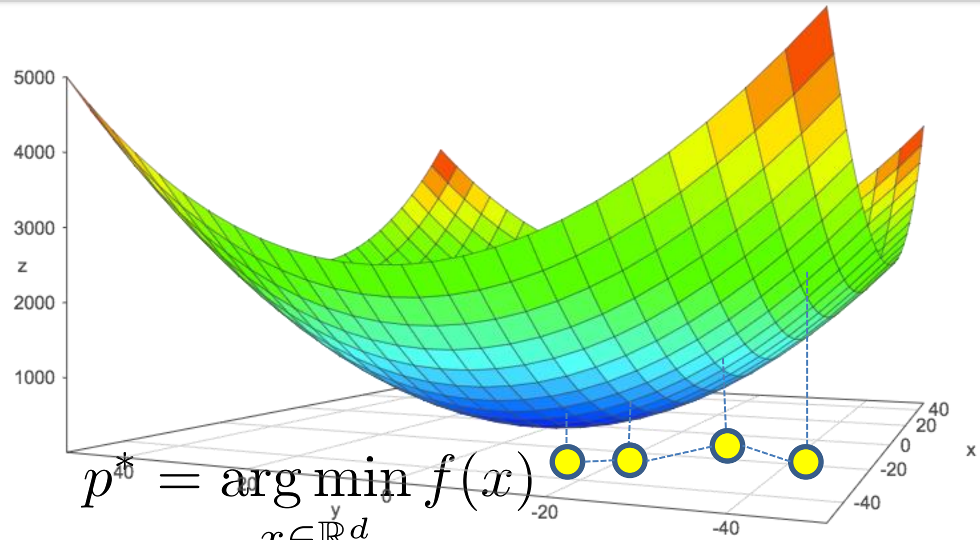
Northeastern

EECE5698
Parallel Processing for Data Analytics

Lecture 13: Stochastic Gradient Descent

Gradient Descent



$$p^* = \arg\min_{x \in \mathbb{R}^d} f(x)$$

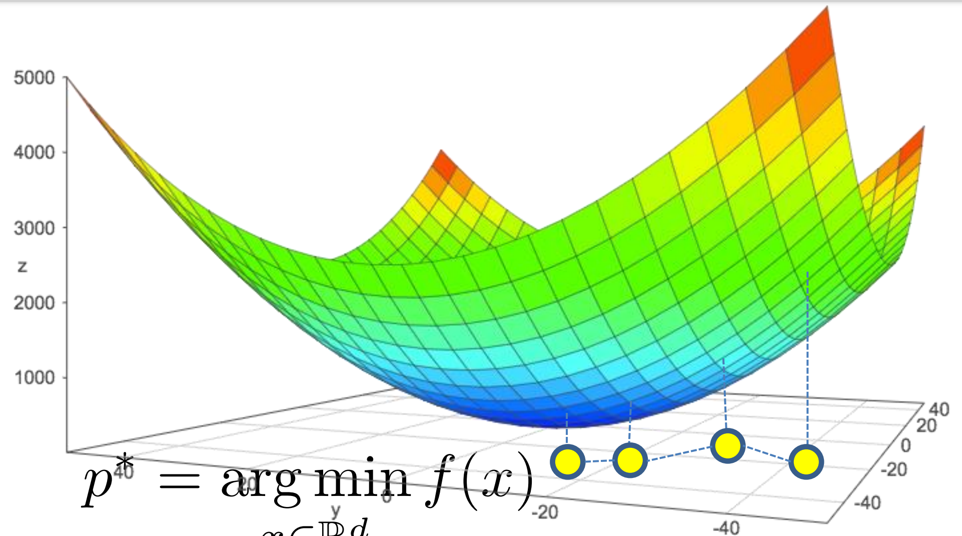
---

given a starting point $x \in \textbf{dom}\, f$.
repeat

1. $\Delta x := -\nabla f(x)$.
2. *Line search.* Choose step size t via exact or backtracking line search.
3. *Update.* $x := x + t\Delta x$.

until $\|\nabla f(x)\|_2 \leq \epsilon$

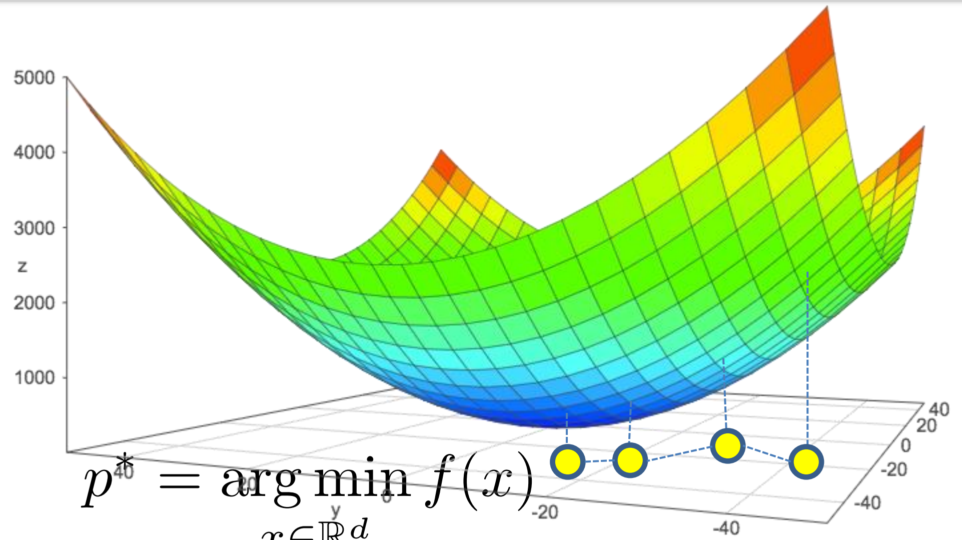
Gradient Descent



$$p^* = \arg\min_{x \in \mathbb{R}^d} f(x)$$

$$x^{k+1} = x^k - \gamma^k \cdot \nabla F(x^k) \qquad k = 0, 1, 2, \ldots$$

Stochastic Gradient Descent



$$p^* = \arg\min_{x \in \mathbb{R}^d} f(x)$$

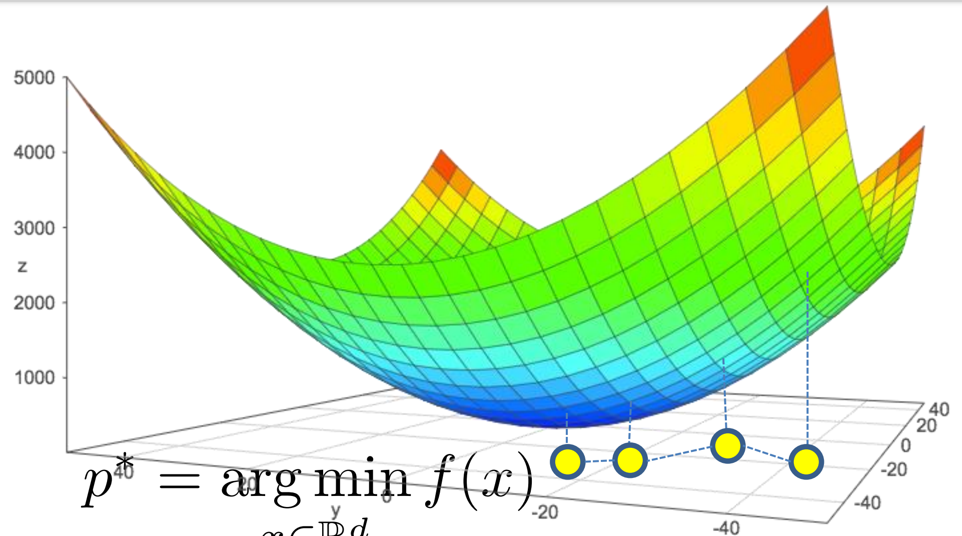
$$x^{k+1} = x^k - \gamma^k \cdot g(x^k, \omega^k) \qquad k = 0, 1, 2, \ldots$$

estimate of gradient — random i.i.d variables

$$\mathbb{E}[g(x^k, \omega^k)] = \nabla f(x^k) \qquad \mathbb{E}[\|g(x^k, \omega^k) - \nabla f(x^k)\|_2^2] \leq M < \infty$$

Think of ω as a "source of randomness"

Stochastic Gradient Descent

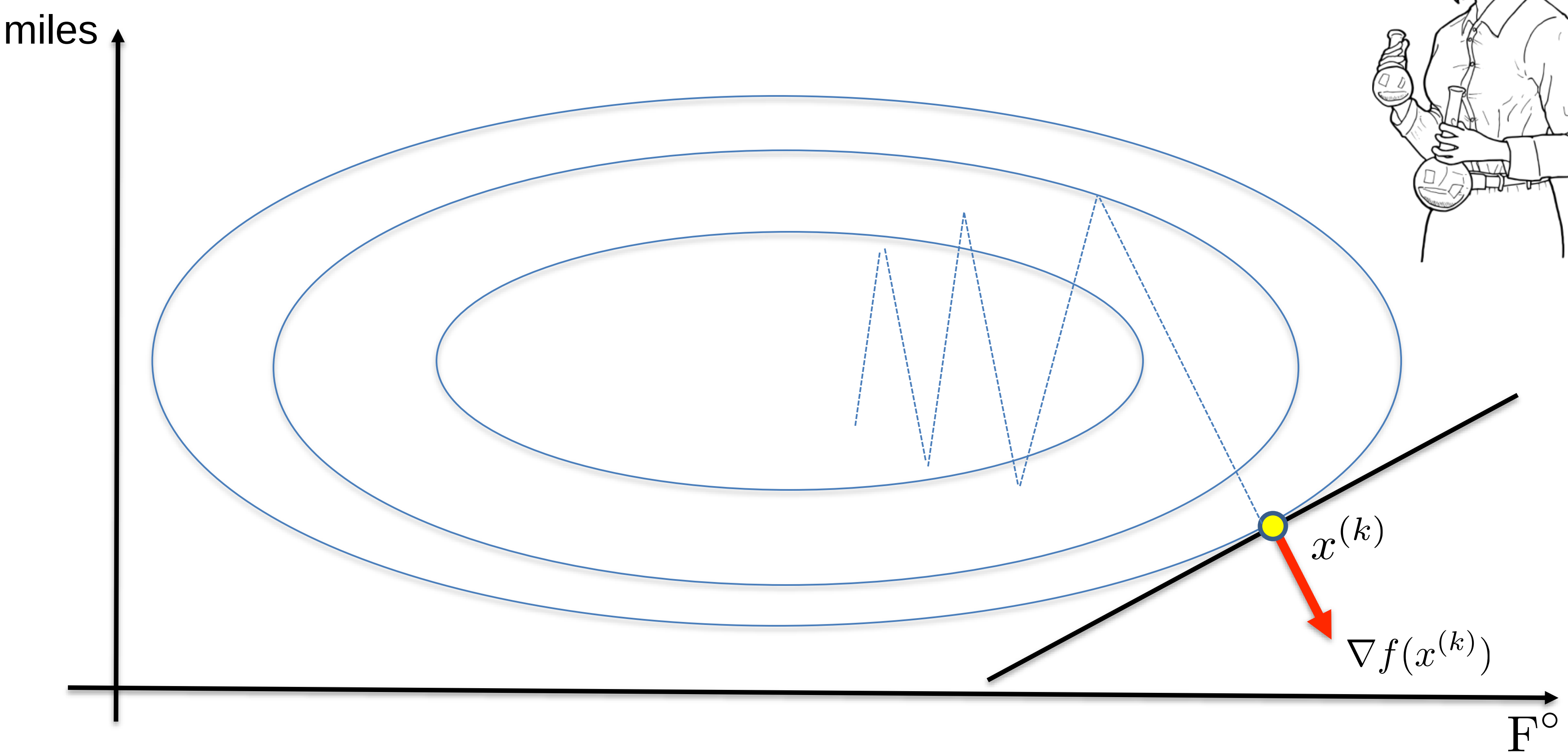


$$p^* = \arg\min_{x \in \mathbb{R}^d} f(x)$$

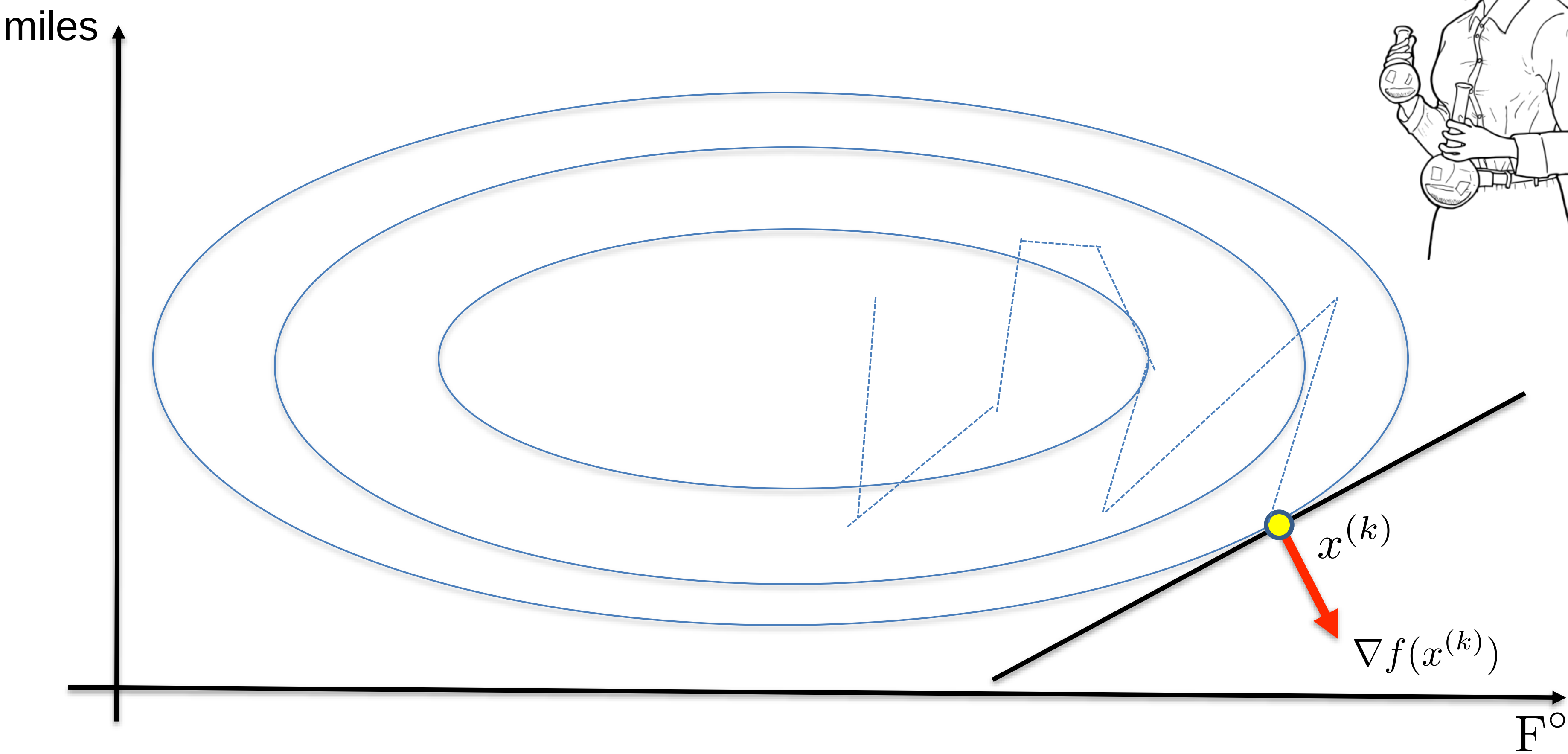
$$x^{k+1} = x^k - \gamma^k \cdot (\nabla F(x^k) + \varepsilon^k) \qquad k = 0, 1, 2, \dots$$

zero mean, finite variance noise

Gradient Descent



Stochastic Approximation

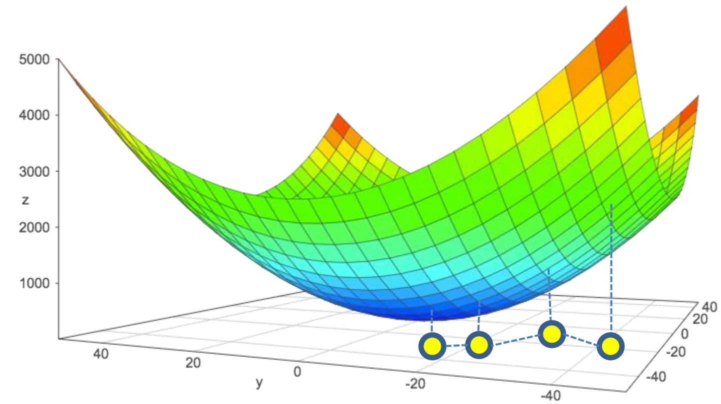


Convergence

i.i.d.

$$x^{k+1} = x^k - \gamma^k \cdot g(x^k, \omega^k) \qquad k = 0, 1, 2, \ldots$$

$$\mathbb{E}[g(x^k, \omega^k)] = \nabla f(x^k) \qquad \texttt{Var}[g(x^k, \omega^k)] \leq M < \infty$$



Theorem: Suppose that $f : \mathbb{R}^d \to \mathbb{R}$ is convex, twice differentiable, and γ^k satisfies

$$\lim_{k\to\infty} \gamma^k = 0, \qquad \lim_{k\to\infty} \sum_{\ell=1}^{k} \gamma^k = \infty \qquad \lim_{k\to\infty} \sum_{\ell=1}^{k} (\gamma^k)^2 < \infty$$

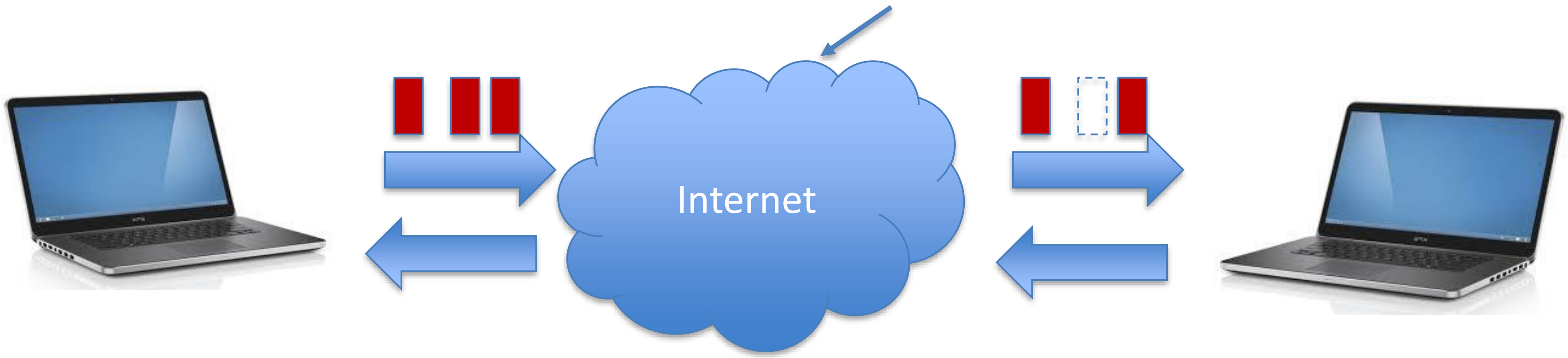
Then,

$$\lim_{k\to\infty} x^k = \arg\min_{x\in\mathbb{R}^d} f(x)$$

Application: Function not known, but can be sampled

Send packets at rate λ

packets dropped with probability $p(\lambda)$



Objective: Maximize throughput $\lambda(1-p(\lambda))$

Application: Function not known, but can be sampled

Probability of jackpot:

p_1

p_2

p_3

…

p_d

Probability distribution over slot machines $x \in [0,1]^d$

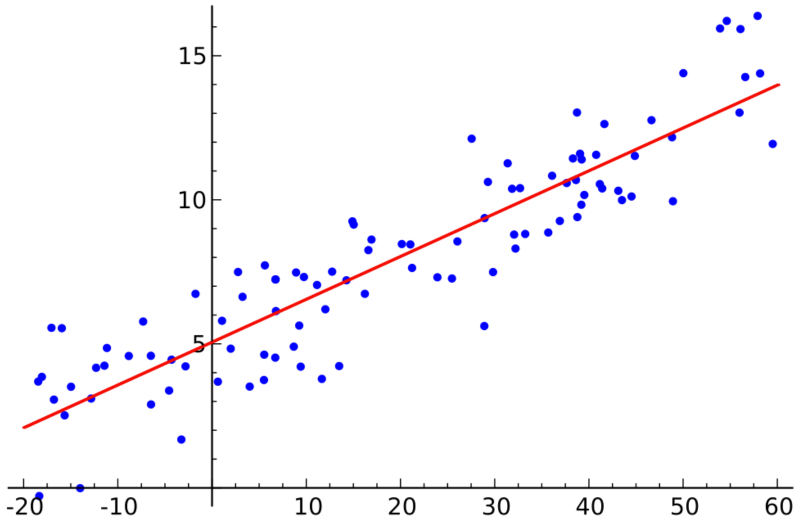
Goal: Maximize expected payoff $x^\top p$

SGD and Parallelism

$$\min_{\beta \in \mathbb{R}^d} F(\beta)$$

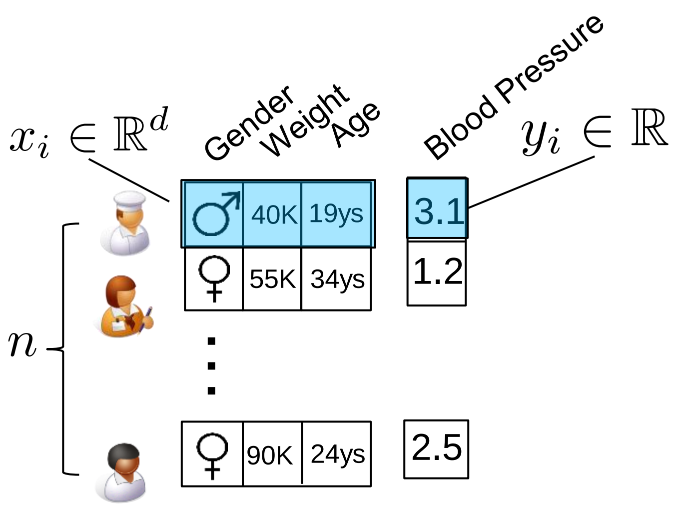
$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$

- If ℓ is convex, so is $F(\beta)$



Parallel Computation and SGD

$$F(\beta) = \sum_{i=1}^{n} \ell(\beta; x_i, y_i)$$

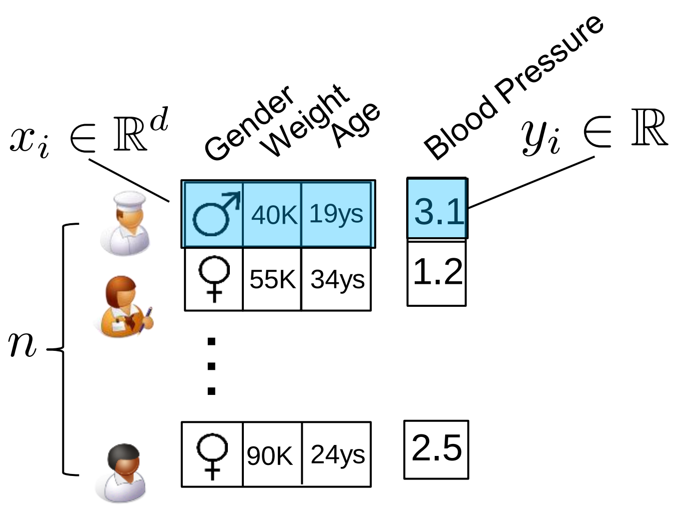



```
rdd = [(x1,y1),
       (x2,y2),
        …
       (xn,yn)]

beta = np.array([0.1,0.4,-2.0])

rdd.map( lambda (x,y):
              loss(beta,x,y))\
       .reduce(add)
```


Northeastern

Parallel Computation

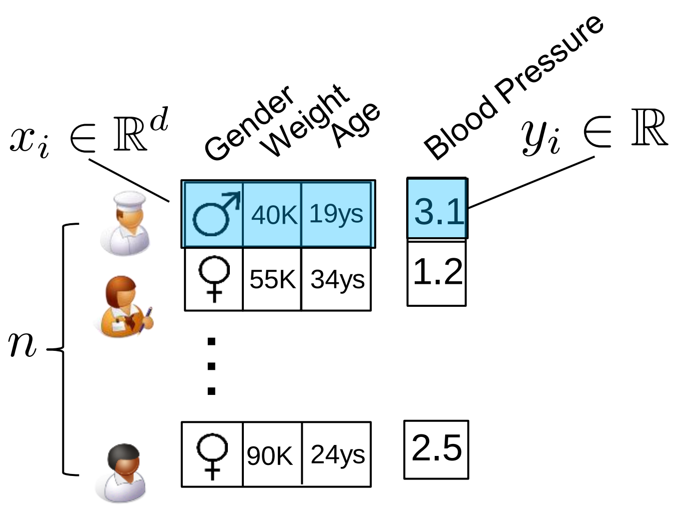
$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_{\beta} \ell(\beta; x_i, y_i)$$




```
rdd = [(x1,y1),
       (x2,y2),
        …
       (xn,yn)]

beta = np.array([0.1,0.4,-2.0])

rdd.map( lambda (x,y):
              gradLoss(beta,x,y))\
     .reduce(add)
```

Parallel Computation: Subsampling!!!!

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i)$$




```
rdd = [(x1,y1),
       (x2,y2),
        …
       (xn,yn)]

beta = np.array([0.1,0.4,-2.0])

sampledRDD=rdd.sample(
            withReplacement=False,
            fraction =0.5 )

sampledRDD.map( lambda (x,y):
                 gradLoss(beta,x,y))\
          .reduce(add)
```

Subsampling as SGD

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i) \qquad \widehat{\nabla F(\beta)} = \sum_{i \in \texttt{Sample}} \nabla_\beta \ell(\beta; x_i, y_i)$$

Let $$\mathbf{1}_{i \in \texttt{Sample}} = \begin{cases} 1, & \text{if } i \in \texttt{Sample} \\ 0, & \text{o.w.} \end{cases}$$

Assume that sampling probability is

$$P\left(\mathbf{1}_{i \in \texttt{Sample}} = 1\right) = p$$

Subsampling as SGD

$$\nabla F(\beta) = \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i) \qquad \widehat{\nabla F(\beta)} = \sum_{i \in \texttt{Sample}} \nabla_\beta \ell(\beta; x_i, y_i)$$

$$\begin{aligned}
\mathbb{E}[\widehat{\nabla F(\beta)}] &= \mathbb{E}\left[\sum_{i \in \texttt{Sample}} \nabla_\beta \ell(\beta; x_i, y_i)\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{n} \mathbf{1}_{i \in \texttt{Sample}} \nabla_\beta \ell(\beta; x_i, y_i)\right] \\
&= \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i) \mathbb{E}[\mathbf{1}_{i \in \texttt{Sample}}] \\
&= \sum_{i=1}^{n} \nabla_\beta \ell(\beta; x_i, y_i) p = p \nabla F(\beta)
\end{aligned}$$

SGD and parallelism

$$\beta^{k+1} = \beta^k - \gamma_k(p\nabla F(\beta^k) + \varepsilon^k)$$

$$\mathbb{E}[\widehat{\nabla F(\beta^k)}] = p\nabla F(\beta^k)$$

- For appropriate step size, converges to global minimizer w.p. 1!

- Tradeoff between
  - computation per iteration and
  - total number of iterations

- Same applies if:
  - Select point i w.p. p
  - Select set Sample uniformly at random so that |Sample|=pn

SGD in the extreme

$$\beta^{k+1} = \beta^k - \gamma_k (p\nabla F(\beta^k) + \varepsilon^k)$$

- Pick only 1 point per iteration!!
    - Small computation per iteration
    - High variance

Northeastern

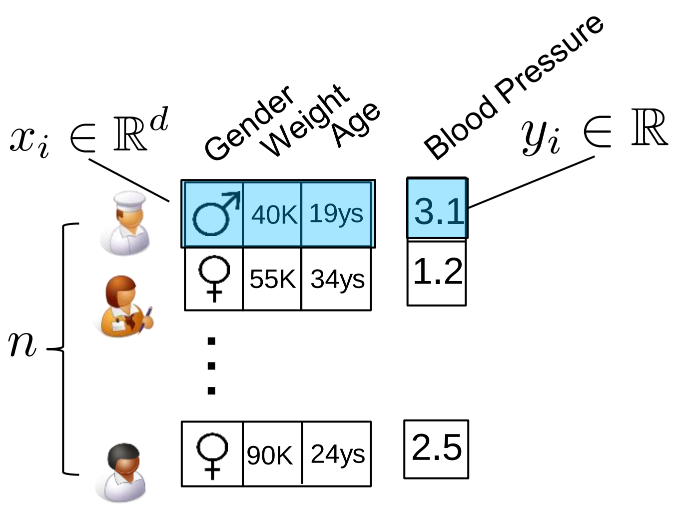
SGD in the extreme

$$\beta^{k+1} = \beta^k - \gamma_k \nabla \ell(\beta^k; x_i, y_i)$$

- Pick only 1 point per iteration!!
  - Small computation per iteration
  - High variance

- Modification: do not select it randomly, just iterate over dataset
  - **This is what people call "SGD"**
  - Akin to perceptron

SGD in the extreme



datalist = [(x1,y1),
(x2,y2),
…
(xn,yn)]

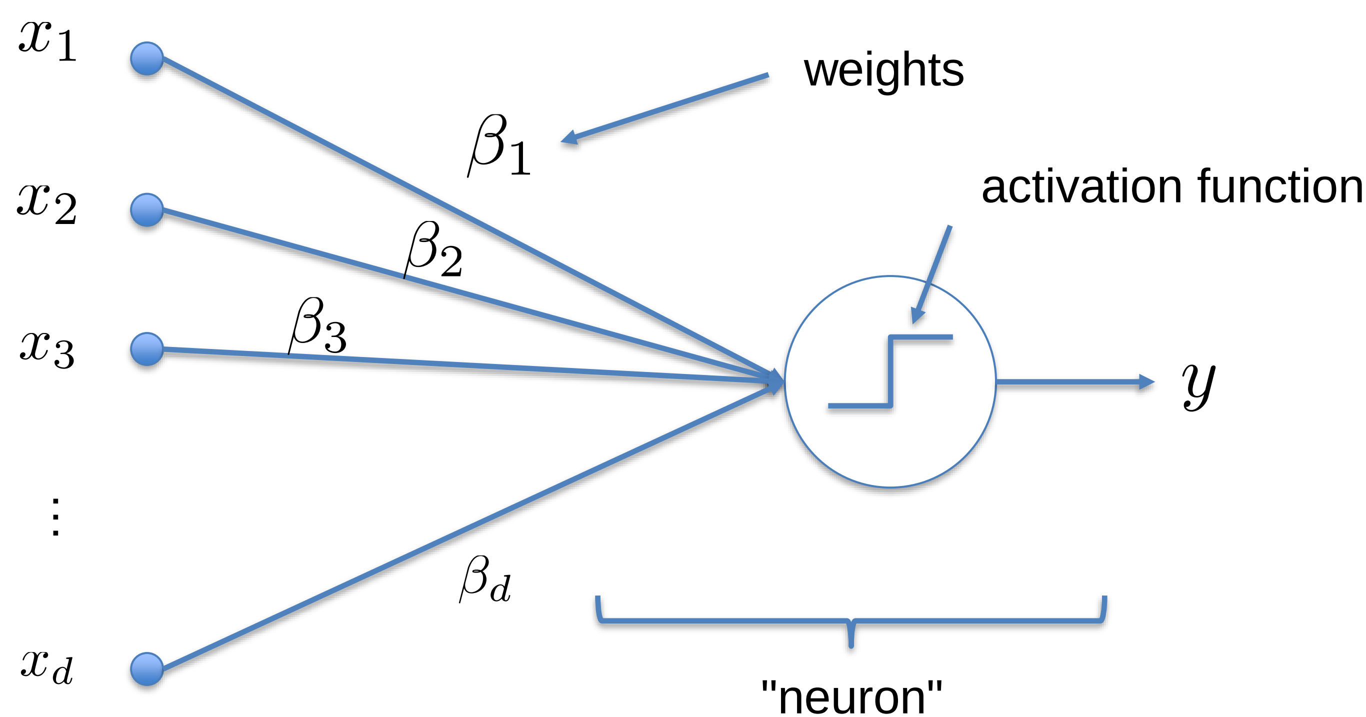
for epoch in range(10):
    for (x_i,y_i) in datalist:
        beta = beta – gamma *gradLoss(beta,x_i,y_i)

EECE5698
Parallel Processing for Data Analytics

Lecture 15: Deep Learning

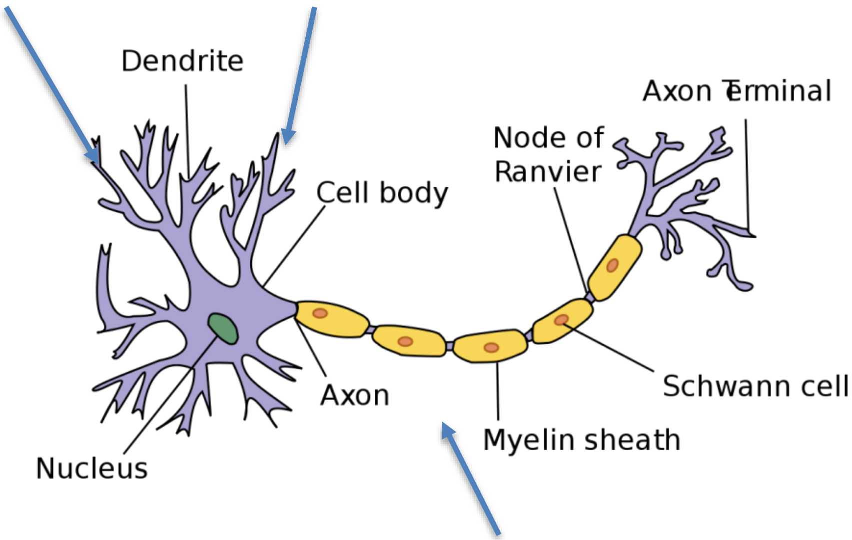
Linear Classifier

$$
\begin{aligned}
y &= \mathrm{sign}(\beta^\top x) \\
&= \mathrm{sign}(\beta_1 x_1 + \beta_2 x_2 + \ldots \beta_d x_d)
\end{aligned}
$$



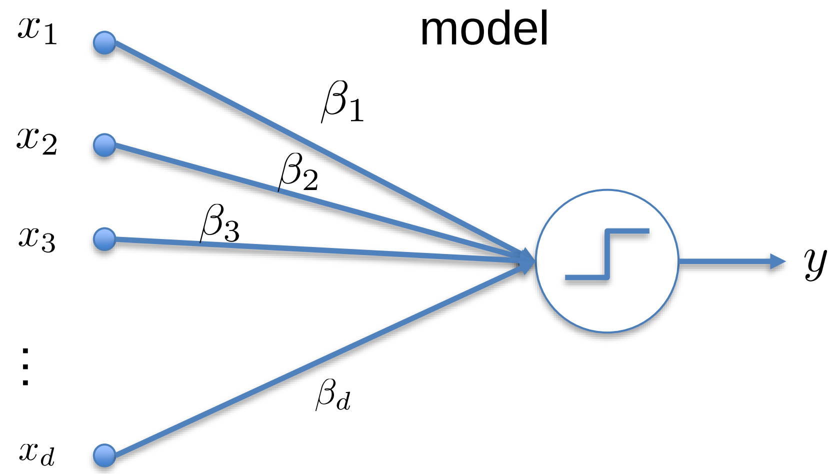
Neurons in biology

Electrical signals come in from other neurons

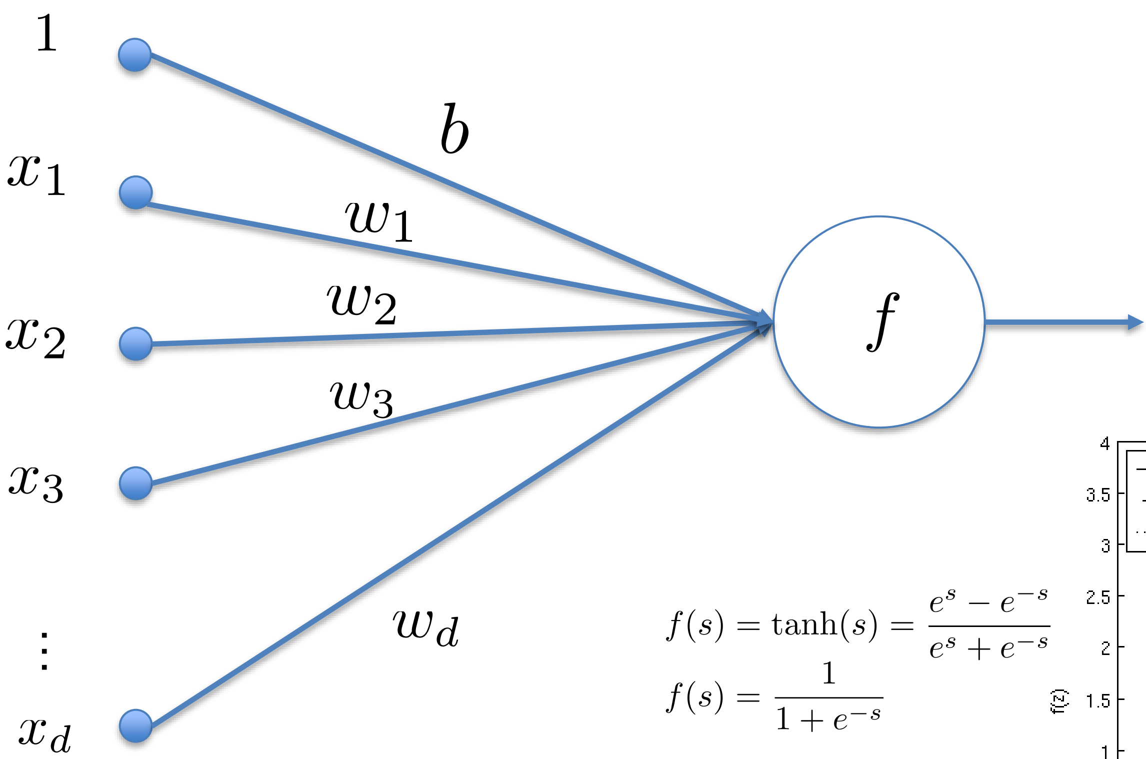


output to other neurons

mathematical model



General Form of a Neuron

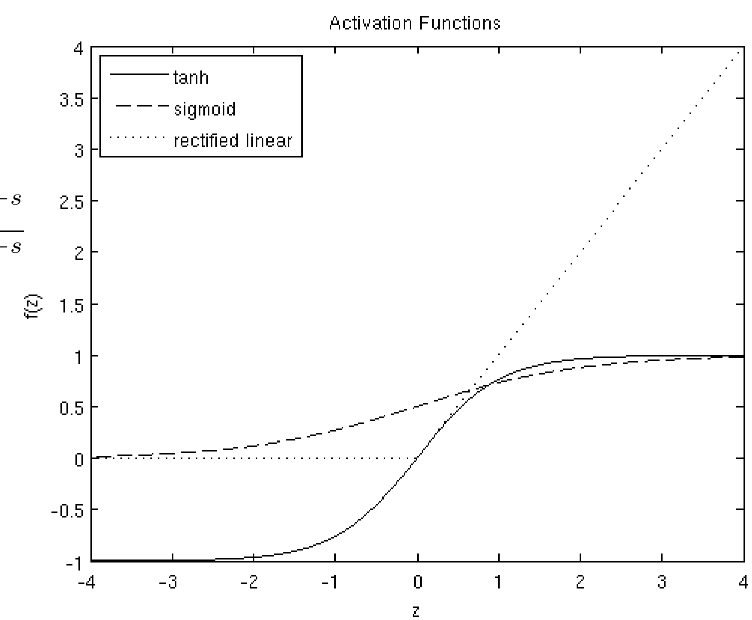


$$y = f(w^\top x + b)$$

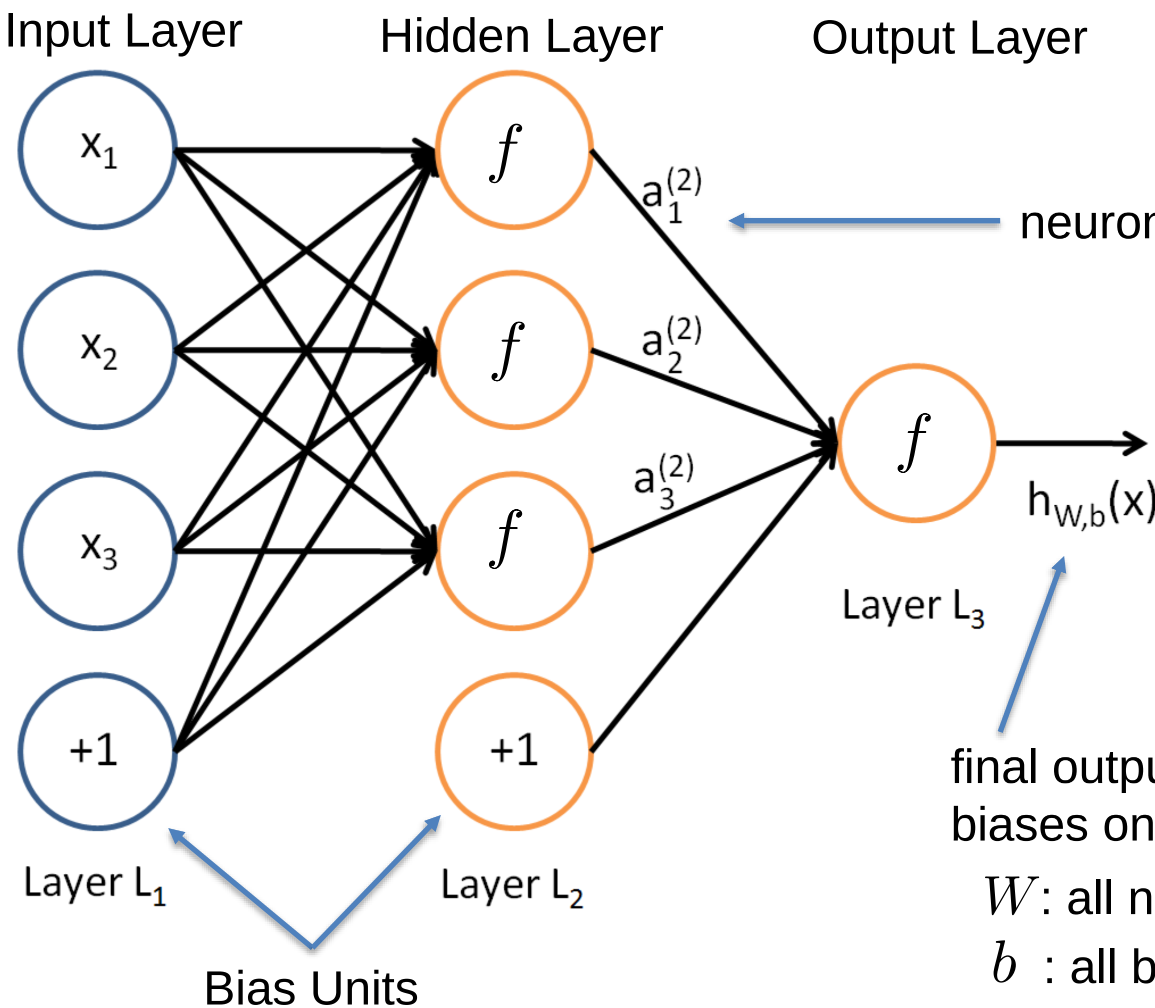
$$f(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

$$f(s) = \frac{1}{1 + e^{-s}}$$

$$f(s) = \max\{0, s\}$$



A 2-Layer Neural Network



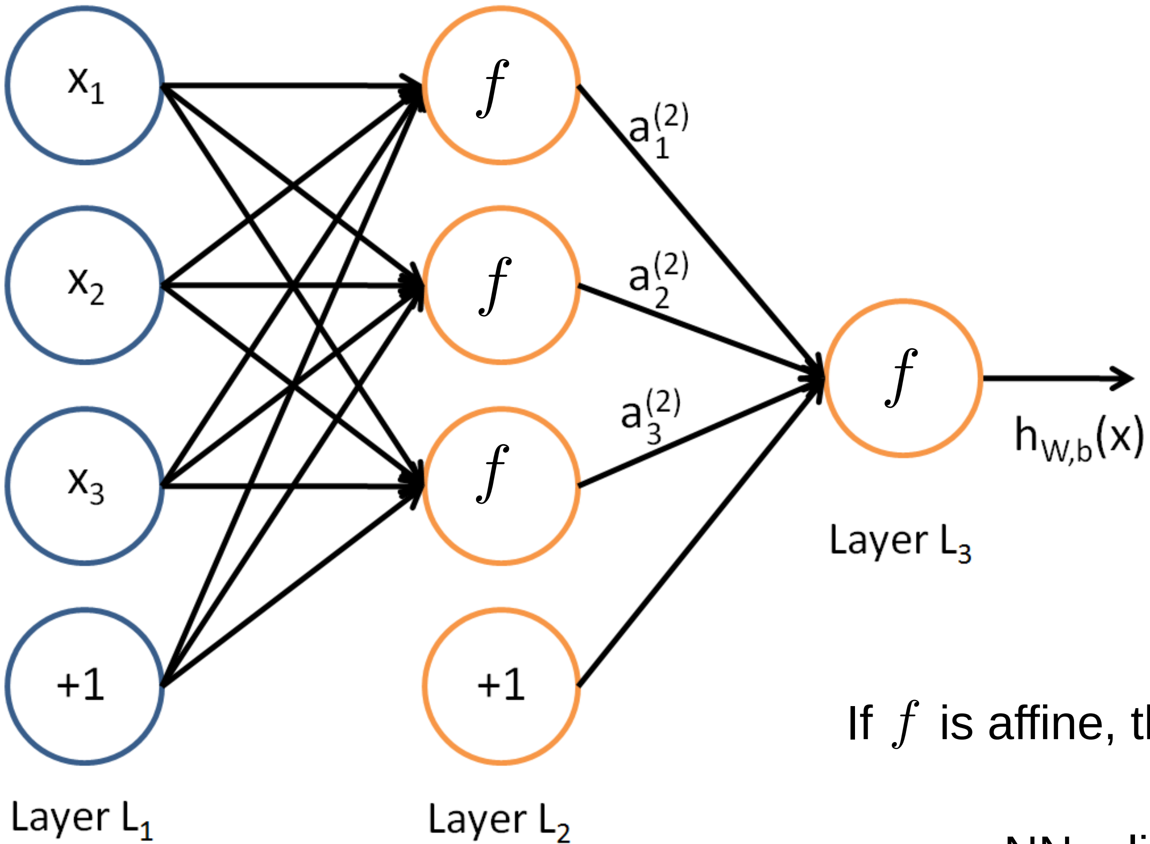
neuron activation/ouput value

final output depends on weights and biases on all edges

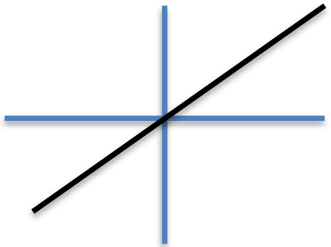
W: all non bias weights

b : all bias weights

Modeling Non-Linearities



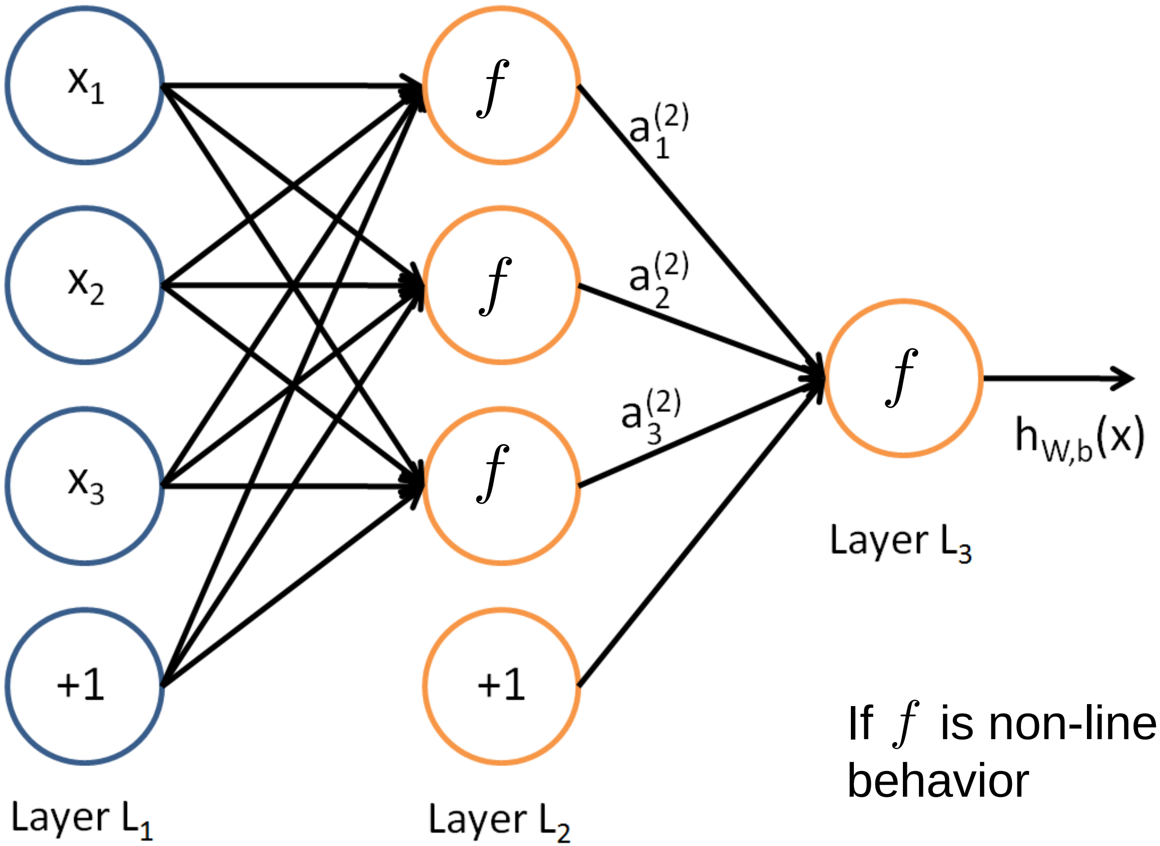
$$f(s) = s$$

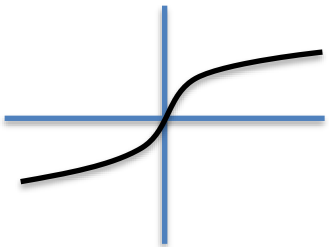


If f is affine, then $h_{W,b}$ is affine.

NN = linear regression

Modeling Non-Linearities



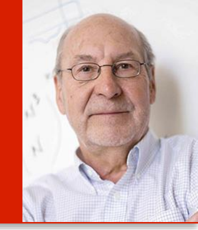


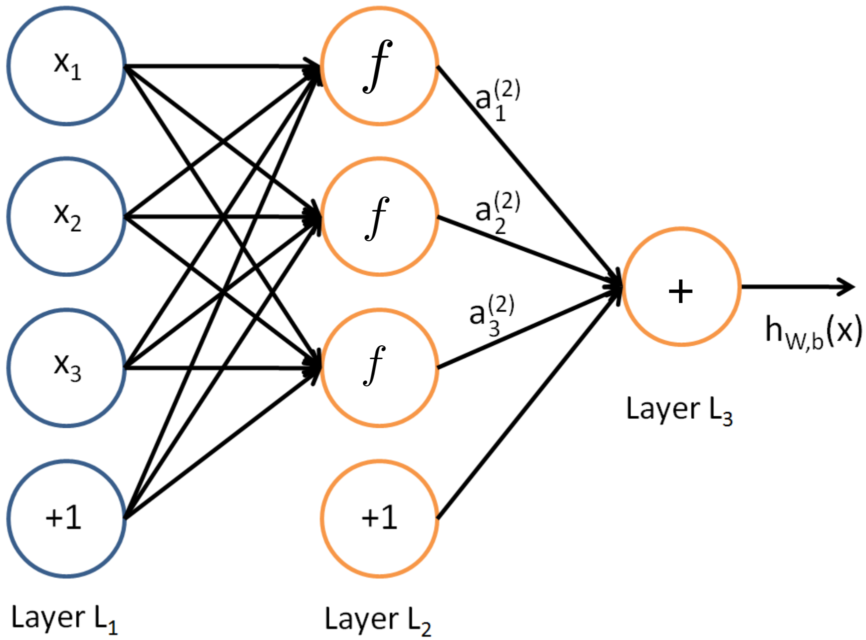
If f is non-linear, $h_{W,b}$ models non-linear behavior

Q: How expressive is it?

Universal Approximation Theorem

George Cybenko





$$f(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

2-Layer NNs can compute continuous functions!

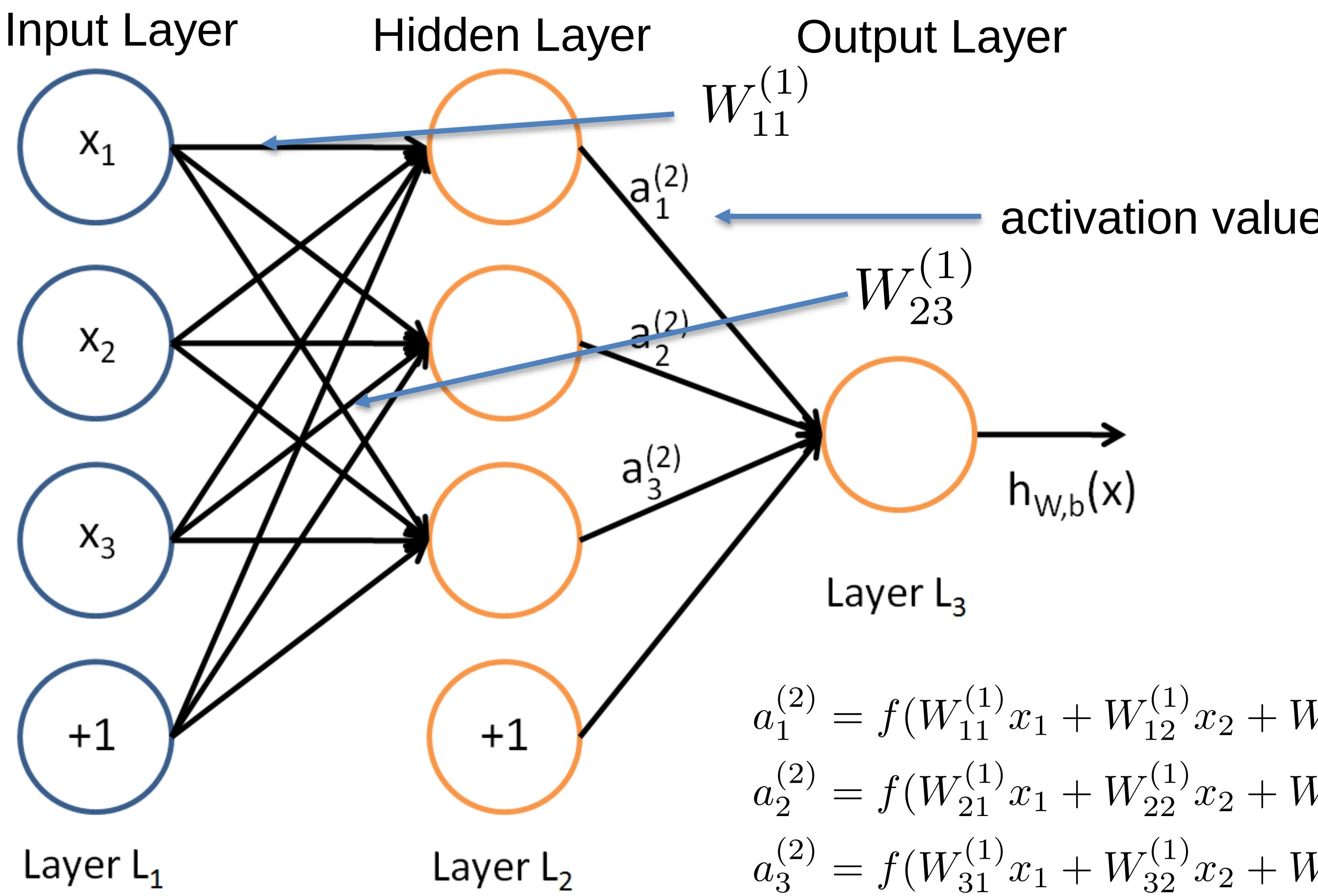
For any continuous function $f_0 : [0,1]^d \to \mathbb{R}$ and any $\varepsilon > 0$, there exists a 2-layer NN $h_{W,b} : [0,1]^d \to \mathbb{R}$ such that:

$$|f_0(x) - h_{W,b}(x)| < \varepsilon$$

for all $x \in [0,1]^d$.

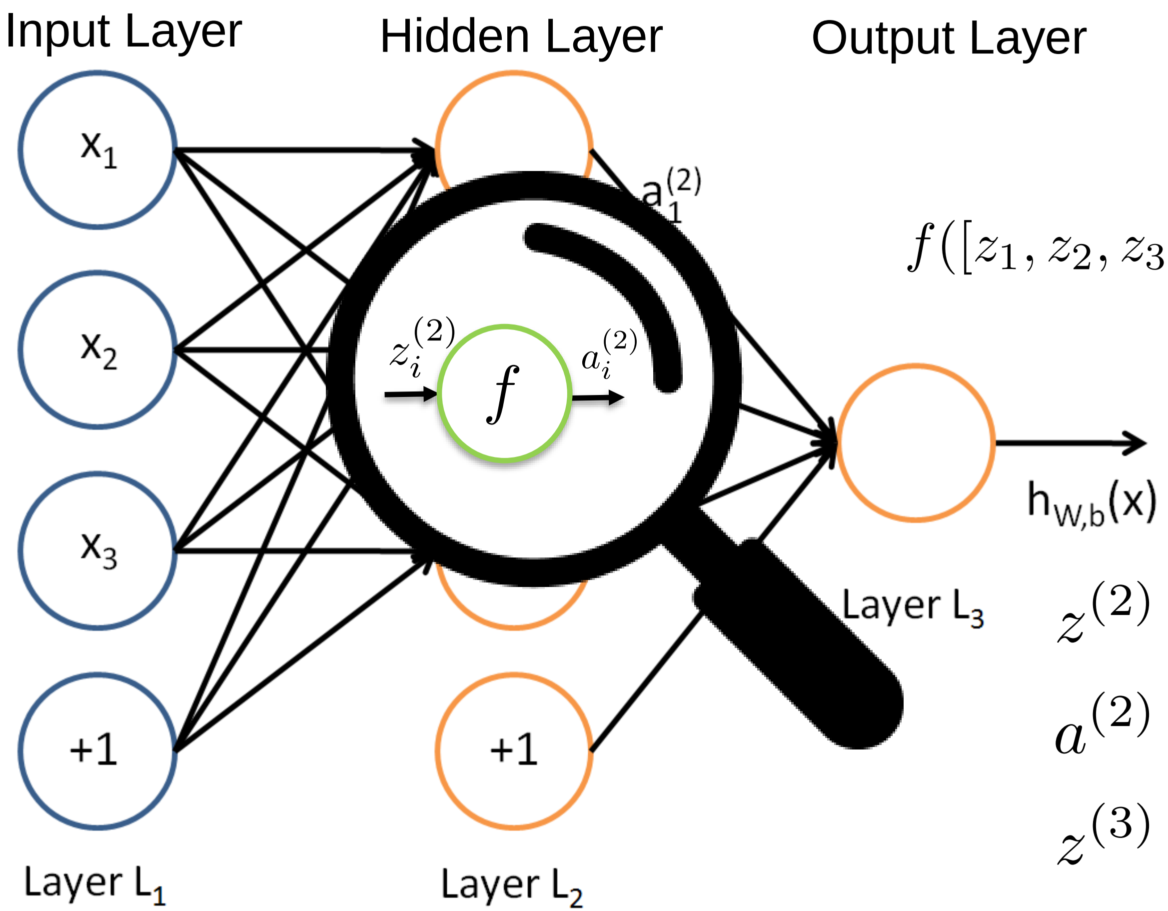
Northeastern

2-Layer Neural Network: Notation



$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

2-Layer Neural Network: Notation



$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

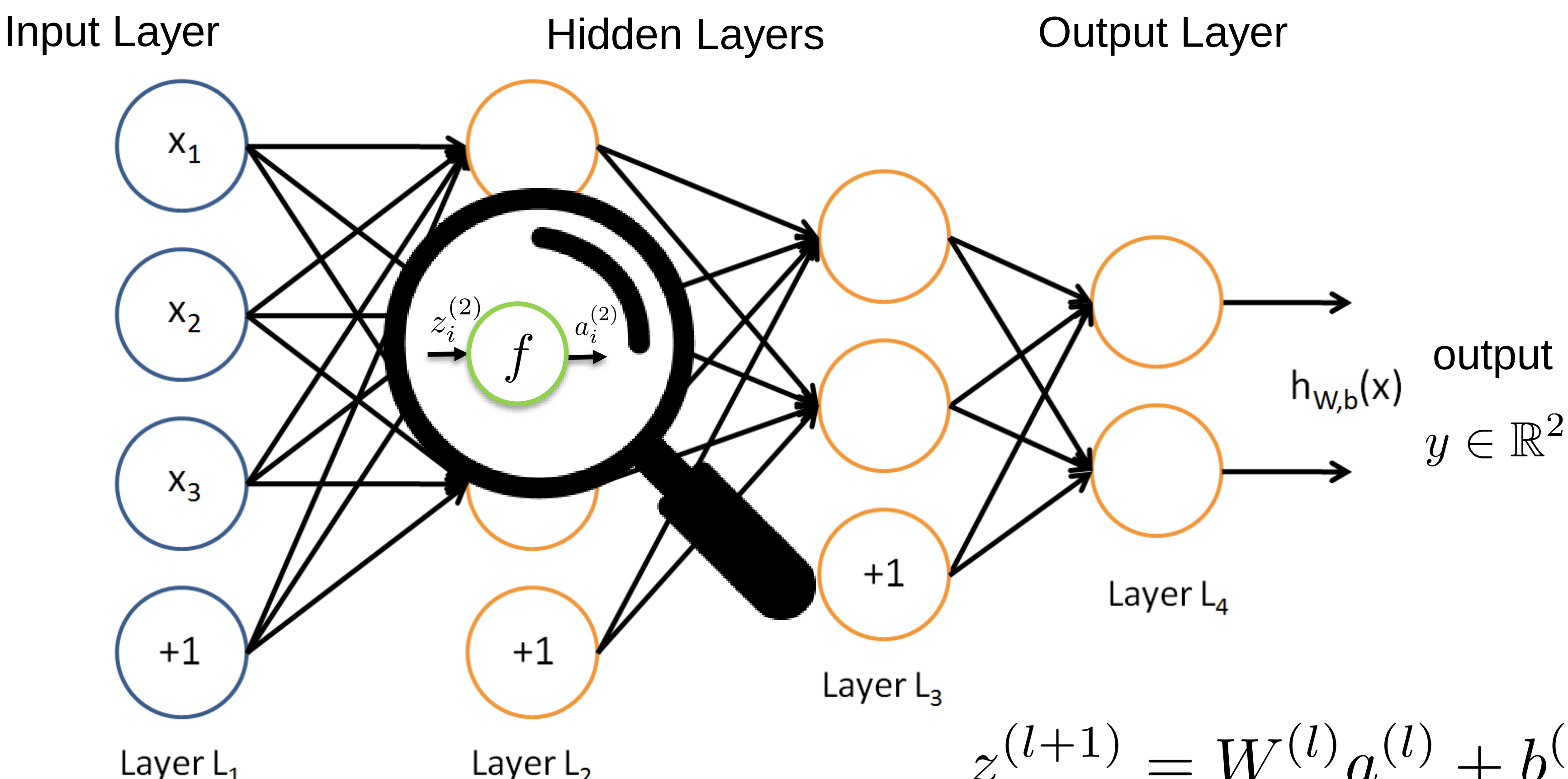
$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

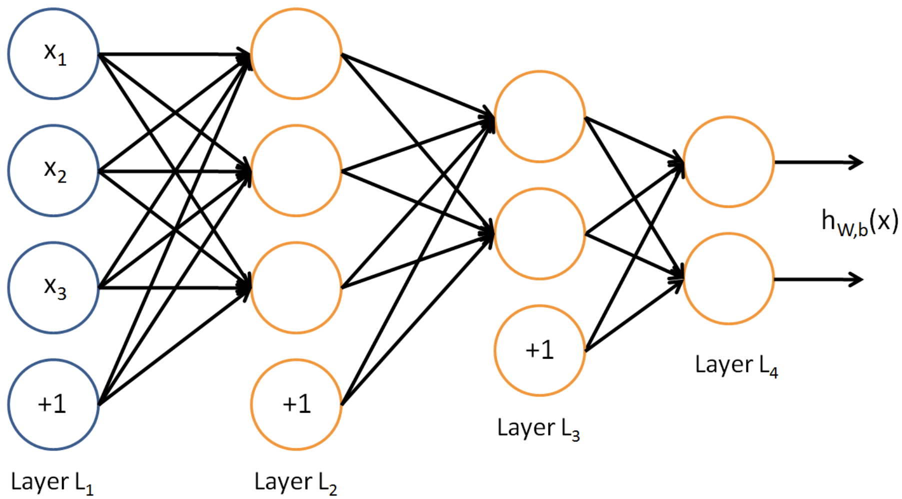
Deep Neural Networks



$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

Training Neural Networks



$y \in \mathbb{R}^2$

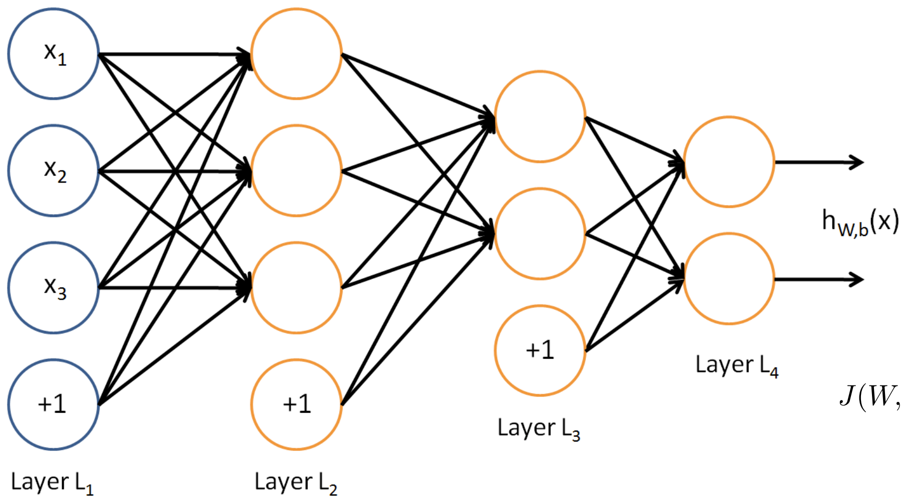
$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2 .$$

Minimize regularized loss over all datapoints $(x^{(i)}, y^{(i)})$ in dataset

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \end{aligned}$$

Gradient Descent



output

$y \in \mathbb{R}^2$

$$J(W,b;x,y) = \frac{1}{2}\left\|h_{W,b}(x) - y\right\|^2 .$$

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$
$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \gamma\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

$$b_i^{(l)} \leftarrow b_i^{(l)} - \gamma\frac{\partial}{\partial b_i^{(l)}}J(W,b)$$

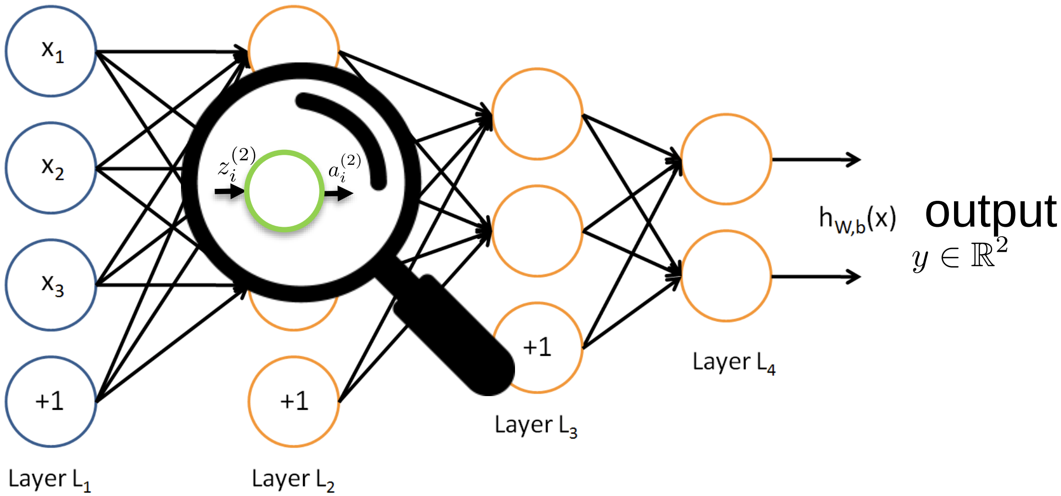
Can be computed **in parallel** or through **SGD** by computing:

$$\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b;x,y) \qquad \frac{\partial}{\partial b_i^{(l)}}J(W,b;x,y)$$

across datapoints.

Northeastern

Computing Gradients per Datapoint

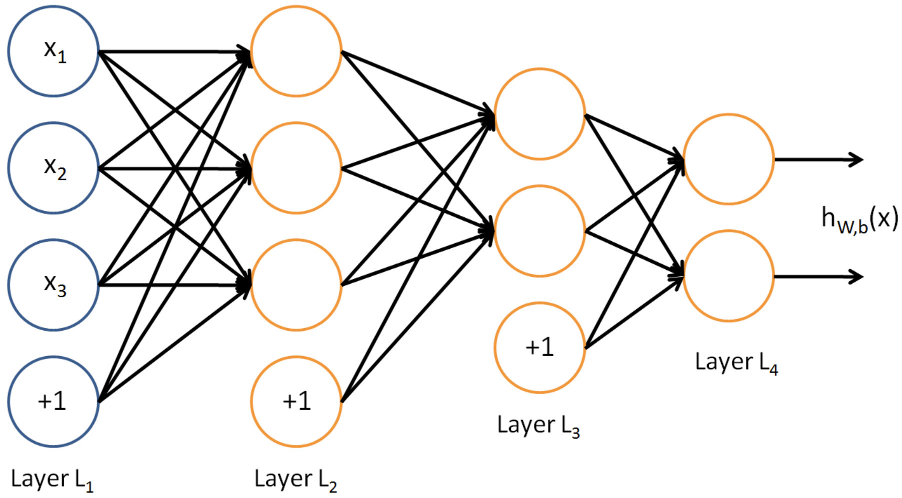


$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$\begin{aligned} h_{W,b}(x) = f(W^{(3)} a^{(3)} + b^{(3)}) &= f\left(W^{(3)} \cdot f\left(W^{(2)} a^{(2)} + b^{(2)}\right) + b^{(3)}\right) \\ &= f\left(W^{(3)} \cdot f\left(W^{(2)} \cdot f\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right) + b^{(3)}\right) \end{aligned}$$

Computing the gradient of *h* involves several applications of the chain rule.

Backpropagation

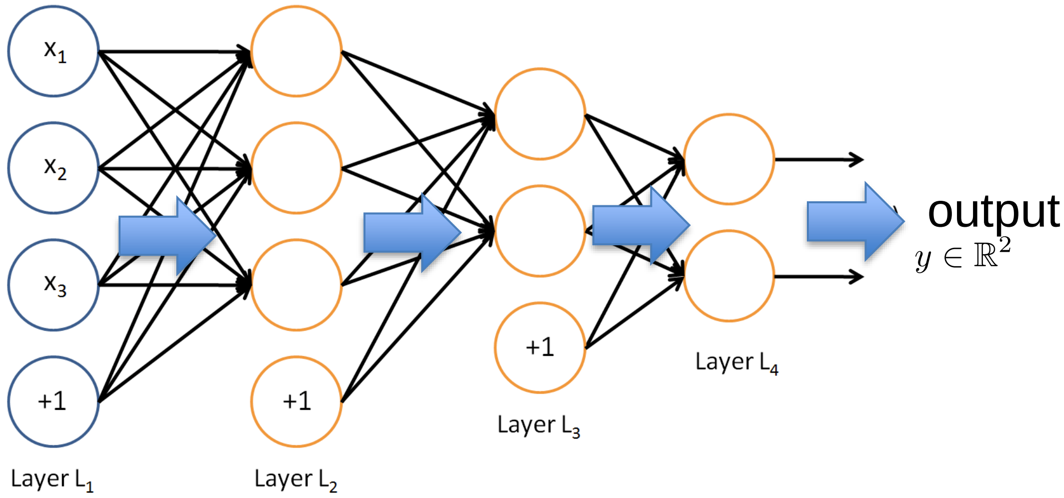


output
$y \in \mathbb{R}^2$

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 .$$

Backpropagation

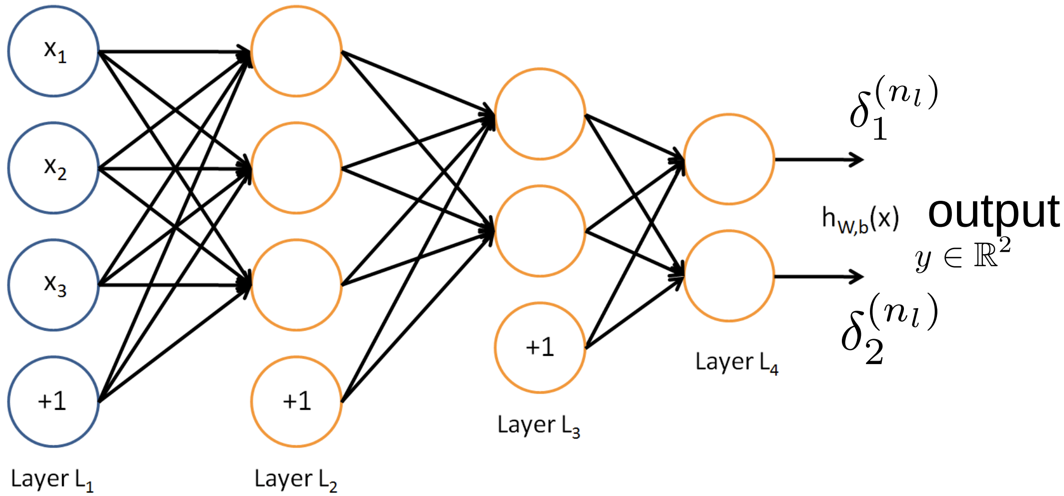


$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 .$$

- **Feed-forward step**: Given input x , compute inputs $z^{(l)}$ and activation values $a^{(l)}$ at every layer l .

Backpropagation



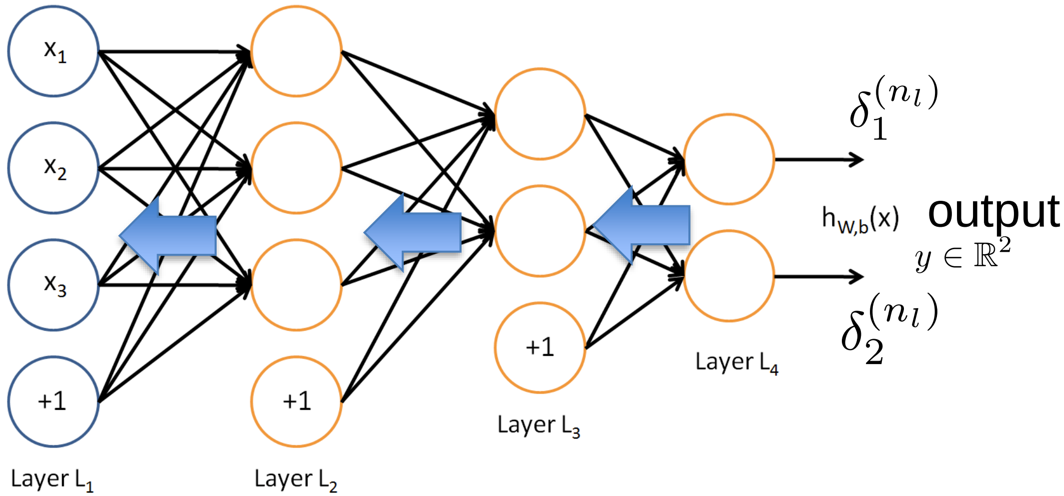
$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 .$$

- **Error computation**: For each output unit i at the output layer n_l, compute errors:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

Backpropagation



$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

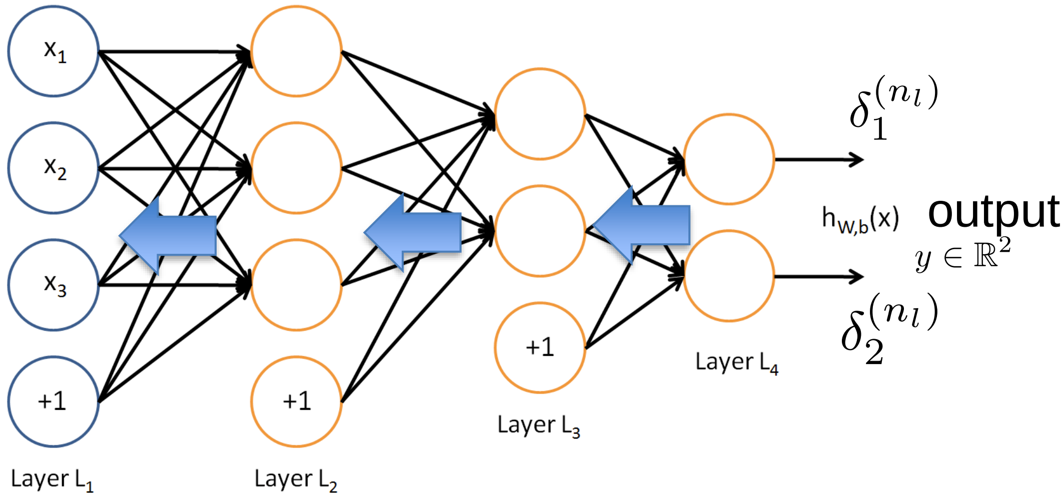
$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2 .$$

- **Error Back-Propagation**: For each layer $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$
  - For each node i in layer l , set:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

Northeastern

Backpropagation



$y \in \mathbb{R}^2$

$\delta_1^{(n_l)}$

$\delta_2^{(n_l)}$

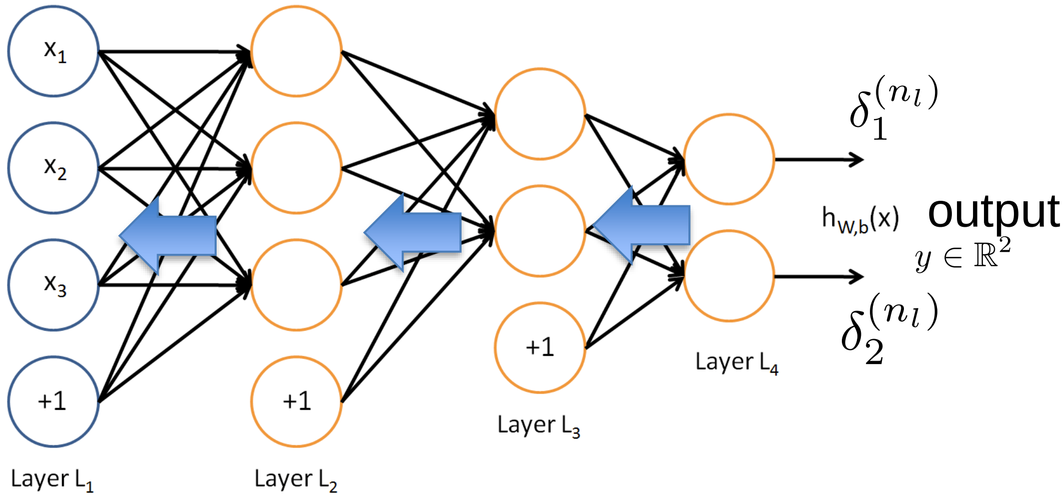
$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 .$$

- **Gradients**: Compute partial derivatives as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

Backpropagation



$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$J(W,b;x,y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2 .$$

- **Intuition**: Error gets attributed according to weights

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

- Same principle applies to other loss functions, e.g., cross-entropy:

$$J(W,b;x,y) = \sum_{k=1}^{d'} y_k \log\left((h_{W,b}(x))_k \right)$$

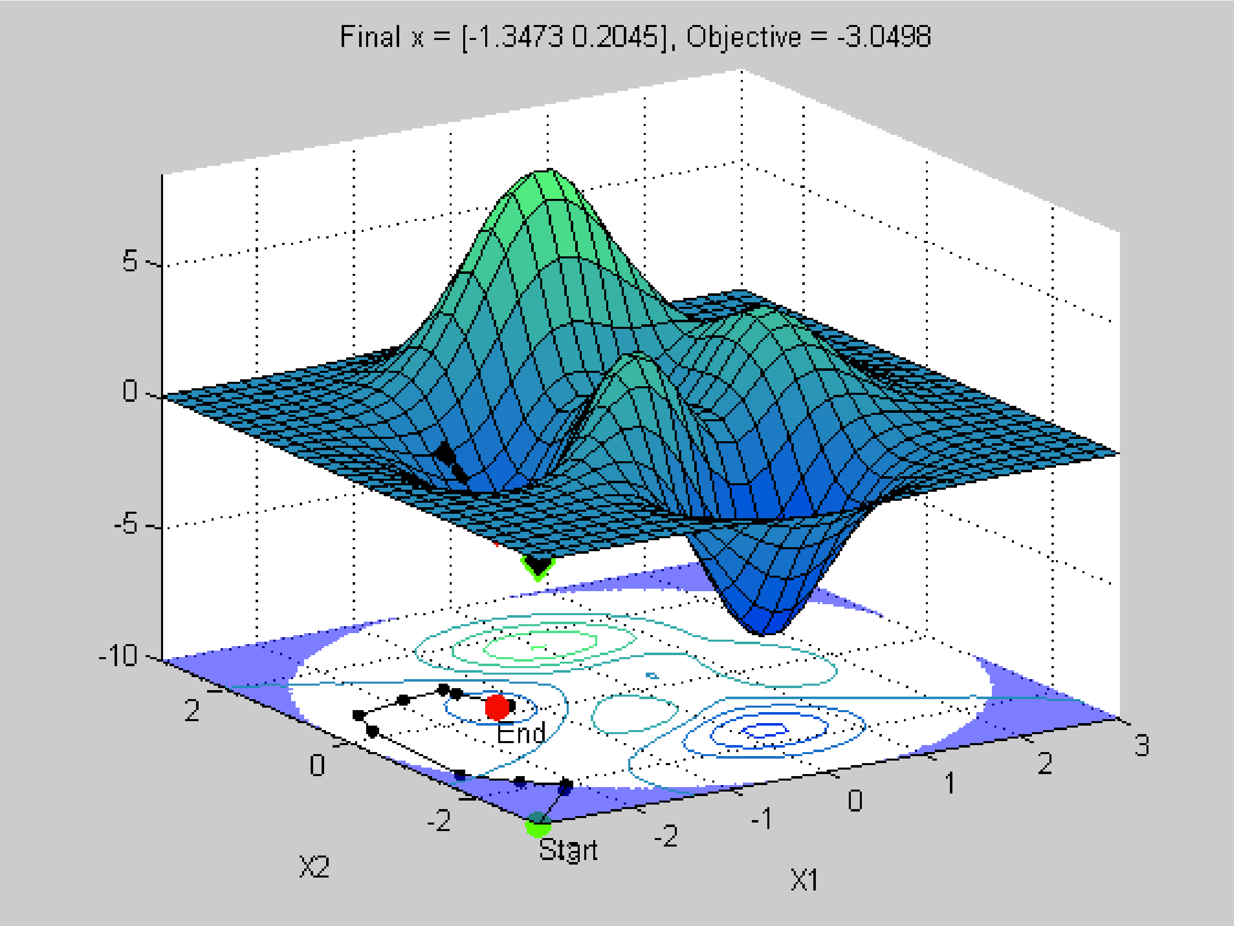
Overfitting

- Deep architectures can suffer from overfitting

- CV used to determine:
  - Number of layers
  - Regularization parameter
  - Number of iterations
  - ...

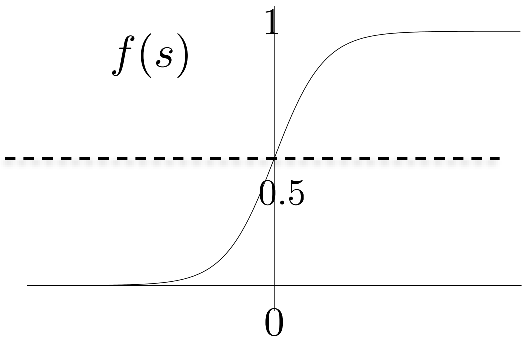
Objective is Non-Convex!

- GD can get stuck in local minima



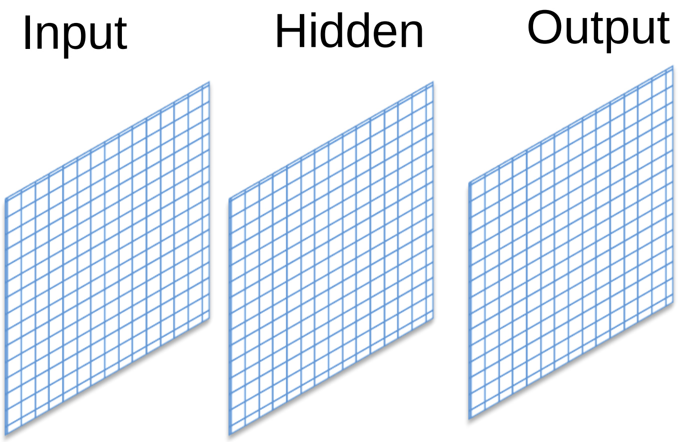
Standard Practices

- Normalize inputs
- Start from values **close to zero**
  - Activation function approx. linear
  - Do not start *exactly* at zero!!



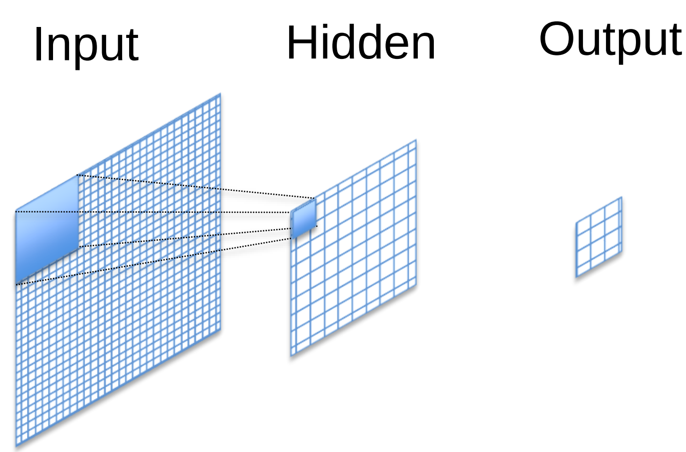
- Multiple starting points
  - Average predictions-**not learned weights!**
  - **Bagging**: average predictions from networks learned from random perturbations of input data $(x^{(i)}, y^{(i)})$
- Tailored Architectures
  - CNNs
  - Auto-Encoders
  - …

Convolutional Neural Nets



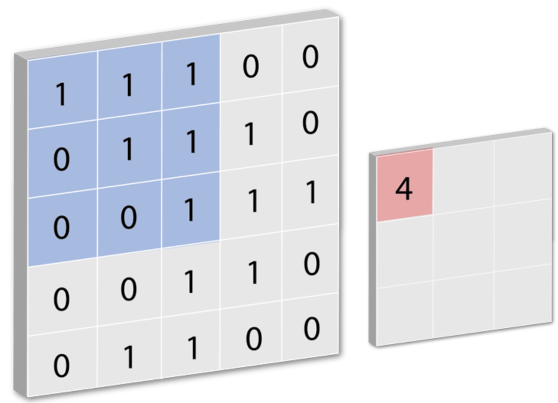
- ❑ Full network extremely dense
- ❑ Ignores locality structure inherent in images

Convolutional Neural Nets



- CNN: use **same weights** on different patches of image

W11	W12	W12
W22	W21	W23
W31	W32	W33



Northeastern

Application: MNIST

Le Cun

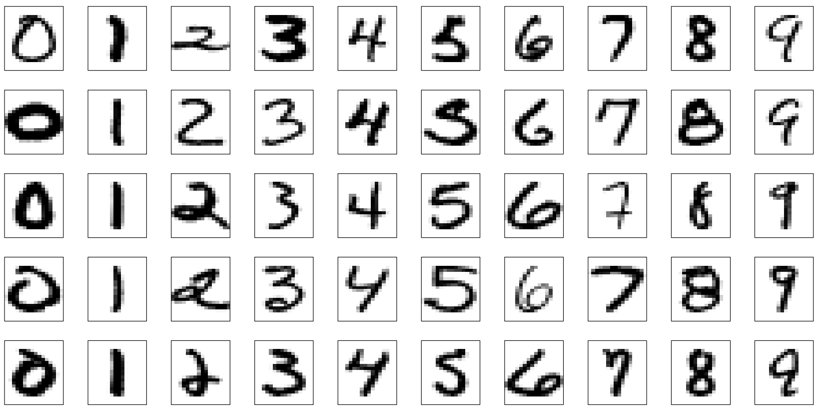
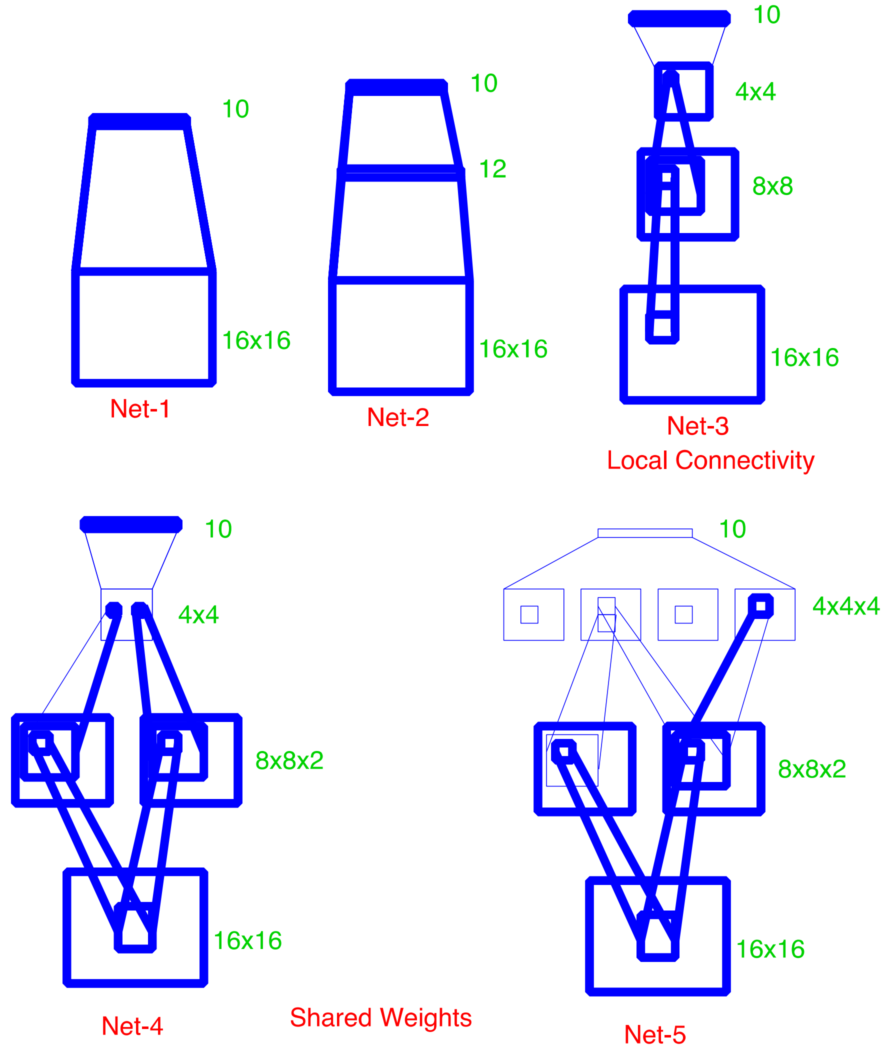


FIGURE 11.9. *Examples of training cases from ZIP code data. Each image is a* 16 × 16 8*-bit grayscale representation of a handwritten digit.*



Net-1: No hidden layer, equivalent to multinomial logistic regression.

Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.

Northeastern

Application: MNIST

Le Cun

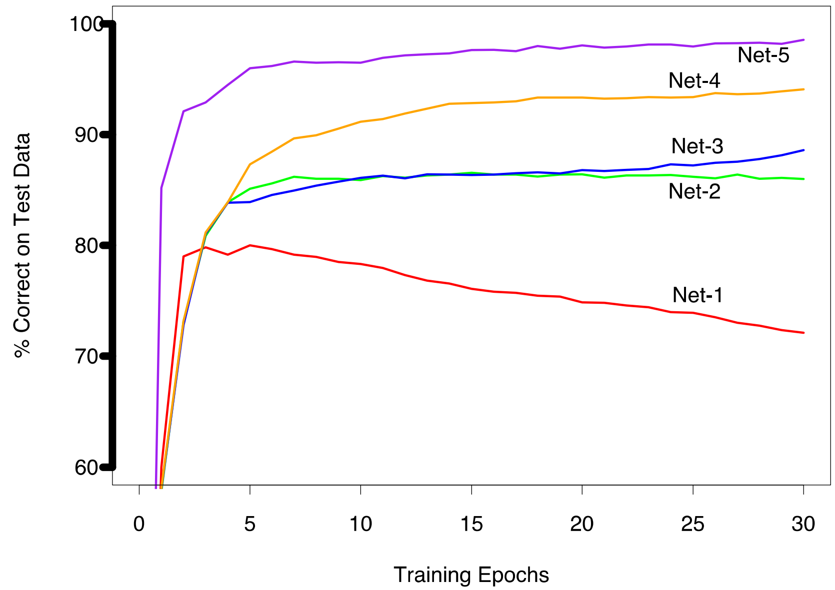


FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data. (Le Cun, 1989)*

TABLE 11.1. *Test set performance of five different neural networks on a handwritten digit classification example (Le Cun, 1989).*

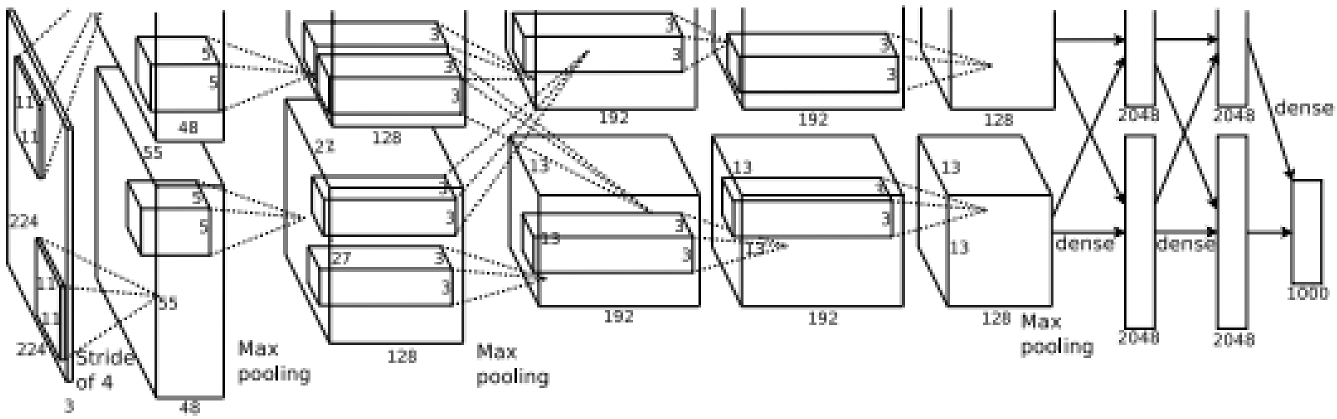
	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

Current SoA: 0.23% http://yann.lecun.com/exdb/mnist/index.html

Northeastern

AlexNet

- ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky, Sutskever, Hinton, NIPS 2012

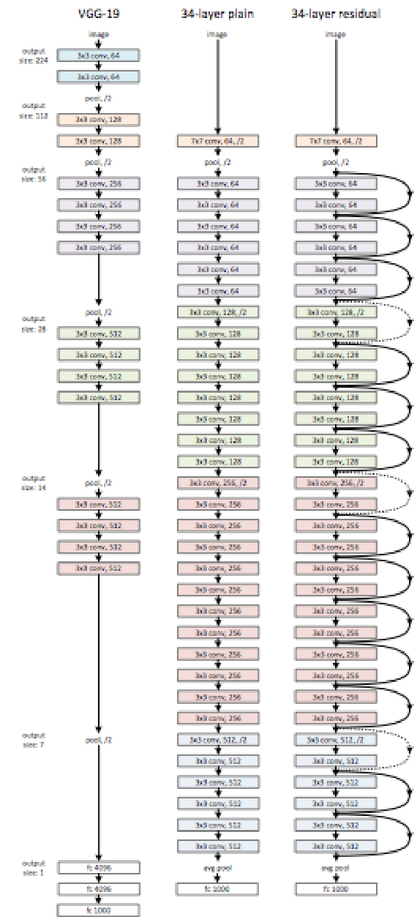


- ILSVRC-2012 competition: achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

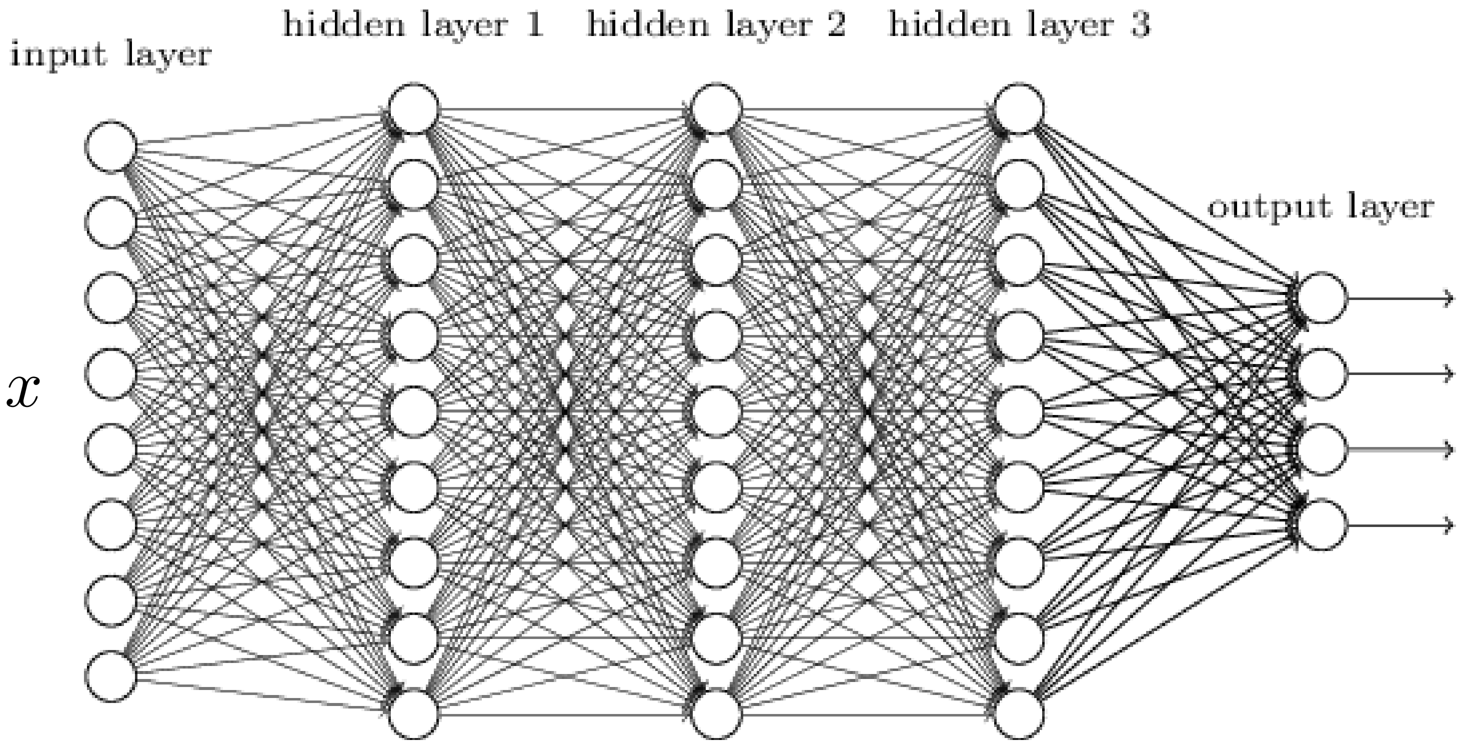
- **GPU**

Ever-Deeper Architectures

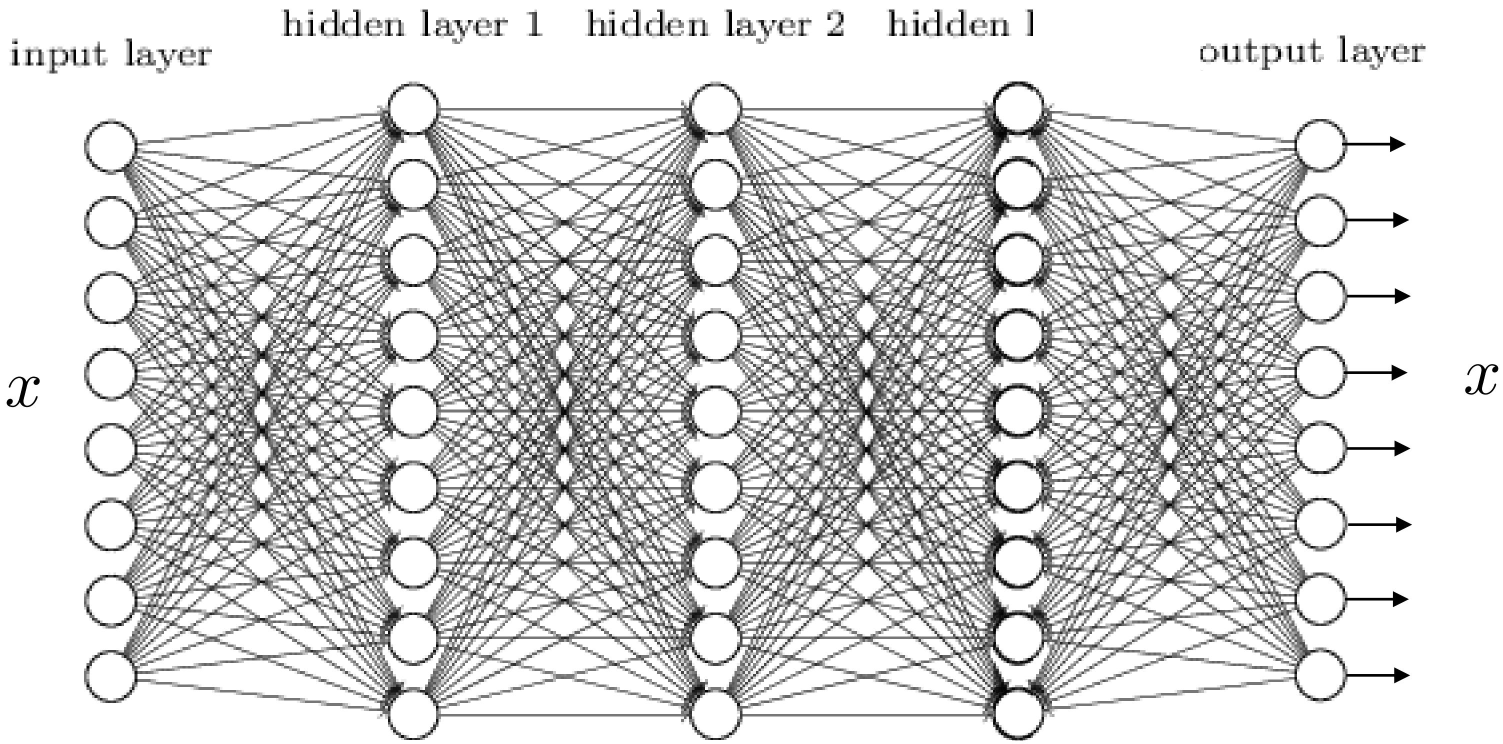
- AlexNet 5 layers
- VGG 19 layers
- GoogLeNet 22 layers
- ResNet 56 layers



AutoEncoders

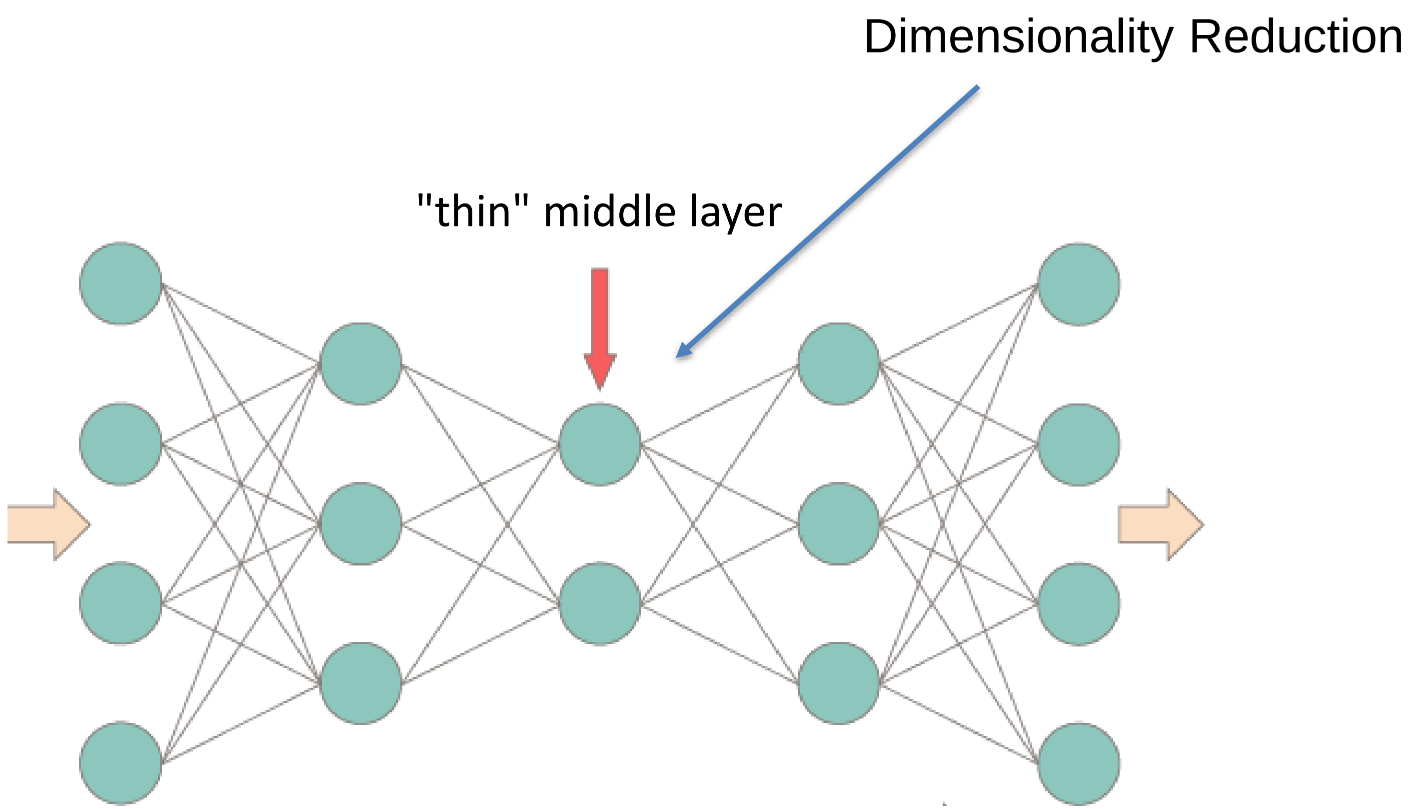


AutoEncoders



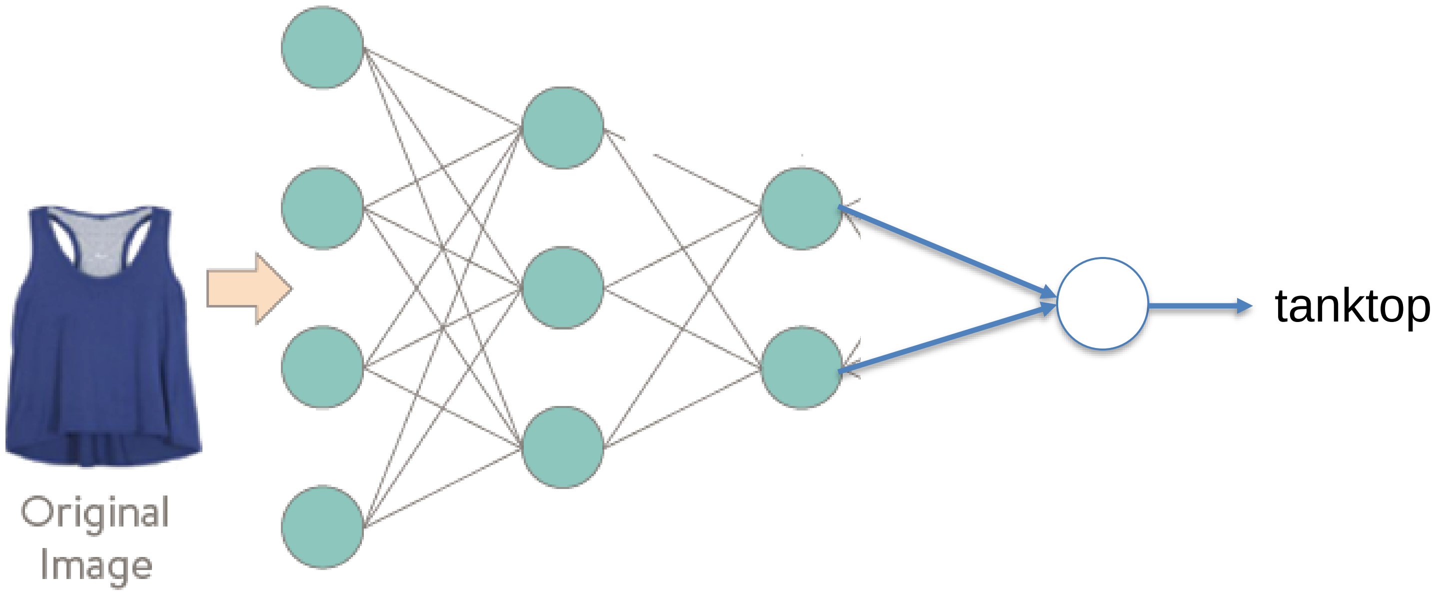
Northeastern

AutoEncoders

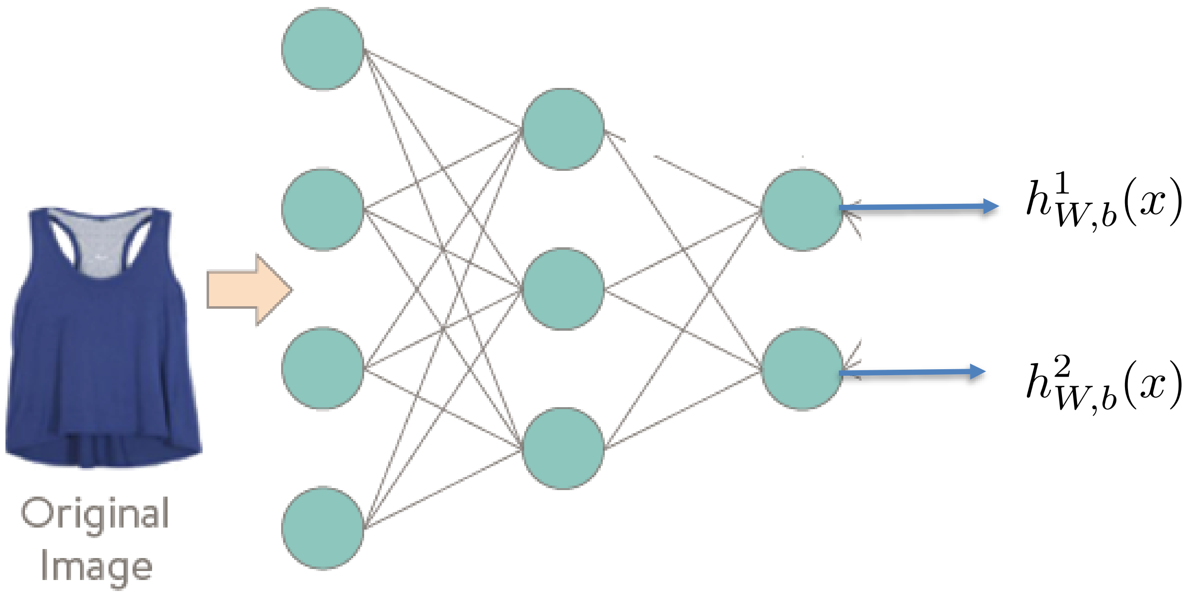


Northeastern

AutoEncoders



AutoEncoders



What does feature 2 mean?

$$\arg\max_{x_i \in \mathtt{data}} h^2_{W,b}(x_i)$$

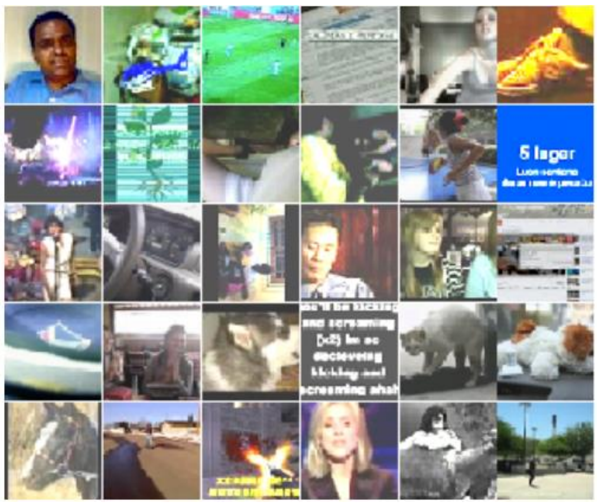
$$\arg\max_{x \in \mathbb{R}^d : \|x\| \leq 1} h^2_{W,b}(x_i)$$

"eigen"-image

Northeastern

Application: YouTube Video Frames

Building High-level Features Using Large Scale Unsupervised Learning, Le, Ranzato, Monga, Devin, Chen, Corrado, Dean, Ng, ICML 2012



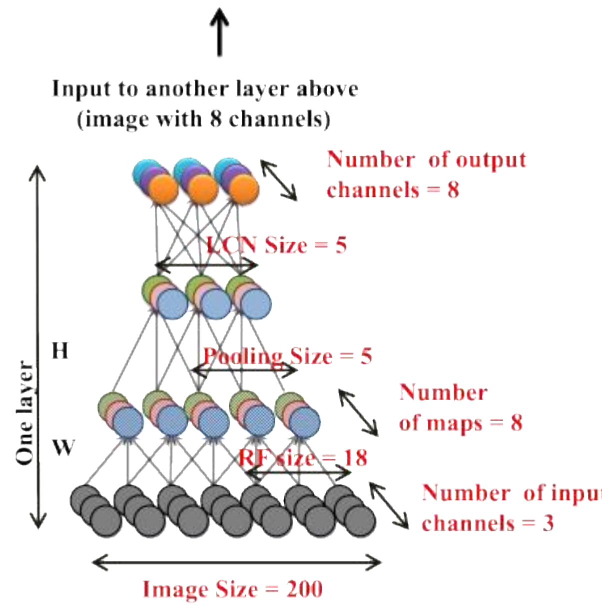


Figure 1. The architecture and parameters in one layer of our network. The overall network replicates this structure three times. For simplicity, the images are in 1D.

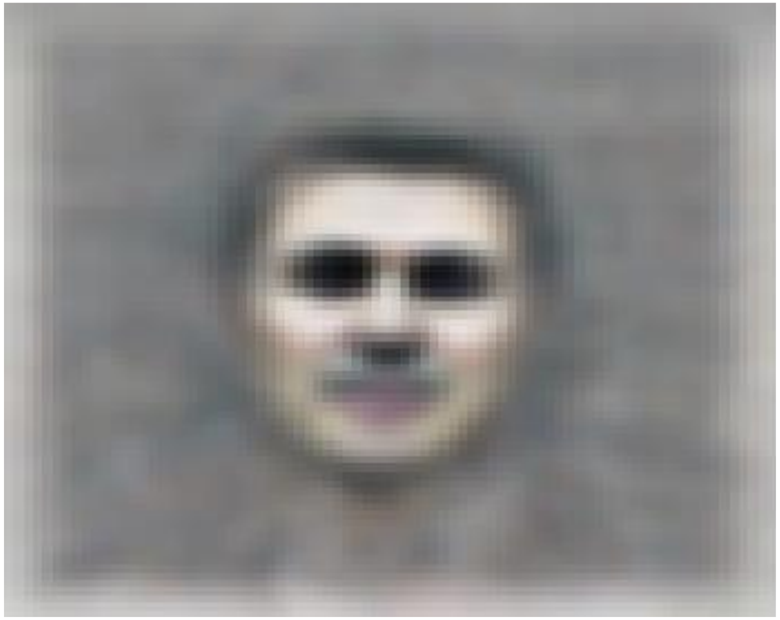
- ❑ Images sampled randomly from 10 million Youtube videos
- ❑ DNN trained over 1000 machines of 16 cores each

Northeastern

Application on Smaller Dataset

- 37K images
  - Labeled Faces in the Wild
  - ImageNet

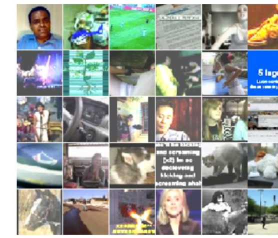
$$\arg\max_{x_i \in \mathtt{data}} h^2_{W,b}(x_i)$$



$$\arg\max_{x \in \mathbb{R}^d : \|x\| \le 1} h^2_{W,b}(x_i)$$

Application: YouTube Video Frames

Building High-level Features Using Large Scale Unsupervised Learning, Le, Ranzato, Monga, Devin, Chen, Corrado, Dean, Ng, ICML 2012



- Images sampled randomly from 10 million Youtube videos

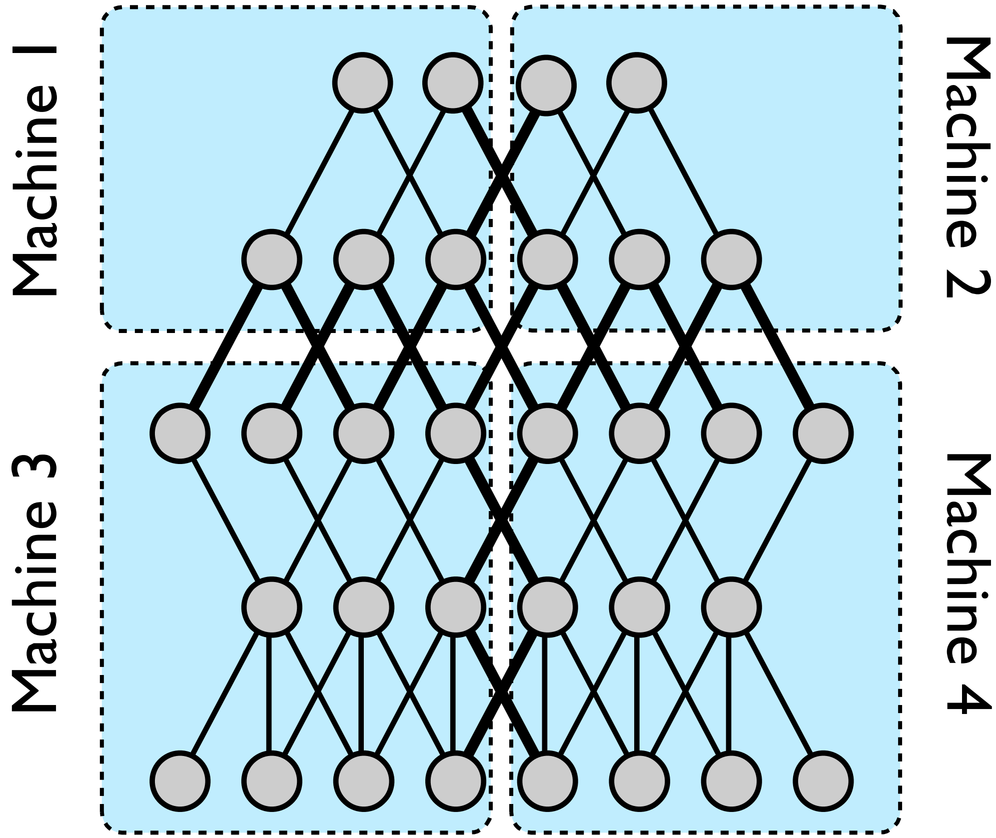
"Eigen"-images:

TECHNOLOGY

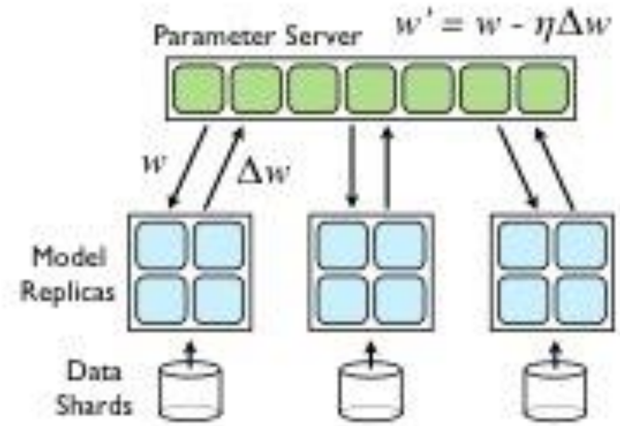
How Many Computers to Identify a Cat? 16,000

By JOHN MARKOFF JUNE 25, 2012

Parallelism and DNNs

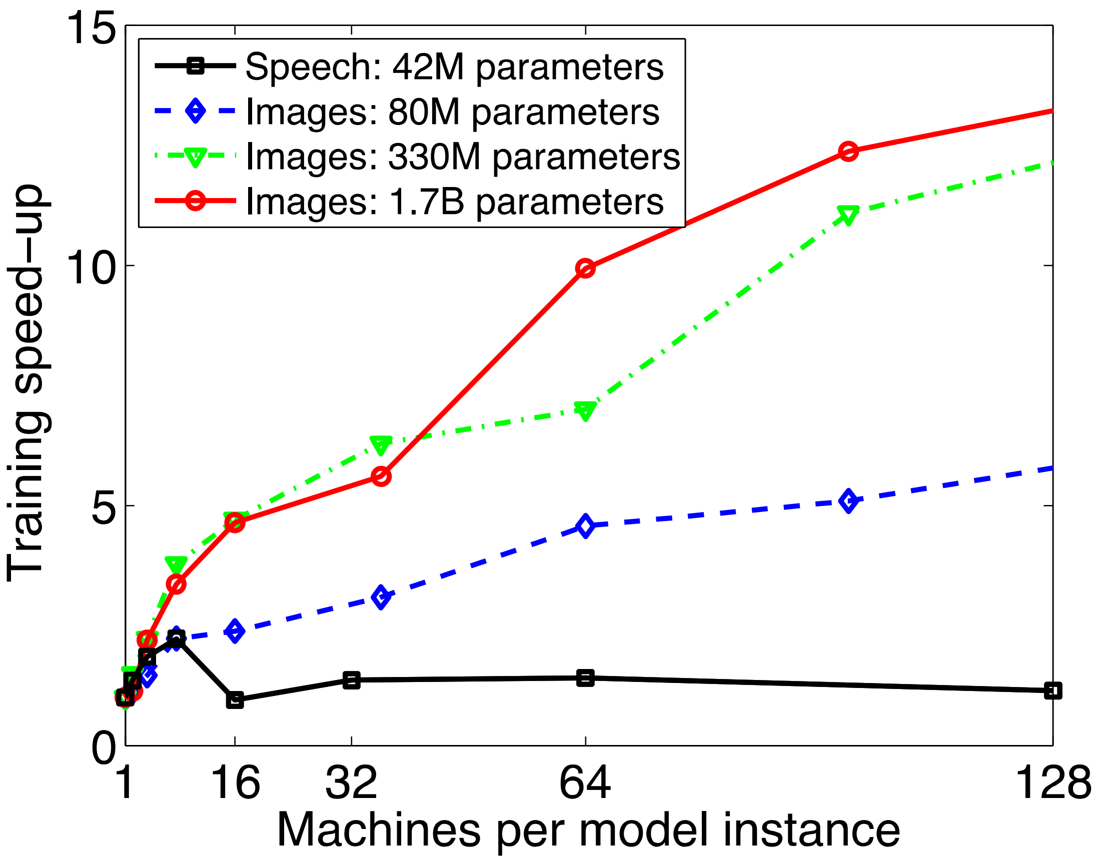


Large Scale Distributed Deep Networks, Dean, Corrado, Monga, Chen, Devin, Mao, Ranzato, Senior, Ng, NIPS 2012.



If **graph is sparse**, you can parallelize to level of neuron!!!

Parallelism and DNNs



Large Scale Distributed Deep Networks, Dean, Corrado, Monga, Chen, Devin, Mao, Ranzato, Senior, Ng, NIPS 2012.

Lots of Software

- GPU based
  - Caffe http://caffe.berkeleyvision.org/
  - Torch http://torch.ch/

- Cluster + GPUs
  - TensorFlow https://www.tensorflow.org/