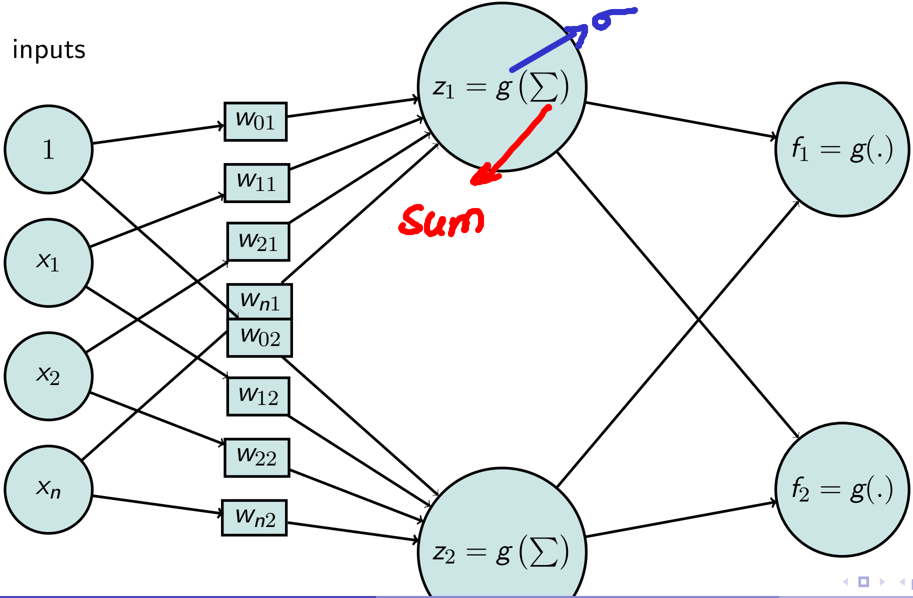
Lecture 20 contd: Neural Network Training using Backpropagation, Convolutional And Recurrent Neural Networks

Instructor: Prof. Ganesh Ramakrishnan

Feed-forward Neural Nets



Training a Neural Network

STEP 0: Pick a network architecture

- Number of input units: Dimension of features $\phi\left(\mathbf{x}^{(i)}\right)$.
- Number of output units: Number of classes.
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same number of hidden units in every layer.
- Number of hidden units in each layer a constant factor (3 or 4) of dimension of x.
- We will use
  - the smooth sigmoidal function $g(s) = \frac{1}{1+e^{-s}}$: **We have now learnt how to train a single node sigmoidal (LR) neural network**
  - instead of the non-smooth step function $g(s) = 1$ if $s \in [\theta, \infty)$ and $g(s) = 0$ otherwise.

High Level Overview of Backpropagation Algorithm for Training NN

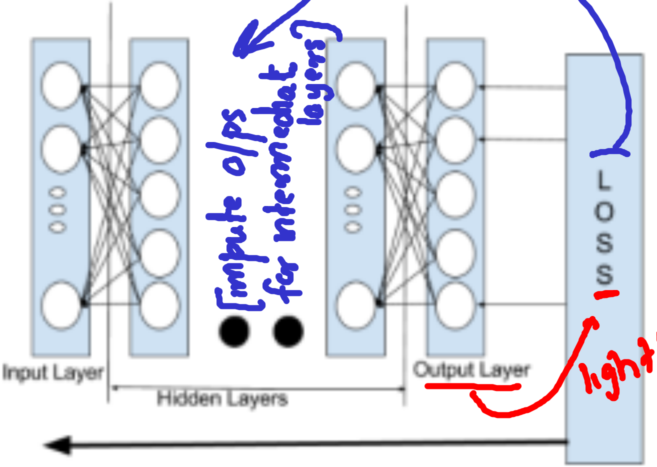
1. Randomly initialize weights w^l_{ij} for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
2. Implement **forward propagation** to get $f_w(\mathbf{x})$ for any $x \in \mathcal{D}$.
3. Execute **backpropagation** on an misclassified example $x \in \mathcal{D}$
   1. by computing partial derivatives $\frac{\partial}{\partial w^{(l)}_{ij}} E(w)$ for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
   2. and using gradient descent to try to minimize (non-convex) $E(w)$ as a function of parameters $\mathbf{w}$.

$$w^l_{ij} = w^l_{ij} - \eta \frac{\partial}{\partial w^{(l)}_{ij}} E(w)$$

4. Verify that the cost function $E(w)$ has indeed reduced, else resort to some random perturbation of weights $\mathbf{w}$.

The Backpropagation Algorithm

indirect relation

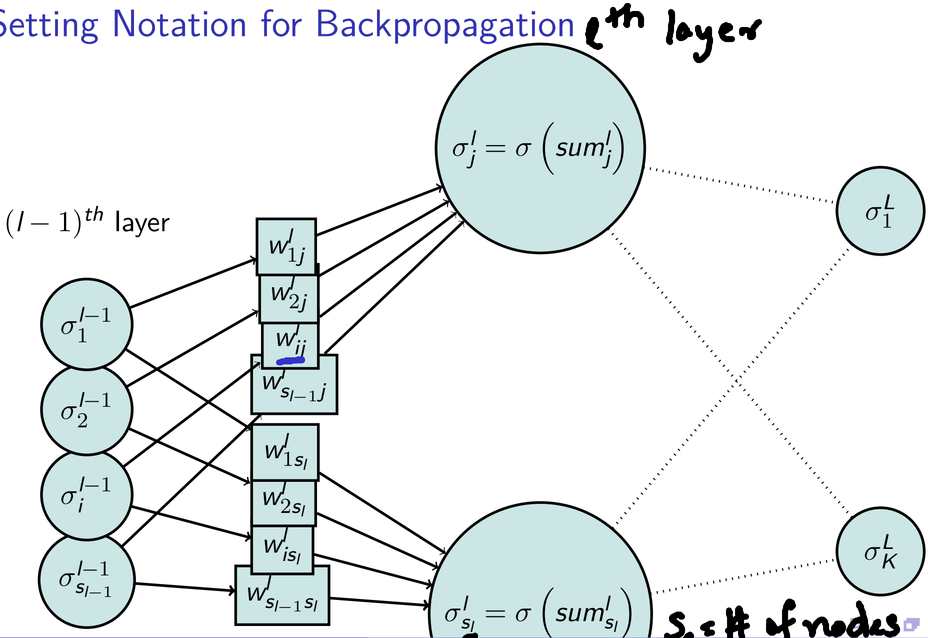


[compute o/ps for intermediate layers]

lightly coupled

Iteratively update the weights along the direction where Loss decreases.

Setting Notation for Backpropagation



s_l = # of nodes in l^{th} layer

Gradient Computation

- The Neural Network objective to be minimized:

$$E(\mathbf{w}) = -\frac{1}{m}\left[\underbrace{\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_k^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^L\left(\mathbf{x}^{(i)}\right)\right)}_{\text{Cross Ent loss computed only on } L^{th} \text{ layer}}\right] + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2}_{\text{Regularization is computed for } l=2 \text{ to } L} \tag{1}$$

Gradient Computation

- The Neural Network objective to be minimized:

$$E(\mathbf{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_k^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^L\left(\mathbf{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \tag{1}$$

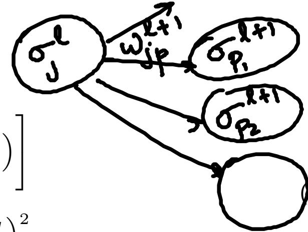
- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial \mathbf{E}}{\partial \mathbf{w}_{\mathbf{ij}}^{\mathbf{l}}} = \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} \frac{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}} \frac{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{w}_{\mathbf{ij}}^{\mathbf{l}}} + \frac{\lambda}{2m} w_{ij}^l$
- $\frac{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}} = \left(\frac{1}{1+e^{-sum_i^l}}\right)\left(1 - \frac{1}{1+e^{-sum_i^l}}\right) = \sigma_{\mathbf{j}}^{\mathbf{l}}(\mathbf{1} - \sigma_{\mathbf{j}}^{\mathbf{l}})$
- $\frac{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{w}_{\mathbf{ij}}^{\mathbf{l}}} = \frac{\partial}{\partial w_{ij}^l}\left(\sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}\right) = \sigma_{\mathbf{i}}^{\mathbf{l-1}}$

E depends on σ_j^l thru (σ^{l+1})

How to compute

$$\frac{\partial E}{\partial \sigma_j^l}$$

Ans: Back propagation!



- For a single example $(\mathbf{x}, y)$:

$$-\left[\sum_{k=1}^{K} y_k \log\left(\sigma_k^L(\mathbf{x})\right) + (1-y_k)\log\left(1-\sigma_k^L(\mathbf{x})\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \quad (2)$$

Recall simple logistic case

$$\frac{\partial E}{\partial \sigma_j^L} = -\frac{y_j}{\sigma_j^L} - \frac{(1-y_j)}{(1-\sigma_j^L)}$$

- $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial sum_p^{l+1}} \frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}} \frac{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}}{\partial \mathbf{sum}_{\mathbf{p}}^{\mathbf{l+1}}} w_{jp}^{l+1}$ since $\frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = w_{jp}^{l+1}$

$\frac{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}}{\partial \mathbf{sum}_{\mathbf{p}}^{\mathbf{l+1}}} \rightarrow \sigma_p^{l+1}(1-\sigma_p^{l+1})$

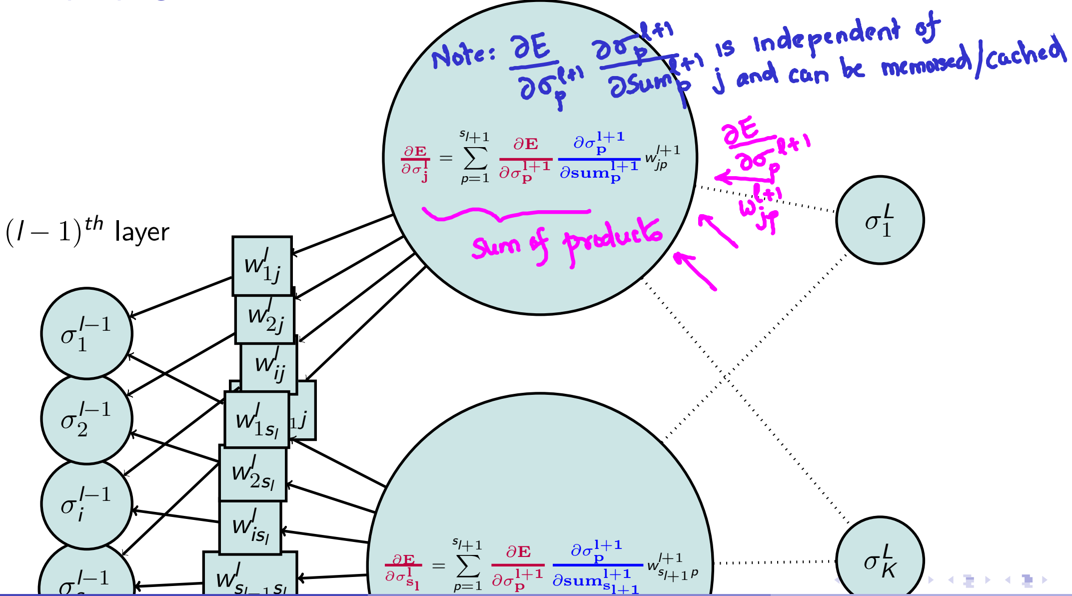
- $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{L}}} = -\frac{\mathbf{y_j}}{\sigma_{\mathbf{j}}^{\mathbf{L}}} + \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_{\mathbf{j}}^{\mathbf{L}}}$

$$E = fn\left(\sigma_{\{k\}}^{L}\left(\sigma_{\{k'\}}^{L-1}\left(\sigma_{\{k''\}}^{L-2} \cdots \left(\sigma_{\{p\}}^{l+1}\left(\sigma_j^{l} \cdots\right)\right)\right)\right)\right)$$

$$\sigma_p^{l+1} = g(sum_p^{l+1}) \qquad sum_p^{l+1} = \sum_j \sigma_j^l w_{jp}^{l+1}$$

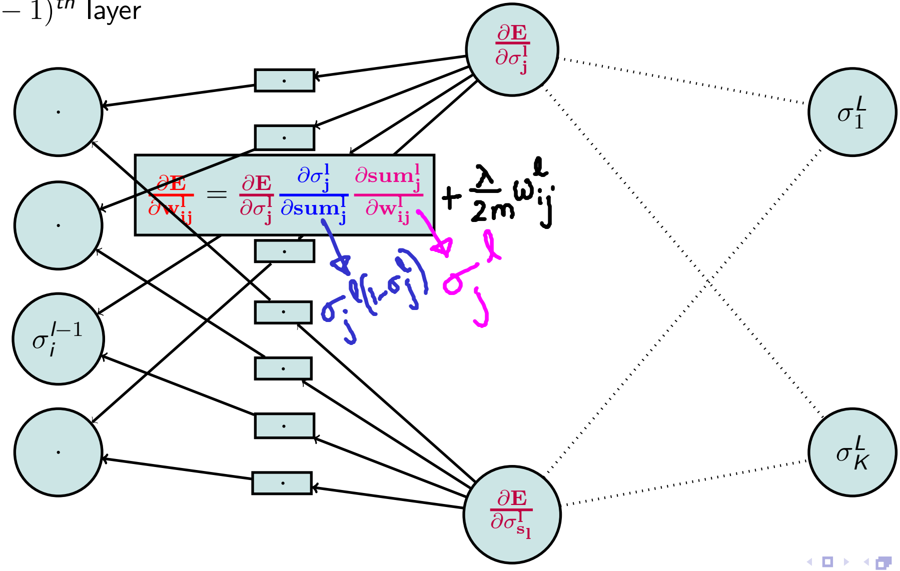
Backpropagation in Action

l^{th} layer



Backpropagation in Action

$(l-1)^{th}$ layer



$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial \sigma_j^l}\underbrace{\frac{\partial \sigma_j^l}{\partial sum_j^l}}_{\sigma_j^l(1-\sigma_j^l)}\underbrace{\frac{\partial sum_j^l}{\partial w_{ij}^l}}_{\sigma_j^l} + \frac{\lambda}{2m} w_{ij}^l$$

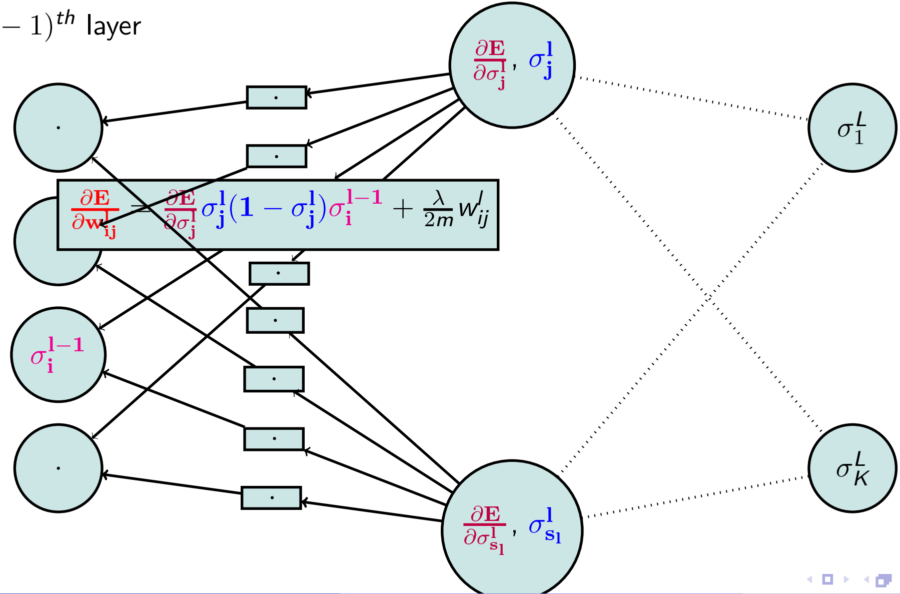
Recall and Substitute

- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} \frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} + \frac{\lambda}{2m} w_{ij}^l$
- $\frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} = \underline{\sigma_\mathbf{j}^\mathbf{l}(\mathbf{1} - \sigma_\mathbf{j}^\mathbf{l})}$
- $\frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} = \underline{\sigma_\mathbf{i}^\mathbf{l-1}}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l+1}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l+1}}{\partial \mathbf{sum_p^{l+1}}} w_{jp}^{l+1}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{L}} = -\frac{\mathbf{y_j}}{\sigma_\mathbf{j}^\mathbf{L}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_\mathbf{j}^\mathbf{L}}$

Already done

Backpropagation in Action

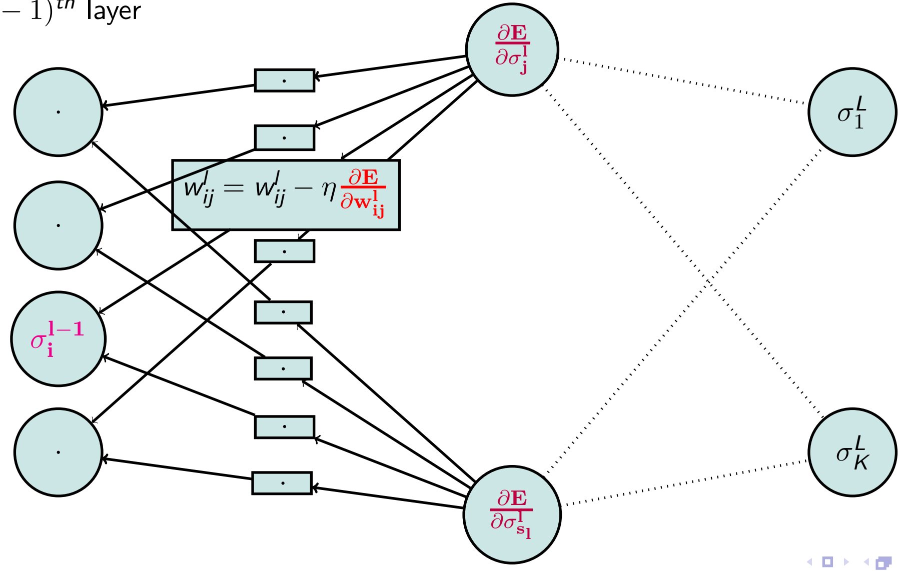
$(l-1)^{th}$ layer



$$\frac{\partial \mathbf{E}}{\partial \mathbf{w}_{\mathbf{ij}}^{\mathbf{l}}} = \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} \sigma_{\mathbf{j}}^{\mathbf{l}}(\mathbf{1} - \sigma_{\mathbf{j}}^{\mathbf{l}})\sigma_{\mathbf{i}}^{\mathbf{l-1}} + \frac{\lambda}{2m} w_{ij}^{l}$$

Backpropagation in Action

$(l-1)^{th}$ layer



The Backpropagation Algorithm for Training NN

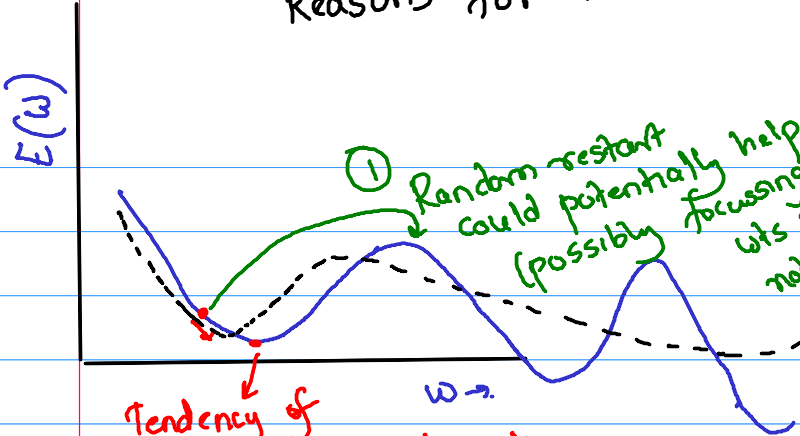
1. Randomly initialize weights w_{ij}^{l} for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
2. Implement **forward propagation** to get $f_{\mathbf{w}}(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{D}$.
3. Execute **backpropagation** on any misclassified $\mathbf{x} \in \mathcal{D}$ by performing gradient descent to minimize (non-convex) $E(\mathbf{w})$ as a function of parameters $\mathbf{w}$.
4. $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{L}}} = -\frac{\mathbf{y_j}}{\sigma_{\mathbf{j}}^{\mathbf{L}}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_{\mathbf{j}}^{\mathbf{L}}}$ for $j = 1$ to s_L. → For L^{th} layer
5. For $l = L-1$ down to 2:
   1. $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l+1}}} \sigma_{\mathbf{j}}^{\mathbf{l+1}}(\mathbf{1} - \sigma_{\mathbf{j}}^{\mathbf{l+1}}) w_{jp}^{l+1}$
   2. $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^{l}}} = \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} \sigma_{\mathbf{j}}^{\mathbf{l}}(\mathbf{1} - \sigma_{\mathbf{j}}^{\mathbf{l}}) \sigma_{\mathbf{i}}^{\mathbf{l-1}} + \frac{\lambda}{2m} w_{ij}^{l}$
   3. $w_{ij}^{l} = w_{ij}^{l} - \eta \frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^{l}}}$

   } Tut 8 problem: Compute min number of multiplications and additions for a single backprop assuming infinite memory
6. Keep picking misclassified examples until the cost function $E(\mathbf{w})$ shows significant reduction; else resort to some random perturbation of weights $\mathbf{w}$ and restart a couple of times.

Additionally estimate max. memory reqd

Reasons for restart

(3)



(1) Random restart could potentially help (possibly focussing on wts that did not change so far)

Error on held out data

Tendency of gradient descent to seek local minima based on "myopic" local linear approx

(2) Avoid overfitting by

(a) Random restart

(b) Validation on held out set

Challenges with Neural Network Training

① Local optima.. only approx correct solution

② Slower, more memory that other methods

③ Architecture design & associating semantics with layers

④ Numerical precision errors

⑤ $\underbrace{\text{Representability}}_{\text{More the \# of layers \& nodes, more is ability to model complex fns}}$ vs learnability tradeoff

More the # of layers & nodes, more is ability to model complex fns [Articulate this claim in Tut8 for all possible activation units]

# of examples required to learn increases exponentially (Curse of dimensionality)

What Changed with Neural Networks?

Reason 1: Better computation infrastructure (GPUs, dist processing, MPI) & lots of data

- Origin: Computational Model of Threshold Logic from Warren McCulloch and Walter Pitts (1943)
- Big Leap: For ImageNet Challenge, AlexNet acheived 85 % accuracy(NIPS 2012). Prev best was 75 % (CVPR 2011).
- Present best is 96.5 % MSRA (arXiv 2015). Comparable to human level accuracy.
- Challenges involved were varied background, same object with different colors(e.g. cats), varied sizes and postures of same objects, varied illuminated conditions.
- Tasks like OCR, Speech recognition are now possible without segmenting the word image/signal into character images/signals.

Pretraining = Unsupervised learning of w's in first (few) layer(s)

Reason 2: ML based principles for sparser/specially constructed n/ws, param sharing, pretraining, convolution

LeNet(1989 and 1998) v/s AlexNet(2012)

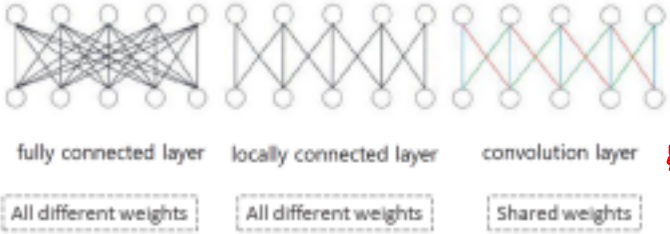
	LeeNet 1989	LeeNet 1998	AlexNet 2012
Task	Digit Recognition	Digit Recognition	Object Recognition
# Classes	10	10	1k
Image Size	16 X 16	28 X 28	256 X 256 X 3
# Examples	7k	60k	1.2 M
units	1256	8084	658 k
parameters	9760	60k	60 M
connections	65k	344k	652M
Summation	Sigmoid	Sigmoid	ReLU
GPU/ Non-GPU	Non-GPU based.	Non-GPU based.	GPU based.
Operations	11 billion	412 billions	200 quadrillions

Reasons for Big Leap

- Why LeeNet was not as successful as AlexNet, though the algorithm was same?
- Right algorithm at wrong time.
- Modern features.
- Advancement in Machine learning.
- Realistic data collection in huge amount due to: regular competitions, evaluation metrics or challenging problem statements.
- Advances in Computational Resources: GPUs, industrial scale clusters.
- Evolution of tasks: Classification of 10 objects to 100 objects to “structure of classes”.

Convolutional Neural Network

- Variation of multi layer feedforward neural network designed to use minimal preprocessing with wide application in image recognition and natural language processing
- Traditional multilayer perceptron(MLP) models do not take into account spatial structure of data and suffer from curse of dimensionality
- Convolution Neural network has smaller number of parameters due to **local connections** and **weight sharing**



H/W: How to compute shared wts?