Neural Networks

Class 22: MLSP, Fall 2016

Instructor: Bhiksha Raj
IMPORTANT ADMINISTRIVIA

• Final week. Project presentations on 6th
Neural Networks are taking over!

• Neural networks have become one of the major thrust areas recently in various pattern recognition, prediction, and analysis problems.

• In many problems they have established the state of the art.
  – Often exceeding previous benchmarks by large margins.
Recent success with neural networks

• Some recent successes with neural networks
  – A bit of hyperbole, but still..
Recent success with neural networks

• Some recent successes with neural networks
  – A bit of hyperbole, but still..
Recent success with neural networks

- Some recent successes with neural networks
  - A bit of hyperbole, but still..
Recent success with neural networks

- Some recent successes with neural networks
  – A bit of hyperbole, but still..
Recent success with neural networks

• Captions generated entirely by a neural network
Successes with neural networks

• And a variety of other problems:
  – Image recognition
  – Signal enhancement
  – Even predicting stock markets!
So what are neural networks??

- What are these boxes?
So what are neural networks??

• It began with this..
• Humans are very good at the tasks we just saw
• Can we model the human brain/ human intelligence?
  – An old question – dating back to Plato and Aristotle..
Observation: *The Brain*

- Mid 1800s: The brain is a mass of interconnected neurons
Brain: Interconnected Neurons

• Many neurons connect *in* to each neuron
• Each neuron connects *out* to many neurons
The brain is a connectionist machine

• The human brain is a connectionist machine

• Neurons connect to other neurons. The processing/capacity of the brain is a function of these connections

• Connectionist machines emulate this structure
Connectionist Machines

• Neural networks are *connectionist* machines
  – As opposed to Von Neumann Machines

  ![Von Neumann Machine](image)
  ![Neural Network](image)

• The machine has many processing units
  – The program is the connections between these units
    • Connections may also define memory
Modelling the brain

- What are the units?
- A neuron:
  - Signals come in through the dendrites into the Soma
  - A signal goes out via the axon to other neurons
    - Only one axon per neuron
  - Factoid that may only interest me: Neurons do not undergo cell division
McCullough and Pitts

• The Doctor and the Hobo..
  – Warren McCulloch: Neurophysician
  – Walter Pitts: Homeless wannabe logician who arrived at his door
The McCulloch and Pitts model

- A mathematical model of a neuron
  - Threshold Logic
- Note: McCullough and Pitts original model was actually slightly different – this model is actually due to Rosenblatt
The solution to everything

• Frank Rosenblatt
  – Psychologist, Logician
  – Inventor of the solution to everything, aka the Perceptron (1958)
    • A mathematical model of the neuron that could solve everything!!!
Simplified mathematical model

- Number of inputs combine linearly
  - Threshold logic: Fire if combined input exceeds threshold

\[
Y = \begin{cases} 
1 & \text{if } \sum_i w_i x_i + b > 0 \\
0 & \text{else}
\end{cases}
\]
Simplified mathematical model

- A mathematical model
  - Originally assumed could represent *any* Boolean circuit
  - Rosenblatt, 1958: “*the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence*”
Perceptron

• Boolean Gates

• But...
Perceptron

No solution for XOR! Not universal!

\[ X \oplus Y \]

• Minsky and Papert, 1968
A single neuron is not enough

- Individual elements are weak computational elements
  - Marvin Minsky and Seymour Papert, 1969, *Perceptrons: An Introduction to Computational Geometry*

- *Networked* elements are required
Multi-layer Perceptron!

- **XOR**
  - The first layer is a “hidden” layer
Multi-Layer Perceptron

- Even more complex Boolean functions can be composed using layered networks of perceptrons
  - Two hidden layers in above model
  - In fact can build any Boolean function over any number of inputs

\[
((A & \bar{X} & Z) | (A & \bar{Y})) \& ((X \& Y) | (X \& Z))
\]
Multi-layer perceptrons are universal Boolean functions

- A multi-layer perceptron is a universal Boolean function
- In fact, an MLP with only one hidden layer is a universal Boolean function!
Constructing Boolean functions with only one hidden layer

- Any Boolean formula can be expressed by an MLP with **one hidden layer**
  - Any Boolean formula can be expressed in Conjunctive Normal Form
- The one hidden layer can be exponentially wide
  - But the same formula can be obtained with a much smaller network if we have *multiple* hidden layers
Neural Networks: Multi-layer Perceptrons

- In reality the input to these systems is not Boolean
- Inputs are continuous valued
  - Signals may be continuous valued
  - Image features are continuous valued
  - Pixels are multi-valued (e.g. 0-255)
MLP on continuous inputs

- The inputs are continuous valued
  - Threshold logic: Fire if combined input exceeds threshold

\[
Y = \begin{cases} 
1 & \text{if } \sum_{i} w_i x_i + b > 0 \\
0 & \text{else}
\end{cases}
\]
A Perceptron on Reals

- A perceptron operates on real-valued vectors
  - This is just a linear classifier

\[
y = \begin{cases} 
1 & \text{if } \sum_i w_i x_i \geq T \\
0 & \text{else}
\end{cases}
\]
Booleans over the reals

- The network must fire if the input is in the coloured area

Can now be composed into “networks” to compute arbitrary classification “boundaries”
Booleans over the reals

• The network must fire if the input is in the coloured area
Booleans over the reals

• The network must fire if the input is in the coloured area
Booleans over the reals

- The network must fire if the input is in the coloured area
Booleans over the reals

- The network must fire if the input is in the coloured area
Booleans over the reals

• The network must fire if the input is in the coloured area
Booleans over the reals

- The network must fire if the input is in the coloured area
Booleans over the reals

- “OR” two polygons
- A third layer is required
How Complex Can it Get

- An arbitrarily complex decision boundary
- Basically any Boolean function over the basic linear boundaries
- **Even with a single hidden layer!**
Composing a polygon

- The polygon net
- Increasing the number of sides shrinks the area outside the polygon that have sum close to N
Composing a circle

The circle net

- Very large number of neurons
- Circle can be of arbitrary diameter, at any location
- Achieved *without using a thresholding function*!!
Adding circles

• The “sum” of two circles sub nets is exactly a net with output 1 if the input falls within either circle

No nonlinearity applied!
Composing an arbitrary figure

• Just fit in an arbitrary number of circles
  – More accurate approximation with greater number of smaller circles
  – A lesson here that we will refer to again shortly.
Story so far..

- **Multi-layer perceptrons are Boolean networks**
  - They represent Boolean functions over linear boundaries
  - They can approximate *any* boundary
    - Using a sufficiently large number of linear units
  - Complex Boolean functions are better modeled with more layers
  - Complex boundaries are more compactly represented using more layers
Let's look at the weights

What do the weights tell us?
- The neuron fires if the inner product between the weights and the inputs exceeds a threshold.

\[
y = \begin{cases} 
1 & \text{if } \sum_i w_i x_i \geq T \\
0 & \text{else}
\end{cases}
\]

\[
y = \begin{cases} 
1 & \text{if } \mathbf{x}^T \mathbf{w} \geq T \\
0 & \text{else}
\end{cases}
\]
The weight as a “template”

- The perceptron fires if the input is within a specified angle of the weight
  - Represents a convex region on the surface of the sphere!
  - The network is a Boolean function over these regions.
    - The overall decision region can be arbitrarily nonconvex
- Neuron fires if the input vector is close enough to the weight vector.
  - If the input pattern matches the weight pattern closely enough

$$X^T W > T$$
$$\cos \theta > \frac{T}{|X|}$$
$$\theta < \cos^{-1} \left( \frac{T}{|X|} \right)$$
The weight as a template

If the correlation between the weight pattern and the inputs exceeds a threshold, fire.

The perceptron is a correlation filter!

\[ y = \begin{cases} 1 & \text{if } \sum_i w_i x_i \geq T \\ 0 & \text{else} \end{cases} \]

Correlation = 0.57

Correlation = 0.82

If the *correlation* between the weight pattern and the inputs exceeds a threshold, fire.

The perceptron is a *correlation filter*!
The MLP as a Boolean function over feature detectors

- The input layer comprises “feature detectors”
  - Detect if certain patterns have occurred in the input
- The network is a Boolean function over the feature detectors
- I.e. it is important for the first layer to capture relevant patterns
The MLP as a cascade of feature detectors

- The network is a cascade of feature detectors
  - Higher level neurons compose complex templates from features represented by lower-level neurons

- **Risk in this perspective**: Upper level neurons may be performing “OR”
  - Looking for a *choice* of compound patterns
Story so far

• **MLPs are Boolean machines**
  – They represent Boolean functions over linear boundaries
  – They can represent arbitrary boundaries

• **Perceptrons are correlation filters**
  – They detect patterns in the input

• MLPs are Boolean formulae over patterns detected by perceptrons
  – Higher-level perceptrons may also be viewed as feature detectors

• **Extra: MLP in classification**
  – The network will fire if the combination of the detected basic features matches an “acceptable” pattern for a desired class of signal
    • E.g. Appropriate combinations of (Nose, Eyes, Eyebrows, Cheek, Chin) → Face
MLP as a continuous-valued regression

- MLPs can actually compose arbitrary functions to arbitrary precision
  - Not just classification/Boolean functions
- 1D example
  - **Left:** A net with a pair of units can create a pulse of any width at any location
  - **Right:** A network of N such pairs approximates the function with N scaled pulses
MLP as a continuous-valued regression

• MLPs can actually compose arbitrary functions
  – Even with only one layer
  – To arbitrary precision
  – The MLP is a universal approximator!
Multi-layer perceptrons are universal function approximators

- A multi-layer perceptron is a universal function approximator
  - Hornik, Stinchcombe and White 1989, several others
What’s inside these boxes?

- Each of these tasks is performed by a net like this one
  - Functions that take the given input and produce the required output
Story so far

• **MLPs are Boolean machines**
  – They represent arbitrary Boolean functions over arbitrary linear boundaries
  – MLPs perform classification

• **MLPs can compute arbitrary real-valued functions of arbitrary real-valued inputs**
  – To arbitrary precision
  – They are *universal approximators*
• Building a network for a task
These tasks are functions

- Each of these boxes is actually a function
  - E.g. \( f: \text{Image} \rightarrow \text{Caption} \)
These tasks are functions

- Each box is actually a function
  - E.g. $f: \text{Image} \rightarrow \text{Caption}$
  - It can be approximated by a neural network
The network as a function

• Inputs are numeric vectors
  – Numeric representation of input, e.g. audio, image, game state, etc.

• Outputs are numeric scalars or vectors
  – Numeric “encoding” of output from which actual output can be derived
  – E.g. a score, which can be compared to a threshold to decide if the input is a face or not
  – Output may be multi-dimensional, if task requires it
The network is a function

• Given an input, it computes the function layer wise to predict an output
A note on activations

- Previous explanations assumed network units used a threshold function for “activation”
- In reality, we use a number of other differentiable functions
  - Mostly, but not always, “squashing” functions
The entire network

- Inputs (D-dimensional): $X_1, \ldots, X_D$
- Weights from $i^{th}$ node in $l^{th}$ layer to $j^{th}$ node in $l+1^{th}$ layer: $W_{i,j}^l$
- Complete set of weights
  $$\{W_{i,j}^l; \ l = 0..L, \ i = 1 \ldots N_l, \ j = 1 \ldots N_{l+1}\}$$
- Outputs (M-dimensional): $O_1, \ldots, O_M$
Making a prediction
(Aka forward pass)

Illustration with a network with N-1 hidden layers
For the jth neuron of the 1st hidden layer:
Input to the activation functions in the first layer

\[ I_j^1 = \sum_i W_{ij}^1 X_i \]
Forward Computation

For the $j$th neuron of the 1st hidden layer:

Input to the activation functions in the first layer:

$$I_j^1 = \sum_i W_{ij}^1 X_i$$

Output of the $j$th neuron of the 1st hidden layer:

$$O_j^1 = f_1(I_j^1)$$
Forward Computation

For the jth neuron of the kth hidden layer:
Input to the activation functions in the kth layer:

\[ I_j^k = \sum_i W_{ij}^k O_i^{k-1} \]
Forward Computation

For the jth neuron of the kth hidden layer:
Input to the activation functions in the kth layer

\[ I_j^k = \sum_i W_{ij}^k O_{i}^{k-1} \]

Output of the jth neuron of the kth hidden layer:

\[ O_j^k = f_k(I_j^k) \]
Forward Computation

\[ I_j^1 = \sum_i W_{ij}^1 X_i \]

ITERATE FOR \( k = 1: N \)

for \( j = 1: \text{layer-width} \)

\[ O_j^k = f_k(I_j^k) \]

\[ I_j^k = \sum_i W_{ij}^k O_i^{k-1} \]

Output \( O_N^N \) is the output of the network.
The problem of training the network

• The network must be composed correctly to produce the desired outputs

• This is the problem of “training” the network
A neural network is effectively just a function
- What makes it an NNet is the structure of the function

Takes an input
- Generally multi-dimensional

Produces an output
- May be uni- or multi-dimensional

Challenge: How to make it produce the desired output for any given input
• Solution 1: Just hand design it
  – What will the network for the above function be?
  • We can hand draw this one..
• Most functions are too complex to hand-design
  – Particularly in high-dimensional spaces
• Instead, we will try to *learn* it from “training” examples
  – Input-output pairs
  – In reality the training data will not be even remotely as dense as shown in the figure
    • It will be several orders of magnitude sparser
General approach to training

• Define an error between the actual network output for any parameter value and the desired output
  – Error can be defined in a number of ways
    • Many “Divergence” functions have been defined
  – Typically defined as the sum of the error over individual training instances
Recall: A note on activations

- Composing/learning networks using threshold “activations” is a combinatorial problem
  - Exponentially hard to find the right solution
- The smoother differentiable activation functions enable learning through optimization
Minimizing Error

• Problem: Find the parameters at which this function achieves a minimum
  – Subject to any constraints we may pose
  – Typically, this cannot be found using a closed-form formula
The Approach of Gradient Descent

• Iterative solution:
  – Start at some point
  – Find direction in which to shift this point to decrease error
    • This can be found from the “slope” of the function
      – A positive slope $\rightarrow$ moving left decreases error
      – A negative slope $\rightarrow$ moving right decreases error
  – Shift point in this direction
    • The size of the shift depends on the slope
The Approach of Gradient Descent

\[ \nabla f = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix} \]

- Multi-dimensional function:
  - “Slope” replaced by vector “gradient”
- From current point shift in direction opposite to gradient
  - Size of shift depends on magnitude of gradient
The Approach of Gradient Descent

• The gradient descent algorithm
  \[ W \leftarrow W - \eta \nabla_W E(X; W) \]
  Until error \( E(X; W) \) converges

\( \eta \) is the “step size”

Steps are smaller when the “slope” is smaller, because the optimum value is generally at a location of near-zero slope
The Approach of Gradient Descent

- The gradient descent algorithm
  \[ W \leftarrow W - \eta \nabla_W E(X; W) \]

  Until error \( E(X; W) \) converges

  \( \eta \) is the “step size”

Steps are smaller when the “slope” is smaller, because the optimum value is generally at a location of near-zero slope.

Needs computation of gradient of error w.r.t. network parameters.
Gradients: Algebraic Formulation

• The network is a nested function

\[ o = f_N(W_N f_{N-1}(W_{N-1} f_{N-2}(W_{N-2} \ldots f_k(W_k f_{k-1}(\ldots f_1(W_1 X) \ldots )) \ldots ))) \]

\[ E = Div(y, o) = ||y - o||^2 \]
Using $f_k$ to represent the output of the activations rather than $O^k$ for easier interpretation

Using subscripts rather than superscripts to represent layer
A note on algebra

\[ \nabla_{I_k} E = \left( \nabla_{I_k} f \right) \nabla_{f_k} E \]

\[ \nabla_{I_k} f_k = \begin{bmatrix} f'_{k,1} & 0 & \cdots & 0 \\ 0 & f'_{k,2} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & f'_{k,L} \end{bmatrix} \]

\[ \left( \nabla_{I_k} f_k \right) \nabla_{f_k} E = \begin{bmatrix} f'_{k,1} & 0 & \cdots & 0 \\ 0 & f'_{k,1} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & f'_{k,1} \end{bmatrix} \nabla_{f_k} E \]

\[ \nabla_{I_k} E = f'_k \circ \nabla_{f_k} E \]
**Graidents: Algebraic Formulation**

- **Generic rule**
  
  \[
  \nabla_{W_k} E = \left( f_k' \circ \nabla_{f_k} E \right) f_{k-1}^T
  \]
  
  \[
  \nabla_{W_1} E = \left( f_1' \circ \nabla_{f_1} E \right) X^T
  \]

- \(f_{k-1}\) is the vector of outputs of the \(k-1\)th layer
- \(\nabla_{f_k} E\) is the gradient of the error w.r.t. the vector of outputs of the \(k\)th layer
- \(f_k'\) is the vector of derivatives of the activations of the \(k\)th layer w.r.t. their inputs
  - taken at current input \(W_k f_{k-1}\)
Graidents: Algebraic Formulation

\[ \nabla_{W_k} E = \left( f'_k \circ \nabla f_k E \right) f_{k-1}^T \]

- What is this?
A note on algebra

\[ I_k = W_k f_{k-1} \]

\[ \nabla_{f_{k-1}} I_k = W_k^T \]

\[ \nabla_{f_{k-1}} f_k = \left( \nabla_{f_{k-1}} I_k \right) \nabla_{I_k} f_k = W_k^T \left[ \begin{array}{cccc} f'_{k,1} & 0 & \cdots & 0 \\ 0 & f'_{k,2} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & f'_{k,L} \end{array} \right] \]
Graidents: Algebraic Formulation

\[ \nabla_{W_k} E = \left( f'_k \circ \nabla_{f_k} E \right) f^T_{k-1} \]

\[ \nabla_{f_k} E = \left( \nabla_{f_k} f_{k+1} \right) \nabla_{f_{k+1}} E = W^T_{k+1} \left( f'_{k+1} \circ \nabla_{f_{k+1}} E \right) \]
Gradients: Algebraic Formulation

\[ \nabla_{W_k} E = \left( f'_k \circ \nabla_{f_k} E \right) f^T_{k-1} \]

\[ \nabla_{f_k} E = W^{T}_{k+1} \left( f'_k \circ \nabla_{f_{k+1}} E \right) \]

Output layer

\[ \nabla_{f_N} E = \nabla_O \text{Div}(O, Y) = \begin{bmatrix} \partial \text{Div}(O, Y) \\ \partial O_1 \\ \vdots \\ \partial \text{Div}(O, Y) \\ \partial O_N \end{bmatrix} \]
Gradients: Algebraic Formulation

\[ \nabla_{W_k} E = (f_k' \circ (W_{k+1}^T (f_{k+1}') \circ W_{k+2}^T (f_{k+2}') \circ \ldots (W_N^T (f_N' \circ \nabla_{O} \text{Div}(O,Y)))))\]
Gradients: Back Propagation

Forward pass:

Initialize

For k = 1 to N:

\[ f_k = f_k (W_k f_{k-1}) \]

Output

\[ O = f_N \]
Gradients: Back Propagation

Error divergence:

For \( k = N \) downto 1:

\[
\nabla_{f_{N}} E = \nabla_{O} \text{Div}(O, Y)
\]

\[
\nabla_{W_{k}} E = \left( f'_{k} \circ \nabla_{f_{k}} E \right) f_{k-1}^T
\]

\[
\nabla_{f_{k-1}} E = W_{k}^T \left( f'_{k} \circ \nabla_{f_{k}} E \right)
\]
BP: Local Formulation

• The network again
Gradients: Local Computation

- Redrawn
- Separately label input and output of each node
Forward Computation

\[ I_j^1 = \sum_i W_{ij}^1 X_i \]
Forward Computation

\[ I_j^1 = \sum_i W_{ij}^1 X_i \]

\[ I_j^k = \sum_i W_{ij}^k O_{i}^{k-1} \]
**Forward Computation**

\[ I_j^1 = \sum_i W_{ij} X_i \]

\[ I_j^k = \sum_i W_{ij}^k O_{i}^{k-1} \]

\[ O_j^k = f_k(I_j^k) \]
Forward Computation

\[ I^1_j = \sum_i W^1_{ij} X_i \]

\[ O^k_j = f_k(I^k_j) \]

\[ I^k_j = \sum_i W^k_{ij} O^k_{i-1} \]

ITERATE FOR \( k = 1:N \)

for \( j = 1:\text{layer-width} \)
Gradients: Backward Computation

\[ f_{N-1} \rightarrow f_N \rightarrow \text{Div}(O,Y) \rightarrow E \]
Gradients: Backward Computation

\[ \frac{\partial E}{\partial O_i^N} = \frac{\partial \text{Div}(O,Y)}{\partial O_i^N} \]
Gradients: Backward Computation

\[
\frac{\partial E}{\partial O_i^N} = \frac{\partial E}{\partial I_i^N} \frac{\partial E}{\partial O_i^N} = f'_N(I_i^N) \frac{\partial E}{\partial O_i^N}
\]

\[
\frac{\partial E}{\partial O_i^N} = \frac{\partial \text{Div}(O,Y)}{\partial O_i^N}
\]
Gradients: Backward Computation

\[
\frac{\partial E}{\partial O_i^{N-1}} = \sum_j \frac{\partial I_j^N}{\partial O_i^{N-1}} \frac{\partial E}{\partial I_j^N} = \sum_j W_{ij}^N \frac{\partial E}{\partial I_j^N}
\]

Note: \( I_j^N = O_i^{N-1} W_{ij}^N + \text{other stuff} \)
Gradients: Backward Computation

\[
\frac{\partial E}{\partial I_i^k} = f_k'(I_i^k) \frac{\partial E}{\partial O_i^k}
\]

\[
\frac{\partial E}{\partial O_i^N} = \frac{\partial E}{\partial \text{Div}(O,Y)}
\]

\[
\frac{\partial E}{\partial I_i^N} = f_N'(I_i^N) \frac{\partial E}{\partial O_i^N}
\]

\[
\frac{\partial E}{\partial O_i^{N-1}} = \sum_j W_{ij}^N \frac{\partial E}{\partial I_j^N}
\]
Gradients: Backward Computation

\[ \frac{\partial E}{\partial O_{i}^{k-1}} = \sum_{j} \frac{\partial I_{j}^{k}}{\partial O_{i}^{k-1}} \frac{\partial E}{\partial I_{j}^{k}} = \sum_{j} W_{ij} \frac{\partial E}{\partial I_{j}^{k}} \]

\[ \frac{\partial E}{\partial O_{i}^{N}} = \frac{\partial \text{Div}(O,Y)}{\partial O_{i}^{N}} \]

\[ \frac{\partial E}{\partial I_{i}^{k}} = f_{k}'(I_{i}^{k}) \frac{\partial E}{\partial O_{i}^{k}} \]
Gradients: Backward Computation

\[
\frac{\partial E}{\partial W_{ij}} = \frac{\partial I_j^k}{\partial W_{ij}} \frac{\partial E}{\partial I_j^k} = O_i^{k-1} \frac{\partial E}{\partial I_j^k}
\]

\[
\frac{\partial E}{\partial O_i^N} = \frac{\partial \text{Div}(O,Y)}{\partial O_i^N}
\]

\[
\frac{\partial E}{\partial I_i^k} = f'_k(I_i^k) \frac{\partial E}{\partial O_i^k}
\]

\[
\frac{\partial E}{\partial O_i^{k-1}} = \sum_j W_{ij}^k \frac{\partial E}{\partial I_j^k}
\]
Gradients: Backward Computation

Initialize: Gradient w.r.t network output
\[
\frac{\partial E}{\partial O_i^N} = \frac{\partial \text{Div}(O,Y)}{\partial O_i^N}
\]

For \( k = N \ldots 1 \)

For \( i = 1 \text{:layer-width} \)
\[
\frac{\partial E}{\partial I_i^k} = f'_k(I_i^k) \frac{\partial E}{\partial O_i^k}
\]
\[
\frac{\partial E}{\partial O_i^{k-1}} = \sum_j W_{ij}^k \frac{\partial E}{\partial I_i^k}
\]
\[
\frac{\partial E}{\partial W_{ij}^k} = O_i^{k-1} \frac{\partial E}{\partial I_j^k}
\]
Backpropagation: Multiplicative Networks

Some types of networks have multiplicative combination

Forward:

Backward:
Overall Approach

- For each data instance
  - **Forward pass**: Pass instance forward through the net. Store all intermediate outputs of all computation
  - **Backward pass**: Sweep backward through the net, iteratively compute all derivatives w.r.t weights
- Actual Error is the sum of the error over all training instances
  \[ E = \sum_x \text{Div}(y(x), o(x)) = \sum_x E(x) \]
- Actual gradient is the sum or average of the derivatives computed for each training instance
  \[ \nabla_W E = \sum_x \nabla_W E(x) \]
  \[ W \leftarrow W - \eta \nabla_W E \]
Issues and Challenges

• What does it learn?
• Speed
  – Will not address this
• Does it do what we want it to do?

• Next class:
  – Variations of nnets
    • MLP, Convolution, recurrence
  – Nnets for various tasks
    • Image recognition, speech recognition, signal enhancement, modelling language..