Neural Networks
Neural Network Neurons

- Receives n inputs (plus a bias term)
- Multiplies each input by its weight
- Applies activation function to the sum of results
- Outputs result
Activation Functions

- Given the weighted sum as input, an activation function controls whether a neuron is “active” or “inactive”
- Commonly used activation functions:
  - **Threshold function**: outputs 1 when input is positive and 0 otherwise (similar to perceptron)
  - **Sigmoid function**:
    \[
    \frac{1}{1+e^{-x}}
    \]
    Differentiable – a good property for optimization
Basic Multilayer Neural Network

- Each layer receives its inputs from the previous layer and forwards its outputs to the next – feed forward structure
- Output layer: sigmoid activation function for classification, and linear activation function for regression
- Referred to as a two-layer network (2 layer of weights)
Representational Power

• **Any Boolean Formula**
  - Consider a formula in disjunctive normal form:
    \[(x_1 \land \neg x_2) \lor (x_2 \land x_3)\]

```
1 -1
-1.5 -0.5
1
1
1
-0.5
1 -1
-1.5 1 1
1
1
1
1
```

OR units

AND units
Example

Neural net is one of the most effective methods when the data include complex sensory inputs such as images.
left  str  rght  up

Learned Weights

30x32 inputs

Typical input images
Training: Backpropagation

• It is traditional to train neural networks to minimize the squared-error

\[ J(W) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2 \]

• This is appropriate for regression problems, but for classification problems it is more appropriate to optimize log-likelihood

• But we will look at MSE as our example

• Gradient descent: must apply the chain rule many times to compute the gradient
Terminology

- $\mathbf{X} = [1, x_1, x_2, x_3, x_4]^T$ – the input vector with the bias term
- $\mathbf{A} = [1, a_6, a_7, a_8]^T$ – the output of the hidden layer with the bias term
- $\mathbf{W}_i$ represents the weight vector leading to node $i$
- $w_{i,j}$ represents the weight connecting from the $j$-th node to the $i$-th node
  - $w_{9,6}$ is the weight connecting from $a_6$ to $a_9$
- We will use $\sigma$ to represent the activation function, so
  \[
  \hat{y} = \sigma(W_9 \cdot [1, a_6, a_7, a_8]^T) = \sigma(W_9 \cdot [1, \sigma(W_6 \cdot \mathbf{X}), \sigma(W_7 \cdot \mathbf{X}), \sigma(W_8 \cdot \mathbf{X})]^T)
  \]
Mean Squared Error

- We adjust the weights of the neural network to minimize the mean squared error (MSE) on training set.

\[
J(W) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2
\]

\[
J_i(W) = \frac{1}{2} (\hat{y}^i - y^i)^2
\]

- **Useful fact:** the derivative of the sigmoid activation function is

\[
\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))
\]
Gradient Descent: Output Unit

\[ w_{9,6} \text{ is a component of } W_9, \text{ connecting from } a_6 \text{ to the output node.} \]

\[
\frac{\partial J_i(W)}{\partial w_{9,6}} = \frac{\partial}{\partial w_{9,6}} \frac{1}{2} (\hat{y}^i - y^i)^2 \\
= \frac{1}{2} \cdot 2 \cdot (\hat{y}^i - y^i) \cdot \frac{\partial}{\partial w_{9,6}} (\sigma(W_9 \cdot A^i) - y^i) \\
= (\hat{y}^i - y^i) \cdot \sigma(W_9 \cdot A^i) \cdot (1 - \sigma(W_9 \cdot A^i)) \cdot \frac{\partial}{\partial w_{9,6}} W_9 \cdot A^i \\
= (\hat{y}^i - y^i) \hat{y}^i (1 - \hat{y}^i) \cdot a_6^i
\]
The Delta Rule

- Define
  \[ \delta_9^i = (\bar{y}^i - y^i)\bar{y}^i(1 - \bar{y}^i) \]

then
  \[ \frac{\partial J_i(W)}{\partial w_{9,6}} = (\bar{y}^i - y^i)\bar{y}^i(1 - \bar{y}^i) \cdot a_6^i \]

  \[ = \delta_9^i \cdot a_6^i \]
Derivation: Hidden Units

\[ \frac{\partial J_i(W)}{\partial w_{6,2}} = \frac{1}{2} \frac{\partial}{\partial w_{6,2}} (\hat{y}^i - y^i)^2 \]

\[ = (\hat{y}^i - y^i) \cdot \sigma(W_9 \cdot A^i)(1 - \sigma(W_9 \cdot A^i)) \cdot \frac{\partial}{\partial w_{6,2}} (W_9 \cdot A^i) \]

\[ = \delta_9 \cdot w_{9,6} \cdot \frac{\partial}{\partial w_{6,2}} \sigma(W_6 \cdot X^i) \]

\[ = \delta_9 \cdot w_{9,6} \cdot \sigma(W_6 \cdot X^i)(1 - \sigma(W_6 \cdot X^i)) \cdot \frac{\partial}{\partial w_{6,2}} (W_6 \cdot X^i) \]

\[ = \delta_9 \cdot w_{9,6} \cdot a_6(1 - a_6) \cdot x_2^i \]
Delta Rule for Hidden Units

Define \( \delta_6^i = \delta_9^i \cdot w_{9,6} \cdot a_6^i (1 - a_6^i) \)
and rewrite as

\[
\frac{\partial J_i(W)}{\partial w_{6,2}} = \delta_6^i \cdot x_2^i.
\]
 Networks with Multiple Output Units

\[ \delta_9 = (\hat{y}_1 - y_1)\hat{y}_1(1 - \hat{y}_1) \]
\[ \delta_{10} = (\hat{y}_2 - y_2)\hat{y}_2(1 - \hat{y}_2) \]

- We get a separate contribution to the gradient from each output unit.
- Hence, for input-to-hidden weights, we must sum up the contributions:

\[ \delta_6 = a_6(1 - a_6)(w_{9,6}\delta_9 + w_{10,6}\delta_{10}) \]
Backpropagation Training

- Initialize all the weights with small random values
- Repeat
  - For all training examples, do
    Begin Epoch
      For each training example do
        – Compute the network output
        – Compute the error
        – Backpropagate this error from layer to layer and adjust weights to decrease this error
    End Epoch
Backpropagation for a Single Example

- **Forward Pass** Given $x$, compute $a_u$ and $\hat{y}_v$ for hidden units $u$ and output units $v$.
- **Compute Errors** Compute $\epsilon_v = (\hat{y}_v - y_v)$ for each output unit $v$.
- **Compute Output Deltas**
  \[ \delta_v = (\hat{y}_v - y_v)\hat{y}_v(1 - \hat{y}_v) \]
- **Compute Hidden Deltas**
  \[ \delta_u = a_u(1 - a_u)\Sigma_v w_{v,u} \delta_v \]
- **Compute Gradient**
  - Compute $\frac{\partial J_i}{\partial w_{v,u}} = \delta_v a^i_u$ for hidden-to-output weights.
  - Compute $\frac{\partial J_i}{\partial w_{u,j}} = \delta_u x^i_j$ for input-to-hidden weights.
- **Take Gradient Descent Step**
  \[ w_{v,u} \leftarrow w_{v,u} - \eta \frac{\partial J_i}{\partial w_{v,u}} \]
  \[ w_{u,j} \leftarrow w_{u,j} - \eta \frac{\partial J_i}{\partial w_{u,j}} \]
Remarks on Training

• No guarantee of convergence, may oscillate or reach a local minima.
• However, in practice many large networks can be adequately trained on large amounts of data for realistic problems, e.g.,
  – Driving a car
  – Recognizing handwritten zip codes
  – Play world championship level Backgammon
• Many epochs (thousands) may be needed for adequate training, large data sets may require hours or days of CPU time.
• Termination criteria can be:
  – Fixed number of epochs
  – Threshold on training set error
  – Increased error on a validation set
• To avoid local minima problems, can run several trials starting from different initial random weights and:
  – Take the result with the best training or validation performance.
  – Build a committee of networks that vote during testing, possibly weighting vote by training or validation accuracy
Notes on Proper Initialization

• Start in the “linear” regions
  – keep all weights near zero, so that all sigmoid units are in their
    linear regions. This makes the whole net the equivalent of one
    linear threshold unit—a relatively simple function.
  – This will also avoid having very small gradient

• Break symmetry
  – If we start with all the weights equal, what would happen?
  – Ensure that each hidden unit has different input weights so that
    the hidden units move in different directions.

• Set each weight to a random number in the range

\[ [-1, +1] \times \frac{1}{\sqrt{\text{fan-in}}} \]

where the “fan-in” of weight \( w_{v,u} \) is the number of inputs to
unit \( v \).
Batch, Online, and Online with Momentum

- **Batch.** Sum the $\nabla_w J_i(W)$ for each example $i$. Then take a gradient descent step.
- **Online.** Take a gradient descent step with each $\nabla_w J_i(W)$ as it is computed (this is the algorithm we described).
- **Momentum factor.** Make the $t+1$-th update dependent on the $t$-th update

\[
\Delta W^{(t+1)} = \nabla_w J(W^t)
\]

\[
\Delta W^{(t+1)} = \alpha \Delta W^{(t)} + \nabla_w J(W^t)
\]

$\alpha$ is called the momentum factor, and typically take values in the range $[0.7, 0.95]$. This tends to keep weight moving in the same direction and improves convergence.
Overtraining Prevention

- Running too many epochs may overtrain the network and result in overfitting.

- Keep a validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and return it when performance decreases significantly beyond this.

- To avoid losing training data to validation:
  - Use 10-fold cross-validation to determine the average number of epochs that optimizes validation performance.
  - Train on the full data set using this many epochs to produce the final result.
Over-fitting Prevention

- Too few hidden units prevent the system from adequately fitting the data and learning the concept.
- Too many hidden units leads to over-fitting.

- Similar cross-validation method can be used to decide an appropriate number of hidden units.
- Another approach to preventing over-fitting is **weight decay**, in which we multiply all weights by some fraction between 0 and 1 after each epoch.
  - Encourages smaller weights and less complex hypotheses.
  - Equivalent to including an additive penalty in the error function proportional to the sum of the squares of the weights of the network.
Input/Output Coding

• Appropriate coding of inputs/outputs can make learning easier and improve generalization.

• Best to encode discrete multi-category features using multiple input units and include one binary unit per value.

• Continuous inputs can be handled by a single input unit, but scaling them between 0 and 1.

• For classification problems, best to have one output unit per class.
  – Continuous output values then represent certainty in various classes.
  – Assign test instances to the class with the highest output.

• Use target values of 0.9 and 0.1 for binary problems rather than forcing weights to grow large enough to closely approximate 0/1 outputs.

• Continuous outputs (regression) can also be handled by scaling to the range between 0 and 1.
Softmax for multi-class classification

- For K classes, we have K nodes in the output layer, one for each class
- Let $a_k$ be the output of the class-$k$ node, i.e. $a_k = (w_k \cdot A)$, where $A$ is the output of the hidden layer, and $w_k$ is the weight vector leading into the class-$k$ node
- We define: $P(y = k|\mathbf{x}) = \frac{\exp a_k}{\sum_{i=1}^{K} \exp a_i}$
Likelihood objective

• Given an instance \((x^i, y^i)\), we compute the outputs for \(x^i\): \(\hat{y}^i_w(1), \hat{y}^i_w(2), \ldots, \hat{y}^i_w(K)\)
  
• These will be the probabilities computed using the softmax, and they will be a function of the weights of the neural net (thus the subscript \(w\))

• Log-likelihood function:

\[
J(w) = \sum_{i=1}^{N} \sum_{j=1}^{K} y^i(j) \log \hat{y}^i_w(j)
\]

• Similar gradient descent algorithm could be derived
Recent Development

• A recent trend in ML is deep learning, which learns feature hierarchies from large amounts of unlabeled data
• The feature hierarchies are expected to capture the inherent structure in the data
• Can often lead to better classification when used the learned features to train with labeled data
• Neural networks provide one approach for deep learning
Shallow vs Deep Architectures

Traditional shallow architecture

Image Video Text ……

Hand-designed feature extraction  →  Trainable classifier  →  Class label

Deep architecture

Image Video Text ……

Layer 1  →  ……  →  Layer N  →  Simple classifier  →  Class label

Learned feature representation
Deep learning with Auto-encoders

- Network is trained to output the input
- The hidden layer serves to “extract” features from the input that is essential for representation
  - Constraining layer 2 to be sparse
- Training is carried out with the same back-prop procedure
Deep structure: Stacked Auto-encoders

- This continues … and learns a feature hierarchy
- The final feature layer can then be used as features for supervised learning
- Can also do supervised training on the entire network to fine-tune all weights
• That’s the basic idea
• There are many many types of deep learning
• Different kinds of autoencoder, variations on architectures and training algorithms, etc…
• Various packages: Theano, Caffe, Torch, Convnet …
• Tremendous impact in vision, speech and natural language processing
• Very fast growing area …

To learn more about deep learning, take Dr. Fuxin Li’s CS519 next term