
Mark Hasegawa-Johnson
All content CC-BY 4.0 unless otherwise specified.

ECE 537, Fall 2022
1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Two-Layer Feedforward Neural Network

\[ \vec{z} = h(\vec{x}, U, V) \]

\[ z_\ell = g(b_\ell) \quad \vec{z} = g(\vec{b}) \]

\[ b_\ell = v_{k0} + \sum_{k=1}^{q} v_{\ell k} y_k \quad \vec{b} = V \vec{y} \]

\[ y_k = f(a_k) \quad \vec{y} = f(\vec{a}) \]

\[ a_k = u_{k0} + \sum_{j=1}^{p} u_{kj} x_j \quad \vec{a} = U \vec{x} \]

\[ \vec{x} \text{ is the input vector} \]
Neural Network = Universal Approximator

Assume . . .

- Linear Output Nodes: \( g(b) = b \)
- Smoothly Nonlinear Hidden Nodes: \( f'(a) = \frac{df}{da} \) finite
- Smooth Target Function: \( \vec{z} = h(\vec{x}, U, V) \) approximates \( \vec{\zeta} = h^*(\vec{x}) \in \mathcal{H} \), where \( \mathcal{H} \) is some class of sufficiently smooth functions of \( \vec{x} \) (functions whose Fourier transform has a first moment less than some finite number \( C \))
- There are \( q \) hidden nodes, \( y_k, 1 \leq k \leq q \)
- The input vectors are distributed with some probability density function, \( p(\vec{x}) \), over which we can compute expected values.

Then (Barron, 1993) showed that . . .

\[
\max_{h^*(\vec{x}) \in \mathcal{H}} \min_{U,V} E \left[ h(\vec{x}, U, V) - h^*(\vec{x}) \right]^2 \leq O \left( \frac{1}{q} \right)
\]
Neural Network Problems: Outline of Remainder of this Talk

1. **Knowledge-Based Design.** Given $U, V, f, g$, what kind of function is $h(\vec{x}, U, V)$? Can we draw $\vec{z}$ as a function of $\vec{x}$? Can we heuristically choose $U$ and $V$ so that $\vec{z}$ looks kinda like $\vec{\zeta}$?

2. **Error Metric.** In what way should $\vec{z} = h(\vec{x})$ be “similar to” $\vec{\zeta} = h^*(\vec{x})$?

3. **Local Optimization: Gradient Descent with Back-Propagation.** Given an initial $U, V$, how do I find $\hat{U}, \hat{V}$ that more closely approximate $\vec{\zeta}$?

4. **Global Optimization: Simulated Annealing.** How do I find the globally optimum values of $U$ and $V$?
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Synapse, First Layer: $a_k = u_{k0} + \sum_{j=1}^{2} u_{kj} x_j$
Axon, First Layer: $y_k = \tanh(a_k)$

First Layer, kth Axon, $y_k(x_1, x_2)$

First Layer, kth Axon, $y_k(x_1, 0)$
Synapse, Second Layer: \( b_\ell = v_{\ell 0} + \sum_{k=1}^{2} v_{\ell k} y_k \)
Axon, Second Layer: $z_\ell = \text{sign}(b_\ell)$
**Step and Logistic nonlinearities**

- **Unit Step:** $g(b) = u(b)$

  ![Unit Step Graph](image)

- **Logistic:** $g(b) = 1/(1+e^{-b})$

  ![Logistic Graph](image)

**Signum and Tanh nonlinearities**

- **Signum:** $g(b) = \text{sign}(b)$

  ![Signum Graph](image)

- **Tanh:** $g(b) = (e^b - e^{-b})/(e^b + e^{-b})$

  ![Tanh Graph](image)
“Linear Nonlinearity” and ReLU

Max and Softmax

Max:

\[
z_{\ell} = \begin{cases} 
1 & b_{\ell} = \max_m b_m \\
0 & \text{otherwise}
\end{cases}
\]

Softmax:

\[
z_{\ell} = \frac{e^{b_{\ell}}}{\sum_m e^{b_m}}
\]
Outline

1 Intro
2 Knowledge-Based Design
3 Error Metric
4 Example
5 Gradient Descent
6 Example: Gradient Descent
7 Simulated Annealing
8 Example: Simulated Annealing
9 Conclusions
Error Metric: How should $h(\vec{x})$ be “similar to” $h^*(\vec{x})$?

Linear output nodes:

**Minimum Mean Squared Error (MMSE)**

$$U^*, V^* = \arg \min E_n = \arg \min \frac{1}{n} \sum_{i=1}^{n} |\vec{\zeta}_i - \vec{z}(x_i)|^2$$

**MMSE Solution:** \(\vec{z} = E\left[\vec{\zeta} | \vec{x}\right]\)

If the training samples \((\vec{x}_i, \vec{\zeta}_i)\) are i.i.d., then

$$E_\infty = E\left[|\vec{\zeta} - \vec{z}|^2\right]$$

\(E_\infty\) is minimized by

$$\vec{z}_{MMSE}(\vec{x}) = E\left[\vec{\zeta} | \vec{x}\right]$$
Error Metric: How should \( h(\vec{x}) \) be “similar to” \( h^*(\vec{x}) \)?

Logistic output nodes:

<table>
<thead>
<tr>
<th>Binary target vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppose</td>
</tr>
</tbody>
</table>
| \( \zeta_\ell = \begin{cases} 
1 & \text{with probability } P_\ell(\vec{x}) \\
0 & \text{with probability } 1 - P_\ell(\vec{x}) 
\end{cases} \) |
| and suppose \( 0 \leq z_\ell \leq 1 \), e.g., logistic output nodes. |

| MMSE Solution: \( z_\ell = \Pr \{ \zeta_\ell = 1 | \vec{x} \} \) |
|-----------------|
| \[ E [ \zeta_\ell | \vec{x} ] = 1 \cdot P_\ell(\vec{x}) + 0 \cdot (1 - P_\ell(\vec{x})) \] |
| \[ = P_\ell(\vec{x}) \] |
| So the MMSE neural network solution is |
| \( z_{\ell, MMSE}(\vec{x}) = P_\ell(\vec{x}) \) |
Error Metric: How should $h(\vec{x})$ be “similar to” $h^*(\vec{x})$?

Softmax output nodes:

One-Hot Vector, CE Solution: $z_\ell = \Pr \{ \zeta_\ell = 1 | \vec{x} \}$

- Suppose $\vec{\zeta}_i$ is a “one hot” vector, i.e., only one element is “hot” ($\zeta_{\ell(i),i} = 1$), all others are “cold” ($\zeta_{mi} = 0$, $m \neq \ell(i)$).
- MMSE will approach the solution $z_\ell = \Pr \{ \zeta_\ell = 1 | \vec{x} \}$, but there’s no guarantee that it’s a correctly normalized pmf ($\sum z_\ell = 1$) until it has fully converged.
- CE also approaches $z_\ell = \Pr \{ \zeta_\ell = 1 | \vec{x} \}$, and guarantees that $\sum z_\ell = 1$. CE is also more computationally efficient, if $\vec{\zeta}$ is a one-hot vector.

CE = Cross-Entropy

$$D_n = -\frac{1}{n} \sum_{i=1}^{n} \sum_{\ell=1}^{r} \zeta_{\ell,i} \log z_{\ell,i} = -\frac{1}{n} \sum_{i=1}^{n} \log z_{\ell(i),i}$$
Error Metrics Summarized

- Use MSE to achieve \( \vec{z} = E \left[ \vec{\zeta} | \vec{x} \right] \). That’s almost always what you want.

- If \( \vec{\zeta} \) is a one-hot vector, then use CE (with a softmax nonlinearity on the output nodes) to guarantee that \( \vec{z} \) is a properly normalized probability mass function, and for better computational efficiency.

- If \( \zeta_\ell \) is signed binary (\( \zeta_\ell \in \{-1, +1\} \)), then use MSE (with a tanh nonlinearity) to achieve \( z_\ell = E \left[ \zeta_\ell | \vec{x} \right] \).

- After you’re done training, you can make your cell phone app more efficient by throwing away the uncertainty:
  - Replace softmax output nodes with max
  - Replace logistic output nodes with unit-step
  - Replace tanh output nodes with signum
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Here's the dataset

WS15 ANN Lab, Reference Labels, ER= 0
Knowledge-based design: set each row of \( U \) to be a line segment, 
\[ u_0 + u_1 x_1 + u_2 x_2 = 0, \]
on the decision boundary. 
\( u_0 \) is an arbitrary scale factor; \( u_0 = -20 \) makes the \( \tanh \) work well.

\[
[x1,x2]=
ginput(2);
\]
\( u0=-20; \quad \% \text{Arbitrary scale factor} \)
\( u = -\text{inv}([x1,x2])*[u0;u0]; \)
\( U(1,:) = [u0,u(1),u(2)]; \)
Check your math by plotting $x_2 = -\frac{u_0}{u_2} - \frac{u_1}{u_2}x_1$

```matlab
nnplot(X,ZETA,ZETA,'Reference Labels',1);
hold on;
plot([0,1],-(u0/u(2))+[0,-u(1)/u(2)],'g-');
hold off;
```
Here are 3 such segments, mapping out the lowest curve:

for m=1:3,
plot([0 1],-U(m,1)/U(m,3)+[0,-U(m,2)/U(m,3)]);
end
(1) Reflect through $x_2 = -0.75$, and (2) Shift upward:

$$U_{foo} = [U; U(:,1)-1.5*U(:,3),U(:,2),-U(:,3)];$$

$$U_{bar} = [U_{foo}; U_{foo}-[0.5*U_{foo}(:,3),zeros(6,2)]];$$

$$U = [U_{bar}; U_{bar} - [U_{bar}(:,3),zeros(12,2)]];$$
nnclassify.m: Error Rate = 14%

function [Z,Y]=nnclassify(X,U,V)
Y = tanh(U*[ones(1,n); X]);
Z = tanh(V*[ones(1,n); Y]);
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Gradient Descent = Local Optimization

Neural Net Error Surface (Schematic)
Gradient Descent = Local Optimization

Given an initial $U, V$, find $\hat{U}, \hat{V}$ with lower error.

$$\hat{u}_{kj} = u_{kj} - \eta \frac{\partial E_n}{\partial u_{kj}}$$

$$\hat{v}_{\ell k} = v_{\ell k} - \eta \frac{\partial E_n}{\partial v_{\ell k}}$$

$\eta =$Learning Rate

- If $\eta$ too large, gradient descent won’t converge. If too small, convergence is slow. Usually we pick $\eta \approx 0.001$ and cross our fingers.
- Second-order methods like L-BFGS and Adam choose an optimal $\eta$ at each step, so they’re MUCH faster.
Computing the Gradient

OK, so how do we find \( \frac{\partial E}{\partial v_{\ell k}} \) and \( \frac{\partial E}{\partial u_{kj}} \)?
\[ \vec{z} = h(\vec{x}, U, V) \]

\[ z_\ell = g(b_\ell) \quad \vec{z} = g(\vec{b}) \]

\[ b_\ell = v_{k0} + \sum_{k=1}^{q} v_{\ell k} y_k \quad \vec{b} = V \vec{y} \]

\[ y_k = f(a_k) \quad \vec{y} = f(\vec{a}) \]

\[ a_k = u_{k0} + \sum_{j=1}^{p} u_{kj} x_j \quad \vec{a} = U \vec{x} \]

\( \vec{x} \) is the input vector

**Computing the Gradient**

**Answer:** Use the chain rule.

\[
\frac{\partial E_n}{\partial v_{\ell k}} = \sum_{i=1}^{n} \left( \frac{\partial E_n}{\partial b_{\ell i}} \right) \left( \frac{\partial b_{\ell i}}{\partial v_{\ell k}} \right) = \sum_{i=1}^{n} \left( \frac{\partial E_n}{\partial b_{\ell i}} \right) y_k
\]
\[ \tilde{z} = h(\tilde{x}, U, V) \]
\[ z_\ell = g(b_\ell) \quad \tilde{z} = g(\tilde{b}) \]
\[ b_\ell = v_{k0} + \sum_{k=1}^{q} v_{\ell k} y_k \quad \tilde{b} = V \tilde{y} \]
\[ y_k = f(a_k) \quad \tilde{y} = f(\tilde{a}) \]
\[ a_k = u_{k0} + \sum_{j=1}^{p} u_{kj} x_j \quad \tilde{a} = U \tilde{x} \]
\[ \tilde{x} \text{ is the input vector} \]

**Chain Rule: Step 1, Back to } b_\ell \]

\[ E_n = \frac{1}{2} \sum_{i} \sum_{\ell} (z_{\ell i} - \zeta_{\ell i})^2 \quad \Rightarrow \quad \frac{\partial E_n}{\partial b_{\ell i}} = \frac{2}{n} (z_{\ell i} - \zeta_{\ell i}) g'(b_{\ell i}) \]
\[
\begin{align*}
\vec{z} &= h(\vec{x}, U, V) \\
\vec{z}_\ell &= g(b_\ell) \\
b_\ell &= v_{k0} + \sum_{k=1}^{q} v_{\ell k} y_k \\
\vec{b} &= V \vec{y} \\
y_k &= f(a_k) \\
\vec{y} &= f(\vec{a}) \\
a_k &= u_{k0} + \sum_{j=1}^{p} u_{kj} x_j \\
\vec{a} &= U \vec{x} \\
\vec{x} \text{ is the input vector}
\end{align*}
\]

Back-Propagating to the First Layer

\[
\frac{\partial E_n}{\partial u_{kj}} = \sum_{i=1}^{n} \left( \frac{\partial E_n}{\partial a_{ki}} \right) \left( \frac{\partial a_{ki}}{\partial u_{kj}} \right) = \sum_{i=1}^{n} \left( \frac{\partial E_n}{\partial a_{ki}} \right) x_{ji}
\]

where...

\[
\left( \frac{\partial E_n}{\partial a_{ki}} \right) = \sum_{\ell=1}^{r} \left( \frac{\partial E_n}{\partial b_{\ell i}} \right) v_{\ell k} f'(a_{ki})
\]
Gradient Descent: Batch Mode = Matrix Multiplications

\[ \hat{V} = V - \eta FY^T, \quad \hat{U} = U - \eta DX^T \]

\[ Y = [\vec{y}_1, \ldots, \vec{y}_n], \quad X = [\vec{x}_1, \ldots, \vec{x}_n] \]

\[ F = \left[ \nabla_{\vec{b}_1} E_n, \ldots, \nabla_{\vec{b}_n} E_n \right], \quad D = \left[ \nabla_{\vec{a}_1} E_n, \ldots, \nabla_{\vec{b}_n} E_n \right] \]
Derivatives of the Nonlinearities

Logistic

Logistic: \( g(b) = 1/(1+e^{-b}) \)

Tanh

Tanh: \( g(b) = (e^b - e^{-b})/(e^b + e^{-b}) \)

ReLU

ReLU: \( g(b) = \max(0, b) \)
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
nnclassify.m: Error Rate = 14%

function [Z,Y]=nnclassify(X,U,V)
Y = tanh(U*[ones(1,n); X]);
Z = tanh(V*[ones(1,n); Y]);
nnbackprop.m: Error Rate = 2.8%

function [EPSILON,DELTA]=nnbackprop(X,Y,Z,ZETA,V)
EPSILON = 2* (1-Z.^2) .* (Z-ZETA);
DELTA = (1-Y.^2) .* (V(:,2:(q+1))' * EPSILON);
But with random initialization: Error Rate = 28%

\[
U_{\text{rand}} = [0.02 \cdot \text{randn}(q,p+1)];
\]
\[
V_{\text{rand}} = [0.02 \cdot \text{randn}(r,q+1)];
\]
\[
[U_c, V_c] = \text{nndescent}(X, ZETA, U_{\text{rand}}, V_{\text{rand}}, 0.1, 1000);
\]
\[
[Z_c, Y_c] = \text{nnclassify}(X, U_c, V_c);
\]
Simulated Annealing: How can we find the globally optimum $U, V$?

- Gradient descent finds a local optimum. The $\hat{U}, \hat{V}$ you end up with depends on the $U, V$ you started with.
- How can you find the **global optimum** of a non-convex error function?
- The answer: Add randomness to the search, in such a way that...

$$P(\text{reach global optimum}) \xrightarrow{t \to \infty} 1$$
- Take a random step. If it goes downhill, do it.
- Take a random step. If it goes downhill, do it.
- If it goes uphill, SOMETIMES do it.
- Take a random step. If it goes downhill, do it.
- If it goes uphill, SOMETIMES do it.
- Uphill steps become less probable as $t \rightarrow \infty$
Simulated Annealing: Algorithm

FOR $t = 1$ TO $\infty$, DO

1. Set $\hat{U} = U + \text{RANDOM}$

2. If your random step caused the error to decrease ($E_n(\hat{U}) < E_n(U)$), then set $U = \hat{U}$
   (prefer to go downhill)

3. Else set $U = \hat{U}$ with probability $P$
   (\ldots but sometimes go uphill!)
   
   1. $P = \exp\left(-\frac{E_n(\hat{U}) - E_n(U)}{\text{Temperature}}\right)$
   (Small steps uphill are more probable than big steps uphill.)
   
   2. Temperature $= \frac{T_{max}}{\log(t + 1)}$
   (Uphill steps become less probable as $t \to \infty$.)

4. Whenever you reach a local optimum ($U$ is better than both the preceding and following time steps), check to see if it’s better than all preceding local optima; if so, remember it.
(Hajek, 1985) proved that, if we start out in a “valley” that is separated from the global optimum by a “ridge” of height $T_{max}$, and if the temperature at time $t$ is $T(t)$, then simulated annealing converges in probability to the global optimum if

$$\sum_{t=1}^{\infty} \exp\left(-\frac{T_{max}}{T(t)}\right) = +\infty$$

For example, this condition is satisfied if

$$T(t) = \frac{T_{max}}{\log(t + 1)}$$
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Intro
Design
Metric
Example
Gradient
Example: Gradient Descent
Annealing
Example: Simulated Annealing
Conclusions
## Simulated Annealing: More Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>c or $\alpha$</th>
<th>$t$</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hajek Cooling ($T = \frac{c}{\log(t + 1)}$)</td>
<td>1</td>
<td>52356</td>
<td>5.1%</td>
</tr>
<tr>
<td></td>
<td>$10^{-4}$</td>
<td>1800</td>
<td>0.70%</td>
</tr>
<tr>
<td>Geometric Annealing ($T(t) = \alpha T(t - 1)$)</td>
<td>0.7</td>
<td>500</td>
<td>0.43%</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>500</td>
<td>0.40%</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>500</td>
<td>0.80%</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>500</td>
<td>0.43%</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>500</td>
<td>0.40%</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>500</td>
<td>0.80%</td>
</tr>
</tbody>
</table>
Outline

1. Intro
2. Knowledge-Based Design
3. Error Metric
4. Example
5. Gradient Descent
6. Example: Gradient Descent
7. Simulated Annealing
8. Example: Simulated Annealing
9. Conclusions
Conclusions

- Back-prop gives you a locally optimal set of weights.
  \[ \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L} \]

- Initialization
  - Random initialization doesn’t always work.
  - Knowledge-based initialization sometimes works.
  - Simulated annealing can recover from a bad starting point, but it can take a long time to train.