Deep Learning Intuition

Ahmed Hosny

SAM Joint Imaging-Therapy Scientific Symposium (Certificate Series Session 2)
Machine Learning for Radiomics - Wednesday, 8/1/2018 10:15 AM - 12:15 PM
Deep Learning

Early efforts
AI with subhuman performance is occasionally used in commercial expert systems with varying degrees of utility

Current state
Narrow task-specific AI has started to match and, in some instances, exceed human performance in tasks including conversational speech recognition, driving vehicles, playing Go and classifying skin cancer

Future outlook
General AI exceeds human performance and reasoning in complex tasks, including writing best-selling novels and performing surgery. Human intelligence improves as we learn from AI
Deep Learning

a) Predefined engineered features + traditional machine learning
   - Feature engineering
   - Histogram
   - Texture
   - Expert knowledge
   - Shape
   - Selection
   - Classification

b) Deep learning
   - Input
   - Hidden layers
     - Increasingly higher-level features
   - Output
   - Convolution layers for feature map extraction
   - Pooling layers for feature aggregation
   - Fully connected layers for classification
What is the intuition behind neural networks?

How do neural networks learn?

How to train neural networks?
What is the intuition behind neural networks?

How do neural networks learn?

How to train neural networks?
Machine Learning: 4 Main Components

- Data
- Model/Representation
- Cost/Error/Loss of model
- Model Optimizer
Logistic Regression

![Graph showing nodules with malignant and benign classifications based on nodule concavity and radius.](image-url)
Logistic Regression

\[ y = wx + b \]

- Red: malignant
- Green: benign

Axes:
- Nodule concavity
- Nodule radius
Logistic Regression

\[ y = wx + b \]

- Malignant
- Benign

\[ \text{log loss} = 3^* + 23^* + 14^* + 11^* \]
Logistic Regression

\[ y = wx + b \]

- Malignant
- Benign

\[ \log \text{loss} = 3^* + 23^* + 14^* + 11^* \]
Logistic Regression

\[ y = wx + b \]

- **malignant**
- **benign**

\[ \text{log loss} = 3^* \text{ red} + 23^* \text{ red} + 14^* \text{ green} + 11^* \text{ green} \]
Logistic Regression

\[ y = wx + b \]

- malignant
- benign

\[ \log \text{loss} = 2^* \text{malignant} + 24^* \text{benign} + 8^* \text{malignant} + 17^* \text{benign} \]
Logistic Regression

\[ y = wx + b \]

- **malignant**
- **benign**

\[
\text{log loss} = 2^* \quad + 
24^* \quad + 
8^* \quad + 
17^*
\]
Logistic Regression

\[ y = wx + b \]

- malignant
- benign

\[ \text{log loss} = 26w^2 + 25b^2 \]
Dealing with Edge Conditions

- Red circles: malignant
- Green circles: benign

Axes:
- Y-axis: nodule concavity
- X-axis: nodule radius
Dealing with Edge Conditions

The graph shows a scatter plot of nodules with their concavity and radius. The points are color-coded: red for malignant and green for benign. A line is drawn to illustrate the edge conditions.
Dealing with Edge Conditions

- **nodule concavity**
- **nodule radius**

- Malignant
- Benign

Diagram showing a scatter plot of nodules with malignant and benign categories, with a line indicating a decision boundary.
Dealing with Edge Conditions

![Graph showing malignant and benign nodules with a decision boundary]

- Malignant
- Benign

Graphical representation of nodules in terms of nodule concavity and nodule radius.
Dealing with Edge Conditions

![Graph showing nodules with malignant and benign categories. The graph plots nodules based on their radius and concavity, with a circle highlighting a specific area.](image-url)
Dealing with Edge Conditions
Quadrant Questioning

- Red dots represent malignant nodules.
- Green dots represent benign nodules.

The diagram shows a scatter plot with axes for nodule concavity and nodule radius, with a line dividing the quadrants A and B.
Quadrant Questioning

- Line A, over or under?
Quadrant Questioning

- Line A, over or under?
- Line B, over or under?
Quadrant Questioning

- Line A, over or under?
- Line B, over or under?
- Both YES?
Graph Representation

Line A, over or under?

Line B, over or under?
Graph Representation

Line A, over or under?

radius
concavity

Line B, over or under?

radius
concavity
Graph Representation

Line A, over or under?

radius
concavity

YES

Line B, over or under?

radius
concavity

YES
Graph Representation

Line A, over or under?

radius
concavity

YES

AND

Both YES?

Line B, over or under?

radius
concavity

YES
Graph Representation

Line A, over or under?

radius
concavity

Line B, over or under?

radius
concavity

(1,1)

Both YES?

YES
A Neural Network

INPUT — HIDDEN LAYER(S) — OUTPUT
A Neural Network

INPUT

HIDDEN LAYER(S)

OUTPUT

\[ W_1 \text{ radius} \times \text{radius} + W_1 \text{ concavity} \times \text{concavity} + b \]

\[ W_2 \text{ radius} \times \text{radius} + W_2 \text{ concavity} \times \text{concavity} + b \]

\[ f(b) \]

\[(1,1)\]
XOR Perceptron

AND
XOR Perceptron

AND

OR
XOR Perceptron

AND

OR

XOR
XOR Perceptron

\[
\begin{align*}
(0,0) & \quad (0,1) \\
(1,0) & \quad (1,1)
\end{align*}
\]
XOR Perceptron

\[ \text{XOR} \]

\[
(0,0) \quad (0,1) \quad (1,0) \quad (1,1)
\]

\[ = \]

\[
\begin{align*}
\text{Passthrough} & \rightarrow \text{NOT} & \text{AND} & \rightarrow \text{OR} \\
\text{Passthrough} & \rightarrow \text{NOT} & \text{AND} & \\
\end{align*}
\]
XOR Perceptron

\[ (0,0), (1,0), (0,1), (1,1) \]

\[ \text{Passthrough} \]

\[ 0, 1 \]

\[ \text{NOT} \]

\[ \text{AND} \]

\[ \text{OR} \]

Lesson 2: Intro To Neural Networks
Udacity Deep Learning Nanodegree Program
XOR Perceptron

Lesson 2: Intro To Neural Networks
Udacity Deep Learning Nanodegree Program
XOR Perceptron

\[ \begin{align*}
(0,0) & \rightarrow (0,1) \\
(1,0) & \rightarrow (1,1)
\end{align*} \]

\[ \begin{align*}
\text{Passthrough} & \rightarrow 0 \\
\text{NOT} & \rightarrow 0 \\
\text{AND} & \rightarrow 0 \\
\text{OR} & \rightarrow 1
\end{align*} \]
What is the intuition behind neural networks?

How do neural networks learn?

How to train neural networks?
Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton & Ronald J. Williams*

*University of California, Los Angeles, Los Angeles, California 90024, USA

We describe a new learning procedure, backpropagation, for training feedforward networks. The procedure essentially assigns the derivative of the error with respect to the weights of the network to the weights themselves. The error is computed by comparing the network output to the desired output at the network's output unit. The desired output is that which would be produced if the network were to be trained on a particular training example. The calculated error is then backpropagated through the network, and the weights are adjusted in such a way that the error is reduced. This process is repeated for each training example in the training set, and the weights are eventually adjusted to minimize the error on the entire training set.

This approach allows for the rapid and efficient training of large neural networks, and is widely used in modern machine learning applications. The backpropagation procedure is simple, and its implementation is straightforward. It is a powerful tool for training feedforward neural networks, and has been used in a wide variety of applications, including pattern recognition, natural language processing, computer vision, and more.
Iris Dataset

Inputs

- Petal length & width
- Sepal length & width

Outputs

- Iris Versicolor
- Iris Setosa
- Iris Virginica
How Neural Networks Learn

✓ Data: iris dataset

label

features

petal length

petal width

sepal length

sepal width
**How Neural Networks Learn**

- **Data:** iris dataset
- **Model:** 3-layer neural network

\[
S(x) = \frac{1}{1 + e^{-x}}
\]

**softmax**

\[
\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.
\]
How Neural Networks Learn

- **Data:** iris dataset
- **Model:** 3-layer neural network
- **Loss:** cross entropy

![Diagram showing neural network structure with sigmoid and softmax activations]
How Neural Networks Learn

- **Data**: iris dataset
- **Model**: 3-layer neural network
- **Loss**: cross entropy
- **Optimizer**: gradient descent
How Neural Networks Learn

✔ Data: iris dataset
✔ Model: 3-layer neural network
✔ Loss: cross entropy
✔ Optimizer: gradient descent

1. parameter initialization
How Neural Networks Learn

- Data: iris dataset
- Model: 3-layer neural network
- Loss: cross entropy
- Optimizer: gradient descent

1. parameter initialization
2. data input

features:
- petal length: 3.2
- petal width: 2.1
- sepal length: 3.6
- sepal width: 1.7

- sigmoid activation
- softmax activation

label:
- 1
- 0
- 0

Cross entropy

Gradient descent
How Neural Networks Learn

✓ Data: iris dataset
✓ Model: 3-layer neural network
✓ Loss: cross entropy
✓ Optimizer: gradient descent

1. parameter initialization
2. data input
3. forward propagation
How Neural Networks Learn

- **Data**: iris dataset
- **Model**: 3-layer neural network
- **Loss**: cross entropy
- **Optimizer**: gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation

```
label
1 0 0
```

```
cross entropy
↓
gradient descent
```

```
features
```
- petal length: 3.2
- petal width: 2.1
- sepal length: 3.6
- sepal width: 1.7

```
0.27
0.67
0.13
0.90
0.49
0.15
0.55
0.30
```

```
sigmoid activation
softmax activation
```

```
-\log(0.15)
0.82
```

1

0

0
How Neural Networks Learn

✓ **Data:** iris dataset
✓ **Model:** 3-layer neural network
✓ **Loss:** cross entropy
✓ **Optimizer:** gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation

```
<table>
<thead>
<tr>
<th>current</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>1.00</td>
</tr>
<tr>
<td>0.55</td>
<td>0.00</td>
</tr>
<tr>
<td>0.30</td>
<td>0.00</td>
</tr>
</tbody>
</table>
```

- **log(0.15)**: 0.82
- **log(1)**: 0.00

**features**
- petal length: 3.2
- petal width: 2.1
- sepal length: 3.6
- sepal width: 1.7

**Cross entropy**

**Gradient descent**
How Neural Networks Learn

✓ Data: iris dataset
✓ Model: 3-layer neural network
✓ Loss: cross entropy
✓ Optimizer: gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation

<table>
<thead>
<tr>
<th>label</th>
<th>cross entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>gradient descent</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

features

- petal length: 3.2
- petal width: 2.1
- sepal length: 3.6
- sepal width: 1.7

sigmoid activation
softmax activation

<table>
<thead>
<tr>
<th>current</th>
<th>best</th>
<th>worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.15</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0.55</td>
<td>0.60</td>
</tr>
<tr>
<td>0</td>
<td>0.30</td>
<td>0.40</td>
</tr>
</tbody>
</table>

- \(-\log(0.15)\) = 0.82
- \(-\log(1)\) = 0.00
- \(-\log(0)\) = \(\infty\)
How Neural Networks Learn

✅ Data: iris dataset
✅ Model: 3-layer neural network
✅ Loss: cross entropy
✅ Optimizer: gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates

\[ \Delta w \propto -\text{gradient} \]
\[ = -\frac{\partial L}{\partial w} \]
\[ = -\eta \frac{\partial L}{\partial w} \]
How Neural Networks Learn

- **Data:** iris dataset
- **Model:** 3-layer neural network
- **Loss:** cross entropy
- **Optimizer:** gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates

\[ \Delta w \propto -\text{gradient} \]
\[ = -\frac{\partial L}{\partial w} \]
\[ = -\eta \frac{\partial L}{\partial w} \]

Diagram:
- Cross entropy
- Gradient descent
- Features:
  - petal length 3.2
  - petal width 2.1
  - sepal length 3.6
  - sepal width 1.7

Activations:
- Sigmoid
- Softmax
How Neural Networks Learn

✓ Data: iris dataset
✓ Model: 3-layer neural network
✓ Loss: cross entropy
✓ Optimizer: gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates

$$f(f(\text{input}))$$
How Neural Networks Learn

- **Data**: iris dataset
- **Model**: 3-layer neural network
- **Loss**: cross entropy
- **Optimizer**: gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates
How Neural Networks Learn

✓ **Data:** iris dataset
✓ **Model:** 3-layer neural network
✓ **Loss:** cross entropy
✓ **Optimizer:** gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates

\[
\Delta w_n = -\eta \frac{\partial L}{\partial w_n} \\
= -\eta \frac{\partial L}{\partial s_0} \frac{\partial s_0}{\partial n} \frac{\partial n}{\partial w_n}
\]
How Neural Networks Learn

- **Data:** iris dataset
- **Model:** 3-layer neural network
- **Loss:** cross entropy
- **Optimizer:** gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates
How Neural Networks Learn

- **Data:** iris dataset
- **Model:** 3-layer neural network
- **Loss:** cross entropy
- **Optimizer:** gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates

\[
\Delta w_m = -\eta \frac{\partial L}{\partial w_m} = -\eta \frac{\partial L}{\partial s_o} \frac{\partial s_o}{\partial n} \frac{\partial n}{\partial s_i} \frac{\partial s_i}{\partial m} \frac{\partial m}{\partial w_m}
\]
How Neural Networks Learn

- **Data**: iris dataset
- **Model**: 3-layer neural network
- **Loss**: cross entropy
- **Optimizer**: gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates

![Diagram of neural network with sigmoid activation nodes and softmax activation nodes](attachment:image.png)
How Neural Networks Learn

- **Data:** iris dataset
- **Model:** 3-layer neural network
- **Loss:** cross entropy
- **Optimizer:** gradient descent

1. parameter initialization
2. data input
3. forward propagation
4. loss calculation
5. backpropagation + updates
6. repeat 2, 3, 4, & 5

![Diagram of neural network with labels and features](image)
Gradient Descent Flavors

vanilla gradient descent - entire dataset
stochastic gradient descent - random batch of samples (IID)
online gradient descent - (need not be IID)
Gradient Descent Flavors

*vanilla gradient descent* - entire dataset
*stochastic gradient descent* - random batch of samples (IID)
*online gradient descent* - (need not be IID)

learning rate  batch size  # of epochs
What is the intuition behind neural networks?

How do neural networks learn?

How to train neural networks?
The Perfect Fit

Ian Goodfellow, Yoshua Bengio & Aaron Courville
Deep Learning
MIT Press - 2016
The Perfect Fit

parameters vs hyperparameters
Hyperparameters

**model-specific** vs **optimizer-specific**

- architecture
- activations
- initializations
- loss functions
- optimizers
- regularizers

- learning rate
- batch size
- # of epochs
Architecture

# of layers
# of units/layer


Going Deeper with Convolutions (GoogleNet/Inception)
CVPR - 2015
Activations

step

\( \mathbf{X} \) non-differentiable
Activations

**step**
- non-differentiable

**sigmoid**
- smooth + step-like
- good activations close to 0
- activations are bound 0-1
- vanishing gradients
Activations

**step**
- non-differentiable

**sigmoid**
- smooth + step-like
- good activations close to 0
- activations are bound 0-1
- vanishing gradients

**tanh**
- scaled sigmoid
- stronger activations
- vanishing gradients
**Activations**

- **step**
  - ✗ non-differentiable

- **sigmoid**
  - ✓ smooth + step-like
  - ✓ good activations close to 0
  - ✓ activations are bound 0-1
  - ✗ vanishing gradients

- **tanh**
  - ✓ scaled sigmoid
  - ✓ stronger activations
  - ✗ vanishing gradients

---

**ImageNet Classification with Deep Convolutional Neural Networks**

*Advances in Neural Information Processing - NIPS 2012*
Activations

**step**
- Non-differentiable

**sigmoid**
- Smooth + step-like
- Good activations close to 0
- Activations are bound 0-1
- Vanishing gradients

**tanh**
- Scaled sigmoid
- Stronger activations
- Vanishing gradients

**ReLU**
- Sparse activations
- Efficient
- Dead nodes
- Not bound

**Leaky ReLU**
- No dead nodes

---
Kaiming He, Xiangyu Zhang, Shaoqing Ren & Jian Sun
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
International Conference on Computer Vision - ICCV 2015
Activations

- **step**
  - non-differentiable

- **sigmoid**
  - smooth + step-like
  - good activations close to 0
  - activations are bound 0-1
  - vanishing gradients

- **tanh**
  - scaled sigmoid
  - stronger activations
  - vanishing gradients

- **ReLU**
  - sparse activations
  - efficient
  - dead nodes
  - not bound

- **Leaky ReLU**
  - no dead nodes

- **ELU**
  - robust to noise
  - expensive

---

**Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)**

*International Conference on Computer Vision - ICCV 2015*
Activations

**step**
- non-differentiable

**sigmoid**
- smooth + step-like
- good activations close to 0
- activations are bound 0-1
- X vanishing gradients

**tanh**
- scaled sigmoid
- stronger activations
- X vanishing gradients

**ReLU**
- sparse activations
- efficient
- X dead nodes
- X not bound

**Leaky ReLU**
- no dead nodes

**ELU**
- robust to noise
- X expensive
Initializations

0 - stuck at a saddle point

**constants** - difficult to break the symmetry

**large random values** - small gradients, slow convergence
Initializations

Tensorflow initializer distribution: 10k samples

- Zeros
- Ones
- Random normal
- Random uniform
- Glorot normal
- Glorot uniform
- Orthogonal
- Identity
### Table B.2.

<table>
<thead>
<tr>
<th>Name</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>$\alpha = 0$</td>
<td>$\beta = 0$</td>
<td>$\gamma \geq 0$</td>
<td>used by [ZF14]</td>
</tr>
<tr>
<td>→ Xavier/Glorot uniform</td>
<td>$\alpha = \sqrt{\frac{6}{n_{in} + n_{out}}}$</td>
<td>$\beta = 0$</td>
<td>$\gamma = 0$</td>
<td>[GB10]</td>
</tr>
<tr>
<td>→ Xavier/Glorot normal</td>
<td>$\alpha = 0$</td>
<td>$\beta = \left(\frac{2}{n_{in} + n_{out}}\right)^2$</td>
<td>$\gamma = 0$</td>
<td>[GB10]</td>
</tr>
<tr>
<td>→ He</td>
<td>$\alpha = 0$</td>
<td>$\beta = \frac{2}{n_{in}}$</td>
<td>$\gamma = 0$</td>
<td>[HZRS15b]</td>
</tr>
<tr>
<td>Orthogonal</td>
<td>—</td>
<td>—</td>
<td>$\gamma = 0$</td>
<td>[SMG13]</td>
</tr>
<tr>
<td>LSUV</td>
<td>—</td>
<td>—</td>
<td>$\gamma = 0$</td>
<td>[MM15]</td>
</tr>
</tbody>
</table>

Table B.2.: Weight initialization schemes of the form $w \sim \alpha \cdot \mathcal{U}[-1, 1] + \beta \cdot \mathcal{N}(0, 1) + \gamma$.

$n_{in}, n_{out}$ are the number of units in the previous layer and the next layer. Typically, biases are initialized with constant 0 and weights by one of the other schemes to prevent unit-coadaptation. However, dropout makes it possible to use constant initialization for all parameters.

LSUV and Orthogonal initialization cannot be described with this simple pattern.
Loss Functions

**regression** - mean squared error

**multiclass classification** - categorical cross entropy

**pixel classification** - dice/ Wasserstein dice coefficient
Optimizers

stochastic gradient descent + momentum
Optimizers

stochastic gradient descent + momentum

adaptive gradient (AdaGrad)
Optimizers

stochastic gradient descent + momentum

adaptive gradient (AdaGrad)

root mean square propagation (RMSProp)
Optimizers

https://imgur.com/a/Hqolp
Regularizers

L1, L2 regularization

\[ \mathcal{L}_{\text{new}} = \mathcal{L} + \frac{\lambda}{2} |W| \]

\[ \mathcal{L}_{\text{new}} = \mathcal{L} + \frac{\lambda}{2} W^2 \]
Regularizers

L1, L2 regularization

\[ \mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2} |W| \]

\[ \mathcal{L}_{new} = \mathcal{L} + \frac{\lambda}{2} W^2 \]
Regularizers

Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever & Ruslan Salakhutdinov

**Dropout: A Simple Way to Prevent Neural Networks from Overfitting**

*Journal of Machine Learning Research - 2014*
Regularizers

batch normalization

Sergey Ioffe & Christian Szegedy

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
Journal of Machine Learning Research - 2014
Optimizer-specific Hyperparameters

**learning rate**
0.1, 0.01, 0.001, 0.0001,...
Optimizer-specific Hyperparameters

learning rate
0.1, 0.01, 0.001, 0.0001,...
Optimizer-specific Hyperparameters

learning rate
0.1, 0.01, 0.001, 0.0001,...
Optimizer-specific Hyperparameters

learning rate

0.1, 0.01, 0.001, 0.0001,...
Optimizer-specific Hyperparameters

learning rate
0.1, 0.01, 0.001, 0.0001,...

batch size
16, 32, 64, 128,...
Optimizer-specific Hyperparameters

**learning rate**
0.1, 0.01, 0.001, 0.0001,...

batch size
16, 32, 64, 128,...

# of epochs
early stopping
lossfunctions

They are a window to your model's heart.

Contribute loss functions to @karpathy. It doesn't matter if your loss functions are flat, converge, diverge, step or oscillate (or any combination of the above). All loss functions are computed beautiful in their own way and are sought after with equal tenacity.

https://lossfunctions.tumblr.com/
Debugging through Learning Curves
Debugging through Learning Curves

cost/error/loss

epochs/steps/time
Debugging through Learning Curves

![Learning Curve Graphs](image-url)
Debugging through Learning Curves
Debugging through Learning Curves
Debugging through Learning Curves
Debugging through Learning Curves
Thank you!