Today’s outline

• Deep neural networks
  • Computational graph (forward and backward propagation)
• Numerical stability in training
  • Gradient vanishing/exploding
• Generalization and regularization
  • Overfitting, underfitting
  • Weight decay and dropout
Part I: Neural Networks as a Computational Graph
Review: neural networks with one hidden layer

- Input \( x \in \mathbb{R}^d \)
- Hidden \( W^{(1)} \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^m \)
- Intermediate output
Review: neural networks with one hidden layer

- Input $x \in \mathbb{R}^d$
- Hidden $W^{(1)} \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$
- Intermediate output
  \[ h = \sigma(W^{(1)}x + b) \]
  \[ h \in \mathbb{R}^m \]
Review: neural networks with one hidden layer
Review: neural networks with one hidden layer

\[
\begin{align*}
W \in \mathbb{R}^{m \times d} & \quad x \in \mathbb{R}^d & \quad b \in \mathbb{R}^m \\
& + & \\
& = \\
& \in \mathbb{R}^m
\end{align*}
\]
Review: neural networks with one hidden layer

\[
\begin{align*}
\mathbf{W} \in \mathbb{R}^{m \times d} & \quad \mathbf{x} \in \mathbb{R}^{d} \quad \mathbf{b} \in \mathbb{R}^{m} \\
\mathbf{W} \mathbf{x} + \mathbf{b} & = \mathbf{y} \\
\end{align*}
\]

Element-wise activation function

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]
Review: neural networks with one hidden layer

**Key elements:** linear operations + Nonlinear activations

\[ m \times d \quad d \times 1 \quad m \times 1 \quad m \times 1 \]

\[ W \quad x \in \mathbb{R}^d \quad b \]

Element-wise activation function

\[ \frac{1}{1 + e^{-x}} \]
Review: Neural network for k-way classification

- K outputs in the final layer

\[ x \in \mathbb{R}^d \]

Input

Hidden layer
m=3 neurons

\[ h_1 = \sigma(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1) \]

\[ h_2 = \sigma(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2) \]

\[ h_3 = \sigma(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3) \]

Output

\[ f_1 = \sum_{i=1}^{m} h_i w_{1i}^{(2)} + b_1' \]
Review: Neural network for k-way classification

- K outputs units in the final layer

Multi-class classification (e.g., ImageNet with k=1000)
Review: Softmax

Turns outputs $f$ into probabilities (sum up to 1 across $k$ classes)

$$p(y \mid x) = \text{softmax}(f) = \frac{\exp f_y(x)}{\sum_{i}^{k} \exp f_i(x)}$$
Softmax

Turns outputs $f$ into probabilities (sum up to 1 across $k$ classes)

Output layer

Softmax activation function

Probabilities

\[
\begin{bmatrix}
1.3 \\
5.1 \\
2.2 \\
0.7 \\
1.1 \\
\end{bmatrix}
\rightarrow
\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)
\rightarrow
\begin{bmatrix}
0.02 \\
0.90 \\
0.05 \\
0.01 \\
0.02 \\
\end{bmatrix}
\]

Normalized
Deep neural networks (DNNs)

\[ h_1 = \sigma(W_1x + b_1) \]

\[ h_2 = \sigma(W_2h_1 + b_2) \]

\[ h_3 = \sigma(W_3h_2 + b_3) \]

\[ f = W_4h_3 + b_4 \]

\[ y = \text{softmax}(f) \]
Deep neural networks (DNNs)

- \( h_1 = \sigma(W_1x + b_1) \)
- \( h_2 = \sigma(W_2h_1 + b_2) \)
- \( h_3 = \sigma(W_3h_2 + b_3) \)
- \( f = W_4h_3 + b_4 \)
- \( y = \text{softmax}(f) \)

NNs are composition of nonlinear functions
Neural networks as variables + operations

\[ a = \text{sigmoid}(Wx + b) \]

- Decompose functions into atomic operations
- Separate data (variables) and computing (operations)
- Known as a computational graph
Neural networks as a computational graph

- A two-layer neural network
Neural networks as a computational graph

- A two-layer neural network
- Forward propagation vs. backward propagation
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables $Z$
Neural networks: forward propagation

- A two-layer neural network
- Intermediate variables $Z$
Neural networks: forward propagation

• A two-layer neural network
• Intermediate variables Z
Neural networks: forward propagation

• A two-layer neural network
• Intermediate variables $Z$
Neural networks: forward propagation

• A two-layer neural network
• Intermediate variables $Z$
Neural networks: backward propagation

- A two-layer neural network
- Assuming forward propagation is done
- Minimize a **loss function** $L$
Neural networks: backward propagation

• A two-layer neural network
• Assuming forward propagation is done
• Minimize a loss function $L$

$$\frac{\partial L}{\partial z_5} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z_5}$$
Neural networks: backward propagation

• A two-layer neural network
• Assuming forward propagation is done
• Minimize a loss function $L$
Neural networks: backward propagation

- A two-layer neural network
- Assuming forward propagation is done

\[
\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial z_4} \frac{\partial z_4}{\partial z_3}
\]

\[
\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial z_4} \frac{\partial z_4}{\partial W^{(2)}},
\]
Backward propagation: A modern treatment

- Define a neural network as a computational graph
- Must be a directed graph
- Nodes as variables and operations
- All operations must be differentiable
Part II: Numerical Stability
Gradients for Neural Networks

- Compute the gradient of the loss $\ell$ w.r.t. $W_t$

$$\frac{\partial \ell}{\partial W^t} = \frac{\partial \ell}{\partial h^d} \frac{\partial h^d}{\partial h^{d-1}} \cdots \frac{\partial h^{t+1}}{\partial h^t} \frac{\partial h^t}{\partial W^t}$$

Multiplication of many matrices
Two Issues for Deep Neural Networks

Gradient Exploding

\[ \left(1.5\right)^{100} \approx 4 \times 10^{17} \]

Gradient Vanishing

\[ \left(0.8\right)^{100} \approx 2 \times 10^{-10} \]
Issues with Gradient Exploding

• Value out of range: infinity value (NaN)
• Sensitive to learning rate (LR)
  • Not small enough LR -> larger gradients
  • Too small LR -> No progress
  • May need to change LR dramatically during training
Gradient Vanishing

• Use sigmoid as the activation function

\[
\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))
\]
Issues with Gradient Vanishing

• Gradients with value 0
• No progress in training
  • No matter how to choose learning rate
• Severe with bottom layers
  • Only top layers are well trained
• No benefit to make networks deeper
How to stabilize training?
Stabilize Training: Practical Considerations
Stabilize Training: Practical Considerations

• Goal: make sure gradient values are in a proper range
  • E.g. in $[1\times10^{-6}, 1\times10^3]$
Stabilize Training: Practical Considerations

• Goal: make sure gradient values are in a proper range
  • E.g. in [1e-6, 1e3]
  • Multiplication -> plus
  • Architecture change (e.g., ResNet)
Stabilize Training: Practical Considerations

- Goal: make sure gradient values are in a proper range
  - E.g. in \([1e-6, 1e3]\)
  - Multiplication -> plus
  - Architecture change (e.g., ResNet)
- Normalize
  - Batch Normalization, Gradient clipping
Stabilize Training: Practical Considerations

• Goal: make sure gradient values are in a proper range
  • E.g. in $[1e-6, 1e3]$  
  • Multiplication -> plus
    • Architecture change (e.g., ResNet)

• Normalize
  • Batch Normalization, Gradient clipping

• Proper activation functions
Part III: Generalization & Regularization
How good are the models?
Training Error and Generalization Error

- Training error: model error on the training data
- **Generalization error**: model error on new data
- Example: practice a future exam with past exams
  - Doing well on past exams (training error) doesn’t guarantee a good score on the future exam (generalization error)
Underfitting

Overfitting

Image credit: hackernoon.com
Model Capacity
Model Capacity

- The ability to fit variety of functions
Model Capacity

- The ability to fit variety of functions
- Low capacity models struggles to fit training set
- Underfitting
Model Capacity

• The ability to fit variety of functions
• Low capacity models struggles to fit training set
  • Underfitting
• High capacity models can memorize the training set
  • Overfitting
# Underfitting and Overfitting

<table>
<thead>
<tr>
<th>Model capacity</th>
<th>Data complexity</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Simple</td>
<td>Complex</td>
</tr>
<tr>
<td>High</td>
<td>Overfitting</td>
<td>Normal</td>
</tr>
</tbody>
</table>

- **Underfitting** occurs when the model has low capacity and is trained on simple data.
- **Overfitting** occurs when the model has high capacity and is trained on complex data.
- **Normal** occurs when the model has normal capacity and is trained on normal data.
Influence of Model Complexity

Also known as “Test error”
Estimate Neural Network Capacity

• It’s hard to compare complexity between different algorithms
  • e.g. tree vs neural network
Estimate Neural Network Capacity

- It’s hard to compare complexity between different algorithms
  - e.g. tree vs neural network
- Given an algorithm family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter

\[
(d + 1)(m + 1)k
\]
Data Complexity

• Multiple factors matters
  • # of examples
  • # of features in each example
  • time/space structure
  • # of labels
How to regularize the model for better generalization?
Weight Decay
Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range
  \[
  \min \ell(w, b) \quad \text{subject to} \quad \|w\|^2 \leq \theta
  \]
  
- Often do not regularize bias \(b\)
  - Doing or not doing has little difference in practice
  - A small \(\theta\) means more regularization
Squared Norm Regularization as Soft Constraint

• We can rewrite the hard constraint version as

\[ \min \ell(w, b) + \frac{\lambda}{2} \|w\|^2 \]
Squared Norm Regularization as Soft Constraint

• We can rewrite the hard constraint version as

\[ \min \ell(w, b) + \frac{\lambda}{2} \|w\|^2 \]

• Hyper-parameter \( \lambda \) controls regularization importance
• \( \lambda = 0 \): no effect
• \( \lambda \to \infty, w^* \to 0 \)
Illustrate the Effect on Optimal Solutions

\[ w^* = \arg \min \ell(w, b) + \frac{\lambda}{2} \|w\|^2 \]

\[ \tilde{w}^* = \arg \min \ell(w, b) \]

\[ \tilde{w}^* = \arg \min \ell(w, b) \]
Dropout

Hinton et al.
Apply Dropout

- Often apply dropout on the output of hidden fully-connected layers

\[
\begin{align*}
  h &= \sigma(W_1 x + b_1) \\
  h' &= \text{dropout}(h) \\
  o &= W_2 h' + b_2 \\
  y &= \text{softmax}(o)
\end{align*}
\]
Figure 2: **Left**: A unit at training time that is present with probability $p$ and is connected to units in the next layer with weights $w$. **Right**: At test time, the unit is always present and the weights are multiplied by $p$. The output at test time is same as the expected output at training time.
Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.
What we’ve learned today...

• Deep neural networks
  • Computational graph (forward and backward propagation)
  • Numerical stability in training
    • Gradient vanishing/exploding
• Generalization and regularization
  • Overfitting, underfitting
  • Weight decay and dropout
Thanks!