Backpropagation
+
RNNs
Reminders

• Homework 5: Neural Networks
  – Out: Sun, Feb 27
  – Due: Fri, Mar 18 at 11:59pm
A 1-Hidden Layer Neural Network

TRAINING A NEURAL NETWORK
Backpropagation

Training

(A) Input
Given $x_i$, $\forall i$

(B) Hidden (linear)
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j$$

(C) Hidden (sigmoid)
$$z_j = \frac{1}{1 + \exp(-a_j)}, \forall j$$

(D) Output (linear)
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(E) Output (sigmoid)
$$y = \frac{1}{1 + \exp(-b)}$$

Input

Weights

Hidden Layer

Weights

Output

$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$

$\mathbf{z}_1$
$\mathbf{z}_2$

$\mathbf{y}$
(A) Input
Given $x_i$, $\forall i$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$, $\forall j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1 + \exp(-a_j)}$, $\forall j$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(E) Output (sigmoid)
y = $\frac{1}{1 + \exp(-b)}$

(F) Loss
$J = \frac{1}{2} (y - y^*)^2$
SGD with Backprop

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

1: procedure SGD(Training data \( \mathcal{D} \), test data \( \mathcal{D}_t \))
2: Initialize parameters \( \alpha, \beta \)
3: for \( e \in \{1, 2, \ldots, E\} \) do
4: for \( (x, y) \in \mathcal{D} \) do
5: Compute neural network layers:
6: \( o = \text{object}(x, a, b, z, \hat{y}, J) = \text{NNFORWARD}(x, y, \alpha, \beta) \)
7: Compute gradients via backprop:
8: \( g_\alpha = \nabla_\alpha J \)
9: \( g_\beta = \nabla_\beta J \)
10: Update parameters:
11: \( \alpha \leftarrow \alpha - \gamma g_\alpha \)
12: \( \beta \leftarrow \beta - \gamma g_\beta \)
13: Evaluate training mean cross-entropy \( J_{\mathcal{D}}(\alpha, \beta) \)
14: Evaluate test mean cross-entropy \( J_{\mathcal{D}_t}(\alpha, \beta) \)
15: return parameters \( \alpha, \beta \)
FORWARD COMPUTATION FOR A NEURAL NETWORK
Training

SGD with Backprop

Example: 1-Hidden Layer Neural Network

Algorithm 1: Stochastic Gradient Descent (SGD)

1: procedure SGD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
2: Initialize parameters $\alpha, \beta$
3: for $e \in \{1, 2, \ldots, E\}$ do
4:   for $(x, y) \in \mathcal{D}$ do
5:     Compute neural network layers:
6:     $o = \text{object}(x, a, b, z, \hat{y}, J) = \text{NNFORWARD}(x, y, \alpha, \beta)$
7:     Compute gradients via backprop:
8:     $g_\alpha = \nabla_\alpha J$
9:     $g_\beta = \nabla_\beta J$
10:    $= \text{NNBACKWARD}(x, y, \alpha, \beta, o)$
11: Update parameters:
12:    $\alpha \leftarrow \alpha - \gamma g_\alpha$
13:    $\beta \leftarrow \beta - \gamma g_\beta$
14: Evaluate training mean cross-entropy $J_\mathcal{D}(\alpha, \beta)$
15: Evaluate test mean cross-entropy $J_\mathcal{D}_t(\alpha, \beta)$
16: return parameters $\alpha, \beta$
Training

Backpropagation

\[ J = \frac{1}{2} (y - y_d)^2 \]

Output (sigmoid):
\[ y = \frac{1}{1 + \exp(-b)} \]

Output (linear):
\[ b = \sum_{j=0}^{D} \beta_j z_j \]

Hidden (sigmoid):
\[ z_j = \frac{1}{1 + \exp(-a_j)}, \quad \forall j \]

Hidden (linear):
\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \quad \forall j \]

Input:
Given \( x_i, \quad \forall i \)
Training

Backpropagation

(A) Input
Given $x_i, \forall i$

(B) Hidden (linear)
$$a_j = \sum_{i=0}^{M} \alpha_{ji}x_i, \forall j$$

(C) Hidden (sigmoid)
$$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$$

(D) Output (linear)
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(E) Output (sigmoid)
$$y = \frac{1}{1+\exp(-b)}$$

(F) Loss
$$J = \frac{1}{2}(y - y^*)^2$$

Input

$\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$

Hidden Layer

$\mathbf{z}_1$, $\mathbf{z}_2$

Weights

$\alpha_{11}, \alpha_{12}, \alpha_{13}, \alpha_{21}, \alpha_{22}, \alpha_{23}$

$\beta_1, \beta_2$

Output

$\mathbf{y}$

Weights
\( J = \frac{1}{2} (y - y^*)^2 \)

\( y = \frac{1}{1 + \exp(-b)} \)

\( b = \sum_{j=0}^{D} \beta_j z_j \)

\( z_j = \frac{1}{1 + \exp(-a_j)}, \forall j \)

\( a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j \)

\( \text{Input} \quad \text{Given} \quad x_i, \forall i \)

\( \text{Output} \quad (\text{sigmoid}) \)

\( y = \frac{1}{1 + \exp(-b)} \)

\( \text{Output (linear)} \)

\( b = \sum_{j=0}^{D} \beta_j z_j \)

\( \text{Hidden (sigmoid)} \)

\( z_j = \frac{1}{1 + \exp(-a_j)}, \forall j \)

\( \text{Hidden (linear)} \)

\( a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j \)

\( \text{Input} \quad \text{Given} \quad x_i, \forall i \)
A 1-Hidden Layer Neural Network

BACKPROPAGATION FOR A NEURAL NETWORK
Training

Backpropagation

(A) Input
Given $x_i$, $\forall i$

(B) Hidden (linear)
$$a_j = \sum_{i=0}^{M} \alpha_{ji}x_i, \forall j$$

(C) Hidden (sigmoid)
$$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$$

(D) Output (linear)
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(E) Output (sigmoid)
$$y = \frac{1}{1+\exp(-b)}$$

Input
$x_1$, $x_2$, $x_3$

Hidden Layer
$z_1$, $z_2$

Weights
$\alpha_{11}$, $\alpha_{12}$, $\alpha_{13}$, $\alpha_{21}$, $\alpha_{22}$, $\alpha_{23}$, $\beta_1$, $\beta_2$

Output
$y$
(A) Input
Given $x_i$, $\forall i$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji}x_i$, $\forall j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1 + \exp(-a_j)}$, $\forall j$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(E) Output (sigmoid)
y = $\frac{1}{1 + \exp(-b)}$

(F) Loss
$J = \frac{1}{2}(y - y^*)^2$
Case 2: Neural Network

Forward

\[ J = y^* \log y + (1 - y^*) \log(1 - y) \]

\[ y = \frac{1}{1 + \exp(-b)} \]

\[ b = \sum_{j=0}^{D} \beta_j z_j \]

\[ z_j = \frac{1}{1 + \exp(-a_j)} \]

\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i \]

Backward

\[ \frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1} \]

\[ \frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2} \]

\[ \frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j \]

\[ \frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j \]

\[ \frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2} \]

\[ \frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i \]

\[ \frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji} \]
## Training

### Backpropagation

### Case 2:

<table>
<thead>
<tr>
<th>Loss</th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J = y^* \log y + (1 - y^*) \log(1 - y) )</td>
<td>( \frac{dJ}{dy} = \frac{y^<em>}{y} + \frac{(1 - y^</em>)}{y - 1} )</td>
<td></td>
</tr>
</tbody>
</table>

### Sigmoid

| \( y = \frac{1}{1 + \exp(-b)} \) | \( \frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db} \) = \( \frac{\exp(-b)}{(\exp(-b) + 1)^2} \) |

### Linear

| \( b = \sum_{j=0}^{D} \beta_j z_j \) | \( \frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j} \) = \( z_j \) |

| \( z_j = \frac{1}{1 + \exp(-a_j)} \) | \( \frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j} \) = \( \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2} \) |

### Linear

| \( a_j = \sum_{i=0}^{M} \alpha_{ji} x_i \) | \( \frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}} \) = \( x_i \) |

| \( \frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j} \frac{da_j}{dx_i} \) = \( \alpha_{ji} \) |
Derivative of a Sigmoid

First suppose that

$$s = \frac{1}{1 + \exp(-b)}$$  \hspace{1cm} (1)

To obtain the simplified form of the derivative of a sigmoid.

$$\frac{ds}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$  \hspace{1cm} (2)

$$= \frac{\exp(-b) + 1 - 1}{(\exp(-b) + 1 + 1 - 1)^2}$$  \hspace{1cm} (3)

$$= \frac{\exp(-b) + 1 - 1}{(\exp(-b) + 1)^2}$$  \hspace{1cm} (4)

$$= \frac{\exp(-b) + 1}{(\exp(-b) + 1)^2} - \frac{1}{(\exp(-b) + 1)^2}$$  \hspace{1cm} (5)

$$= \frac{1}{(\exp(-b) + 1)} - \frac{1}{(\exp(-b) + 1)^2}$$  \hspace{1cm} (6)

$$= \frac{1}{(\exp(-b) + 1)} - \left( \frac{1}{(\exp(-b) + 1)} \frac{1}{\exp(-b) + 1} \right)$$  \hspace{1cm} (7)

$$= \frac{1}{(\exp(-b) + 1)} \left( 1 - \frac{1}{(\exp(-b) + 1)} \right)$$  \hspace{1cm} (8)

$$= s(1 - s)$$  \hspace{1cm} (9)
### Case 2: Backpropagation

#### Forward

**Loss**

\[
J = y^* \log y + (1 - y^*) \log(1 - y)
\]

**Sigmoid**

\[
y = \frac{1}{1 + \exp(-b)}
\]

\[
b = \sum_{j=0}^{D} \beta_j z_j
\]

**Sigmoid**

\[
z_j = \frac{1}{1 + \exp(-a_j)}
\]

**Linear**

\[
a_j = \sum_{i=0}^{M} \alpha_{ji} x_i
\]

#### Backward

\[
\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}
\]

**Sigmoid**

\[
\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}
\]

\[
\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j
\]

\[
\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j
\]

**Sigmoid**

\[
\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}
\]

**Linear**

\[
\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i
\]

\[
\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}
\]
## Backpropagation

**Case 2:**

### Forward

<table>
<thead>
<tr>
<th>Loss</th>
<th>( J = y^* \log y + (1 - y^*) \log(1 - y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid</td>
<td>( y = \frac{1}{1 + \exp(-b)} )</td>
</tr>
<tr>
<td>Linear</td>
<td>( b = \sum_{j=0}^{D} \beta_j z_j )</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>( z_j = \frac{1}{1 + \exp(-a_j)} )</td>
</tr>
<tr>
<td>Linear</td>
<td>( a_j = \sum_{i=0}^{M} \alpha_{ji} x_i )</td>
</tr>
</tbody>
</table>

### Backward

<table>
<thead>
<tr>
<th>Loss</th>
<th>( \frac{dJ}{dy} = \frac{y^<em>}{y} + \frac{(1 - y^</em>)}{y - 1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid</td>
<td>( \frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db} )</td>
</tr>
<tr>
<td>Linear</td>
<td>( \frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j} )</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>( \frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j} )</td>
</tr>
<tr>
<td>Linear</td>
<td>( \frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j} )</td>
</tr>
</tbody>
</table>

\[ \frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j} \frac{da_j}{dx_i} \]
Training

SGD with Backprop

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

1: procedure SGD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
2: Initialize parameters $\alpha, \beta$
3: for $e \in \{1, 2, \ldots, E\}$ do
4:   for $(x, y) \in \mathcal{D}$ do
5:     Compute neural network layers:
6:     $o = \text{object}(x, a, b, z, \hat{y}, J) = \text{NNFORWARD}(x, y, \alpha, \beta)$
7:     Compute gradients via backprop:
8:     $g_\alpha = \nabla_\alpha J$
9:     $g_\beta = \nabla_\beta J$
10:    $= \text{NNBACKWARD}(x, y, \alpha, \beta, o)$
11: Update parameters:
12:    $\alpha \leftarrow \alpha - \gamma g_\alpha$
13:    $\beta \leftarrow \beta - \gamma g_\beta$
14: Evaluate training mean cross-entropy $J_\mathcal{D}(\alpha, \beta)$
15: Evaluate test mean cross-entropy $J_\mathcal{D}_t(\alpha, \beta)$
16: return parameters $\alpha, \beta$
THE BACKPROPAGATION ALGORITHM
Backpropagation

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation
1. Write an algorithm for evaluating the function $y = f(x)$. The algorithm defines a directed acyclic graph, where each variable is a node (i.e. the “computation graph”)
2. Visit each node in topological order.
   For variable $u_i$ with inputs $v_1, \ldots, v_N$
   a. Compute $u_i = g_i(v_1, \ldots, v_N)$
   b. Store the result at the node

Backward Computation (Version A)
1. Initialize $dy/dy = 1$.
2. Visit each node $v_j$ in reverse topological order.
   Let $u_1, \ldots, u_M$ denote all the nodes with $v_j$ as an input
   Assuming that $y = h(u) = h(u_1, \ldots, u_M)$
   and $u = g(v)$ or equivalently $u_i = g_i(v_1, \ldots, v_j, \ldots, v_N)$ for all $i$
   a. We already know $dy/du_i$ for all $i$
   b. Compute $dy/dv_j$ as below (Choice of algorithm ensures computing $(du_i/dv_j)$ is easy)
   $\frac{dy}{dv_j} = \sum_{i=1}^{M} \frac{dy}{du_i} \frac{du_i}{dv_j}$

Return partial derivatives $dy/du_i$ for all variables
Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation
1. Write an **algorithm** for evaluating the function \( y = f(x) \). The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the “**computation graph**”)
2. Visit each node in **topological order**. For variable \( u_i \) with inputs \( v_1, \ldots, v_N \)
   a. Compute \( u_i = g_i(v_1, \ldots, v_N) \)
   b. Store the result at the node

Backward Computation (Version B)
1. **Initialize** all partial derivatives \( \frac{dy}{du_i} \) to 0 and \( \frac{dy}{dy} = 1 \).
2. Visit each node in **reverse topological order**. For variable \( u_i = g_i(v_1, \ldots, v_N) \)
   a. We already know \( \frac{dy}{du_i} \)
   b. Increment \( \frac{dy}{dv_j} \) by \( (\frac{dy}{du_i})(\frac{du_i}{dv_j}) \)
   (Choice of algorithm ensures computing \( \frac{du_i}{dv_j} \) is easy)

**Return** partial derivatives \( \frac{dy}{du_i} \) for all variables
Why is the backpropagation algorithm efficient?

1. Reuses **computation from the forward pass** in the backward pass

2. Reuses **partial derivatives** throughout the backward pass (*but only if the algorithm reuses shared computation in the forward pass*)

(Key idea: partial derivatives in the backward pass should be thought of as variables stored for reuse)
A Recipe for Machine Learning

1. Given training data:
   \( \{x_i, y_i\}_{i=1}^{N} \)

2. Choose each of these:
   – Decision function
     \( \hat{y} = f_\theta(x_i) \)
   – Loss function
     \( \ell(\hat{y}, y_i) \in \mathbb{R} \)

3. Define goal:

4. Train with SGD:
   (take small steps opposite the gradient)

   Gradients
   Backpropagation can compute this gradient!

   And it’s a special case of a more general algorithm called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!
MATRIX CALCULUS
Q: Do I need to know matrix calculus to derive the backprop algorithms used in this class?

A: Well, we’ve carefully constructed our assignments so that you do not need to know matrix calculus. That said, it’s pretty handy. So we added matrix calculus to our learning objectives for backprop.
Matrix Calculus

Let $y, x \in \mathbb{R}$ be scalars, $y \in \mathbb{R}^M$ and $x \in \mathbb{R}^P$ be vectors, and $Y \in \mathbb{R}^{M \times N}$ and $X \in \mathbb{R}^{P \times Q}$ be matrices.

<table>
<thead>
<tr>
<th>Types of Derivatives</th>
<th>scalar</th>
<th>vector</th>
<th>matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Numerator</strong></td>
<td>$\frac{\partial y}{\partial x}$</td>
<td>$\frac{\partial y}{\partial x}$</td>
<td>$\frac{\partial Y}{\partial x}$</td>
</tr>
<tr>
<td><strong>Denominator</strong></td>
<td>$\frac{\partial y}{\partial X}$</td>
<td>$\frac{\partial y}{\partial X}$</td>
<td>$\frac{\partial Y}{\partial X}$</td>
</tr>
</tbody>
</table>
# Matrix Calculus

<table>
<thead>
<tr>
<th>Types of Derivatives</th>
<th>scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x} \end{bmatrix} )</td>
<td></td>
</tr>
</tbody>
</table>

## Scalar

\[ \frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix} \]

## Matrix

\[ \frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix} \]
# Matrix Calculus

<table>
<thead>
<tr>
<th>Types of Derivatives</th>
<th>scalar</th>
<th>vector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>scalar</strong></td>
<td>( \frac{\partial y}{\partial x} = [ \frac{\partial y}{\partial x} ] )</td>
<td>( \frac{\partial y}{\partial x} = [ \frac{\partial y_1}{\partial x}, \frac{\partial y_2}{\partial x}, \ldots, \frac{\partial y_N}{\partial x} ] )</td>
</tr>
<tr>
<td><strong>vector</strong></td>
<td>( \frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \ \frac{\partial y}{\partial x_2} \ \vdots \ \frac{\partial y}{\partial x_P} \end{bmatrix} )</td>
<td>( \frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} &amp; \frac{\partial y_2}{\partial x_1} &amp; \ldots &amp; \frac{\partial y_N}{\partial x_1} \ \frac{\partial y_1}{\partial x_2} &amp; \frac{\partial y_2}{\partial x_2} &amp; \ldots &amp; \frac{\partial y_N}{\partial x_2} \ \vdots &amp; \vdots &amp; \ddots &amp; \vdots \ \frac{\partial y_1}{\partial x_P} &amp; \frac{\partial y_2}{\partial x_P} &amp; \ldots &amp; \frac{\partial y_N}{\partial x_P} \end{bmatrix} )</td>
</tr>
</tbody>
</table>
Matrix Calculus

Whenever you read about matrix calculus, you’ll be confronted with two layout conventions:

Let \( y, x \in \mathbb{R} \) be scalars, \( y \in \mathbb{R}^M \) and \( x \in \mathbb{R}^P \) be vectors.

1. In numerator layout:

   \[
   \frac{\partial y}{\partial x} \quad \text{is a } 1 \times P \text{ matrix, i.e. a row vector}
   \]

   \[
   \frac{\partial y}{\partial x} \quad \text{is an } M \times P \text{ matrix}
   \]

2. In denominator layout:

   \[
   \frac{\partial y}{\partial x} \quad \text{is a } P \times 1 \text{ matrix, i.e. a column vector}
   \]

   \[
   \frac{\partial y}{\partial x} \quad \text{is an } P \times M \text{ matrix}
   \]

In this course, we use denominator layout.

Why? This ensures that our gradients of the objective function with respect to some subset of parameters are the same shape as those parameters.
Matrix Calculus

Common Vector Derivatives

Let \( \frac{df(x)}{dx} = \nabla_x f(x) \) be the vector derivative of \( f \), \( \nabla \in \mathbb{R}^{m \times m} \), \( x \in \mathbb{R}^m \)

Scalar Derivatives

\[ f(x) \rightarrow \frac{df}{dx} \]

\[ b_x \rightarrow b \]

\[ xb \rightarrow bx \]

\[ x^2 \rightarrow 2x \]

\[ bx^2 \rightarrow 2bx \]

Vector Derivatives

\[ f(x) \rightarrow \frac{df}{dx} \]

\[ x^TB \rightarrow B \]

\[ x^Tb \rightarrow b \]

\[ x^Tx \rightarrow 2x \]

\[ x^TBx \rightarrow 2Bx \]

\( B \) symmetric
Matrix Calculus

**Question:**
Suppose \( y = g(u) \) and \( u = h(x) \)

\[ y \]
\[ g \]
\[ u \]
\[ h \]
\[ x \]

Which of the following is the correct definition of the chain rule?

**Answer:**

\[
\frac{\partial y}{\partial x} = \ldots
\]

A. \( \frac{\partial y}{\partial u} \frac{\partial u}{\partial x} \)

B. \( \frac{\partial y}{\partial u} \frac{\partial u^T}{\partial x} \)

C. \( \frac{\partial y}{\partial u} \frac{\partial u^T}{\partial x} \)

D. \( \frac{\partial y}{\partial u} \frac{\partial u^T}{\partial x} \)

E. \( \left( \frac{\partial y}{\partial u} \frac{\partial u}{\partial x} \right)^T \)

F. None of the above
DRAWING A NEURAL NETWORK
Ways of Drawing Neural Networks

Neural Network Diagram

- The diagram represents a neural network
- Nodes are circles
- One node per hidden unit
- Node is labeled with the variable corresponding to the hidden unit
- For a fully connected feed-forward neural network, a hidden unit is a nonlinear function of nodes in the previous layer
- Edges are directed
- Each edge is labeled with its weight (side note: we should be careful about ascribing how a matrix can be used to indicate the labels of the edges and pitfalls there)

Other details:
- Following standard convention, the intercept term is NOT shown as a node, but rather is assumed to be part of the nonlinear function that yields a hidden unit. (i.e. its weight does NOT appear in the picture anywhere)
- The diagram does NOT include any nodes related to the loss computation
Ways of Drawing Neural Networks

Computation Graph

- The diagram represents an algorithm
- Nodes are rectangles
- One node per intermediate variable in the algorithm
- Node is labeled with the function that it computes (inside the box) and also the variable name (outside the box)
- Edges are directed
- Edges do not have labels (since they don’t need them)
- For neural networks:
  - Each intercept term should appear as a node (if it’s not folded in somewhere)
  - Each parameter should appear as a node
  - Each constant, e.g. a true label or a feature vector should appear in the graph
  - It’s perfectly fine to include the loss
Ways of Drawing Neural Networks

**Computation Graph**

- The diagram represents an algorithm
- Nodes are rectangles
- One node per intermediate variable in the algorithm
- Node is labeled with the function that it computes (inside the box) and also the variable name (outside the box)
- Edges are directed
- Edges do not have labels (since they don’t need them)
- For neural networks:
  - Each intercept term should appear as a node (if it’s not folded in somewhere)
  - Each parameter should appear as a node
  - Each constant, e.g. a true label or a feature vector should appear in the graph
  - It’s perfectly fine to include the loss

(F) Loss
\[ J = \frac{1}{2} (y - y^*)^2 \]

(E) Output (sigmoid)
\[ y = \frac{1}{1 + \exp(-b)} \]

(D) Output (linear)
\[ b = \sum_{j=0}^{D} \beta_j z_j \]

(C) Hidden (sigmoid)
\[ z_j = \frac{1}{1 + \exp(-a_j)}, \forall j \]

(B) Hidden (linear)
\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j \]

(A) Input
Given \( x_i, \forall i \)
Ways of Drawing Neural Networks

**Neural Network Diagram**
- The diagram represents a neural network
- Nodes are circles
- One node per hidden unit
- Node is labeled with the variable corresponding to the hidden unit
- For a fully connected feed-forward neural network, a hidden unit is a nonlinear function of nodes in the previous layer
- Edges are directed
- Each edge is labeled with its weight (side note: we should be careful about ascribing how a matrix can be used to indicate the labels of the edges and pitfalls there)
- Other details:
  - Following standard convention, the intercept term is NOT shown as a node, but rather is assumed to be part of the nonlinear function that yields a hidden unit. (i.e. its weight does NOT appear in the picture anywhere)
  - The diagram does NOT include any nodes related to the loss computation

**Computation Graph**
- The diagram represents an algorithm
- Nodes are rectangles
- One node per intermediate variable in the algorithm
- Node is labeled with the function that it computes (inside the box) and also the variable name (outside the box)
- Edges are directed
- Edges do not have labels (since they don’t need them)
- For neural networks:
  - Each intercept term should appear as a node (if it’s not folded in somewhere)
  - Each parameter should appear as a node
  - Each constant, e.g. a true label or a feature vector should appear in the graph
  - It’s perfectly fine to include the loss

**Important!**
Some of these conventions are specific to 10-301/601. The literature abounds with variations on these conventions, but it’s helpful to have some distinction nonetheless.
Summary

1. Neural Networks…
   – provide a way of learning features
   – are highly nonlinear prediction functions
   – (can be) a highly parallel network of logistic regression classifiers
   – discover useful hidden representations of the input

2. Backpropagation…
   – provides an efficient way to compute gradients
   – is a special case of reverse-mode automatic differentiation
Backprop Objectives

You should be able to...

• Differentiate between a neural network diagram and a computation graph
• Construct a computation graph for a function as specified by an algorithm
• Carry out the backpropagation on an arbitrary computation graph
• Construct a computation graph for a neural network, identifying all the given and intermediate quantities that are relevant
• Instantiate the backpropagation algorithm for a neural network
• Instantiate an optimization method (e.g. SGD) and a regularizer (e.g. L2) when the parameters of a model are comprised of several matrices corresponding to different layers of a neural network
• Apply the empirical risk minimization framework to learn a neural network
• Use the finite difference method to evaluate the gradient of a function
• Identify when the gradient of a function can be computed at all and when it can be computed efficiently
• Employ basic matrix calculus to compute vector/matrix/tensor derivatives.
DEEP LEARNING
Why is everyone talking about Deep Learning?

• Because a lot of money is invested in it...
  – DeepMind: Acquired by Google for $400 million
  – Deep Learning startups command millions of VC dollars
  – Demand for deep learning engineers continually outpaces supply

• Because it made the front page of the New York Times
Why is everyone talking about Deep Learning?

Deep learning:

- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn’t always the case!

Since 1980s: Form of models hasn’t changed much, but lots of new tricks...

- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)
FIRST EXAMPLE OF A DEEP NETWORK
10-601 course staff
BACKGROUND: HUMAN LANGUAGE TECHNOLOGIES
Human Language Technologies

Speech Recognition

Machine Translation

기계 번역은 특히 영어와 한국어와 같은 언어 쌍의 경우 매우 어렵습니다.

Summarization
Bidirectional RNN

RNNs are a now commonplace backbone in deep learning approaches to natural language processing.
BACKGROUND: COMPUTER VISION
Example: Image Classification

• ImageNet LSVRC-2011 contest:
  – Dataset: 1.2 million labeled images, 1000 classes
  – Task: Given a new image, label it with the correct class
  – Multiclass classification problem
• Examples from http://image-net.org/
Bird

Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings

- marine animal, marine creature, sea animal, sea creature (1)
- scavenger (1)
- biped (0)
- predator, predatory animal (1)
- larva (49)
- acrodont (0)
- feeder (0)
- stunt (0)
- chordate (3087)
  - tunicate, urochordate, urochord (6)
  - cephalochordate (1)
  - vertebrate, craniate (3077)
    - mammal, mammalian (1169)
    - bird (871)
      - dickybird, dicky-bird, dickybird, dicky-bird (0)
      - cock (1)
      - hen (0)
      - nester (0)
      - night bird (1)
      - bird of passage (0)
      - protoavis (0)
      - archaeopteryx, archеopteryx, Archaeopteryx lithographi
      - Sinornis (0)
      - Ibero-mesornis (0)
      - archaeornis (0)
    - ratite, ratite bird, flightless bird (10)
      - carinate, carinate bird, flying bird (0)
      - passerine, passeriform bird (279)
      - nonpasserine bird (0)
    - bird of prey, raptor, reptorial bird (80)
    - gallinaceous bird, gallinacean (114)
German iris, Iris kochii

Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica

- halophyte (0)
- succulent (39)
- cultivar (0)
- cultivated plant (0)
- weed (54)
- evergreen, evergreen plant (0)
- deciduous plant (0)
- vine (272)
- creeper (0)
- woody plant, ligneous plant (1868)
- geophyte (0)
- desert plant, xerophyte, xerophytic plant, xerophile, xerophilic (0)
- mesophyte, mesophytic plant (0)
- aquatic plant, water plant, hydrophyte, hydrophytic plant (11)
- tuberous plant (0)
- bulbous plant (179)
- indaceous plant (27)
  - iris, flag, fleur-de-lis, sword lily (19)
    - bearded iris (4)
      - Florentine iris, orris, Iris germanica florentina, Iris germanica (0)
      - German iris, Iris germanica (0)
      - German iris, Iris kochii (0)
      - Dalmatian iris, Iris pallida (0)
  - beardless iris (4)
  - bulbous iris (0)
  - dwarf iris, Iris cristata (0)
  - stinking iris, gladdon, gladdon iris, stinking gladwyn, 
    Persian iris, Iris persica (0)
  - yellow iris, yellow flag, yellow water flag, Iris pseudacorus (0)
  - dwarf iris, vernal iris, Iris verna (0)
  - blue flag, Iris versicolor (0)
Court, courtyard
An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"
Feature Engineering for CV

Edge detection (Canny)

Corner Detection (Harris)

Scale Invariant Feature Transform (SIFT)

Figures from http://opencv.org

Figure from Lowe (1999) and Lowe (2004)
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.
CNNs for Image Recognition

Revolution of Depth

ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

ImageNet Classification top-5 error (%)

Backpropagation and Deep Learning

Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are simply fancy computation graphs (aka. hypotheses or decision functions).

Our recipe also applies to these models and (again) relies on the backpropagation algorithm to compute the necessary gradients.
BACKGROUND:
N-GRAM LANGUAGE MODELS
n-Gram Language Model

- **Goal**: Generate realistic looking sentences in a human language
- **Key Idea**: condition on the last n-1 words to sample the n\textsuperscript{th} word

\[ p(· | \text{START}) \quad p(· | \text{START}, \text{The}) \quad p(· | \text{The}, \text{bat}) \quad p(· | \text{bat}, \text{made}) \quad p(· | \text{made}, \text{noise}) \quad p(· | \text{noise}, \text{at}) \]
n-Gram Language Model

**Question:** How can we **define** a probability distribution over a sequence of length $T$?

\[
p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1})
\]

For a 2-gram model, this becomes:

\[
p(w_1, w_2, w_3, \ldots, w_6) = p(w_1) p(w_2 \mid w_1) p(w_3 \mid w_2) p(w_4 \mid w_3) p(w_5 \mid w_4) p(w_6 \mid w_5)
\]
**n-Gram Language Model**

**Question:** How can we define a probability distribution over a sequence of length $T$?

$n$-Gram Model ($n=3$)

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})$$

$$p(w_1, w_2, w_3, \ldots, w_6) =$$

- $p(w_1)$
- $p(w_2 \mid w_1)$
- $p(w_3 \mid w_2, w_1)$
- $p(w_4 \mid w_3, w_2)$
- $p(w_5 \mid w_4, w_3)$
- $p(w_6 \mid w_5, w_4)$
n-Gram Language Model

**Question:** How can we **define** a probability distribution over a sequence of length T?

![Sequence of Words]

$n$-Gram Model (n=3)

\[
p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid w_{t-1}, w_{t-2})
\]

\[
p(w_1, w_2, w_3, \ldots, w_6) = \frac{p(w_1) \cdot p(w_2 \mid w_1) \cdot p(w_3 \mid w_2, w_1) \cdot p(w_4 \mid w_3, w_2) \cdot p(w_5 \mid w_4, w_3) \cdot p(w_6 \mid w_5, w_4)}
\]

**Note:** This is called a **model** because we made some **assumptions** about how many previous words to condition on (i.e. only n-1 words)
# Learning an n-Gram Model

**Question**: How do we **learn** the probabilities for the n-Gram Model?

\[
p(w_t | w_{t-2} = \text{The, } w_{t-1} = \text{bat})
\]

| \(w_t\) | \(p(\cdot | \cdot, \cdot)\) |
|---|---|
| ate | 0.015 |
| ... | ... |
| flies | 0.046 |
| ... | ... |
| zebra | 0.000 |

\[
p(w_t | w_{t-2} = \text{made, } w_{t-1} = \text{noise})
\]

| \(w_t\) | \(p(\cdot | \cdot, \cdot)\) |
|---|---|
| at | 0.020 |
| ... | ... |
| pollution | 0.030 |
| ... | ... |
| zebra | 0.000 |

\[
p(w_t | w_{t-2} = \text{cows, } w_{t-1} = \text{eat})
\]

| \(w_t\) | \(p(\cdot | \cdot, \cdot)\) |
|---|---|
| corn | 0.420 |
| ... | ... |
| grass | 0.510 |
| ... | ... |
| zebra | 0.000 |
Learning an n-Gram Model

Question: How do we learn the probabilities for the n-Gram Model?
Answer: From data! Just count n-gram frequencies

| $w_t$ | $p(\cdot | \cdot, \cdot)$ |
|-------|--------------------------|
| corn  | 4/11                     |
| grass | 3/11                     |
| hay   | 2/11                     |
| if    | 1/11                     |
| which | 1/11                     |

...the cows eat grass...
...our cows eat hay daily...
...factory-farm cows eat corn...
...on an organic farm, cows eat hay and...
...do your cows eat grass or corn?...
...what do cows eat if they have...
...cows eat corn when there is no...
...which cows eat which foods depends...
...if cows eat grass...
...when cows eat corn their stomachs...
...should we let cows eat corn?...
Sampling from a Language Model

**Question:** How do we sample from a Language Model?

**Answer:**
1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to \( p(w_t | w_{t-2}, w_{t-1}) \)
3. Roll that die and generate whichever word \( w_t \) lands face up
4. Repeat

\[
\begin{align*}
p(\cdot | \text{START}) & \rightarrow \text{The} \\
p(\cdot | \text{START}, \text{The}) & \rightarrow \text{bat} \\
p(\cdot | \text{The}, \text{bat}) & \rightarrow \text{made} \\
p(\cdot | \text{bat}, \text{made}) & \rightarrow \text{noise} \\
p(\cdot | \text{made}, \text{noise}) & \rightarrow \text{at} \\
p(\cdot | \text{noise}, \text{at}) & \rightarrow \text{night}
\end{align*}
\]
Sampling from a Language Model

**Question**: How do we sample from a Language Model?

**Answer**:
1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to \( p(w_t | w_{t-2}, w_{t-1}) \)
3. Roll that die and generate whichever word \( w_t \) lands face up
4. Repeat

### Training Data (Shakespeaere)

I tell you, friends, most charitable care
ave the patricians of you. For your
wants, Your suffering in this dearth,
you may as well Strike at the heaven
with your staves as lift them Against
the Roman state, whose course will on
The way it takes, cracking ten thousand
curbs Of more strong link asunder than
can ever Appear in your impediment.
For the dearth, The gods, not the
patricians, make it, and Your knees to
them, not arms, must help.

### 5-Gram Model

Approacheth, denay. dungy
Thither! Julius think: grant,—
Yead linens, sheep's Ancient,
Agreed: Petrarch plaguy Resolved
pear! observingly honourest
adulteries wherever scabbard
guess; affirmation—his monsieur;
died. jealousy, chequins me.
Daphne building. weakness: sun-
rise, cannot stays carry't,
unpurposed. prophet-like drink;
back-return 'gainst surmise
Bridget ships? wane; interim?
She's striving wet;
RECURRENT NEURAL NETWORK (RNN) LANGUAGE MODELS
Recurrent Neural Networks (RNNs)

inputs: \( x = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \)

hidden units: \( h = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \)

outputs: \( y = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \)

nonlinearity: \( \mathcal{H} \)

Definition of the RNN:
\[
\begin{align*}
h_t &= \mathcal{H} (W_{xh} x_t + W_{hh} h_{t-1} + b_h) \\
y_t &= W_{hy} h_t + b_y
\end{align*}
\]
The Chain Rule of Probability

Question: How can we define a probability distribution over a sequence of length $T$?

Chain rule of probability: \[ p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t | w_{t-1}, \ldots, w_1) \]

Note: This is called the chain rule because it is always true for every probability distribution.
RNN Language Model:

\[ p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid f_{\theta}(w_{t-1}, \ldots, w_1)) \]

\[ p(w_1, w_2, w_3, \ldots, w_6) = \]

\[ p(w_1) \]
\[ p(w_2 \mid f_{\theta}(w_1)) \]
\[ p(w_3 \mid f_{\theta}(w_2, w_1)) \]
\[ p(w_4 \mid f_{\theta}(w_3, w_2, w_1)) \]
\[ p(w_5 \mid f_{\theta}(w_4, w_3, w_2, w_1)) \]
\[ p(w_6 \mid f_{\theta}(w_5, w_4, w_3, w_2, w_1)) \]

Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution \( p(w_t \mid f_{\theta}(w_{t-1}, \ldots, w_1)) \) that conditions on the vector
Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution \( p(w_t | f_\theta(w_{t-1}, \ldots, w_1)) \) that conditions on the vector \( h_t = f_\theta(w_{t-1}, \ldots, w_1) \)
RNN Language Model

Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$
RNN Language Model

Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$
RNN Language Model

Key Idea:
(1) convert all previous words to a fixed length vector
(2) define distribution \( p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1)) \) that conditions on the vector \( h_t = f_\theta(w_{t-1}, \ldots, w_1) \)
RNN Language Model

**Key Idea:**

1. convert all previous words to a **fixed length vector**
2. define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$
**Key Idea:**

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$
Key Idea:
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t | f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$
RNN Language Model

**Key Idea:**
(1) convert all previous words to a **fixed length vector**
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $h_t = f_\theta(w_{t-1}, \ldots, w_1)$
RNN Language Model

\[ p(w_1, w_2, w_3, \ldots, w_T) = p(w_1 | h_1) \ p(w_2 | h_2) \ldots \ p(w_T | h_T) \]
**Sampling from a Language Model**

**Question:** How do we sample from a Language Model?

**Answer:**
1. Treat each probability distribution like a (50k-sided) weighted die
2. Pick the die corresponding to \( p(w_t \mid w_{t-2}, w_{t-1}) \)
3. Roll that die and generate whichever word \( w_t \) lands face up
4. Repeat

The same approach to sampling we used for an n-Gram Language Model also works here for an RNN Language Model.
Sampling from an RNN-LM

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him of his but young and tender; and, I would be loath to foil him, as I honour, if he come in: for my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from [http://karpathy.github.io/2015/05/21/rnn-effectiveness/](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)
Sampling from an RNN-LM

Shakespeare’s As You Like It

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

RNN-LM Sample

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Sampling from an RNN-LM

RNN-LM Sample

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

Shakespeare’s As You Like It

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretely to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him well. Your brother is but young and tender; and, for your love, I would be loath to foil him, as I must, for my own honour, if he come in: therefore, out of my love to you, I came hither to acquaint you withal, that either you might stay him from his intendment or brook such disgrace well as he shall run into, in that it is a thing of his own search and altogether against my will.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Sampling from an RNN-LM

VIOLA: Why, Salisbury must find his flesh and thought That which I am not aps, not a man and in fire, To show the reining of the raven and the wars To grace my hand reproach within, and not a fair are hand, That Caesar and my goodly father's world; When I was heaven of presence and our fleets, We spare with hours, but cut thy council I am great, Murdered and by thy master's ready there My power to give thee but so much as hell: Some service in the noble bondman here, Would show him to her wine.

KING LEAR: O, if you were a feeble sight, the courtesy of your law, Your sight and several breath, will wear the gods With his heads, and my hands are wonder'd at the deeds, So drop upon your lordship's head, and your opinion Shall be against your honour.

CHARLES: Marry, do I, sir; and I came to acquaint you with a matter. I am given, sir, secretly to understand that your younger brother Orlando hath a disposition to come in disguised against me to try a fall. To-morrow, sir, I wrestle for my credit; and he that escapes me without some broken limb shall acquit him so much as hell: Some service in the noble bondman here, Would show him to her wine.

TOUCHSTONE: For my part, I had rather bear with you than bear you; yet I should bear no cross if I did bear you, for I think you have no money in your purse.

Example from http://karpathy.github.io/2015/05/21/rnn-effectiveness/
SEQUENCE TO SEQUENCE MODELS
Sequence to Sequence Model

Speech Recognition

Machine Translation

기계 번역은 특히 영어와 한국어와 같은 언어 쌍의 경우 매우 어렵습니다.

Summarization
Now suppose you want generate a sequence conditioned on another input

**Key Idea:**

1. Use an **encoder** model to generate a vector representation of the **input**
2. Feed the output of the encoder to a **decoder** which will generate the **output**

**Applications:**
- translation: Spanish $\rightarrow$ English
- summarization: article $\rightarrow$ summary
- speech recognition: speech signal $\rightarrow$ transcription