EE 511
Neural Networks

Instructor: Hanna Hajishirzi
hannaneh@washington.edu

Slides adapted from Ali Farhadi, Mari Ostendorf, Pedro Domingos, Carlos Guestrin, and Luke Zettelmoyer, Andrei Karpathy
Computational Graphs

\[ s = f(x; W) = Wx \quad \text{scores function} \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad \text{SVM loss} \]

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2 \quad \text{data loss + regularization} \]

want \[ \nabla_W L \]

\[ f = Wx \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \(x = -2, y = 5, z = -4\)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
"local gradient"
"local gradient"

\[
\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial z}
\]

\[
\frac{\partial z}{\partial y}
\]

\[
\frac{\partial z}{\partial z}
\]

\[
x
\]

\[
y
\]

\[
z
\]

\[
gadients
\]
\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

“local gradient”

\[
\frac{\partial L}{\partial z}
\]

gradients
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
    f(x) &= e^x \\
    \frac{df}{dx} &= e^x \\
    f_a(x) &= ax \\
    \frac{df}{dx} &= a \\
    f_c(x) &= c + x \\
    \frac{df}{dx} &= 1 \\
    f(x) &= \frac{1}{x} \\
    \frac{df}{dx} &= -\frac{1}{x^2}
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example: 

\[
f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

\[
f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x
\]

\[
f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a
\]

\[
f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}
\]

\[
f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1
\]
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = a x \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ (e^{-1})(-0.53) = -0.20 \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]
\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]
\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]

\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x & \Rightarrow & & \frac{df}{dx} &= e^x \\
  f_a(x) &= ax & \Rightarrow & & \frac{df}{dx} &= a \\
  f(x) &= \frac{1}{x} & \Rightarrow & & \frac{df}{dx} &= -\frac{1}{x^2} \\
  f_c(x) &= c + x & \Rightarrow & & \frac{df}{dx} &= 1
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{- (w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  &\quad \frac{df}{dx} = e^x \\
  \quad \frac{df}{dx} = a \\
  \quad \frac{df}{dx} = 1
\end{align*}
\]

\[
\begin{align*}
  &\quad \frac{df}{dx} = -1/x^2 \\
  \quad \frac{df}{dx} = -1/x \\
  \quad \frac{df}{dx} = 1
\end{align*}
\]
Another example: 

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
[w_0 \cdot 2.00] \times [x_0 \cdot -1.00] & = -2.00 \\
[w_1 \cdot -3.00] \times [x_1 \cdot -2.00] & = 6.00 \\
\end{align*}
\]

\[
\begin{align*}
[0.2] \times [1] & = 0.2 \\
[1] \times [0.2] & = 0.2 \quad \text{(both inputs!)}
\end{align*}
\]

\[
\begin{align*}
f(x) &= e^x \\ f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
\frac{df}{dx} &= e^x \\ \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
f(x) &= \frac{1}{x} \\ f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
\frac{df}{dx} &= -1/x^2 \\ \frac{df}{dx} &= 1
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x \\
  \frac{df}{dx} &= e^x \\
  f_a(x) &= ax \\
  \frac{df}{dx} &= a \\
  f_c(x) &= c + x \\
  \frac{df}{dx} &= 1 \\
  f(x) &= \frac{1}{x} \\
  \frac{df}{dx} &= -\frac{1}{x^2}
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

[local gradient] x [upstream gradient]

x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a \\
  \frac{df}{dx} &= \frac{1}{x} \\
  \frac{df}{dx} &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= -1/x^2 \\
  \frac{df}{dx} &= 1
\end{align*}
\]
$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}}\right) \left(\frac{1}{1 + e^{-x}}\right) = (1 - \sigma(x)) \sigma(x)$$
The sigmoid function is defined as:

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

Its derivative is:

\[ \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x))\sigma(x) \]

The expression \((0.73) \times (1 - 0.73) = 0.2\) is highlighted in the diagram, indicating it as a part of the computation process.
**add** gate: gradient distributor

**Q:** What is a **max** gate?

**max** gate: gradient router

**Q:** What is a **mul** gate?

**mul** gate: gradient switcher
Gradients add at branches
Gradients for vectorized code

This is now the Jacobian matrix (derivative of each element of \( z \) w.r.t. each element of \( x \))

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

(x, y, z are now vectors)
Vectorized operations

Q: what is the size of the Jacobian matrix?

\[ [4096 \times 4096] \]

**f(x) = max(0,x)** (elementwise)

**4096-d input vector**

**4096-d output vector**

in practice we process an entire minibatch (e.g. 100) of examples at one time:
A vectorized example: \( f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \)

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2
\]
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \)

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.22 \\
0.26
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.44 \\
0.52
\end{bmatrix}
\]

\[q = W \cdot x = \left( \begin{array}{c} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{array} \right)\]

\[f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2\]

\[
\frac{\partial f}{\partial q_i} = 2q_i
\]

\[\nabla_q f = 2q\]
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$W = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \\ 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix}, \quad x = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

$$\nabla_W f = 2q \cdot x^T$$

Always check: The gradient with respect to a variable should have the same shape as the variable.
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x_i)^2 \)

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial q_k}{\partial x_i} = W_{k,i}
\]

\[
\frac{\partial f}{\partial x_i} = \sum_{k} \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_{k} 2q_k W_{k,i}
\]
A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W_i \cdot x_i)^2 \]

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4 \\
-0.112 & 0.636
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\nabla_x f = 2W^T \cdot q
\]
Modularized implementation: forward / backward API

Graph (or Net) object  \textit{(rough pseudo code)}

class ComputationalGraph(object):
    
    def forward(inputs):
        
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss

    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward()  # little piece of backprop (chain rule applied)
        return inputs_gradients
Modularized implementation: forward / backward API

(x, y, z are scalars)
Modularized implementation: forward / backward API

(x, y, z are scalars)
Summary

- neural nets will be very large: impractical to write down gradient formula by hand for all parameters
- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- implementations maintain a graph structure, where the nodes implement the **forward()** / **backward()** API
- **forward**: compute result of an operation and save any intermediates needed for gradient computation in memory
- **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs
Neural networks: Architectures

“2-layer Neural Net”, or “1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or “2-hidden-layer Neural Net”
Example feed-forward computation of a neural network

```python
class Neuron:
    # ...

def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))  # sigmoid activation function
    return firing_rate
```

We can efficiently evaluate an entire layer of neurons.
Example feed-forward computation of a neural network

```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```
Loop:
1. **Sample** a batch of data
2. **Forward** prop it through the graph (network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient
Activation functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^Tx + b_1, w_2^Tx + b_2) \]

**ELU**
\[
\begin{cases}
  x & x \geq 0 \\
  \alpha(e^x - 1) & x < 0
\end{cases}
\]
Activation Functions

**Sigmoid**

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

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

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. \(\exp()\) is a bit compute expensive
What happens when $x = -10$?
What happens when $x = 0$?
What happens when $x = 10$?
Activation Functions

- Computes $f(x) = \max(0, x)$
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

ReLU
(Rectified Linear Unit)
Training Neural Networks

1. One time setup
   activation functions, preprocessing, weight initialization, regularization, gradient checking
2. Training dynamics
   babysitting the learning process, parameter updates, hyperparameter optimization
3. Evaluation
Step 1: Preprocess the data

- **original data**
- **zero-centered data**
- **normalized data**
Consider what happens when the input to a neuron is always positive...

\[ f \left( \sum_i w_i x_i + b \right) \]

What can we say about the gradients on \( w \)?
Always all positive or all negative :(
(this is also why you want zero-mean data!)
Step 2: Choose the architecture: say we start with one hidden layer of 50 neurons:

- **Input layer**: CIFAR-10 images, 3072 numbers
- **Hidden layer**: 50 hidden neurons
- **Output layer**: 10 output neurons, one per class
Hyperparameters

Cross-validation strategy

coarse -> fine cross-validation in stages

First stage: only a few epochs to get rough idea of what params work
Second stage: longer running time, finer search
... (repeat as necessary)

Tip for detecting explosions in the solver:
If the cost is ever > 3 * original cost, break out early
Monitor and visualize the accuracy:

big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?
Overfitting

How to improve single-model performance?

Regularization
Regularization: Add term to loss

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W) \]

In common use:

**L2 regularization**

\[ R(W) = \sum_k \sum_l W_{k,l}^2 \quad \text{(Weight decay)} \]

**L1 regularization**

\[ R(W) = \sum_k \sum_l |W_{k,l}| \]

**Elastic net (L1 + L2)**

\[ R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \]
Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common

How can this possibly be a good idea?
Regularization: Dropout

How can this possibly be a good idea?

Forces the network to have a redundant representation; Prevents co-adaptation of features

Another interpretation: Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model.
Dropout: Test time

Dropout makes our output random!

\[ y = f_W(x, z) \]

Want to “average out” the randomness at test-time

\[ y = f(x) = E_z [f(x, z)] = \int p(z) f(x, z) dz \]

But this integral seems hard …
Dropout: Test time

Want to approximate the integral

\[ y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz \]

Consider a single neuron.

At test time we have:

\[ E[a] = w_1x + w_2y \]

During training we have:

\[ E[a] = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) = \frac{1}{2}(w_1x + w_2y) \]

At test time, multiply by dropout probability
Regularization: A common pattern

**Training**: Add some kind of randomness

\[ y = f_W(x, z) \]

**Testing**: Average out randomness (sometimes approximate)

\[ y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz \]
Regularization: Data Augmentation

Load image and label  
“cat”  

Compute loss
Regularization: Data Augmentation

Load image and label

"cat"

Transform image

CNN

Compute loss
Data Augmentation
Random crops and scales

**Training**: sample random crops / scales

ResNet:
1. Pick random \( L \) in range \([256, 480]\)
2. Resize training image, short side = \( L \)
3. Sample random 224 x 224 patch

**Testing**: average a fixed set of crops

ResNet:
1. Resize image at 5 scales: \( \{224, 256, 384, 480, 640\} \)
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips
Summary

• Backpropagation
• Training neural nets strategies
• Regularizations