Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 3: Introduction to Deep Learning (continued)
Course Logistics

- Update on course registrations:
  - New Room (SWING 409) — 47 seats capacity (instead of 40)
  - Registrations (39 students currently registered) — approx. 8 seats left
- Microsoft Azure credits and tutorial next week
- Assignment 1 … any questions?
Short Review …

- Introduced the basic building block of Neural Networks (MLP/FC) layer
- Introduced the basic building block of Neural Networks (MLP/FC) layer

- How do we **stack these layers** up to make a Deep NN
Short Review …

- Introduced the basic building block of Neural Networks *(MLP/FC) layer*

- How do we *stack these layers* up to make a Deep NN

- Basic *NN operations* (implemented using *computational graph*)
Short Review …

- Introduced the basic building block of Neural Networks (MLP/FC) layer
- How do we stack these layers up to make a Deep NN
- Basic NN operations (implemented using computational graph)
- Introduced the basic building block of Neural Networks *(MLP/FC) layer*

- How do we **stack these layers** up to make a Deep NN

- Basic **NN operations** (implemented using computational graph)

### Prediction / Inference

<table>
<thead>
<tr>
<th>Function evaluation</th>
<th>(a.k.a. <strong>ForwardProp</strong>)</th>
</tr>
</thead>
</table>

### Parameter Learnings

<table>
<thead>
<tr>
<th>(Stochastic) Gradient Descent</th>
<th>(needs <strong>derivatives</strong>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>Numerical</strong> differentiation</td>
<td>(not accurate)</td>
</tr>
<tr>
<td>- <strong>Symbolic</strong> differential</td>
<td>(intractable)</td>
</tr>
<tr>
<td>- AutoDiff <strong>Forward</strong></td>
<td>(computationally expensive)</td>
</tr>
<tr>
<td>- AutoDiff <strong>Backward / BackProp</strong></td>
<td></td>
</tr>
</tbody>
</table>
Short **Review** …

- Introduced the basic building block of Neural Networks *(MLP/FC)* layer
- How do we **stack these layers** up to make a Deep NN
- Basic **NN operations** (implemented using computational graph)

### Prediction / Inference

- Function evaluation  
  (a.k.a. **ForwardProp**)

### Parameter Learnings

**(Stochastic) Gradient Descent** (needs derivatives)
- **Numerical** differentiation (not accurate)
- **Symbolic** differential (intractable)
- AutoDiff **Forward** (computationally expensive)
- AutoDiff **Backward / BackProp**

- Different **activation functions** and saturation problem
Regularization: L2 or L1 on the weights

**L2 Regularization:** Learn a more (dense) distributed representation

\[ R(W) = ||W||_2 = \sum \sum W_{i,j}^2 \]

**L1 Regularization:** Learn a sparse representation (few non-zero weight elements)

\[ R(W) = ||W||_1 = \sum \sum |W_{i,j}| \]

(other regularizers are also possible)

Example:

\[ x = [1, 1, 1, 1] \]
\[ W_1 = [1, 0, 0, 0] \]
\[ W_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \]

\[ W_1 \cdot x^T = W_2 \cdot x^T \]

two networks will have identical loss

L2 Regularizer:
\[ R_{L2}(W_1) = 1 \]
\[ R_{L2}(W_2) = 0.25 \]

L1 Regularizer:
\[ R_{L1}(W_1) = 1 \]
\[ R_{L1}(W_2) = 1 \]
Computational Graph: 1-layer with PReLU + Regularizer
Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

\[ \bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}} \]

Benefit:
Improves learning (better gradients, higher learning rate)

[ Ioffe and Szegedy, NIPS 2015 ]
Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

\[
\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}
\]

Benefit:

Improves learning (better gradients, higher learning rate)

Why?

[ Ioffe and Szegedy, NIPS 2015 ]
Activation Function: Sigmoid

\[ a = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \]

Cons:
- Saturated neurons “kill” the gradients
- Non-zero centered
- Could be expensive to compute

* slide adopted from Li, Karpathy, Johnson’s CS231n at Stanford
Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

\[
\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}
\]

Benefit:

Improves learning (better gradients, higher learning rate)

Typically inserted before activation layer

[ Ioffe and Szegedy, NIPS 2015 ]
Activation Function: Sigmoid vs. Tanh

Pros:
- Squishes everything in the range [-1,1]
- Centered around zero
- Has well defined gradient everywhere

Cons:
- Saturated neurons “kill” the gradients

\[
a(x) = \tanh(x) = 2 \cdot \text{sigmoid}(2x) - 1
\]

\[
a(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1
\]
Regularization: Batch Normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

\[
\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}
\]

In practice, also learn how to scale and offset:

\[
y^{(k)} = \gamma^{(k)} \bar{x}^{(k)} + \beta^{(k)}
\]

Benefit:

Improves learning (better gradients, higher learning rate)

Typically inserted before activation layer

[Ioffe and Szegedy, NIPS 2015]
Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to **dropout rate** (between 0 to 1)

[ Srivastava et al, JMLR 2014 ]

* adopted from slides of **CS231n at Stanford**
Regularization: Dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)

1. Compute output of the linear/fc layer $o_i = W_i \cdot x + b_i$
2. Compute a mask with probability proportional to dropout rate $m_i = \text{rand}(1, |o_i|) < \text{dropout rate}$
3. Apply the mask to zero out certain outputs $o_i = o_i \odot m_i$

[ Srivastava et al, JMLR 2014 ]

* adopted from slides of CS231n at Stanford
Regularization: Dropout

Randomly set some neurons to zero in the forward pass, with probability proportional to dropout rate (between 0 to 1)

Why is this a good idea?

[ Srivastava et al, JMLR 2014 ]

* adopted from slides of CS231n at Stanford
Regularization: Dropout

Randomly set some neurons to zero in the forward pass, with probability proportional to dropout rate (between 0 to 1).

Why is this a good idea?

Dropout is training an ensemble of models that share parameters.

Each binary mask (generated in the forward pass) is one model that is trained on (approximately) one data point.

[ Srivastava et al, JMLR 2014 ]

* adopted from slides of CS231n at Stanford
Regularization: Dropout (at test time)

Randomly set some neurons to zero in the forward pass, with probability proportional to dropout rate (between 0 to 1)

After Applying Dropout

At test time, integrate out all the models in the ensemble

Monte Carlo approximation: many forward passes with different masks and average all predictions

[Srivastava et al, JMLR 2014]

* adopted from slides of CS231n at Stanford
Regularization: Dropout (at test time)

Randomly set some neurons to zero in the forward pass, with probability proportional to dropout rate (between 0 to 1)

At test time, integrate out all the models in the ensemble

Monte Carlo approximation: many forward passes with different masks and average all predictions

Equivalent to forward pass with all connections on and scaling of the outputs by dropout rate

For derivation see Lecture 6 of CS231n at Stanford

*S* adopted from slides of CS231n at Stanford

[Srivastava et al, JMLR 2014]
**Regularization: Dropout**

Randomly **set some neurons to zero** in the forward pass, with probability proportional to dropout rate (between 0 to 1)

1. Compute output of the linear/fc layer \( o_i = W_i \cdot x + b_i \)
2. Compute a mask with probability proportional to dropout rate \( m_i = \text{rand}(1, |o_i|) < \text{dropout rate} \)
3. Apply the mask to zero out certain outputs \( o_i = o_i \odot m_i \)

[ Srivastava et al, JMLR 2014 ]

* adopted from slides of CS231n at Stanford
Deep Learning **Terminology**

- **Network structure**: number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

  - generally kept fixed, requires some knowledge of the problem and NN to sensibly set

  - deeper = better

Google’s “Inception” network
Deep Learning **Terminology**

- **Network structure**: number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
  - Generally kept fixed, requires some knowledge of the problem and NN to sensibly set
  - Deeper = better

- **Loss function**: objective function being optimized (softmax, cross entropy, etc.)
  - Requires knowledge of the nature of the problem
Deep Learning **Terminology**

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
  - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
  - deeper = better

- **Loss function:** objective function being optimized (softmax, cross entropy, etc.)
  - requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, etc.
  - optimized using SGD or variants
Deep Learning Terminology

- **Network structure**: number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)
  - generally kept fixed, requires some knowledge of the problem and NN to sensibly set
  - deeper = better

- **Loss function**: objective function being optimized (softmax, cross entropy, etc.)
  - requires knowledge of the nature of the problem

- **Parameters**: trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, etc.
  - optimized using SGD or variants

- **Hyper-parameters**: parameters, including for optimization, that are not optimized directly as part of training (e.g., learning rate, batch size, drop-out rate)
  - grid search

Google’s “Inception” network generally kept fixed, requires some knowledge of the problem and NN to sensibly set deeper = better

Loss function: objective function being optimized (softmax, cross entropy, etc.) requires knowledge of the nature of the problem

Parameters: trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, etc. optimized using SGD or variants

Hyper-parameters: parameters, including for optimization, that are not optimized directly as part of training (e.g., learning rate, batch size, drop-out rate) grid search
Loss Functions ...

This is where all the **fun** is ... we will only look at most common ones
Multivariate Regression

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$

**Output:** output vector $\mathbf{y} \in \mathbb{R}^m$
Multivariate Regression

**Input:** feature vector \( \mathbf{x} \in \mathbb{R}^n \)  
**Output:** output vector \( \mathbf{y} \in \mathbb{R}^m \)

**Neural Network** (input + intermediate hidden layers) \( f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k \)

- with **sigmoid** activations: \( 0 \leq f(\mathbf{x}; \Theta) \leq 1 \)
- with **Tanh** activations: \( -1 \leq f(\mathbf{x}; \Theta) \leq 1 \)
- with **ReLU** activations: \( 0 \leq f(\mathbf{x}; \Theta) \)
Multivariate Regression

**Input:** feature vector \( \mathbf{x} \in \mathbb{R}^n \)

**Output:** output vector \( \mathbf{y} \in \mathbb{R}^m \)

**Neural Network** (input + intermediate hidden layers) \( f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k \)

- with **sigmoid** activations: \( 0 \leq f(\mathbf{x}; \Theta) \leq 1 \)
- with **Tanh** activations: \( -1 \leq f(\mathbf{x}; \Theta) \leq 1 \)
- with **ReLU** activations: \( 0 \leq f(\mathbf{x}; \Theta) \)

**Neural Network** (output): linear layer

\[
\hat{\mathbf{y}} = g(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W} f(\mathbf{x}; \Theta) + \mathbf{b} : \mathbb{R}^k \rightarrow \mathbb{R}^m
\]
Multivariate Regression

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$  

**Output:** output vector $\mathbf{y} \in \mathbb{R}^m$

**Neural Network** (input + intermediate hidden layers)  
$f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k$

- with **sigmoid** activations: $0 \leq f(\mathbf{x}; \Theta) \leq 1$
- with **Tanh** activations: $-1 \leq f(\mathbf{x}; \Theta) \leq 1$
- with **ReLU** activations: $0 \leq f(\mathbf{x}; \Theta)$

**Neural Network** (output): linear layer

$\hat{\mathbf{y}} = g(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W} f(\mathbf{x}; \Theta) + \mathbf{b} : \mathbb{R}^k \rightarrow \mathbb{R}^m$

**Loss:**

$L(\mathbf{y}, \hat{\mathbf{y}}) = \| \mathbf{y} - \hat{\mathbf{y}} \|^2$
Binary Classification (Bernoulli)

**Input:** feature vector $x \in \mathbb{R}^n$  

**Output:** binary label $y \in \{0, 1\}$

**Neural Network** (input + intermediate hidden layers) $f(x; \Theta) : \mathbb{R}^n \to \mathbb{R}$

with **sigmoid** activations: $0 \leq f(x; \Theta) \leq 1$
Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$  
Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with sigmoid activations: $0 \leq f(\mathbf{x}; \Theta) \leq 1$

Neural Network (output): threshold hidden output (which is a sigmoid)

$\hat{y} = 1[f(\mathbf{x}; \Theta) > 0.5]$
**Binary Classification** (Bernoulli)

**Input:** feature vector \( x \in \mathbb{R}^n \)  \hspace{1cm} **Output:** binary label \( y \in \{0, 1\} \)

**Neural Network** (input + intermediate hidden layers) \( f(x; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R} \)

with **sigmoid** activations: \( 0 \leq f(x; \Theta) \leq 1 \)

**Neural Network** (output): threshold hidden output (which is a sigmoid)

\[
\hat{y} = 1[ f(x; \Theta) > 0.5 ]
\]

**Problem:** Not differentiable, probabilistic interpretation maybe desirable
Binary Classification (Bernoulli)

**Input**: feature vector $\mathbf{x} \in \mathbb{R}^n$  

**Output**: binary label $y \in \{0, 1\}$

**Neural Network** (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $0 \leq f(\mathbf{x}; \Theta) \leq 1$

**Neural Network** (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

can interpret the score as the log-odds of $y = 1$ (a.k.a. the logits)
Binary Classification (Bernoulli)

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$  

**Output:** binary label $y \in \{0, 1\}$

**Neural Network** (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$  

with *sigmoid* activations: $0 \leq f(\mathbf{x}; \Theta) \leq 1$

**Neural Network** (output): interpret sigmoid output as probability  

\[
p(y = 1) = f(\mathbf{x}; \Theta)
\]

can interpret the score as the log-odds of $y = 1$ (a.k.a. the *logits*).

**Loss:** similarity between two distributions
**Binary Classification** (Bernoulli)

**Input:** feature vector \( x \in \mathbb{R}^n \)  

**Output:** binary label \( y \in \{0, 1\} \)

We can measure similarity between distribution and using cross-entropy:

\[
H(p, q) = -\mathbb{E}_{x \sim p} \left[ \log q(x) \right]
\]

For discrete distributions this ends up being:

\[
H(p, q) = -\sum_x p(x) \log q(x)
\]

**Loss:** similarity between two distributions
Binary Classification (Bernoulli)

**Input:** feature vector  \( \mathbf{x} \in \mathbb{R}^n \)  

**Output:** binary label  \( y \in \{0, 1\} \)

**Neural Network** (input + intermediate hidden layers)  \( f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R} \)

with **sigmoid** activations:  \( 0 \leq f(\mathbf{x}; \Theta) \leq 1 \)

**Neural Network** (output): interpret sigmoid output as probability

\[
p(y = 1) = f(\mathbf{x}; \Theta)
\]

can interpret the score as the log-odds of  \( y = 1 \) (a.k.a. the **logits**)

**Loss:**  
\[
\mathcal{L}(y, \hat{y}) = -y \log[f(\mathbf{x}; \Theta)] - (1 - y) \log[1 - f(\mathbf{x}; \Theta)]
\]
**Binary Classification** (Bernoulli)

**Input:** feature vector $x \in \mathbb{R}^n$  

**Output:** binary label $y \in \{0, 1\}$

**Neural Network** (input + intermediate hidden layers) $f(x; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $0 \leq f(x; \Theta) \leq 1$

**Neural Network** (output): interpret sigmoid output as probability

$$p(y = 1) = f(x; \Theta)$$

*can interpret the score as the log-odds of $y = 1$ (a.k.a. the **logits**)*

**Loss:**

$$\mathcal{L}(y, \hat{y}) = \begin{cases} -\log(1 - f(x; \Theta)) & y = 0 \\ -\log(f(x; \Theta)) & y = 1 \end{cases}$$
Binary Classification (Bernoulli)

**Input:** feature vector \( \mathbf{x} \in \mathbb{R}^n \)  
**Output:** binary label \( y \in \{0, 1\} \)

**Neural Network** (input + intermediate hidden layers)  
\( f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R} \)

with **sigmoid** activations:  
\( 0 \leq f(\mathbf{x}; \Theta) \leq 1 \)

**Neural Network** (output): interpret sigmoid output as probability  
\[
p(y = 1) = f(\mathbf{x}; \Theta)
\]

Minimizing this **loss** is the same as maximizing log likelihood of data

**Loss:**  
\[
\mathcal{L}(y, \hat{y}) = \begin{cases} 
-\log[1 - f(\mathbf{x}; \Theta)] & y = 0 \\
-\log[f(\mathbf{x}; \Theta)] & y = 1 
\end{cases}
\]
Binary Classification (Bernoulli)

**Input:** feature vector \( \mathbf{x} \in \mathbb{R}^n \)

**Output:** binary label \( y \in \{0, 1\} \)

**Neural Network** (input + intermediate hidden layers)

\[
f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k
\]

with **ReLU** activations:

\[
0 \leq f(\mathbf{x}; \Theta)
\]
**Binary Classification** (Bernoulli)

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$  

**Output:** binary label $y \in \{0, 1\}$

**Neural Network** (input + intermediate hidden layers)  

$f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k$

with ReLU activations: $0 \leq f(\mathbf{x}; \Theta)$

**Neural Network** (output): linear layer with one neuron and sigmoid activation
Multiclass Classification (e.g., ImageNet)

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$

**Output:** muticlass label $\mathbf{y} \in \{0, 1\}^m$

(one-hot encoding)
**Multiclass Classification** (e.g., ImageNet)

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$  
**Output:** muticlass label $\mathbf{y} \in \{0, 1\}^m$  

(One-hot encoding)

**Neural Network** (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

with **ReLU** activations: $0 \leq f(\mathbf{x}; \Theta)$
**Multiclass Classification (e.g., ImageNet)**

**Input:** feature vector \( \mathbf{x} \in \mathbb{R}^n \)

**Output:** multiclass label \( \mathbf{y} \in \{0, 1\}^m \) (one-hot encoding)

**Neural Network** (input + intermediate hidden layers)

\[
f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m
\]

with **ReLU** activations:

\[
0 \leq f(\mathbf{x}; \Theta)
\]

**Neural Network** (output): **softmax** function, where probability of class \( k \) is:

\[
p(y_k = 1) = \frac{\exp [f(\mathbf{x}; \Theta)_i]}{\sum_{j=1}^{C} \exp [f(\mathbf{x}; \Theta)_j]}
\]
**Multiclass Classification (e.g., ImageNet)**

**Input:** feature vector $\mathbf{x} \in \mathbb{R}^n$

**Output:** muticlass label $\mathbf{y} \in \{0, 1\}^m$

(One-hot encoding)

**Neural Network** (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

With **ReLU** activations: $0 \leq f(\mathbf{x}; \Theta)$

**Neural Network** (output): **softmax** function, where probability of class k is:

$$p(y_k = 1) = \frac{\exp [f(\mathbf{x}; \Theta)_i]}{\sum_{j=1}^{C} \exp [f(\mathbf{x}; \Theta)_j]}$$

**Loss:**

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i} y_i \log \hat{y}_i$$
Multiclass Classification (e.g., ImageNet)

**Input:** feature vector \( \mathbf{x} \in \mathbb{R}^n \)

**Output:** muticlass label \( \mathbf{y} \in \{0, 1\}^m \)

(\text{one-hot encoding})

**Neural Network** (input + intermediate hidden layers) \( f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m \)

with \textbf{ReLU} activations: \( 0 \leq f(\mathbf{x}; \Theta) \)

**Neural Network** (output): \textbf{softmax} function, where probability of class \( k \) is:

\[
p(y_k = 1) = \frac{\exp [f(\mathbf{x}; \Theta)_i]}{\sum_{j=1}^{C} \exp [f(\mathbf{x}; \Theta)_j]}
\]

**Loss:** \( \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = H(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i} y_i \log \hat{y}_i = - \log \hat{y}_i \)

Special case for multi-class single label
Monitoring Learning: Visualizing the (training) loss

* slide from Li, Karpathy, Johnson's CS231n at Stanford
Monitoring Learning: Visualizing the (training) loss

Big gap = overfitting

Solution: increase regularization

No gap = undercutting

Solution: increase model capacity

Small gap = ideal

* slide from Li, Karpathy, Johnson's CS231n at Stanford