• Linear Classifier (cont.)
• SVM / Hinge Loss
• Softmax Classifier and Cross-Entropy Loss
• Gradient Descent
MISC

- PS0 due yesterday
- Check the Piazza post if you still have trouble running Colab.
- PS1 release 08/31
- Use Piazza!
Reinforcement Learning - Supervision in form of reward

Supervised Learning
- Train Input: \( \{X, Y\} \)
- Learning output: \( f : X \rightarrow Y \), e.g. \( P(y|x) \)

Unsupervised Learning
- Input: \( \{X\} \)
- Learning output: \( P(x) \)
- Example: Clustering, density estimation, etc.

Reinforcement Learning
- Supervision in form of reward
- No supervision on what action to take

Very often combined, sometimes within the same model!
Recap:

Types of Errors and Generalization
Recap:

**Algebraic Viewpoint**

\[ f(x, W) = WX \]

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space

Adapted from CS 231n slides
This time:

\[ f(x, W) = Wx \]

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x,W)=Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

A **loss function** that tells how good the current classifier is:

Given a dataset of examples:

$$\{(x_i, y_i)\}_{i=1}^N$$

Where $x_i$ is image and $y_i$ is (integer) label.

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum L(f(x_i, W), y_i)$$

Adapted from CS 231n slides.
Multiclass SVM loss:

Given an example \((x_i, y_i)\) where \(x_i\) is the image and where \(y_i\) is the (integer) label,

and using the shorthand for the scores vector: \(s_i = f(x_i, W)\)

the SVM loss has the form:

\[
L_i = \sum_{j \neq y_i} \begin{cases} 
0 & \text{if } s_{y_i} \geq s_j + 1 \\
 s_j - s_{y_i} + 1 & \text{otherwise}
\end{cases}
\]

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

**Notation:** \(s_{y_i}\) is the score given by the classifier for the correct label class of the i-th example \((y_i)\)
Multiclass SVM loss:

Given an example \((x_i, y_i)\) where \(x_i\) is the image and \(y_i\) is the (integer) label,

and using the shorthand for the scores vector: \(s = f(x_i, W)\)

the SVM loss has the form:

\[
L_i = \sum_{j \neq y_i} \begin{cases} 
0 & \text{if } s_{y_i} \geq s_j + 1 \\
{s_j - s_{y_i} + 1} & \text{otherwise}
\end{cases}
\]

\[
= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\]
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x,W) = Wx$ are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Score $s$</th>
<th>Loss $L_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>2.9</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Multiclass SVM loss:

Given an example $(x_i, y_i)$ where $x_i$ is the image and $y_i$ is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$, the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$

Adapted from CS 231n slides
SVM Loss Example

Multiclass SVM loss:

Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>Cat</th>
<th>Car</th>
<th>Frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Losses</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
<tr>
<td></td>
<td>2.9</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

Given an example $(x_i, y_i)$ where $x_i$ is the image and where $y_i$ is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= $\max(0, 1.3 - 4.9 + 1)$ + $\max(0, 2.0 - 4.9 + 1)$
= $\max(0, -2.6) + \max(0, -1.9)$
= 0 + 0
= 0

Adapted from from CS 231n slides
**Multiclass SVM loss:**

Given an example \((x_i,y_i)\) where \(x_i\) is the image and where \(y_i\) is the (integer) label,

and using the shorthand for the scores vector: \(s = f(x_i,W)\)

the SVM loss has the form:

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

\[
L = \frac{(2.9 + 0 + 12.9)}{3}
\]

\[
= 5.27
\]

Suppose: 3 training examples, 3 classes. With some \(W\) the scores \(f(x,W) = Wx\) are:

<table>
<thead>
<tr>
<th>Class</th>
<th>Loss 1</th>
<th>Loss 2</th>
<th>Loss 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

**Losses:**

\[
\text{Losses: } 2.9 \quad 0 \quad 12.9
\]

Adapted from CS 231n slides
Multiclass SVM loss:

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

Q: What happens to loss if car image scores change a bit (e.g., ± 0.1)?

No change for small values

Suppose: 3 training examples, 3 classes. With some \( W \) the scores \( f(x,W) = Wx \) are:

- **cat**
  - 3.2
  - 1.3
  - 2.2
- **car**
  - 5.1
  - 4.9
  - 2.5
- **frog**
  - -1.7
  - 2.0
  - -3.1

Adapted from CS 231n slides
Multiclass SVM loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What is min/max of loss value?

\[ [0, \infty] \]

Suppose: 3 training examples, 3 classes. With some \( W \) the scores \( f(x, W) = Wx \) are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>cat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
<td></td>
</tr>
<tr>
<td>frog</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adapted from CS 231n slides
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = Wx$ are:

<table>
<thead>
<tr>
<th>Class</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Q: At initialization $W$ is close to 0 so all $s \approx 0$. What is the loss?

Multiclass SVM loss:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

num_class - 1

Adapted from CS 231n slides
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x,W)=Wx$ are:

- **cat**: 3.2, 1.3, 2.2
- **car**: 5.1, 4.9, 2.5
- **frog**: -1.7, 2.0, -3.1

Multiclass SVM loss:

$$L_i = \frac{1}{C} \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What if we used mean instead of sum?

No difference

Scaling by constant

Adapted from CS 231n slides
Multiclass SVM loss:

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

```python
def hinge_loss_vec(x, y, W):
    
    x (d): input example vectors
    y (int): class label
    W (C x d): weight matrix
    
    scores = W.dot(x)  # calculate raw scores
    margins = np.maximum(0, scores - scores[y] + 1)  # calculate margins s_j - s_{yi} + 1
    margins[y] = 0  # exclude yi from the loss sum
    loss_i = np.sum(margins). # sum across all j (classes)
    return loss_i
```

Adapted from CS 231n slides
\[ f(x, W) = Wx \]

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

E.g. Suppose that we found a \( W \) such that \( L = 0 \).
Q: Is this \( W \) unique?

Let’s look at an example

Adapted from CS 231n slides
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x,W)=Wx$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>3.2</th>
<th>1.3</th>
<th>2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

Before:

$= \max(0, 1.3 - 4.9 + 1) + \max(0, 2.0 - 4.9 + 1)$
$= \max(0, -2.6) + \max(0, -1.9)$
$= 0 + 0$
$= 0$

With $W$ twice as large:

$= \max(0, 2.6 - 9.8 + 1) + \max(0, 4.0 - 9.8 + 1)$
$= \max(0, -6.2) + \max(0, -4.8)$
$= 0 + 0$
$= 0$

Adapted from CS 231n slides
E.g. Suppose that we found a $W$ such that $L = 0$.

**Q: Is this $W$ unique?**

No, $2W$ also has $L=0$

How do we choose between $W$, $2W$, and $1e+7W$?

Adapted from from CS 231n slides
Regularization intuition: fitting a polynomial function

Adapted from CS 231n slides
Regularization intuition: fitting a polynomial function

Adapted from CS 231n slides
Regularization intuition: fitting a polynomial function

Regularization balances the simplicity of the function and loss, so we don’t overfit to the noises in the data.

Adapted from CS 231n slides
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

**Data loss**: Model predictions should match training data

**Regularization**: Prevent the model from doing too well on training data

*Adapted from CS 231n slides*
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

- **Data loss**: Model predictions should match training data
- **Regularization**: Prevent the model from doing *too* well on training data

\[ \lambda = \text{regularization strength (hyperparameter)} \]

**Simple examples**
- **L2 regularization**: \( R(W) = \sum_k \sum_l W_{k,l}^2 \)
- **L1 regularization**: \( R(W) = \sum_k \sum_l |W_{k,l}| \)
- **Elastic net (L1 + L2)**: \( R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \)

**More complex (DNN-specific)**:
- Dropout
- Batch/layer normalization
- Stochastic depth, fractional pooling, etc

Regularization: Implement a simple L2 regularizer

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

```python
def l2_regularized_hinge_loss(x, y, W, reg_coeff):
    data_loss = 0
    # calculate dataset loss
    for i in range(x.shape[0]):
        data_loss += hinge_loss_vec(x[i], y[i], W)

    # calculate weight regularization loss
    reg_loss = np.sum(np.square(W)) * reg_coeff

    return data_loss + reg_loss
```
What if we want probabilities?

We need a different classifier!*

```
cat  3.2
car  5.1
frog -1.7
```

Raw class scores

*Technically we can get probability from SVM classifiers too, see Platt scaling
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; \theta) \]

Probabilities must be >= 0

\[ p_\theta(Y = y_i|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \]

Probabilities must sum to 1

<table>
<thead>
<tr>
<th></th>
<th>Unnormalized log-probabilities / logits</th>
<th>Unnormalized probabilities</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>24.5</td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>164.0</td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>0.18</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Adapted from CS 231n slides

How do we compute the loss?

Cross-Entropy Loss Example
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; \theta) \]

\[ p_\theta(Y = y_i | X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \]

We maximize the probability of \( p_\theta(y_i | x_i) \)!

Finding a set of weights \( \theta \) that maximizes the probability of correct prediction: \( \arg\max_{\theta} \prod p_\theta(y_i | x_i) \)

This is equivalent to:

\[ \arg\max_{\theta} \sum \ln p_\theta(y_i | x_i) \]

\[ L_i = -\ln p_\theta(y_i | x_i) = -\ln \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) = -\ln(0.13) \]

1. Maximum Likelihood Estimation (MLE):
Choose weights to maximize the likelihood of observed data. In this case, the loss function is the Negative Log-Likelihood (NLL).
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

$$s = f(x_i; \theta)$$

$$p_{\theta}(Y = y_i|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

We maximize the probability of $$p_{\theta}(y_i|x_i)!$$

Finding a set of weights $$\theta$$ that maximizes the probability of correct prediction: $$\text{argmax}_{\theta} \prod p_{\theta}(y_i|x_i)$$

This is equivalent to:

$$\text{argmax}_{\theta} \sum \ln p_{\theta}(y_i|x_i)$$

$$L_i = -\ln p_{\theta}(y_i|x_i) = -\ln \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) = -\ln(0.13)$$

Cross-Entropy Loss Example
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; \theta) \]

\[ p_\theta(Y = y_i|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \]

2. Information theory view

Unnormalized log-probabilities / logits

Predicted Probs (softmax)

maximize agreement

Correct probs

Adapted from CS 231n slides
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; \theta) \]

\[ p_\theta(Y = y_i|X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \]

Cross Entropy:

\[ H(p, q) = -\sum x p(x) \ln q(x) \]

Cross Entropy Loss -> NLL

\[ H_i(p, p_\theta) = -\sum_{y \in Y} p(y|x_i) \ln p_\theta(y|x_i) \]

\[ = -\ln p_\theta(y_i|x_i) \]

\[ L = \sum H_i(p, p_\theta) = -\sum \ln p_\theta(y_i|x_i) \equiv NLL \]

2. Information theory view

Unnormalized log-probabilities / logits

<table>
<thead>
<tr>
<th>Class</th>
<th>Unnormalized Logits</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Predicted Probs (softmax)

<table>
<thead>
<tr>
<th>Class</th>
<th>Softmax Predicted Probs</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Correct Probs

<table>
<thead>
<tr>
<th>Class</th>
<th>Correct Probs</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1.00</td>
</tr>
<tr>
<td>car</td>
<td>0.00</td>
</tr>
<tr>
<td>frog</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Adapted from CS 231n slides
Softmax Classifier (Multinomial Logistic Regression)

NLL and CrossEntropy are different loss functions in PyTorch!

**CROSSENTROPYLOSS**

```python
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0) [SOURCE]
```

Expects unformalized logits as input (the function will apply softmax & log on top)

**NLLLOSS**

```python
CLASS torch.nn.NLLLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean') [SOURCE]
```

Expects log probabilities as input (do softmax yourself!)
**Softmax Classifier (Multinomial Logistic Regression)**

Want to interpret raw classifier scores as **probabilities**

\[ s = f(x_i; \theta) \]

\[ p_\theta(Y = y_i | X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \]  

**Softmax Function**

**Cross-entropy loss:**

\[ L_i = -\log(p_\theta(y_i | x_i)) \]

**Q:** What is the min/max of possible loss \( L_i \)?

Infimum is 0, max is unbounded (inf)

*Adapted from from CS 231n slides*
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; \theta)$$

$$p_\theta(Y = y_i | X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

**Softmax Function**

**Cross-entropy loss:**

$$L_i = -\log(p_\theta(y_i | x_i))$$

Q: At initialization all $s$ will be approximately equal; what is the loss?

Log(C), e.g. $\log(3) \approx 1.1$

Adapted from CS 231n slides
Q: Why softmax?

Use logistic function as example. Same as general softmax but for binary classification

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

Consider the following three basis for NLL:
1. Squash and clip value to (0, 1]
2. Logistic function
3. Logistic function but no log (just negative likelihood)
Q: Why softmax?

Why this?

Use logistic function as example. Same as general softmax but for binary classification

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

Consider the following three basis for NLL:
1. Squash and clip value to (0, 1]
2. Logistic function
3. Logistic function but no log (just negative likelihood)

\[ p_\theta(Y = y_i | X = x_i) = \frac{e^{s_i y_i}}{\sum_j e^{s_j}} \]
Q: Why softmax?

Use logistic function as example. Same as general softmax but for binary classification

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

Consider the following three basis for NLL:
1. Squash and clip value to (0, 1]
2. Logistic function
3. Logistic function but no log (just negative likelihood)

Why this?

\[ p_\theta(Y = y_i | X = x_i) = \frac{e^{s y_i}}{\sum_j e^{s_j}} \]

3. Negative likelihood w/ logistic function: saturated loss when classifier is very wrong
Q: Why softmax?

Use logistic function as example. Same as general softmax but for binary classification

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

Consider the following three basis for NLL:

1. Squash and clip value to (0, 1]
2. Logistic function
3. Logistic function but no log (just negative likelihood)

2. NLL w/ logistic: Strong guidance when classifier is wrong

$$p_\theta(Y = y_i | X = x_i) = \frac{e^{s y_i}}{\sum_j e^{s_j}}$$

Only saturate at convergence, e.g. \(\sigma(3) \approx 0.95\)
Loss functions: SVM and Softmax Classifier

• Loss function: performance measure to improve
  • Find weights that better satisfies the objective

• Multiclass SVM Classifier
  • Predicts class score
  • Hinge loss: “maximum margin” objective: \( L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \)

• Regularization
  • Prevent overly complex function that only works well on the training set

• Softmax Classifier
  • Predicts class probabilities
  • To train softmax classifiers: use NLL and Cross Entropy Loss
- Input (and representation)
- Functional form of the model
  - Including parameters
- Performance measure to improve
  - Loss or objective function
- **Algorithm for finding best parameters**
  - Optimization algorithm

Data: Image

Model: \( f(x, W) = Wx + b \)

Loss Function

Optimizer
Strategy #1: A first very bad idea solution: **Random search**

```python
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf")  # Python assigns the highest possible float value
for num in xrange(1000):
    W = np.random.randn(10, 3073) * 0.0001  # generate random parameters
    loss = L(X_train, Y_train, W)  # get the loss over the entire training set
    if loss < bestloss:  # keep track of the best solution
        bestloss = loss
        bestW = W
    print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (truncated: continues for 1000 lines)
```
Let's see how well this works on the test set...

```python
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% accuracy! not bad! (SOTA is ~99.7%)

*Adapted from CS 231n slides*
Given a model and loss function, finding the best set of weights is a **search problem**

- Find the best combination of weights that minimizes our loss function

**Several classes of methods:**

- Random search
- Genetic algorithms (population-based search)
- Gradient-based optimization

In deep learning, **gradient-based methods are dominant** although not the only approach possible

\[
\begin{bmatrix}
w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\
w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\
w_{21} & w_{22} & \cdots & w_{3m} & b_3 \\
\end{bmatrix}
\]

1. Calculate the gradients of a loss function with respect to a set of parameters (w’s).
2. Update the parameters towards the gradient direction that minimizes the loss.
Gradient Descent: Follow the Slope!
As weights change, the gradients change as well.

- This is often somewhat-smooth locally, so small changes in weights produce small changes in the loss.

We can therefore think about iterative algorithms that take current values of weights and modify them a bit.
We can find the steepest descent direction by computing the **derivative**:

\[
\frac{\partial f}{\partial w} = \lim_{h \to 0} \frac{f(w + h) - f(w)}{h}
\]

- **Gradient** is multi-dimensional derivatives
- Notation: \( \frac{\partial f}{\partial w} \) is the gradient of \( f \) (e.g., a loss function) with respect to variable \( w \) (e.g., a weight vector).
- \( \frac{\partial f}{\partial w} \) is of the **same shape** as \( w \)
- **Intuitively**: Measures how the function changes as the variable \( w \) changes by a small step size
- Steepest descent direction is the **negative gradient**
- **Gradient descent**: Minimize loss by changing parameters

Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current $W$:</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]</td>
<td>[?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,…]</td>
</tr>
</tbody>
</table>

loss 1.25347

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
## Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current W:</th>
<th>( W + h ) (first dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[?, ?, ?, ?, ?, ?, ?, ?, ?,...]</td>
</tr>
<tr>
<td><strong>loss 1.25347</strong></td>
<td><strong>loss 1.25322</strong></td>
<td></td>
</tr>
</tbody>
</table>

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (first dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34 + 0.0001, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, ?, ?,... ]</td>
</tr>
</tbody>
</table>

loss 1.25347

\[ \frac{df(x)}{dx} = \lim_{{h \to 0}} \frac{f(x + h) - f(x)}{h} \]

\[ \frac{(1.25322 - 1.25347)}{0.0001} = -2.5 \]

loss 1.25322

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current W:</th>
<th>W + ( h ) (second dim):</th>
<th>gradient ( dW ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, ?, ?, ?, ?, ?, ?, ?, ...]</td>
</tr>
</tbody>
</table>

loss 1.25347 | loss 1.25353 |
Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current W:</th>
<th>W + h (second dim):</th>
<th>gradient dW:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11 + 0.0001, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, 0.6, ?, ?, ?,...]</td>
</tr>
</tbody>
</table>

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

\[
\frac{1.25353 - 1.25347}{0.0001} = 0.6
\]

loss 1.25347 | loss 1.25353

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (third dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11, 0.78 + \textbf{0.0001}, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, 0.6, ?, ?, ?, ?, ?, ?, ...]</td>
</tr>
</tbody>
</table>

loss 1.25347 | loss 1.25347 |

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Calculate gradients: finite differences

<table>
<thead>
<tr>
<th>current $W$:</th>
<th>$W + h$ (third dim):</th>
<th>gradient $dW$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[0.34, -1.11, 0.78 + 0.0001, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]</td>
<td>[-2.5, 0.6, 0, ?, ?, ?, ?, ...]</td>
</tr>
</tbody>
</table>

loss 1.25347 loss 1.25347

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

\[
\frac{(1.25347 - 1.25347)}{0.0001} = 0
\]
Several ways to compute $\frac{\partial L}{\partial w_i}$

- Manual differentiation
- Symbolic differentiation
- Numerical differentiation
- Automatic differentiation

More on autodiff:
Numerical vs Analytic Gradients

\[
d\frac{f(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

**Numerical gradient:** slow :(, approximate :(, easy to write :)

**Analytic gradient:** fast :), exact :), error-prone :(  

Almost all differentiable functions that you can think of have analytical gradients implemented in popular libraries, e.g., PyTorch, TensorFlow.

If you want to derive your own gradients, check your implementation with numerical gradient. This is called a **gradient check**.
Composing simple functions creates complex analytical gradients

\[ \sin(x) \quad \log(x) \quad \cos(x) \quad x^3 \quad \exp(x) \]

Compose into a complex function

\[-\log \left( \frac{1}{1 + e^{-w \cdot x}} \right) \]

Adapted from slides by: Marc'Aurelio Ranzato, Yann LeCun
Decomposing a Function

Next time: Chain rule and Backpropagation!

Adapted from slides by: Marc'Aurelio Ranzato, Yann LeCun
The gradient descent algorithm

1. Choose a model: \( f(x, W) = Wx \)
2. Choose loss function: \( L_i = |y - Wx_i|^2 \)
3. Calculate partial derivative for each parameter: \( \frac{\partial L}{\partial w_i} \)
4. Update the parameters: \( w_i = w_i - \frac{\partial L}{\partial w_i} \)
5. Add learning rate to prevent too big of a step: \( w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \)

Repeat 3-5
\[
\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} \frac{\partial u}{\partial w}
\]

Next time: Chain rule and Backpropagation!