Lecture 7 Recap
Naïve Losses: L2 vs L1

• L2 Loss:
  \[ L^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 \]
  – Sum of squared differences (SSD)
  – Prone to outliers
  – Compute-efficient (optimization)
  – Optimum is the mean

• L1 Loss:
  \[ L^1 = \sum_{i=1}^{n} |y_i - f(x_i)| \]
  – Sum of absolute differences
  – Robust
  – Costly to compute
  – Optimum is the median
Binary Classification: Sigmoid

\[ \sigma(x, \theta) = \frac{1}{1 + e^{-\sum \theta_i x_i}} \]

Can be interpreted as a probability:

\[ p(y = 1|x, \theta) \]
Softmax Formulation

• What if we have multiple classes?

\[
p(y = 1|x, \theta) = \frac{e^{s_1}}{e^{s_1} + e^{s_2} + e^{s_3}}
\]

\[
p(y = 2|x, \theta) = \frac{e^{s_2}}{e^{s_1} + e^{s_2} + e^{s_3}}
\]

\[
p(y = 3|x, \theta) = \frac{e^{s_3}}{e^{s_1} + e^{s_2} + e^{s_3}}
\]
Example: Hinge vs Cross-Entropy

Hinge Loss: \( L_i = \sum_{k \neq y_i} \max(0, s_k - s_{y_i} + 1) \)

Cross Entropy: \( L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right) \)

Given the following scores for \( x_i \):

<table>
<thead>
<tr>
<th>Model</th>
<th>( s )</th>
<th>Hinge loss:</th>
<th>Cross Entropy loss:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>([5, -3, 2])</td>
<td>( \max(0, -3 - 5 + 1) + \max(0, 2 - 5 + 1) = 0 )</td>
<td>(- \ln \left( \frac{e^5}{e^5 + e^{-3} + e^2} \right) = 0.05 )</td>
</tr>
<tr>
<td>Model 2</td>
<td>([5, 10, 10])</td>
<td>( \max(0, 10 - 5 + 1) + \max(0, 10 - 5 + 1) = 12 )</td>
<td>(- \ln \left( \frac{e^5}{e^5 + e^{10} + e^{10}} \right) = 5.70 )</td>
</tr>
<tr>
<td>Model 3</td>
<td>([5, -20, -20])</td>
<td>( \max(0, -20 - 5 + 1) + \max(0, -20 - 5 + 1) = 0 )</td>
<td>(- \ln \left( \frac{e^5}{e^5 + e^{-20} + e^{-20}} \right) = 2 \times 10^{-11} )</td>
</tr>
</tbody>
</table>

- Cross Entropy *always* wants to improve! (loss never 0)
- Hinge Loss saturates.
Sigmoid Activation

Forward

\[
\frac{\partial L}{\partial w} = \frac{\partial s}{\partial w} \frac{\partial L}{\partial s}
\]

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

Saturated neurons kill the gradient flow

\[
\frac{\partial L}{\partial s} = \frac{\partial \sigma}{\partial s} \frac{\partial L}{\partial \sigma}
\]

\[
\frac{\partial \sigma}{\partial s} = \frac{\partial L}{\partial \sigma}
\]
TanH Activation

Still saturates

Zero-centered

[LeCun et al. 1991] Improving Generalization Performance in Character Recognition
Rectified Linear Units (ReLU)

- **Dead ReLU**

  What happens if a ReLU outputs zero?

- **Fast convergence**

- **Does not saturate**

[Krizhevsky et al. NeurIPS 2012] ImageNet Classification with Deep Convolutional Neural Networks

I2DL: Prof. Niessner, Prof. Leal-Taixé
Quick Guide

- Sigmoid is not really used.

- ReLU is the standard choice.

- Second choice are the variants of ReLU or Maxout.

- Recurrent nets will require TanH or similar.
Initialization is Extremely Important

- Optimum

\[ x^* = \arg \min f(x) \]

Initialization

Not guaranteed to reach the optimum
Xavier Initialization

• How to ensure the variance of the output is the same as the input?

\[
\frac{(n\text{Var}(w)\text{Var}(x))}{\text{Var}(w)} = 1
\]

\[
\text{Var}(w) = \frac{1}{n}
\]
ReLU Kills Half of the Data

$$\text{Var}(w) = \frac{2}{n}$$

It makes a huge difference!

[He et al., ICCV’15] He Initialization
Lecture 8
Data Augmentation
Data Augmentation

• A classifier has to be invariant to a wide variety of transformations
Google

Images

Cute
And Kittens
Clipart
Drawing
Cute Baby
White Cats And Kittens

Pose
Appearance
Illumination

I2DL: Prof. Niessner, Prof. Leal-Taixé
Data Augmentation

• A classifier has to be invariant to a wide variety of transformations

• Helping the classifier: synthesize data simulating plausible transformations
Data Augmentation

- a. No augmentation (= 1 image)
- b. Flip augmentation (= 2 images)
- c. Crop+Flip augmentation (= 10 images)

[Krizhevsky et al., NIPS’12] ImageNet
Data Augmentation: Brightness

- Random brightness and contrast changes

[Krizhevsky et al., NIPS'12] ImageNet
Data Augmentation: Random Crops

• Training: random crops
  – Pick a random $L$ in $[256, 480]$  
  – Resize training image, short side $L$
  – Randomly sample crops of 224x224

• Testing: fixed set of crops
  – Resize image at $N$ scales
  – 10 fixed crops of 224x224: (4 corners + 1 center) $\times$ 2 flips
Data Augmentation

• When comparing two networks make sure to use the same data augmentation!

• Consider data augmentation a part of your network design
Advanced Regularization
Weight Decay

- L2 regularization

\[ \Theta_{k+1} = \Theta_k - \epsilon \nabla_{\Theta}(\Theta_k, x, y) - \lambda \Theta_k \]

- Penalizes large weights
- Improves generalization
Early Stopping

Learning curves

Overfitting
Early Stopping

• Easy form of regularization

![Diagram of early stopping with θ values and overfitting]

I2DL: Prof. Niessner, Prof. Leal-Taixé
Bagging and Ensemble Methods

- Train multiple models and average their results

- E.g., use a different algorithm for optimization or change the objective function / loss function.

- If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size.
Bagging and Ensemble Methods

- Bagging: uses k different datasets

![Bagging Diagrams]

*Image Source: [Srivastava et al., JMLR'14] Dropout*
Dropout

• Disable a random set of neurons (typically 50%)

(a) Standard Neural Net
(b) After applying dropout.
Dropout: Intuition

- Using half the network = half capacity

(b) After applying dropout.

Furry
Has two eyes
Has a tail
Has paws
Has two ears

Redundant representations

[Srivastava et al., JMLR'14] Dropout
Dropout: Intuition

• Using half the network = half capacity
  – Redundant representations
  – Base your scores on more features

• Consider it as a model ensemble
Dropout: Intuition

• Two models in one

(b) After applying dropout.

[Srivastava et al., JMLR'14] Dropout
Dropout: Intuition

- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as two models in one
  - Training a large ensemble of models, each on different set of data (mini-batch) and with SHARED parameters

Reducing co-adaptation between neurons

[Srivastava et al., JMLR'14] Dropout
Dropout: Test Time

- All neurons are “turned on” – no dropout

[Srivastava et al., JMLR’14] Dropout

Conditions at train and test time are not the same
Dropout: Test Time

- **Test:**
  - $z = (\theta_1 x_1 + \theta_2 x_2) \cdot p$
  - $p = 0.5$

- **Train:**
  - $E[z] = \frac{1}{4} (\theta_10 + \theta_20$
  - $+ \theta_1 x_1 + \theta_20$
  - $+ \theta_10 + \theta_2 x_2$
  - $+ \theta_1 x_1 + \theta_2 x_2)$
  - $= \frac{1}{2} (\theta_1 x_1 + \theta_2 x_2)$

*Weight scaling inference rule*

[Srivastava et al., JMLR'14] Dropout
Dropout: Verdict

• Efficient bagging method with parameter sharing

• Try it!

• Dropout reduces the effective capacity of a model → larger models, more training time
Batch Normalization
Our Goal

• All we want is that our activations do not die out
Batch Normalization

- Wish: Unit Gaussian activations (in our example)
- Solution: let's do it

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

\( N = \text{mini-batch size} \)
\( D = \text{num of features} \)

Ioffe and Szegedy, PMLR'15 | Batch Normalization
Batch Normalization

- In each dimension of the features, you have a unit gaussian (in our example)

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

Mean of your mini-batch examples over feature k

Unit gaussian

D = num of features

N = mini-batch size
Batch Normalization

• In each dimension of the features, you have a unit gaussian (in our example)

• For NN in general \(\rightarrow\) BN normalizes the mean and variance of the inputs to your activation functions
BN Layer

- A layer to be applied after Fully Connected (or Convolutional) layers and before non-linear activation functions
Batch Normalization

• 1. Normalize

\[ \hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}, \]

Differentiable function so we can backprop through it.

• 2. Allow the network to change the range

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

These parameters will be optimized during backprop.

[Ioffe and Szegedy, PMLR’15] Batch Normalization
Batch Normalization

• 1. Normalize

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

• 2. Allow the network to change the range

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

The network can learn to undo the normalization

\[ \gamma^{(k)} = \sqrt{Var[x^{(k)}]} \]

\[ \beta^{(k)} = E[x^{(k)}] \]

[Ioffe and Szegedy, PMLR’15] Batch Normalization
Batch Normalization

• Ok to treat dimensions separately?
  Shown empirically that even if features are not correlated, convergence is still faster with this method

• You can set all biases of the layers before BN to zero, because they will be cancelled out by BN anyway
BN: Train vs Test

- Train time: mean and variance is taken over the mini-batch

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

- Test-time: what happens if we can just process one image at a time?
  - No chance to compute a meaningful mean and variance

Ioffe and Szegedy, PMLR’15 | Batch Normalization
BN: Train vs Test

Training: Compute mean and variance from mini-batch 1, 2, 3 …

Testing: Compute mean and variance by running an exponentially weighted averaged across training mini-batches. Use them as $\sigma_{test}^2$ and $\mu_{test}$.

\[
\begin{align*}
\text{Var}_{running} &= \beta_m \times \text{Var}_{running} + (1 - \beta_m) \times \text{Var}_{minibatch} \\
\mu_{running} &= \beta_m \times \mu_{running} + (1 - \beta_m) \times \mu_{minibatch}
\end{align*}
\]

$\beta_m$: momentum (hyperparameter)

[Ioffe and Szegedy, PMLR’15] Batch Normalization
BN: What do you get?

- Very deep nets are much easier to train → more stable gradients

- A much larger range of hyperparameters works similarly when using BN
BN: A Milestone

[w/o BN] [w/ BN]

[wu and he, ECCV'18] Group Normalization
BN: Drawbacks

Batch: also source of drawbacks

- Small batch
- Varying batch val error

[Wu and He, ECCV'18] Group Normalization
Other Normalizations

Our Method: Group Normalization

- GN is batch-independent
- Small batch
- Varying batch: val error

[Wu and He, ECCV’18] Group Normalization
Other Normalizations

Figure 2. Normalization methods. Each subplot shows a feature map tensor. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. Group Norm is illustrated using a group number of 2.

Group-wise computation. Group convolutions have been presented by AlexNet [28] for distributing a model into two GPUs. The concept of groups as a dimension for model design has been more widely studied recently. The work of ResNeXt [7] investigates the trade-off between depth, width, and groups, and it suggests that a larger number of groups can improve accuracy under similar computational cost. MobileNet [38] and Xception [39] exploit channel-wise (also called "depth-wise") convolutions, which are group convolutions with a group number equal to the channel number. ShuffleNet [40] proposes a channel shuffle operation that permutes the axes of grouped features. These methods all involve dividing the channel dimension into groups. Despite the relation to these methods, GN does not require group convolutions. GN is a generic layer, as we evaluate in standard ResNets [3].

3 Group Normalization

The channels of visual representations are not entirely independent. Classical features of SIFT [14], HOG [15], and GIST [41] are group-wise representations by design, where each group of channels is constructed by some kind of histogram. These features are often processed by group-wise normalization over each histogram or each orientation. Higher-level features such as VLAD [42] and Fisher Vectors (FV) [43] are also group-wise features where a group can be thought of as the sub-vector computed with respect to a cluster. Analogously, it is not necessary to think of deep neural network features as unstructured vectors. For example, for conv1 (the first convolutional layer) of a network, it is reasonable to expect a filter and its horizontal flipping to exhibit similar distributions of filter responses on natural images. If conv1 happens to approximately learn this pair of filters, or if the horizontal flipping (or other transformations) is made into the architectures by design [44, 45], then the corresponding channels of these filters can be normalized together.

The higher-level layers are more abstract and their behaviors are not as intuitive. However, in addition to orientations (SIFT [14], HOG [15], or [44, 45]), there are many factors that could lead to grouping, e.g., frequency, shapes, illumination, textures. Their coefficients can be interdependent. In fact, a well-accepted computational model in neuroscience is to normalize across the entire cell. [Wu and He, ECCV’18] Group Normalization
What We Know
What do we know so far?

[Diagram of a neural network with input layer, hidden layer 1, hidden layer 2, hidden layer 3, and output layer.]
What do we know so far?

Concept of a ‘Neuron’

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

\[ s = \sum \theta_i x_i \]

I2DL: Prof. Niessner, Prof. Leal-Taixé
What do we know so far?

Activation Functions (non-linearities)

- **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$
- **TanH**: $\tanh(x)$
- **ReLU**: $\max(0, x)$
- **Leaky ReLU**: $\max(0.1x, x)$
What do we know so far?

Backpropagation

I2DL: Prof. Niessner, Prof. Leal-Taixé
What do we know so far?

SGD Variations (Momentum, etc.)
What do we know so far?

Data Augmentation

- a. No augmentation (= 1 image)
- b. Flip augmentation (= 2 images)

Batch-Norm

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

Weight Initialization (e.g., Xavier/2)

Dropout

(b) After applying dropout.

Weight Regularization

e.g., \( L^2 \)-reg: \( R^2(W) = \sum_{i=1}^{N} w_i^2 \)
Why not simply more Layers?

- We cannot make networks arbitrarily complex

- Why not just go deeper and get better?
  - No structure!!
  - It is just brute force!
  - Optimization becomes hard
  - Performance plateaus / drops!
See you next week!
References

  – Chapter 6: Deep Feedforward Networks

• Bishop “Pattern Recognition and Machine Learning” (2006),
  – Chapter 5.5: Regularization in Network Nets

• http://cs231n.github.io/neural-networks-1/

• http://cs231n.github.io/neural-networks-2/