Lecture 39: Training Neural Networks (Cont’d)

CS 4670/5670
Sean Bell

Strawberry  Goblet  Throne
(Side Note for PA5) AlexNet: 1 vs 2 parts

Caffe represents caffe like the above image, but computes as if it were the bottom image using 2 “groups”
(Recall) Each iteration of training

(1) Forward Propagation:

\[ x \xrightarrow{\text{Function}} h \xrightarrow{\cdots} s \xrightarrow{\text{Function}} L \]

(2) Backward Propagation:

\[ \frac{\partial L}{\partial x} \xleftarrow{\text{Function}} \quad \frac{\partial L}{\partial h} \xleftarrow{\cdots} \quad \frac{\partial L}{\partial s} \xleftarrow{\text{Function}} \]

(3) Weight update:

\[ \theta \leftarrow \theta - \lambda \frac{\partial L}{\partial \theta} \]
(Recall) Babysitting the training process

Typical loss
(Recall) Babysitting the training process

![Graph showing loss over time with a note about bad initialization being a prime suspect.](image)

*Figure: Andrej Karpathy*
Weight Initialization

For deep nets, initialization is subtle and important:

Initialize weights to be smaller if there are more input connections:

\[ W = \text{np.random.randn}(n) \times \sqrt{2.0 / n} \]

For neural nets with ReLU, this will ensure all activations have the same variance

Initialization matters

Training can take much longer if not carefully initialized:

22 layer model  
30 layer model

Proper initialization is an active area of research

*Understanding the difficulty of training deep feedforward neural networks* by Glorot and Bengio, 2010

*Exact solutions to the nonlinear dynamics of learning in deep linear neural networks* by Saxe et al, 2013

*Random walk initialization for training very deep feedforward networks* by Sussillo and Abbott, 2014

*Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* by He et al., 2015

*Data-dependent Initializations of Convolutional Neural Networks* by Krähenbühl et al., 2015

*All you need is a good init*, Mishkin and Matas, 2015

...
(Recall) Regularization reduces overfitting

\[ L = L_{\text{data}} + L_{\text{reg}} \]

\[ L_{\text{reg}} = \lambda \frac{1}{2} ||W||_2^2 \]

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]
Example Regularizers

**L2 regularization**

\[ L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2 \]

(L2 regularization encourages small weights)

**L1 regularization**

\[ L_{\text{reg}} = \lambda \|W\|_1 = \lambda \sum_{ij} |W_{ij}| \]

(L1 regularization encourages sparse weights: weights are encouraged to reduce to exactly zero)

**“Elastic net”**

\[ L_{\text{reg}} = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_2^2 \]

(combine L1 and L2 regularization)

**Max norm**

Clamp weights to some max norm

\[ \|W\|_2^2 \leq c \]
"Weight decay"

Regularization is also called "weight decay" because the weights "decay" each iteration:

\[ L_{\text{reg}} = \lambda \frac{1}{2} ||W||^2 \quad \leftrightarrow \quad \frac{\partial L}{\partial W} = \lambda W \]

Gradient descent step:

\[ W \leftarrow W - \alpha \lambda W - \frac{\partial L_{\text{data}}}{\partial W} \]

Weight decay: \( \alpha \lambda \) (weights always decay by this amount)

**Note:** biases are sometimes excluded from regularization

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]
Dropout

Simple but powerful technique to reduce overfitting:

Dropout

Simple but powerful technique to reduce overfitting:

[Figure showing the comparison between models with and without dropout]

Dropout

Simple but powerful technique to reduce overfitting:

Note: Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

Dropout

How much dropout?    Around $p = 0.5$

(a) Keeping $n$ fixed.

(b) Keeping $pn$ fixed.

Dropout

Case study: [Krizhevsky 2012]

“Without dropout, our network exhibits substantial overfitting.”

[Barlow et al, “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS 2012]
**Dropout**

Dropout is a regularization technique that involves randomly setting input units to a neural network to 0 at test time with a certain probability. This prevents overfitting by reducing the reliance on any single input feature.

```python
p = 0.5  # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p  # first dropout mask
    H1 *= U1  # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p  # second dropout mask
    H2 *= U2  # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

*Example forward pass with a 3-layer network using dropout*

*(note, here X is a single input)*

*Figure: Andrej Karpathy*
Dropout

**Test time:** scale the activations

Expected value of a neuron $h$ with dropout:

$$E[h] = ph + (1 - p)0 = ph$$

```python
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p  # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p  # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

We want to keep the same expected value

*Figure: Andrej Karpathy*
Batch Normalization

“you want unit gaussian activations? just make them so.”

consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

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

this is a vanilla differentiable function...

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$
Batch Normalization

**Input:** Values of $x$ over a mini-batch: $B = \{x_1...m\}$; Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

\[
\begin{align*}
\mu_B & \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad // \text{mini-batch mean} \\
\sigma_B^2 & \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad // \text{mini-batch variance} \\
\hat{x}_i & \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad // \text{normalize} \\
y_i & \leftarrow \gamma\hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad // \text{scale and shift}
\end{align*}
\]

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

[IOFFE AND SZEGEDY, 2015]
Batch Normalization

Input: Values of \( x \) over a mini-batch: \( B = \{x_1 \ldots m\} \); Parameters to be learned: \( \gamma, \beta \)

Output: \( \{y_i = \text{BN}_{\gamma,\beta}(x_i)\} \)

\[
\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad // \text{mini-batch mean}
\]

\[
\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad // \text{mini-batch variance}
\]

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad // \text{normalize}
\]

\[
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad // \text{scale and shift}
\]

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)
Batch Normalization

Place after a FC or Convolutional layer, and before nonlinearity

[ioffe and Szegedy, 2015]
Transfer Learning ("fine-tuning")

1. Train on ImageNet

ImageNet data

2. Finetune network on your own data

your data
Transfer Learning ("fine-tuning")

This is not just a special trick; this is "the" method used by most papers
Transfer Learning ("fine-tuning")

E.g. Caffe Model Zoo: Lots of pretrained ConvNets
https://github.com/BVLC/caffe/wiki/Model-Zoo

Slide: Andrej Karpathy
Summary

- Preprocess the data (subtract mean, sub-crops)
- Initialize weights carefully
- Use Dropout and/or Batch Normalization
- Use SGD + Momentum
- Fine-tune from ImageNet
- Babysit the network as it trains
You are now ready.
You are now ready.
You are now ready.