

Lecture 3: Regularization and Optimization

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 1

April 9, 2024

Regularization -

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}}$$

Data loss: Model predictions should match training data

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \lambda R(W)$$

Data loss: Model predictions should match training data

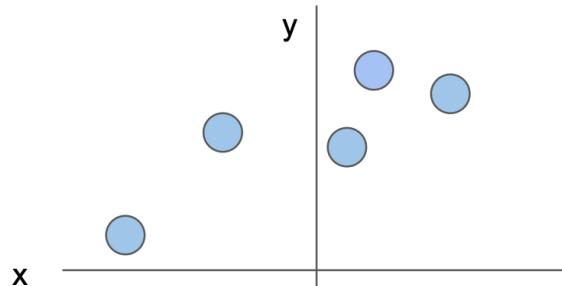
Regularization: Prevent the model from doing *too well* on training data

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 20

April 9, 2024

Regularization intuition: toy example training data

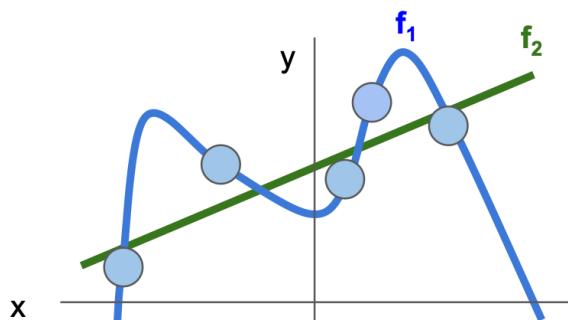


Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 21

April 9, 2024

Regularization intuition: Prefer Simpler Models

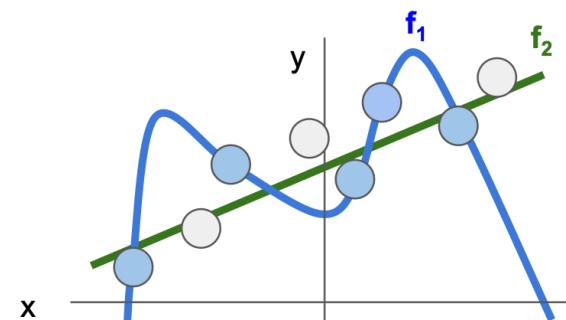


Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 22

April 9, 2024

Regularization: Prefer Simpler Models



Regularization pushes against fitting the data too well so we don't fit noise in the data

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 23

April 9, 2024

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

Data loss: Model predictions should match training data

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

Occam's Razor: Among multiple competing hypotheses, the simplest is the best, William of Ockham 1285-1347

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 24

April 9, 2024

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions}} + \underbrace{\lambda R(W)}_{\text{Regularization: Prevent the model from doing too well on training data}}$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

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

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 25

April 9, 2024

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

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

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

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

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

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:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \lambda R(W)$$

λ = regularization strength (hyperparameter)

Data loss: Model predictions should match training data

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

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

Regularization: Expressing Preferences

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Which of w1 or w2 will the L2 regularizer prefer?

$$w_1^T x = w_2^T x = 1$$

Regularization: Expressing Preferences

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Which of w1 or w2 will the L2 regularizer prefer?

L2 regularization likes to "spread out" the weights

$$w_1^T x = w_2^T x = 1$$

Regularization: Expressing Preferences

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Which of w1 or w2 will the L2 regularizer prefer?

L2 regularization likes to "spread out" the weights

$$w_1^T x = w_2^T x = 1$$

Which one would L1 regularization prefer?

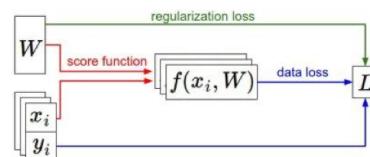
Recap

- We have some dataset of (x, y)
- We have a **score function**: $s = f(x; W) = Wx$ e.g.
- We have a **loss function**:

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right) \text{ Softmax}$$

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

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W) \text{ Full loss}$$



Lecture 6 (Part 2): Training Neural Networks

Overview

1. **One time set up:** activation functions, preprocessing, weight initialization, regularization, gradient checking
1. **Training dynamics:** babysitting the learning process, parameter updates, hyperparameter optimization
1. **Evaluation:** model ensembles, test-time augmentation, transfer learning

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 6

April 17, 2024

Activation Functions

Fei-Fei Li, Ehsan Adeli, Zane Durante

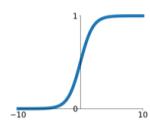
Lecture 7 - 7

April 17, 2024

Activation Functions

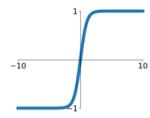
Sigmoid

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



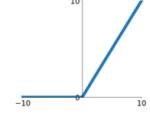
tanh

$$\tanh(x)$$

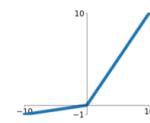


ReLU

$$\max(0, x)$$



$$\text{Leaky ReLU}$$
$$\max(0.1x, x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

Fei-Fei Li, Ehsan Adeli, Zane Durante

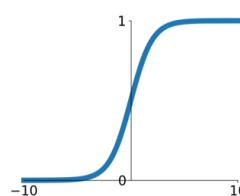
Lecture 7 - 9

April 17, 2024

Activation Functions

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

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



Sigmoid

3 problems:

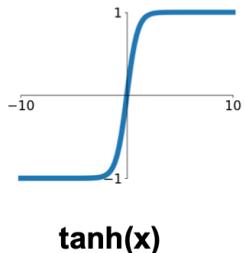
1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered
3. $\exp()$ is a bit compute expensive

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 109

April 17, 2024

Activation Functions



- Squashes numbers to range [-1, 1]
- zero centered (nice)
- still kills gradients when saturated :)

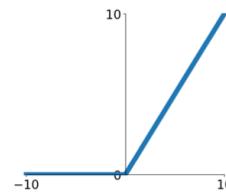
[LeCun et al., 1991]

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 19

April 17, 2024

Activation Functions



ReLU
(Rectified Linear Unit)

- 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)

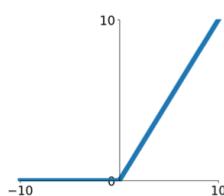
[Krizhevsky et al., 2012]

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 20

April 17, 2024

Activation Functions



ReLU
(Rectified Linear Unit)

- 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)

- Not zero-centered output
- An annoyance:

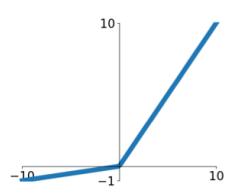
hint: what is the gradient when $x < 0$?

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 22

April 17, 2024

Activation Functions



Leaky ReLU
 $f(x) = \max(0.01x, x)$

[Maas et al., 2013]
[He et al., 2015]

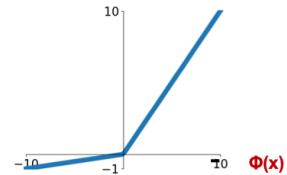
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- will not “die”.

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 26

April 17, 2024

Activation Functions



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

[Mass et al., 2013]
[He et al., 2015]

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- will not “die”.

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

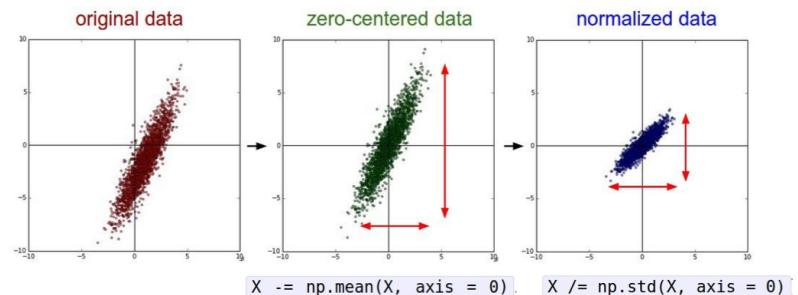
backprop into α (parameter)

TLDR: In practice:

- Use **ReLU**. Be careful with your learning rates
- Try out **Leaky ReLU / PReLU / GELU**
 - To squeeze out some marginal gains
- Don't use **sigmoid or tanh**

Data Preprocessing

Data Preprocessing



(Assume X [NxD] is data matrix,
each example in a row)

Weight Initialization

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 35

April 17, 2024

- First idea: **Small random numbers**
(gaussian with zero mean and 1e-2 standard deviation)

```
W = 0.01 * np.random.randn(Din, Dout)
```

Works ~okay for small networks, but problems with deeper networks.

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 38

April 17, 2024

Weight Initialization: Activation statistics

```
dims = [4096] * 7    Forward pass for a 6-layer  
hs = []               net with hidden size 4096  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.01 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

What will happen to the activations for the last layer?

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 39

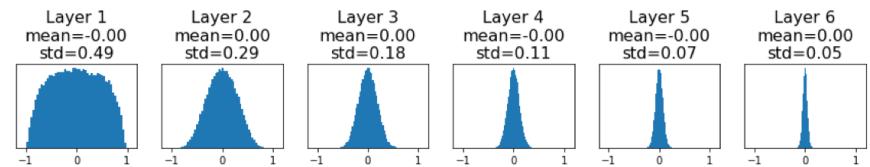
April 17, 2024

Weight Initialization: Activation statistics

```
dims = [4096] * 7    Forward pass for a 6-layer  
hs = []               net with hidden size 4096  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.01 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

All activations tend to zero for deeper network layers

Q: What do the gradients dL/dW look like?



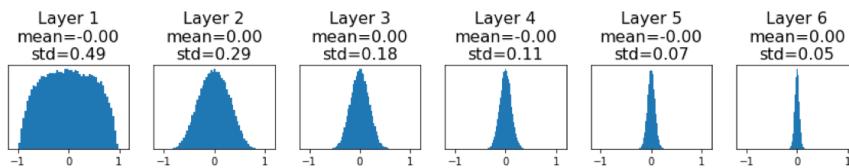
Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 40

April 17, 2024

Weight Initialization: Activation statistics

```
dims = [4096] * 7    Forward pass for a 6-layer  
hs = []  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.01 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```



Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 41

April 17, 2024

Weight Initialization: Activation statistics

```
dims = [4096] * 7    Increase std of initial  
hs = []  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.05 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

What will happen to the activations for the last layer?

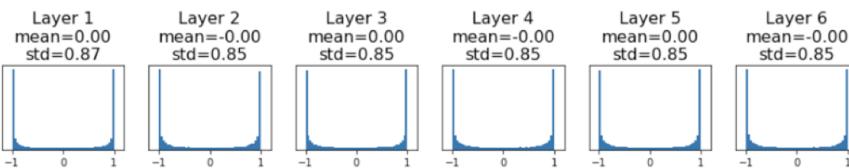
Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 42

April 17, 2024

Weight Initialization: Activation statistics

```
dims = [4096] * 7    Increase std of initial  
hs = []  
weights from 0.01 to 0.05  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.05 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```



Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 43

April 17, 2024

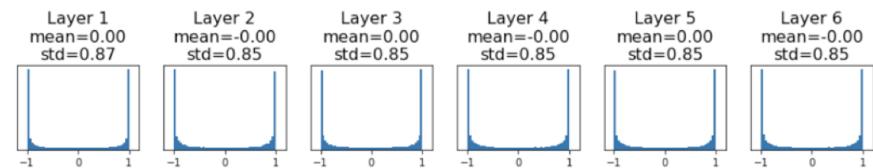
Weight Initialization: Activation statistics

```
dims = [4096] * 7    Increase std of initial  
hs = []  
weights from 0.01 to 0.05  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = 0.05 * np.random.randn(Din, Dout)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

All activations saturate

Q: What do the gradients look like?

A: Local gradients all zero, no learning =(



Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 44

April 17, 2024

Weight Initialization: “Xavier” Initialization

```
dims = [4096] * 7      "Xavier" initialization:  
hs = []  
std = 1/sqrt(Din)  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = np.random.randn(Din, Dout) / np.sqrt(Din)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

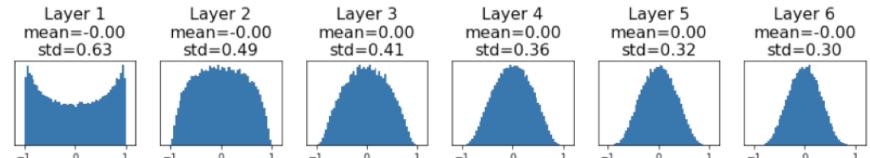
Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 45

April 17, 2024

Weight Initialization: “Xavier” Initialization

```
dims = [4096] * 7      "Xavier" initialization:  
hs = []  
std = 1/sqrt(Din)  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = np.random.randn(Din, Dout) / np.sqrt(Din)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

“Just right”: Activations are nicely scaled for all layers!



Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 46

April 17, 2024

Weight Initialization: What about ReLU?

```
dims = [4096] * 7      Change from tanh to ReLU  
hs = []  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = np.random.randn(Din, Dout) / np.sqrt(Din)  
    x = np.maximum(0, x.dot(W))  
    hs.append(x)
```

Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 48

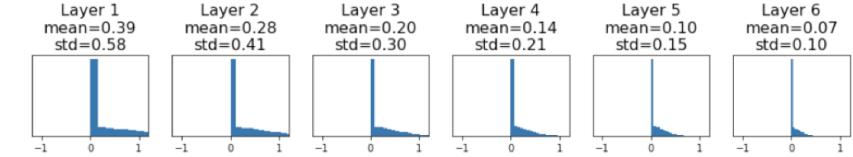
April 17, 2024

Weight Initialization: What about ReLU?

```
dims = [4096] * 7      Change from tanh to ReLU  
hs = []  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = np.random.randn(Din, Dout) / np.sqrt(Din)  
    x = np.maximum(0, x.dot(W))  
    hs.append(x)
```

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning =(

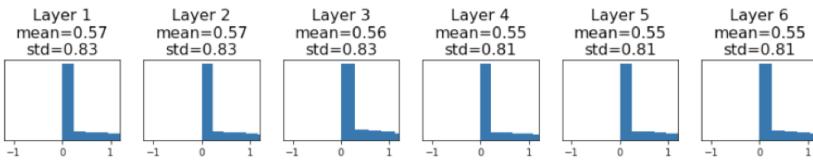


Fei-Fei Li, Ehsan Adeli, Zane Durante Lecture 7 - 49

April 17, 2024

Weight Initialization: Kaiming / MSRA Initialization

```
dims = [4096] * 7  ReLU correction: std = sqrt(2 / Din) "Just right": Activations are
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```



He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Proper initialization is an ongoing 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

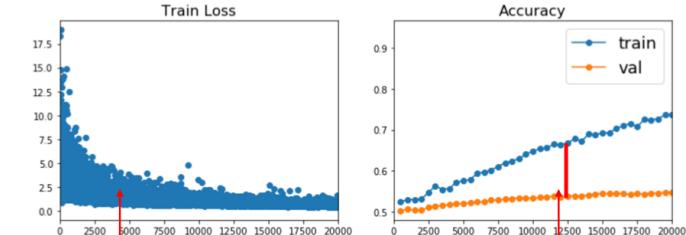
All you need is a good init, Mishkin and Matas, 2015

Fixup Initialization: Residual Learning Without Normalization, Zhang et al, 2019

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, Frankle and Carbin, 2019

Training vs. Testing Error

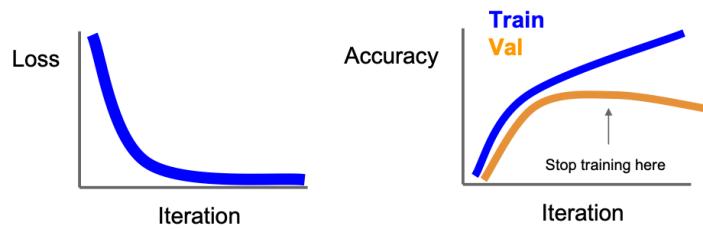
Beyond Training Error



Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

Early Stopping: Always do this



Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot
that worked best on val

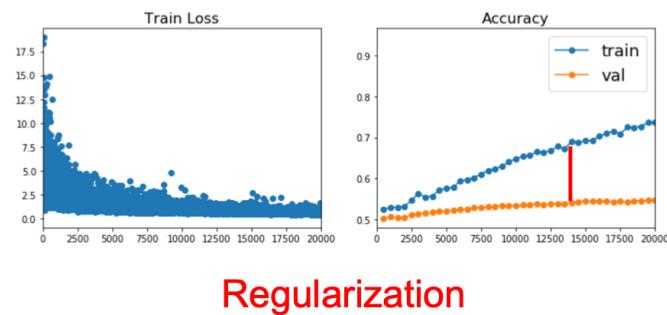
Model Ensembles

1. Train multiple independent models
2. At test time average their results

(Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

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

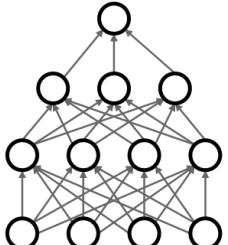
$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Elastic net (L1 + L2)

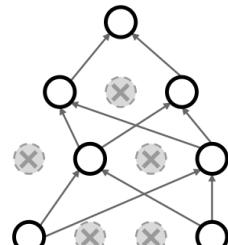
$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Regularization: Dropout

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



Srivastava et al., "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014



Regularization: Dropout

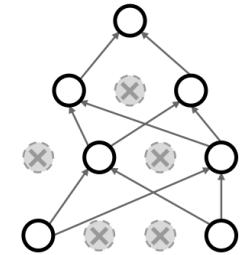
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

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

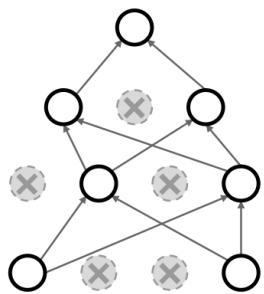
    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout



Regularization: Dropout

How can this possibly be a good idea?

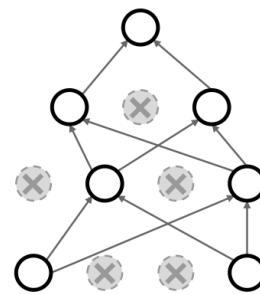


Forces the network to have a redundant representation;
Prevents co-adaptation of features



Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

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

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Dropout: Test time

Dropout makes our output random!

$$\text{Output (label)} \quad \text{Input (image)}$$
$$y = f_W(x, z)$$

Random mask

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

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

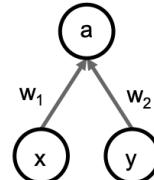
But this integral seems hard ...

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.



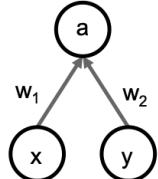
Dropout: Test time

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: $E[a] = w_1x + w_2y$



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: $E[a] = w_1x + w_2y$

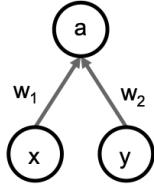
$$\begin{aligned} \text{During training we have: } E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

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: $E[a] = w_1x + w_2y$

$$\begin{aligned} \text{During training we have: } E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

At test time, multiply by dropout probability

Dropout: Test time

```

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
  
```

At test time all neurons are active always

=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

```

"""
Vanilla Dropout: Not recommended implementation (see notes below)
"""

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)

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
  
```

Dropout Summary

drop in train time

scale at test time

More common: “Inverted dropout”

```

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

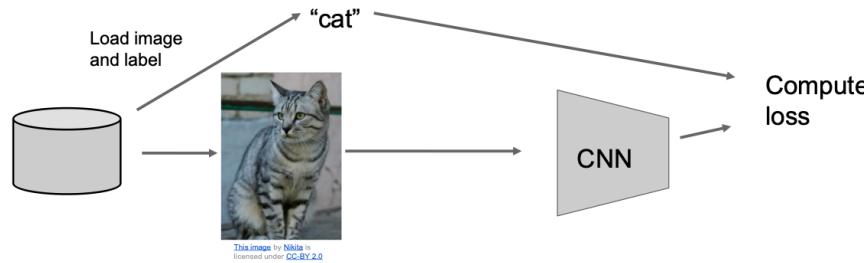
def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) /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

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

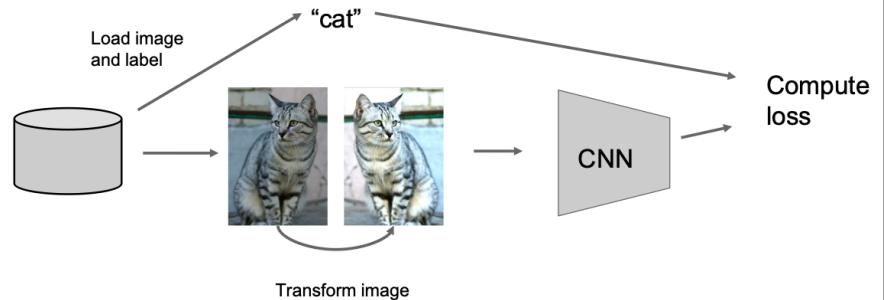
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
  
```

test time is unchanged!

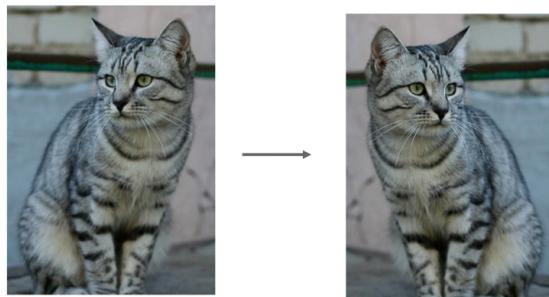
Regularization: Data Augmentation



Regularization: Data Augmentation



Data Augmentation Horizontal Flips

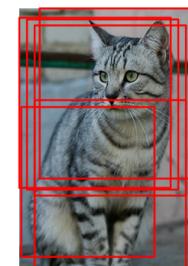


Data Augmentation Random crops and scales

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



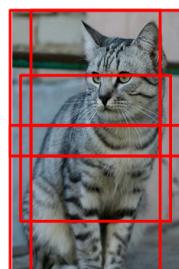
Data Augmentation

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

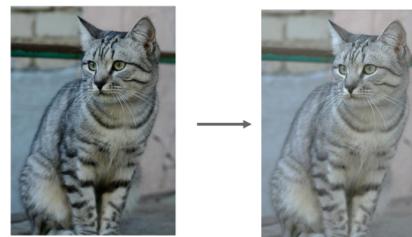
ResNet:

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

Data Augmentation

Color Jitter

Simple: Randomize contrast and brightness



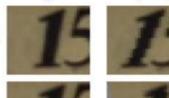
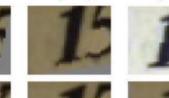
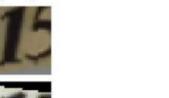
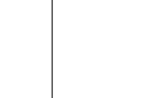
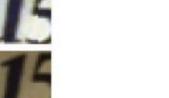
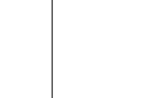
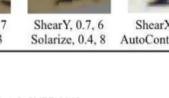
Data Augmentation

Get creative for your problem!

Examples of data augmentations:

- translation
- rotation
- stretching
- shearing,
- lens distortions, ... (go crazy)

Automatic Data Augmentation

	Original	Sub-policy 1	Sub-policy 2	Sub-policy 3	Sub-policy 4	Sub-policy 5
Batch 1						
Batch 2						
Batch 3						

Regularization: Cutout

Training: Set random image regions to zero

Testing: Use full image

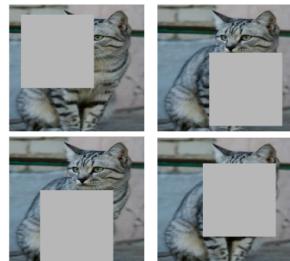
Examples:

Dropout

Batch Normalization

Data Augmentation

Cutout / Random Crop



Works very well for small datasets like CIFAR,
less common for large datasets like ImageNet

DeVries and Taylor, "Improved Regularization of
Convolutional Neural Networks with Cutout", arXiv 2017

Regularization - In practice

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout

Batch Normalization

Data Augmentation

Cutout / Random Crop

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout especially for small classification datasets

Choosing Hyperparameters

(without tons of GPUs)

Choosing Hyperparameters

Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization
e.g. $\log(C)$ for softmax with C classes

Random guessing $\rightarrow 1/C$ probability for each class
Softmax Loss $\rightarrow -\log(1/C) = \log(C)$

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization

Loss not going down? LR too low, bad initialization

Loss explodes to Inf or NaN? LR too high, bad initialization

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-4

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try: 1e-4, 1e-5, 0

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

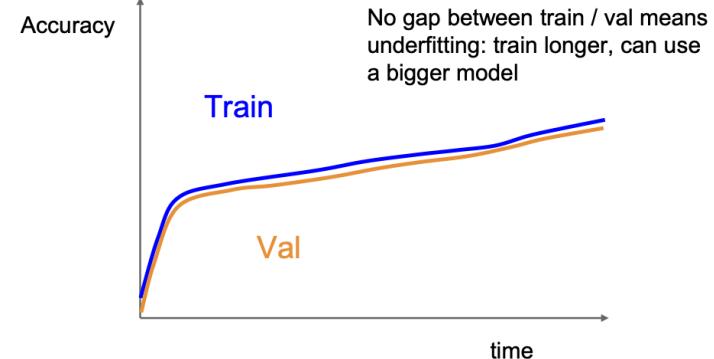
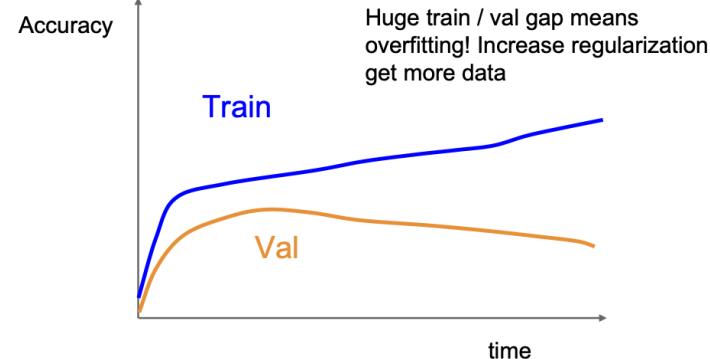
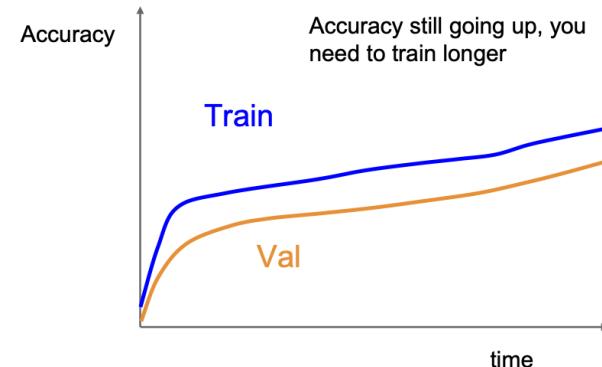
Step 4: Coarse grid, train for ~1-5 epochs

Step 5: Refine grid, train longer

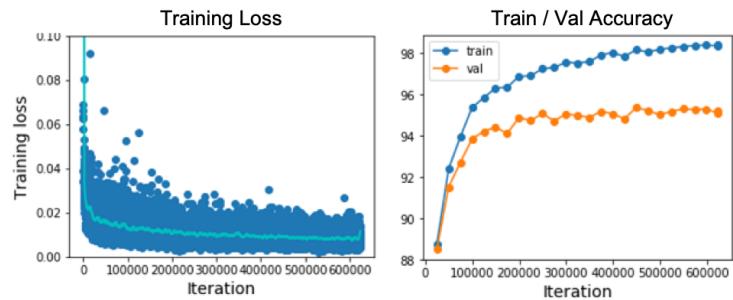
Pick best models from Step 4, train them for longer (~10-20 epochs) with constant learning rate

Choosing Hyperparameters

- Step 1:** Check initial loss
- Step 2:** Overfit a small sample
- Step 3:** Find LR that makes loss go down
- Step 4:** Coarse grid, train for ~1-5 epochs
- Step 5:** Refine grid, train longer
- Step 6:** Look at loss and accuracy curves



Look at learning curves!



Losses may be noisy, use a scatter plot and also plot moving average to see trends better

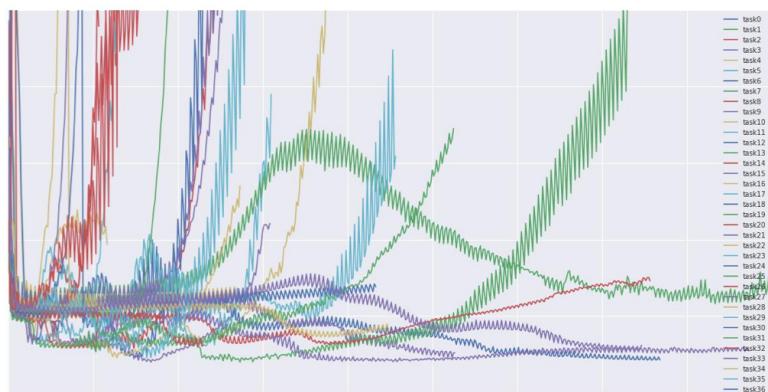
Cross-validation

We develop "command centers" to visualize all our models training with different hyperparameters

check out [weights and biases](#)



You can plot all your loss curves for different hyperparameters on a single plot



Choosing Hyperparameters

- Step 1:** Check initial loss
- Step 2:** Overfit a small sample
- Step 3:** Find LR that makes loss go down
- Step 4:** Coarse grid, train for ~1-5 epochs
- Step 5:** Refine grid, train longer
- Step 6:** Look at loss and accuracy curves
- Step 7:** GOTO step 5