

Lecture 6 (Part 2): Training Neural Networks

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 1

April 17, 2024

Gradient Descent

```
# Vanilla Gradient Descent
```

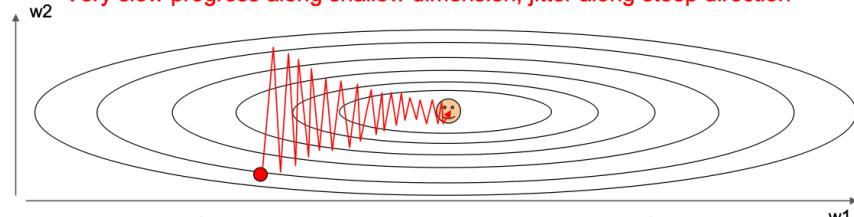
```
while True:  
    weights_grad = evaluate_gradient(loss_fun, data, weights)  
    weights += - step_size * weights_grad # perform parameter update
```

Optimization: Problem #1 with SGD

What if loss changes quickly in one direction and slowly in another?

What does gradient descent do?

Very slow progress along shallow dimension, jitter along steep direction



Aside: Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Fei-Fei Li, Ehsan Adeli, Zane Durante

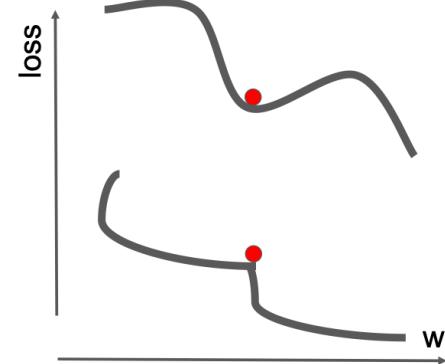
Lecture 3 - 60

April 9, 2024

Optimization: Problem #2 with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient,
gradient descent
gets stuck



Fei-Fei Li, Ehsan Adeli, Zane Durante

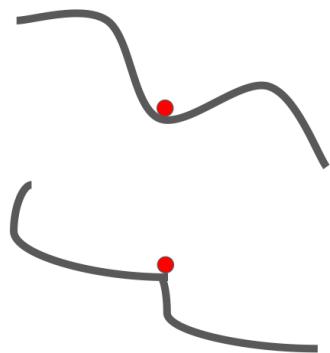
Lecture 3 - 62

April 9, 2024

Optimization: Problem #2 with SGD

What if the loss function has a **local minima** or **saddle point**?

Saddle points much more common in high dimension



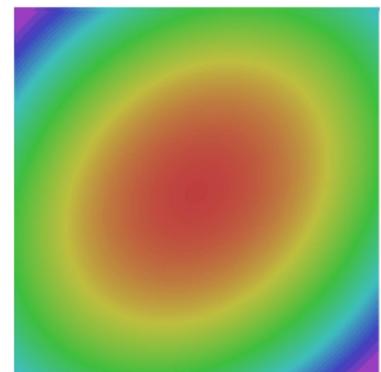
Dauphin et al, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization", NIPS 2014

Optimization: Problem #3 with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum:

continue moving in the general direction as the previous iterations

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:  
    dx = compute_gradient(x)  
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

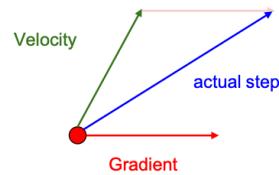
```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x -= learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

SGD+Momentum

Momentum update:



Combine gradient at current point with velocity to get step used to update weights

Nesterov, "A method of solving a convex programming problem with convergence rate $O(1/k^2)$ ", 1983
Nesterov, "Introductory lectures on convex optimization: a basic course", 2004
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

More Complex Optimizers: RMSProp

SGD +
Momentum

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

Adds element-wise scaling of the
gradient based on the historical sum of
squares in each dimension (with decay)

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 71

April 9, 2024

More Complex Optimizers: RMSProp

SGD +
Momentum

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

"Per-parameter learning rates"
or "adaptive learning rates"

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 72

April 9, 2024

RMSProp

RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Q: What happens with RMSProp?

Progress along "steep" directions is damped;
progress along "flat" directions is accelerated

Tieleman and Hinton, 2012

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 74

April 9, 2024

Adam (almost)

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7)
```

Momentum
RMSProp

Sort of like RMSProp with momentum

Q: What happens at first timestep?

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 77

April 9, 2024

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that
first and second moment
estimates start at zero

Adam with **beta1 = 0.9**,
beta2 = 0.999, and **learning_rate = 1e-3 or 5e-4**
is a great starting point for many models!

Kingma and Ba, "Adam: A method for stochastic optimization", ICLR 2015

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 79

April 9, 2024

Learning rate schedules

```
# Vanilla Gradient Descent
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

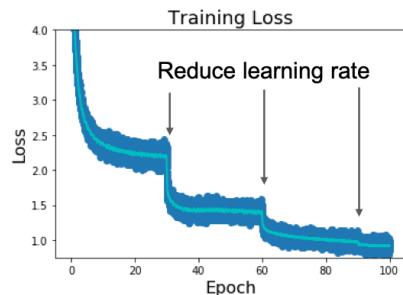
Learning rate

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 86

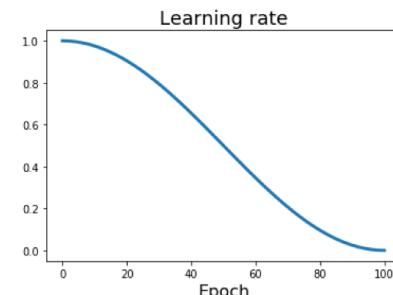
April 9, 2024

Learning rate decays over time



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

$$\text{Cosine: } \alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(t\pi/T))$$

Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
Feichtenhofer et al, "SlowFast Networks for Video Recognition", arXiv 2018
Child et al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 89

April 9, 2024

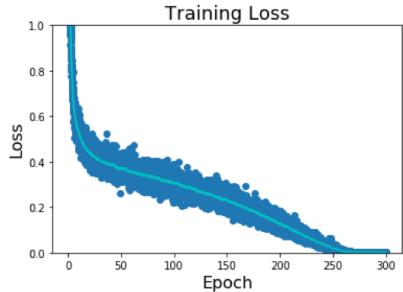
Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 90

April 9, 2024

α_0 : Initial learning rate
 α_t : Learning rate at epoch t
 T : Total number of epochs

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

$$\text{Cosine: } \alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$$

α_0 : Initial learning rate
 α_t : Learning rate at epoch t
 T : Total number of epochs

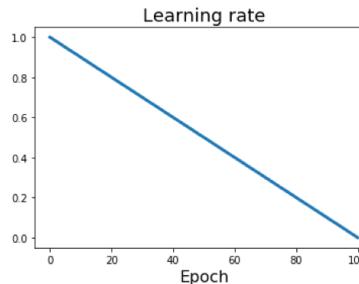
Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
Feichtenhofer et al, "SlowFast Networks for Video Recognition", arXiv 2018
Child et al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 91

April 9, 2024

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

$$\text{Cosine: } \alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$$

$$\text{Linear: } \alpha_t = \alpha_0(1 - t/T)$$

α_0 : Initial learning rate
 α_t : Learning rate at epoch t
 T : Total number of epochs

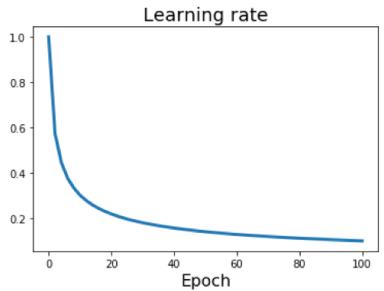
Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 92

April 9, 2024

Learning Rate Decay



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

$$\text{Cosine: } \alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$$

$$\text{Linear: } \alpha_t = \alpha_0(1 - t/T)$$

$$\text{Inverse sqrt: } \alpha_t = \alpha_0/\sqrt{t}$$

α_0 : Initial learning rate
 α_t : Learning rate at epoch t
 T : Total number of epochs

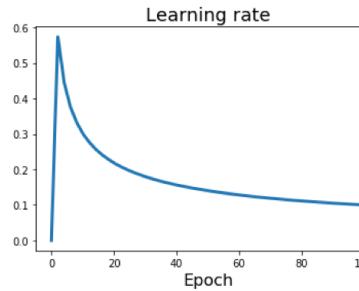
Vaswani et al, "Attention is all you need", NIPS 2017

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 93

April 9, 2024

Learning Rate Decay: Linear Warmup



High initial learning rates can make loss explode; linearly increasing learning rate from 0 over the first ~5,000 iterations can prevent this.

Empirical rule of thumb: If you increase the batch size by N, also scale the initial learning rate by N

Goyal et al, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", arXiv 2017

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 3 - 94

April 9, 2024

In practice:

- **Adam(W)** is a good default choice in many cases; it often works ok even with constant learning rate
- **SGD+Momentum** can outperform Adam but may require more tuning of LR and schedule
- If you can afford to do full batch updates then look beyond 1st order optimization (**2nd order and beyond**)

Fei-Fei Li, Ehsan Adeli, Zane Durante

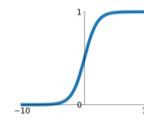
Lecture 3 - 100

April 9, 2024

Activation Functions

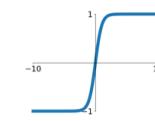
Sigmoid

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



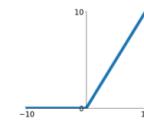
tanh

$$\tanh(x)$$



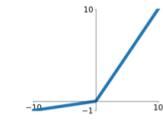
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

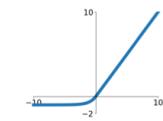


Maxout

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

ELU

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



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**

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 31

April 17, 2024

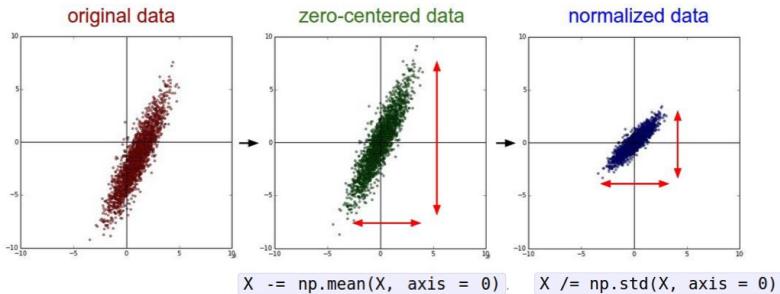
Data Preprocessing

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 7 - 32

April 17, 2024

Data Preprocessing



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

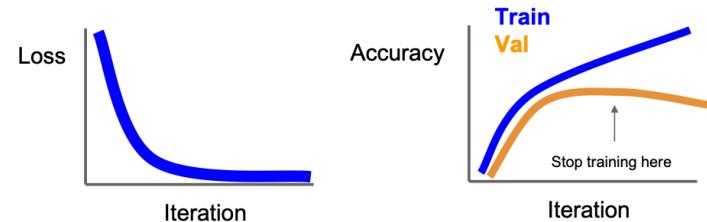
TLDR: In practice for Images: center only

e.g. consider CIFAR-10 example with $[32,32,3]$ images

- Subtract the mean image (e.g. AlexNet)
(mean image = $[32,32,3]$ array)
- Subtract per-channel mean (e.g. VGGNet)
(mean along each channel = 3 numbers)
- Subtract per-channel mean and
Divide by per-channel std (e.g. ResNet and beyond)
(mean along each channel = 3 numbers)

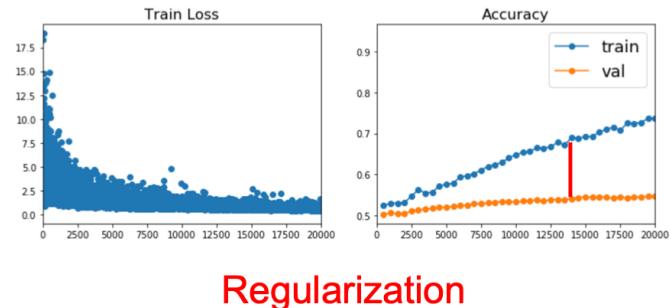
Training vs. Testing Error

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

How to improve single-model performance?



Regularization: Add term to loss

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

In common use:

L2 regularization

L1 regularization

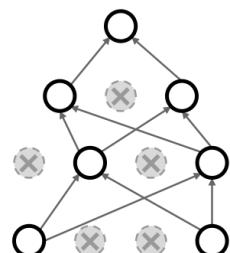
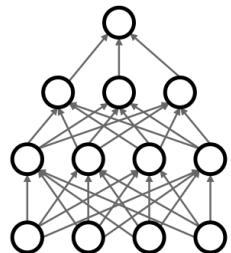
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

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

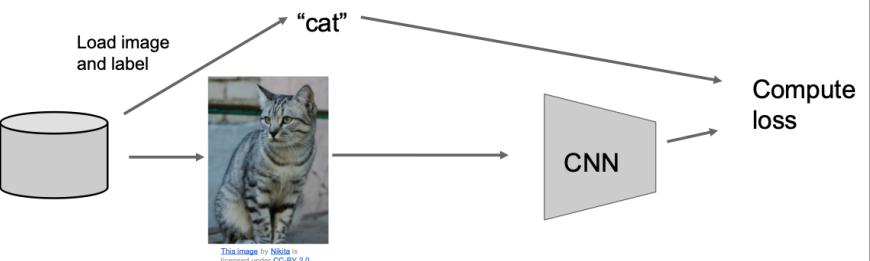
$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Regularization: Dropout

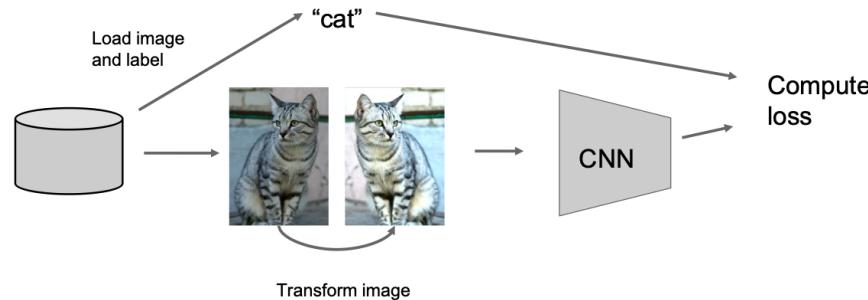
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



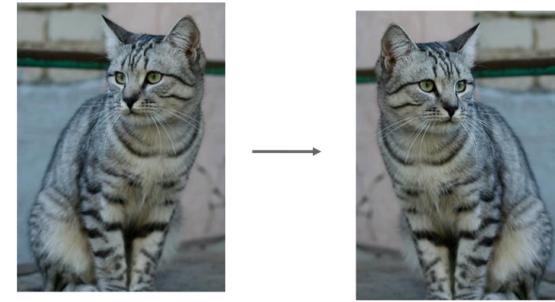
Regularization: Data Augmentation



Regularization: Data Augmentation

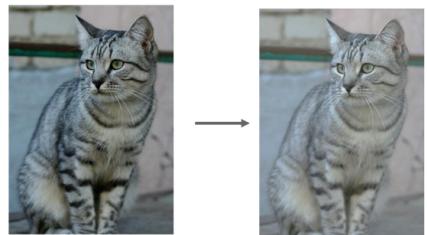


Data Augmentation Horizontal Flips



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						

Cubuk et al., "AutoAugment: Learning Augmentation Strategies from Data", CVPR 2019

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

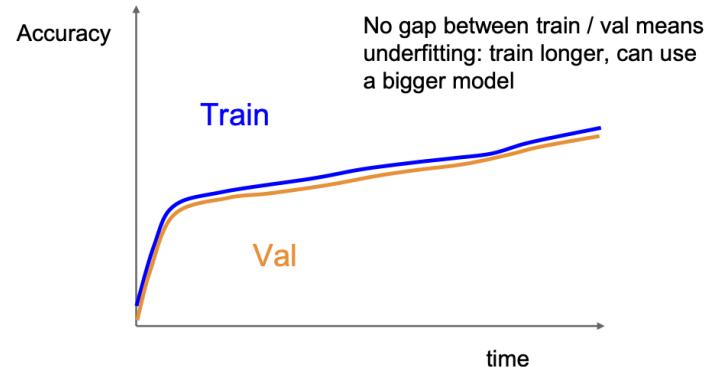
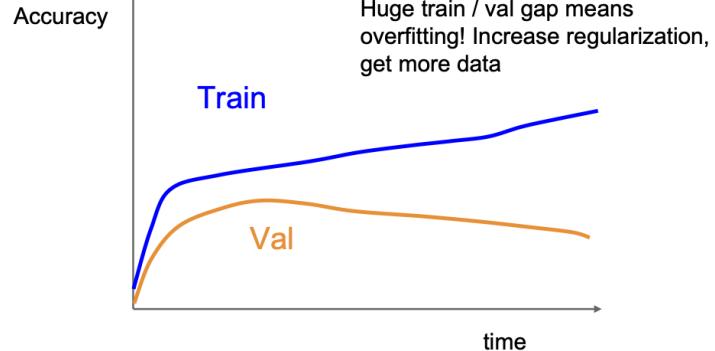
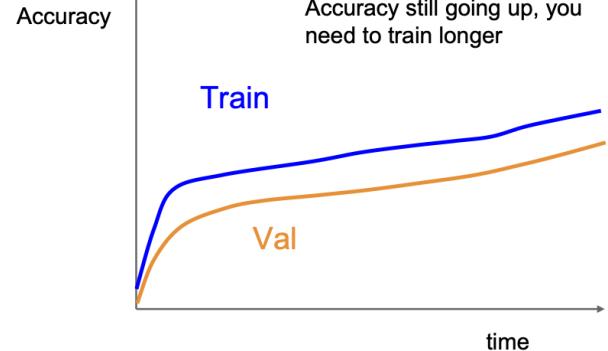
Step 2: Overfit a small sample

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



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**

Summary

TLDRs

We looked in detail at:

- Activation Functions ([use ReLU](#))
- Data Preprocessing ([images: subtract mean](#))
- Weight Initialization ([use Xavier/Kaiming init](#))
- Batch Normalization ([use this!](#))
- Transfer learning ([use this if you can!](#))