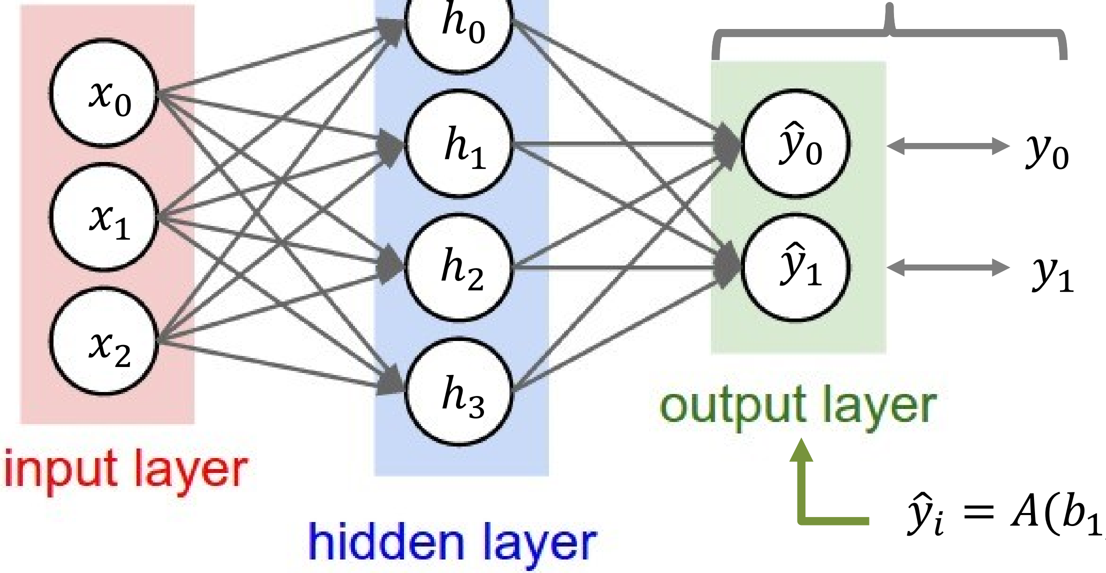
Training Neural Networks

Lecture 5 Recap

Gradient Descent for Neural Networks

Loss function

$$L_i = (\hat{y}_i - y_i)^2$$



$$\nabla_{\boldsymbol{W},\boldsymbol{b}} f_{\{\boldsymbol{x},\boldsymbol{y}\}}(\boldsymbol{W}) = \begin{bmatrix} \frac{\partial f}{\partial w_{0,0,0}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial w_{l,m,n}} \\ \cdots \\ \cdots \\ \frac{\partial f}{\partial b_{l,m}} \end{bmatrix}$$

$$\hat{y}_i = A(b_{1,i} + \sum_j h_j w_{1,i,j})$$

$$h_j = A(b_{0,j} + \sum_k x_k w_{0,j,k})$$

Just simple:

$A(x) = \max(0, x)$

Stochastic Gradient Descent (SGD)

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{k}, \underbrace{\boldsymbol{x}_{\{1..m\}}, \boldsymbol{y}_{\{1..m\}}}_{\text{Mini batch}})$$

$\boldsymbol{k}$ now refers to $\boldsymbol{k}$-th iteration

$$\nabla_{\boldsymbol{\theta}} L = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L_i$$

$\boldsymbol{m}$ training samples in the current minibatch

Gradient for the $\boldsymbol{k}$-th minibatch

Gradient Descent with Momentum

$$\boldsymbol{v}^{k+1} = \beta \cdot \boldsymbol{v}^{k} + \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{k})$$

accumulation rate
('friction', momentum)

velocity

Gradient of current minibatch

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \alpha \cdot \boldsymbol{v}^{k+1}$$

model

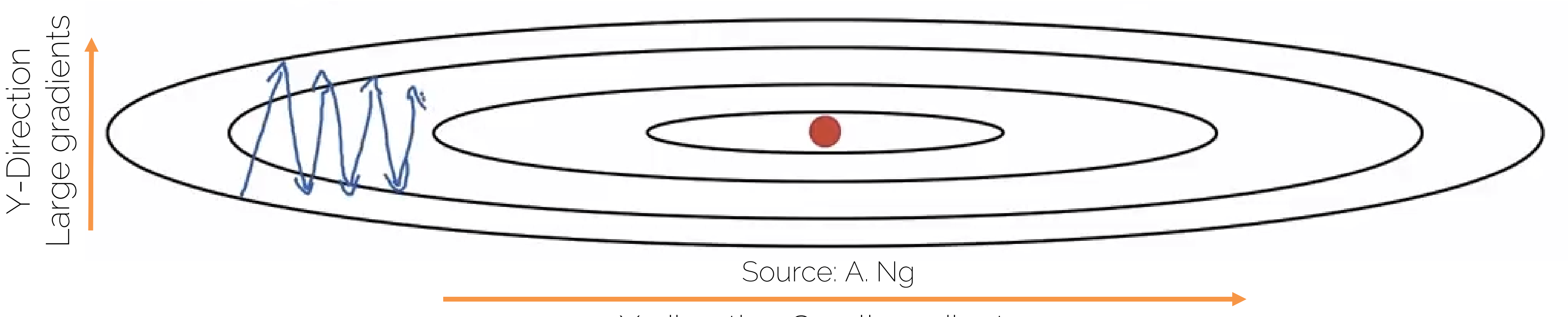
learning rate

velocity

Exponentially-weighted average of gradient

Important: velocity $\boldsymbol{v}^{k}$ is vector-valued!

RMSProp



(Uncentered) variance of gradients
→ second momentum

$$\boldsymbol{s}^{k+1} = \beta \cdot \boldsymbol{s}^{k} + (1-\beta)[\nabla_{\boldsymbol{\theta}} L \circ \nabla_{\boldsymbol{\theta}} L]$$

We're dividing by square gradients:
- Division in Y-Direction will be large
- Division in X-Direction will be small

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^{k} - \alpha \cdot \frac{\nabla_{\boldsymbol{\theta}} L}{\sqrt{\boldsymbol{s}^{k+1}} + \epsilon}$$

Can increase learning rate!

Adam

- Combines Momentum and RMSProp

$$\boldsymbol{m}^{k+1} = \beta_1 \cdot \boldsymbol{m}^k + (1-\beta_1)\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^k) \qquad \boldsymbol{v}^{k+1} = \beta_2 \cdot \boldsymbol{v}^k + (1-\beta_2)[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^k) \circ \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^k)$$

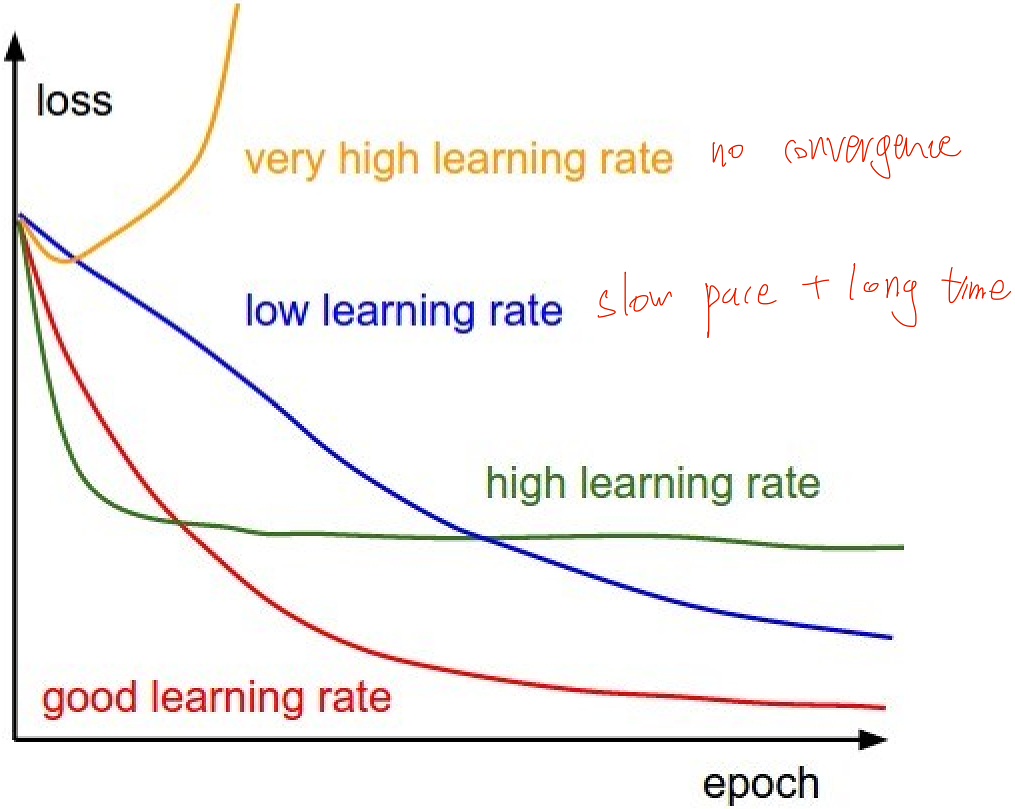
- $\boldsymbol{m}^{k+1}$ and $\boldsymbol{v}^{k+1}$ are initialized with zero
  → bias towards zero
  → Typically, bias-corrected moment updates

$$\boxed{\hat{\boldsymbol{m}}^{k+1} = \frac{\boldsymbol{m}^{k+1}}{1-\beta_1{}^{k+1}} \qquad \hat{\boldsymbol{v}}^{k+1} = \frac{\boldsymbol{v}^{k+1}}{1-\beta_2{}^{k+1}} \longrightarrow \boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \cdot \frac{\hat{\boldsymbol{m}}^{k+1}}{\sqrt{\hat{\boldsymbol{v}}^{k+1}}+\epsilon}}$$

Training Neural Nets

Learning Rate: Implications

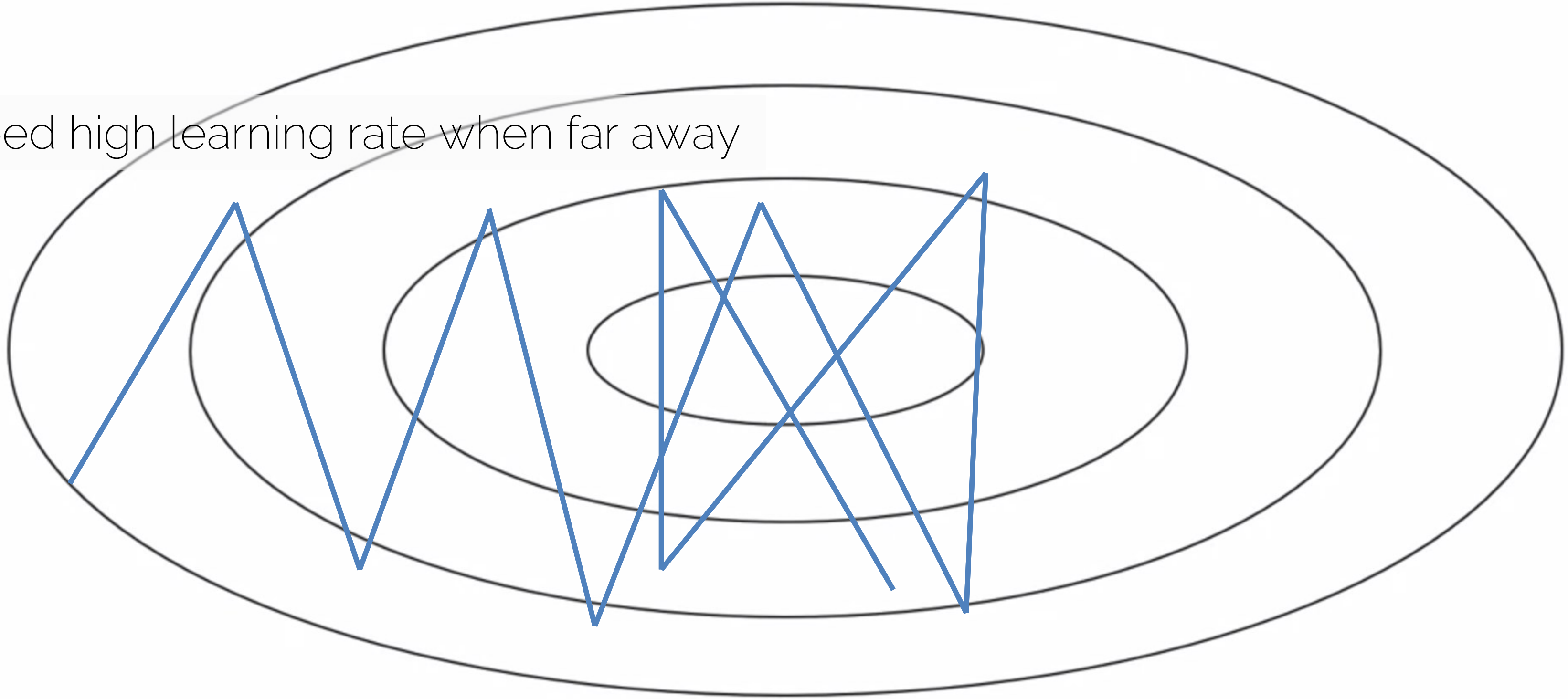
- What if too high?
- What if too low?



Source: http://cs231n.github.io/neural-networks-3/

Learning Rate

Need high learning rate when far away



Need low learning rate when close

Learning Rate Decay

Fixed → Decayed

- $\alpha = \frac{1}{1+decay\_rate*epoch} \cdot \alpha_0$

  – E.g., $\alpha_0 = 0.1$, $decay\_rate = 1.0$

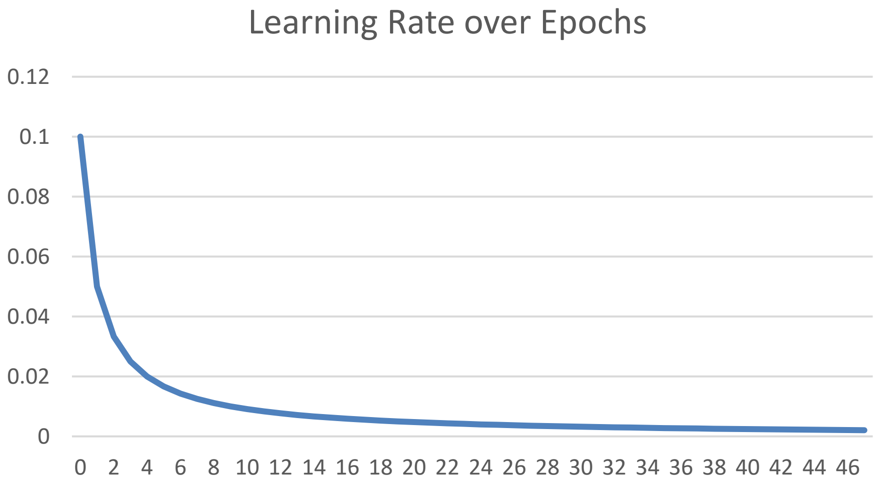
→ Epoch 0: 0.1

→ Epoch 1: 0.05

→ Epoch 2: 0.033

→ Epoch 3: 0.025

...



Learning Rate Decay

Many options:

- Step decay $\alpha = \alpha - t \cdot \alpha$ (only every n steps)
  - T is decay rate (often 0.5)
- Exponential decay $\alpha = t^{epoch} \cdot \alpha_0$
  - t is decay rate (t < 1.0)
- $\alpha = \frac{t}{\sqrt{epoch}} \cdot a_0$
  - t is decay rate
- Etc.

Training Schedule

Manually specify learning rate for entire training process

- Manually set learning rate every n-epochs
- How?
  - Trial and error (the hard way)
  - Some experience (only generalizes to some degree)

Consider: #epochs, training set size, network size, etc.

Basic Recipe for Training

- Given a dataset with labels
  - $\{x_i, y_i\}$
    - $\boldsymbol{x_i}$ is the $\boldsymbol{i^{th}}$ training image, with label $\boldsymbol{y_i}$
    - Often $\mathbf{dim}(\boldsymbol{x}) \gg \mathbf{dim}(\boldsymbol{y})$ (e.g., for classification)
    - $\boldsymbol{i}$ is often in the 100-thousands or millions
  - Take network $\boldsymbol{f}$ and its parameters $\boldsymbol{w}, \boldsymbol{b}$
  - Use SGD (or variation) to find optimal parameters $\boldsymbol{w}, \boldsymbol{b}$
    - Gradients from backpropagation

Gradient Descent on Train Set

- Given large train set with $(\boldsymbol{n})$ training samples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$
  - Let's say 1 million labeled images
  - Let's say our network has 500k parameters

- Gradient has 500k dimensions
- $n = 1\ million$
- Extremely expensive to compute

Learning

- Learning means generalization to unknown dataset
  - (So far no 'real' learning)
  - i.e., train on known dataset → test with optimized parameters on unknown dataset

- Basically, we hope that based on the train set, the optimized parameters will give similar results on different data (i.e., test data)

Learning

- Training set (‘*train*’):
  - Use for training your neural network
- Validation set (‘*val*’):
  - Hyperparameter optimization
  - Check generalization progress
- Test set (‘*test*’):
  - Only for the very end
  - NEVER TOUCH DURING DEVELOPMENT OR TRAINING

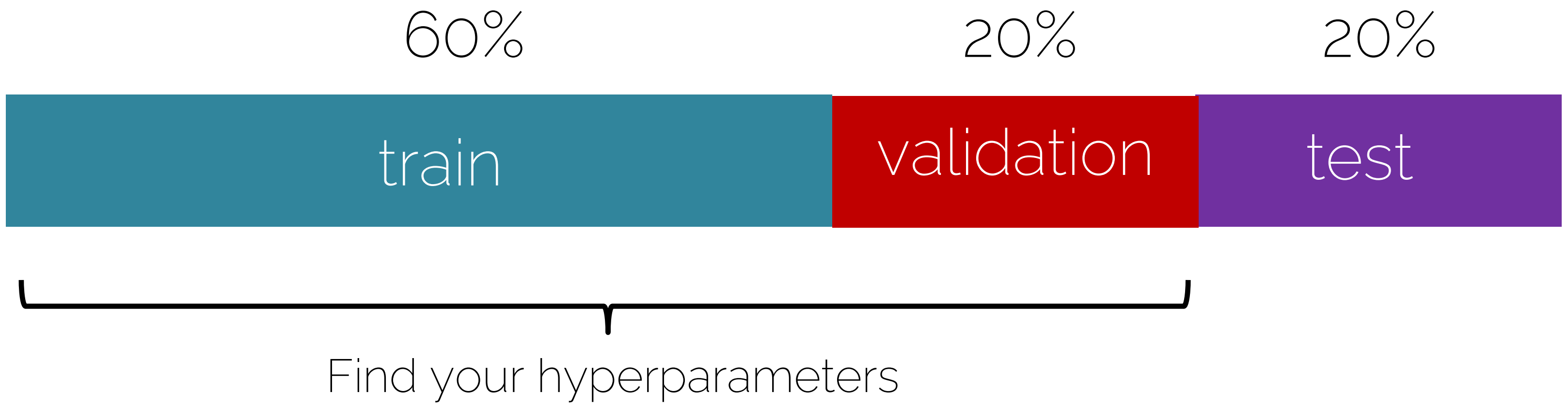
Learning

- Typical splits
  - Train (60%), Val (20%), Test (20%)
  - Train (80%), Val (10%), Test (10%)

- During training:
  - Train error comes from average minibatch error
  - Typically take subset of validation every n iterations

Basic Recipe for Machine Learning

- Split your data



Basic Recipe for Machine Learning

- Split your data

60%	20%	20%
train	validation	test

Example scenario

Ground truth error 1%

Bias

Training set error 5%

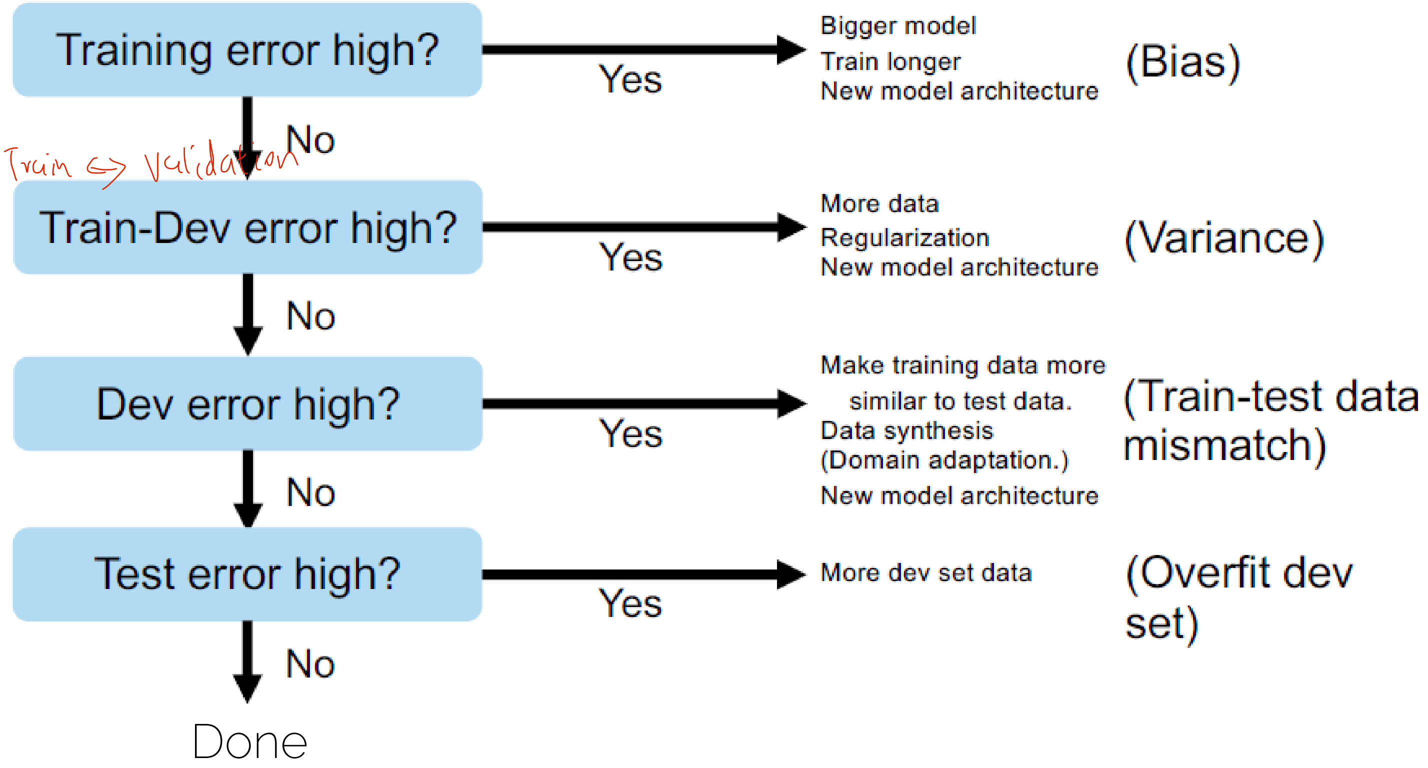
Variance

Val/test set error 8%

Bias (underfitting)

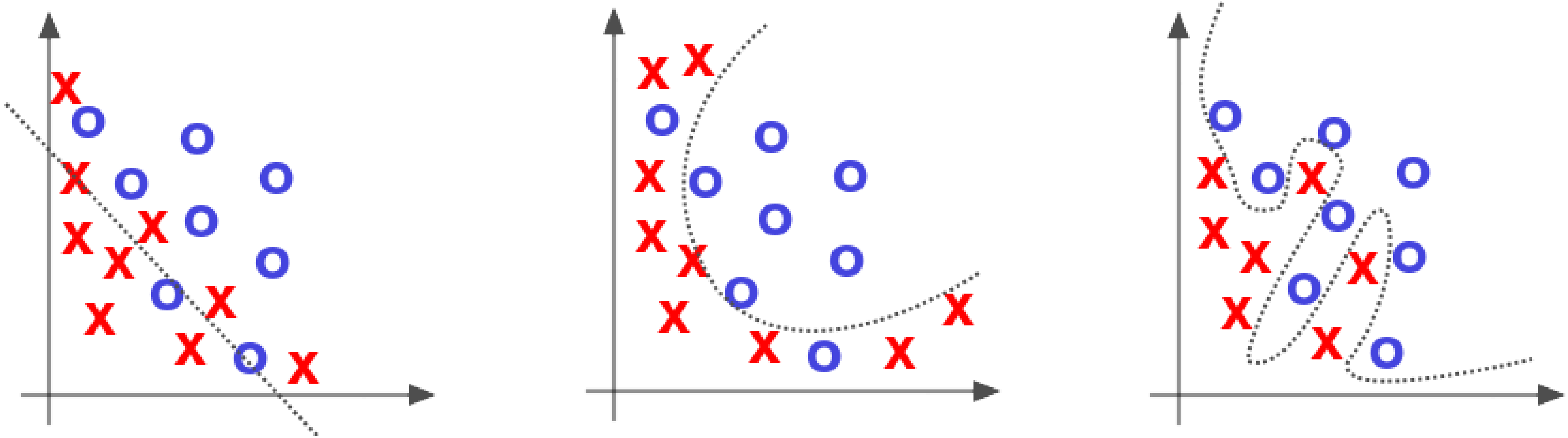
Variance (overfitting)

Basic Recipe for Machine Learning



Credits: A. Ng

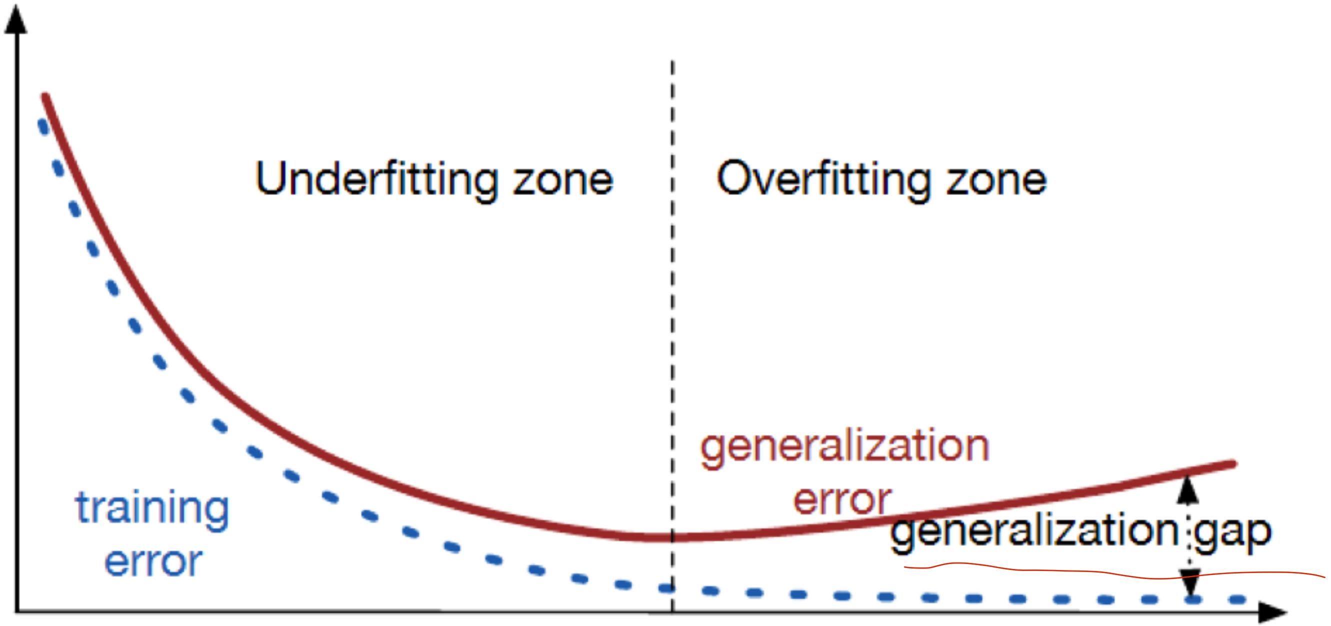
Over- and Underfitting



Underfitted | Appropriate | Overfitted

Source: Deep Learning by Adam Gibson, Josh Patterson, O'Reily Media Inc., 2017

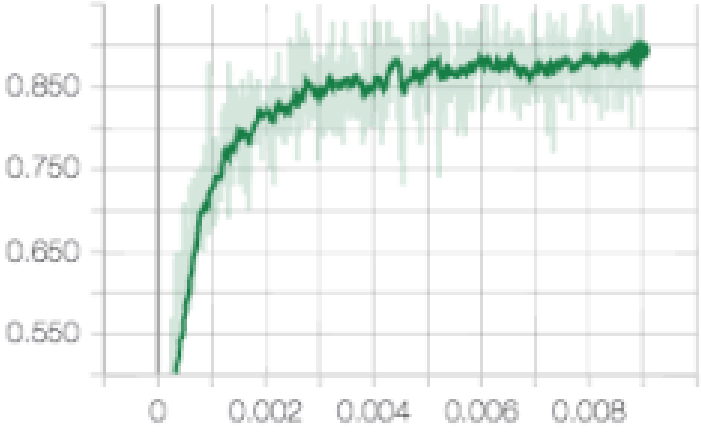
Over- and Underfitting

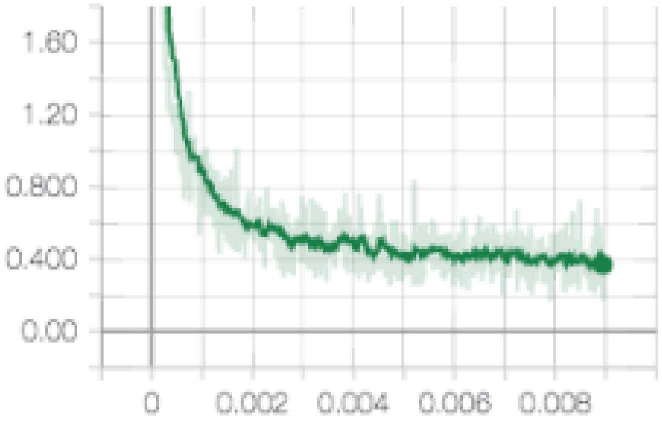


Source: https://srdas.github.io/DLBook/ImprovingModelGeneralization.html

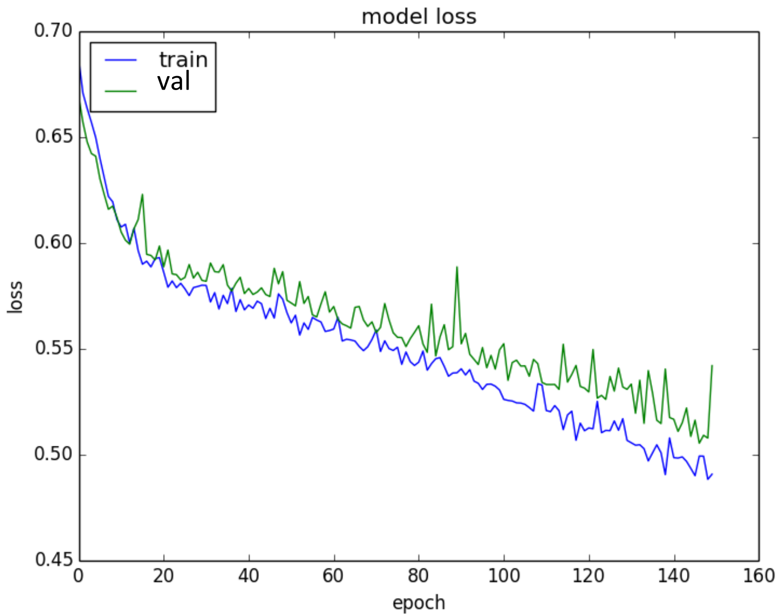
Learning Curves

- Training graphs
  - Accuracy
  - Loss



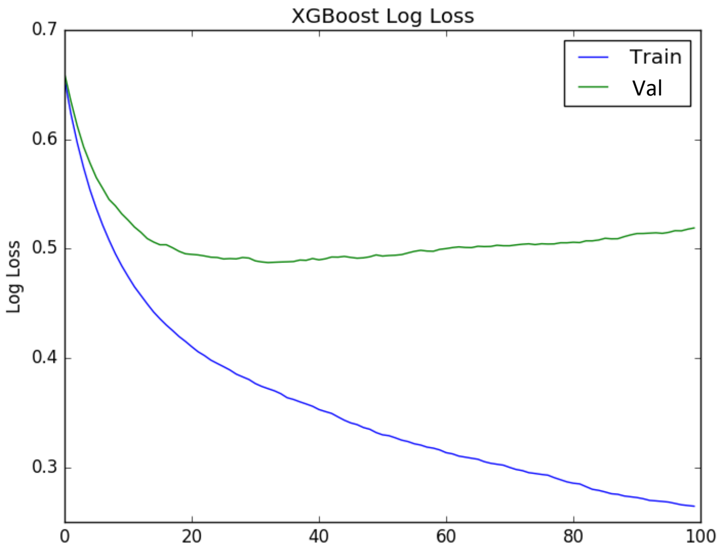


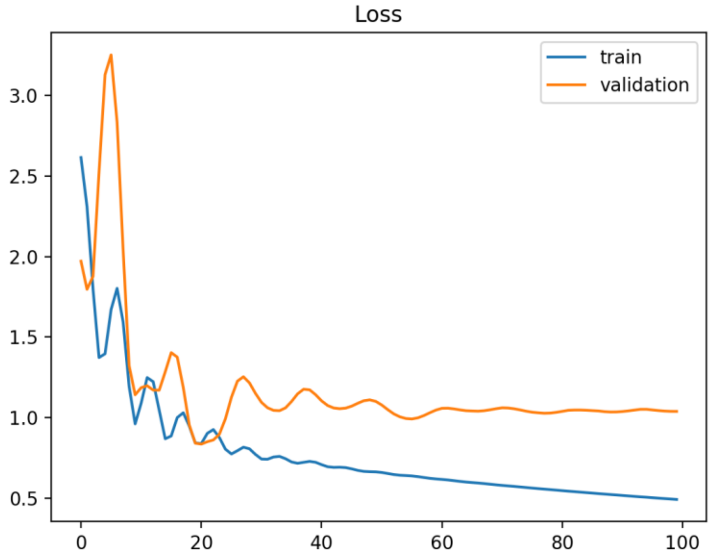
Learning Curves



Source: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

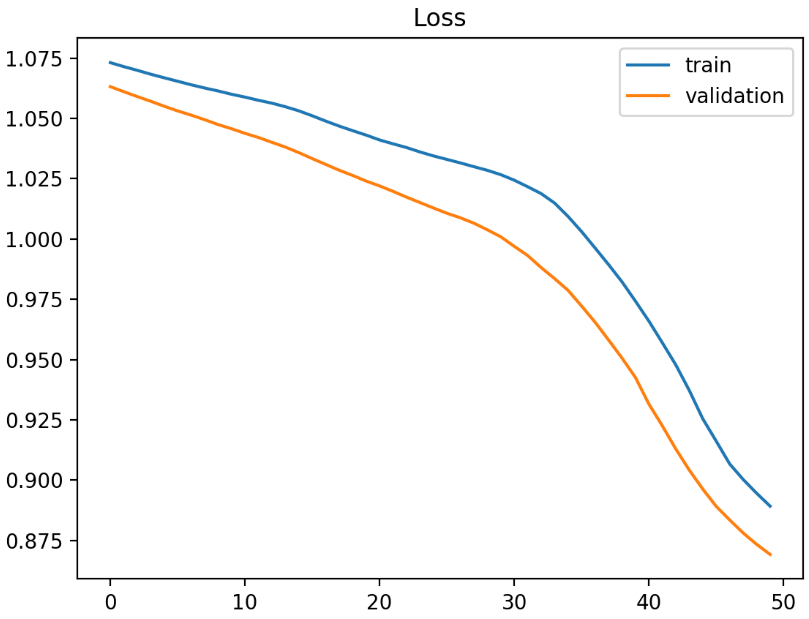
Overfitting Curves



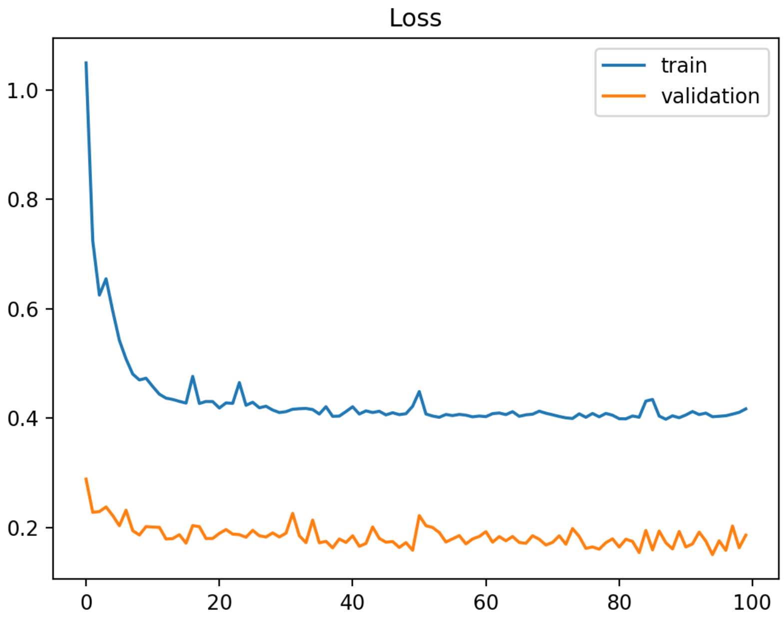


Source: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Other Curves



Underfitting (loss still decreasing)



Validation Set is easier than Training set

Source: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

To Summarize

- Underfitting
  - Training and validation losses decrease even at the end of training
- Overfitting
  - Training loss decreases and validation loss increases
- Ideal Training
  - Small gap between training and validation loss, and both go down at same rate (stable without fluctuations).

To Summarize

- Bad Signs
  - Training error not going down
  - Validation error not going down
  - Performance on validation better than on training set
  - Tests on train set different than during training
- Bad Practice
  - Training set contains test data
  - Debug algorithm on test data

Never touch during development or training

Hyperparameters

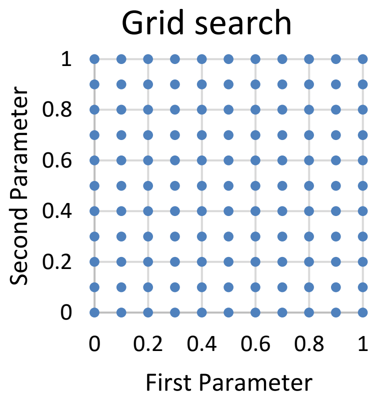
- Network architecture (e.g., num layers, #weights)
- Number of iterations
- Learning rate(s) (i.e., solver parameters, decay, etc.)
- Regularization (more later next lecture)
- Batch size
- …
- Overall:
  learning setup + optimization = hyperparameters

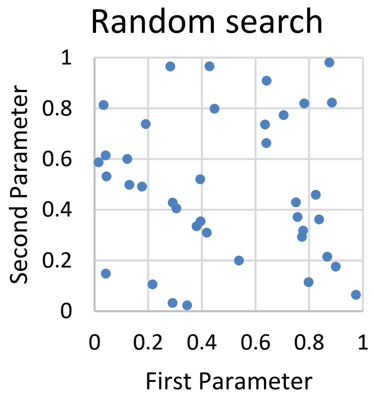
Hyperparameter Tuning

- Methods:
  - Manual search:
    - most common ☺
  - Grid search (structured, for 'real' applications)
    - Define ranges for all parameters spaces and select points
    - Usually pseudo-uniformly distributed

    → Iterate over all possible configurations
  - Random search:

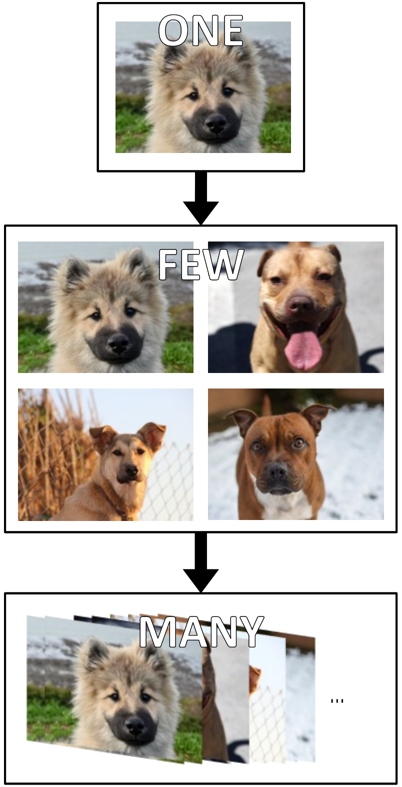
    Like grid search but one picks points at random in the predefined ranges





How to Start

- Start with single training sample
  - Check if output correct
  - Overfit → train accuracy should be 100% because input just memorized
- Increase to handful of samples (e.g., 4)
  - Check if input is handled correctly
- Move from overfitting to more samples
  - 5, 10, 100, 1000, …
  - At some point, you should see generalization



Find a Good Learning Rate

Karpathy's constant

Andrej Karpathy
@karpathy

3e-4 is the best learning rate for Adam, hands down.

4:01 AM · Nov 24, 2016 · Twitter Web Client

123 Retweets **30** Quote Tweets **562** Likes

Karpathy's constant

Andrej Karpathy
@karpathy

3e-4 is the best learning rate for Adam, hands down.

4:01 AM · Nov 24, 2016 · Twitter Web Client

123 Retweets **30** Quote Tweets **562** Likes

Andrej Karpathy @karpathy · Nov 24, 2016
Replying to @karpathy
(i just wanted to make sure that people understand that this is a joke...)

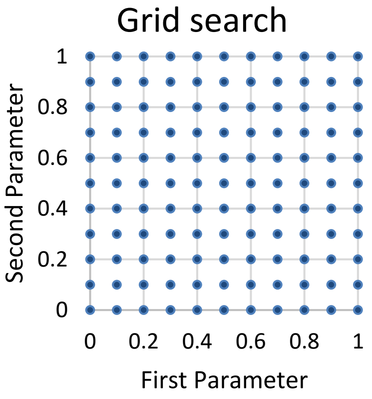
9 6 144

Find a Good Learning Rate

- Use all training data with small weight decay
- Perform initial loss sanity check e.g., $\log(C)$ for softmax with C classes
- Find a learning rate that makes the loss drop significantly (exponentially) within 100 iterations
- Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-4

Coarse Grid Search

- Choose a few values of learning rate and weight decay around what worked from
- Train a few models for a few epochs.
- Good weight decay to try: 1e-4, 1e-5, 0

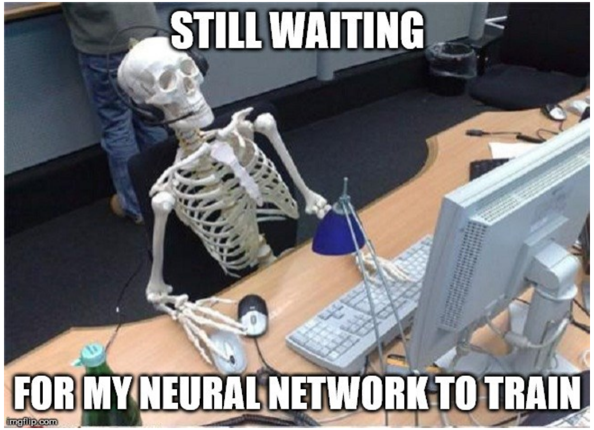


Refine Grid

- Pick best models found with coarse grid.
- Refine grid search around these models.
- Train them for longer (10-20 epochs) without learning rate decay
- Study loss curves <- most important debugging tool!

Timings

- How long does each iteration take?
  - Get **precise** timings!
  - If an iteration exceeds **500**ms, things get dicey
- Look for bottlenecks
  - Dataloading: smaller resolution, compression, train from SSD
  - Backprop
- Estimate total time
  - How long until you see some pattern?
  - How long till convergence?



Network Architecture

- Frequent mistake: "*Let's use this super big network, train for two weeks and we see where we stand.*"

- Instead: start with simplest network possible
  - Rule of thumb divide #layers you started with by 5
- Get debug cycles down
  - Ideally, minutes

Debugging

- Use train/validation/test curves
  - Evaluation needs to be consistent
  - Numbers need to be comparable

- Only make **one change at a time**
  - "I've added 5 more layers and double the training size, and now I also trained 5 days longer. Now it's better, but why?"

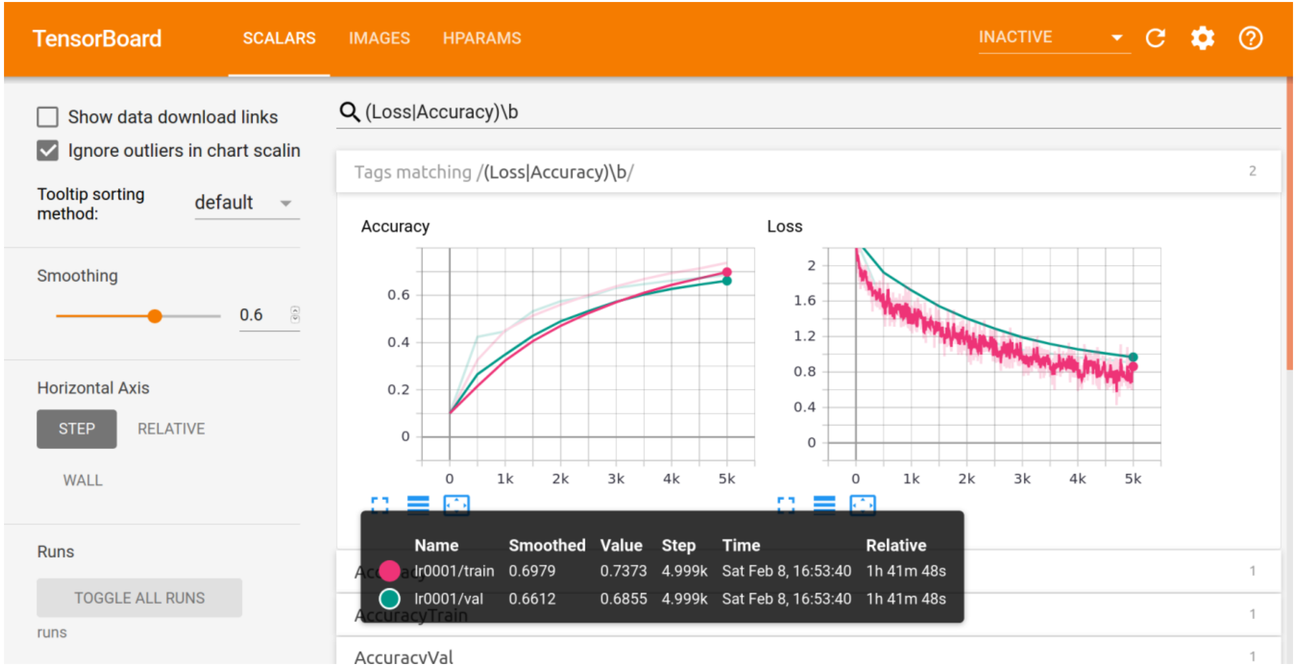
- Visualize input, prediction, ground truth

Common Mistakes in Practice

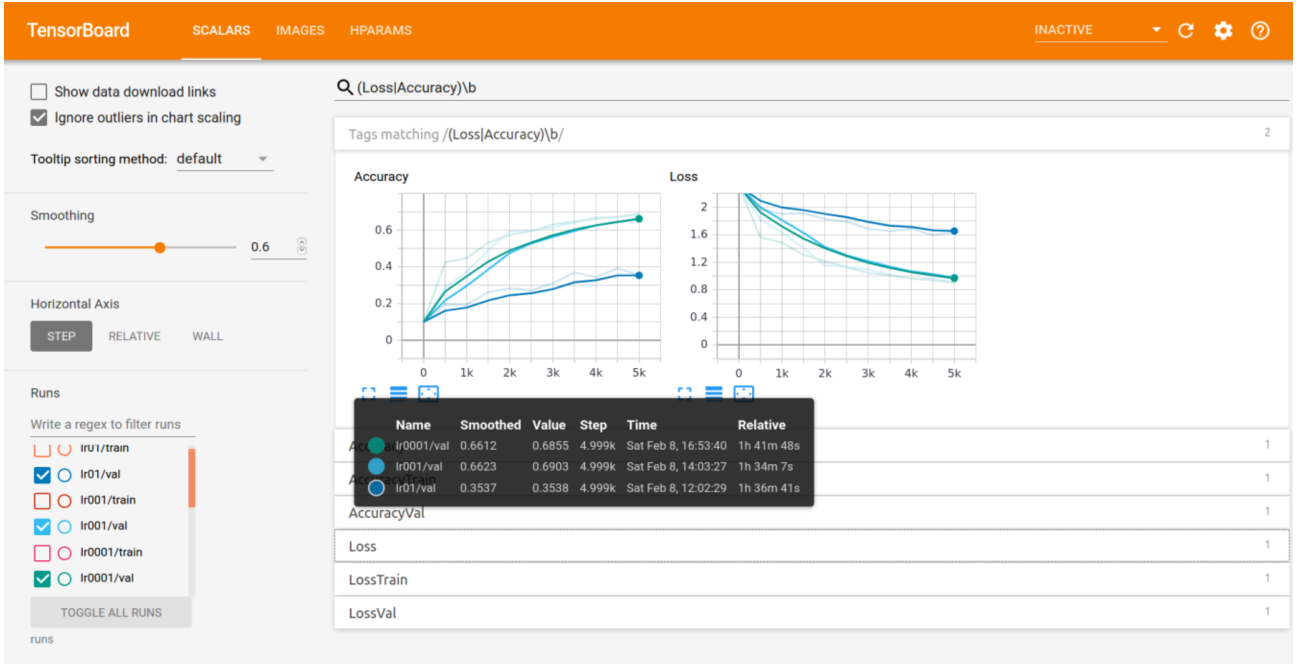
- Did not overfit to single batch first
- Forgot to toggle train/eval mode for network
  - Check later when we talk about dropout…
- Forgot to call **.zero_grad()** (*in PyTorch*) before calling **.backward()**
- Passed softmaxed outputs to a loss function that expects raw logits

Tensorboard: Visualization in Practice

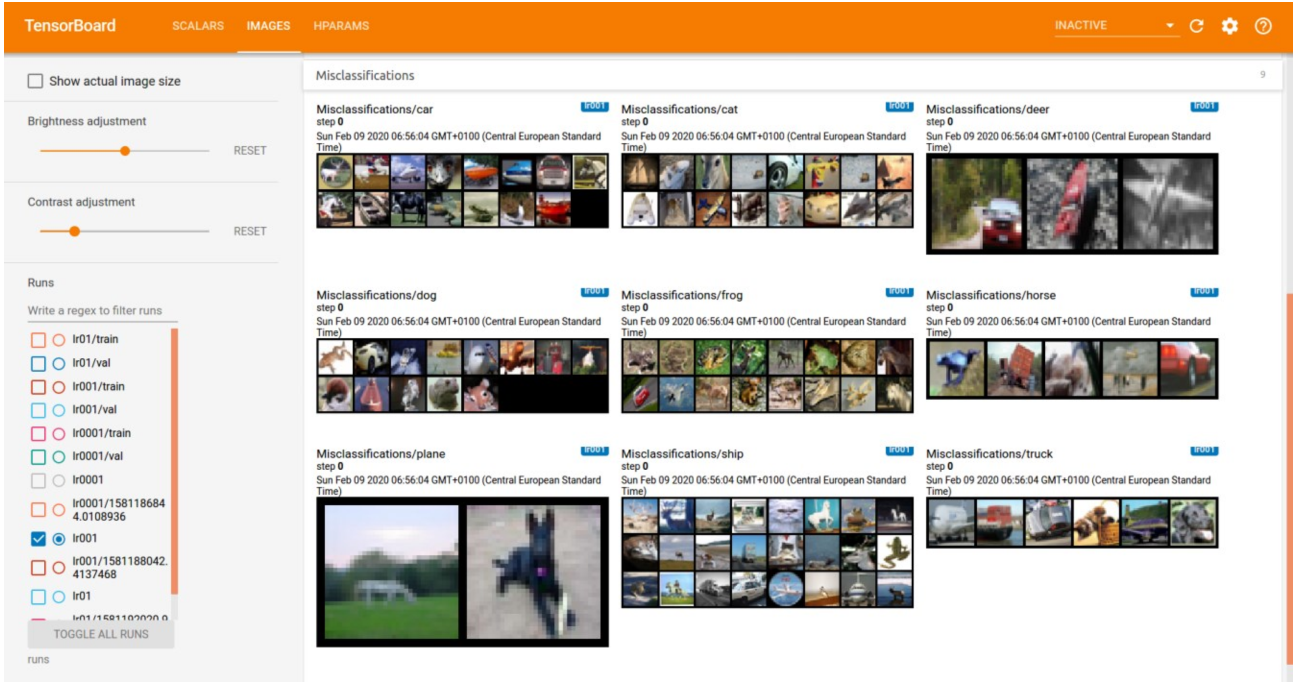
Tensorboard: Compare Train/Val Curves



Tensorboard: Compare Different Runs

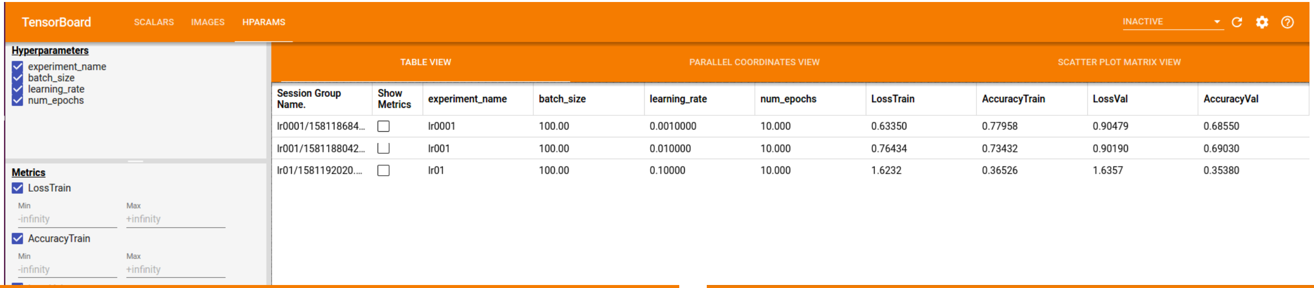


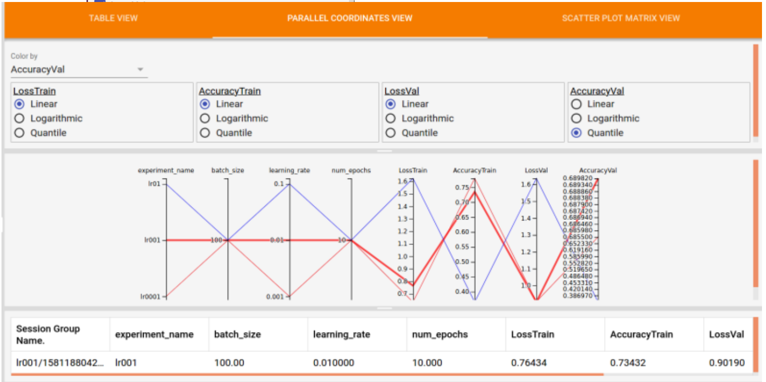
Tensorboard: Visualize Model Predictions

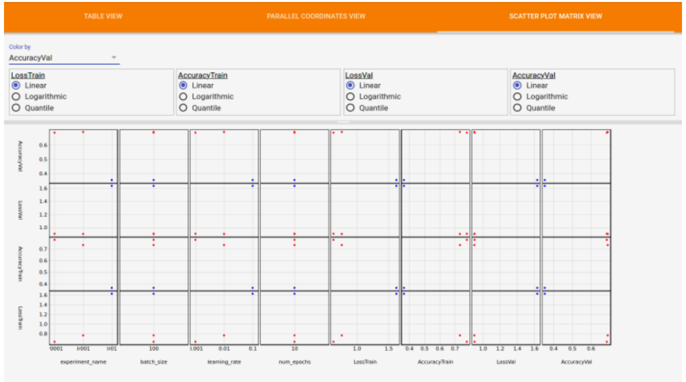


Tensorboard: Visualize Model Predictions

Tensorboard: Compare Hyperparameters







Next Lecture

- Next lecture
  - More about training neural networks: output functions, loss functions, activation functions

- Check the exercises ☺

See you next week ☺

References

- Goodfellow et al. "Deep Learning" (2016),
  - 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/
- http://cs231n.github.io/neural-networks-3/