Economics 2355: Classification and Training Neural Nets

Melissa Dell

Harvard University

February 2021

Classification

- Last time, we learned about CNN architectures
- CNNs underlie image classification
- This lecture, we will learn more about how to implement classification in practice, with a heavy emphasis on how to train neural nets
- The discussion of training applies to many different types of neural nets, not just those with a classification head at the final layer

Classification in Neural Nets

- Recall that we can think of neural nets as legos, stacking together different types of legos
- Classification is very straightforward: just stack a classifier layer on top of the CNN features extractor
- We already saw this when we introduced CNNs last lecture, i.e. a final layer after the features network output the ImageNet class probabilities

SVM Loss

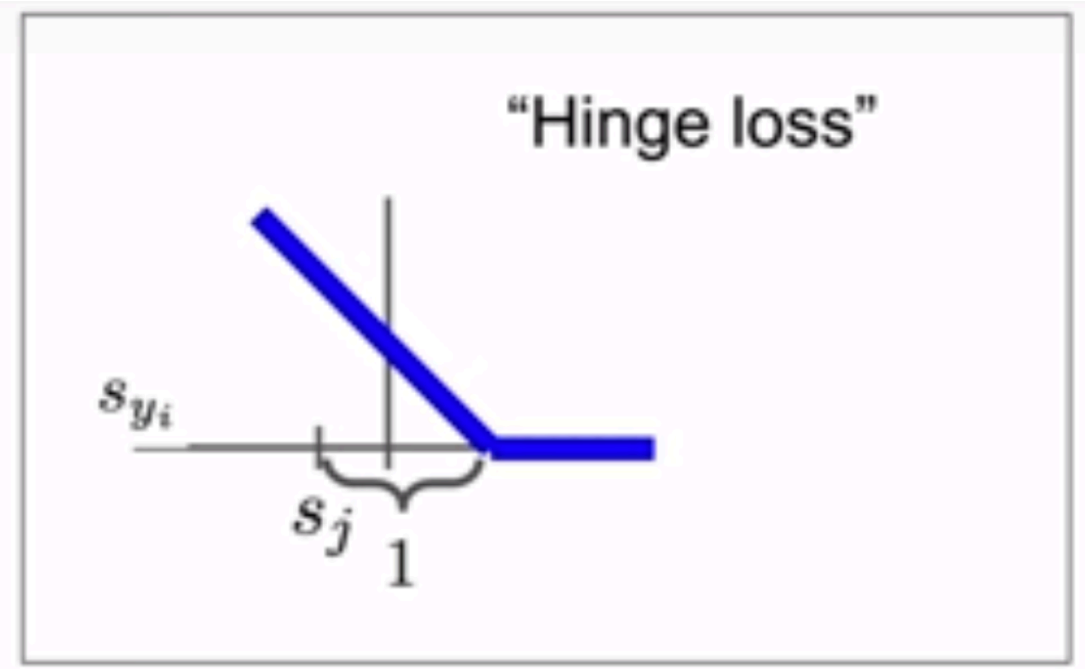
The SVM loss for the i'th example:

$$L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1) \tag{1}$$

This equation sums over the incorrect classes ($j \neq y_i$)
Suppose we have three classes that receive the scores $s = [3, -7, 2.5]$ and the first class is correct. The SVM loss is $L_i = max(0, -7 - 3 + 1) + max(0, 2.5 - 3 + 1) = 0.5$. For the second class, the score is much lower than the true class. The loss doesn't care how much, as long as the difference is above the threshold. For the third class, even though the score is less than the true class, there is a loss associated with it because it doesn't exceed the threshold.

It is possible to show that we can set the threshold to one WOLG (weights could scale to make this difference smaller or larger)

SVM (Hinge Loss)



Softmax Loss

Softmax

$$L_i = -log\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \tag{2}$$

Softmax takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one. Can interpret as the normalized "probability" assigned to the correct label y_i given the image x_i (how peaky probabilities are will depend on regularization: ordering interpretable but magnitudes technically not)

The loss then minimizes cross-entropy between estimated class probabilities and true distribution, where all the probability mass is on the true class. Alternatively interpreted as minimizing the negative log likelihood of the correct class. If probability on true class is 0.7, loss is $-ln0.7 = 0.357$. If probability on true class was 0.3, the loss would be 1.2. With a probability of 1, the loss would be zero.

Loss for Classification

- Both SVM and Softmax are frequently used for classification tasks
- Once you get the right answer (above a threshold), SVM doesn't care if you push the score for the correct class up higher
- In contrast, softmax cares about pushing probability of the correct class to 1: this means early in training your accuracy can jump suddenly without much change in your loss
- SVM is a more local objective, which could be thought of either as a bug or a feature (i.e. spend most effort distinguishing cars from trucks, versus cars from cats)
- In practice, they tend to yield very similar results

Training Neural Nets

- Today we will cover, from a theoretical perspective, how to train neural networks
- After we introduce object detection, we will return to a discussion of training in practice, geared towards object detection

Outline

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

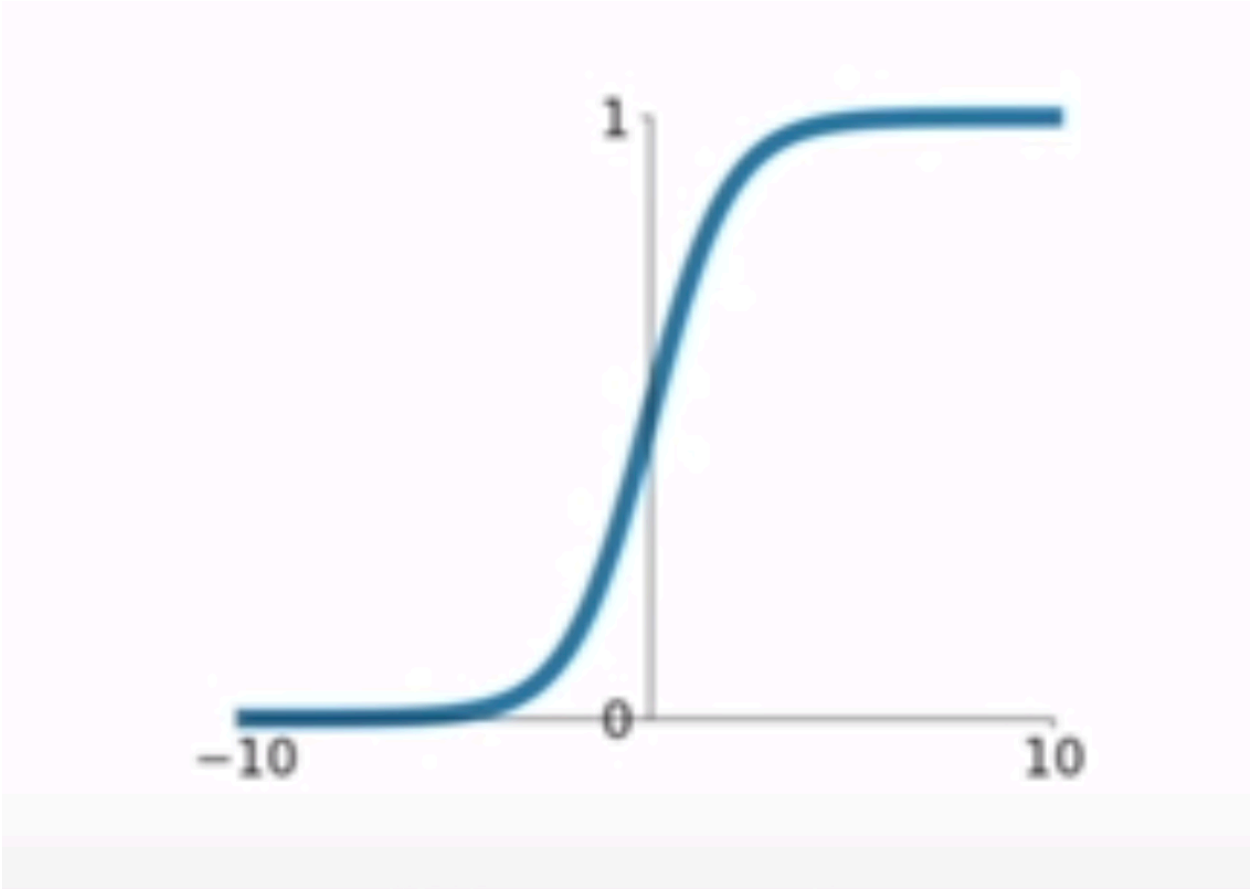
Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

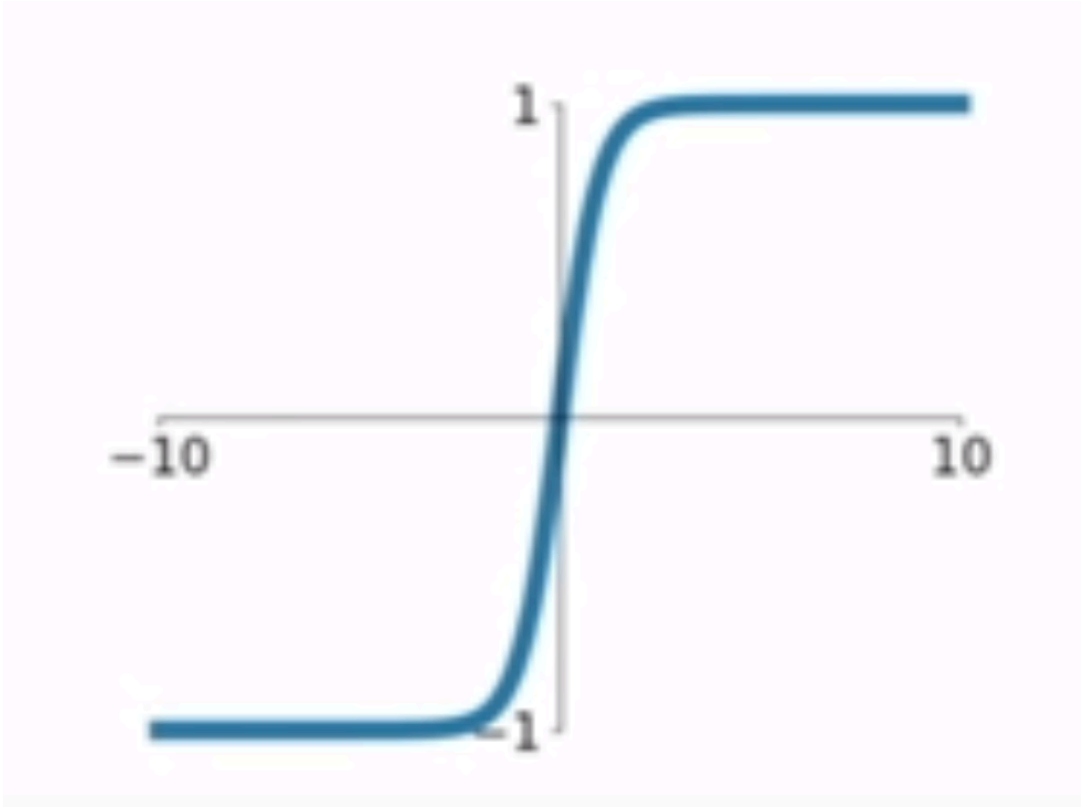
Sigmoid Activation



Sigmoid Problems

- Saturates across most of its domain. This flat derivative makes it difficult to improve the weights through gradient descent
- The vanishing gradient problem gets worse as the number of layers increases
- Very sensitive to small changes over part of its range, which also complicates training
- Not zero-centered
- Costly to compute

Inverse Hyperbolic Tangent (tanh) Activation



tanh

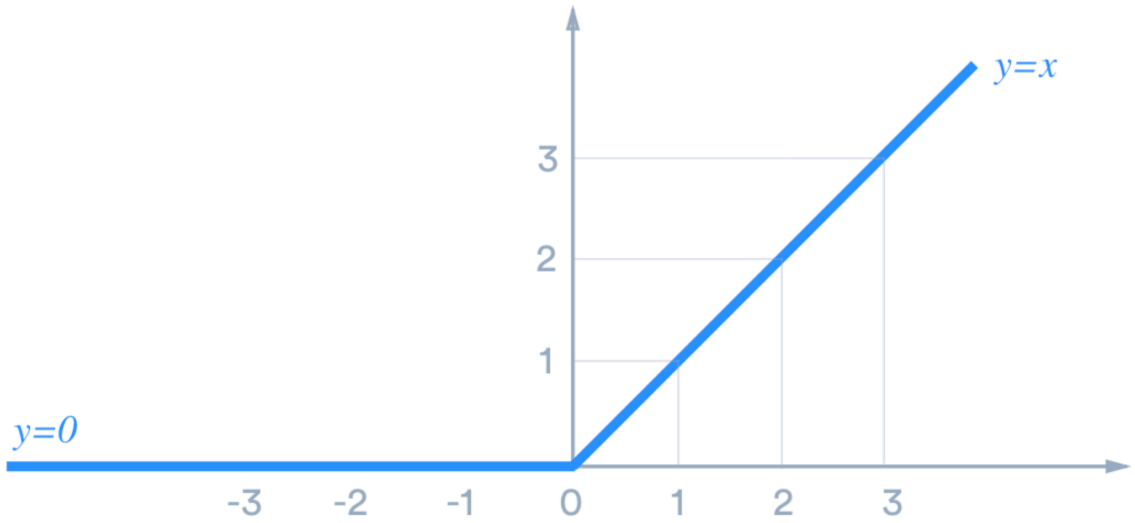
Advantages:

- It is non-linear everywhere. Accounting for non-linearities is one of the activation function's main purposes.
- It is zero-centered.

Disadvantages:

- It is relatively flat except for a very narrow range. The derivative of the function is very small unless the input is in this narrow range.
- Not an accident that a key part of AlexNet was to use ReLU instead of tanh.

Rectified Linear Unit Activation



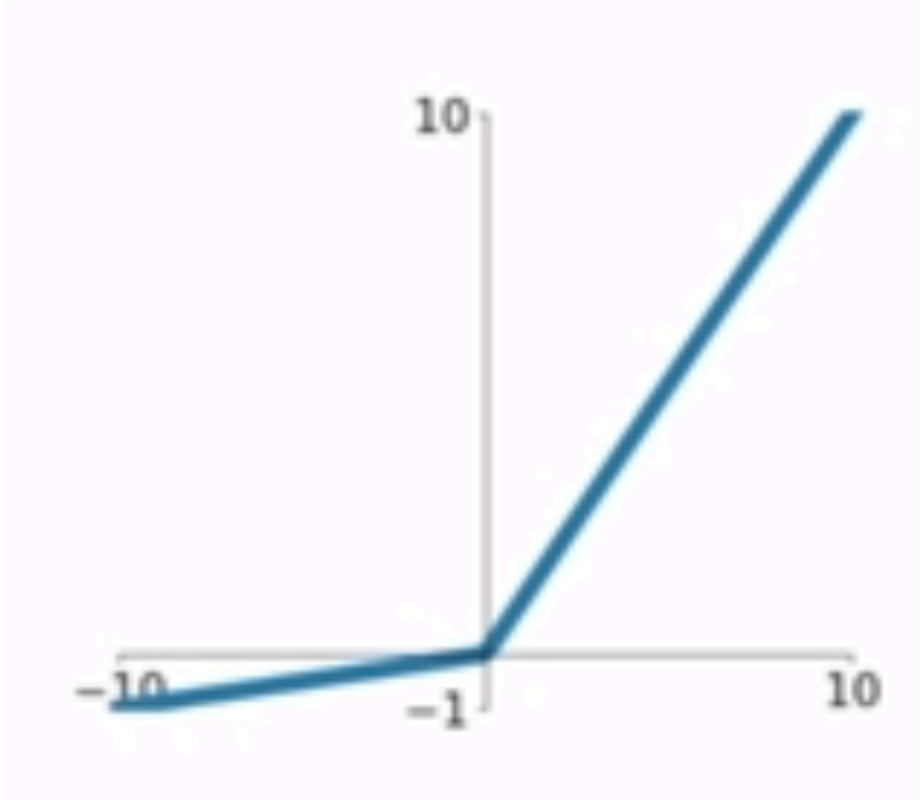
Why does ReLU work?

- Linearity means that the slope doesn't saturate when x gets large. It doesn't have the severe vanishing gradient problem suffered by other activation functions like sigmoid or tanh
- When training on a reasonable sized batch, there will usually be some data points giving positive values to any given node. So the average derivative is rarely close to 0, which allows gradient descent to keep progressing
- Can output true zeros, which is better in estimating a sparse network. May be less subject to overfitting
- Max very fast to compute (6x faster than tanh)

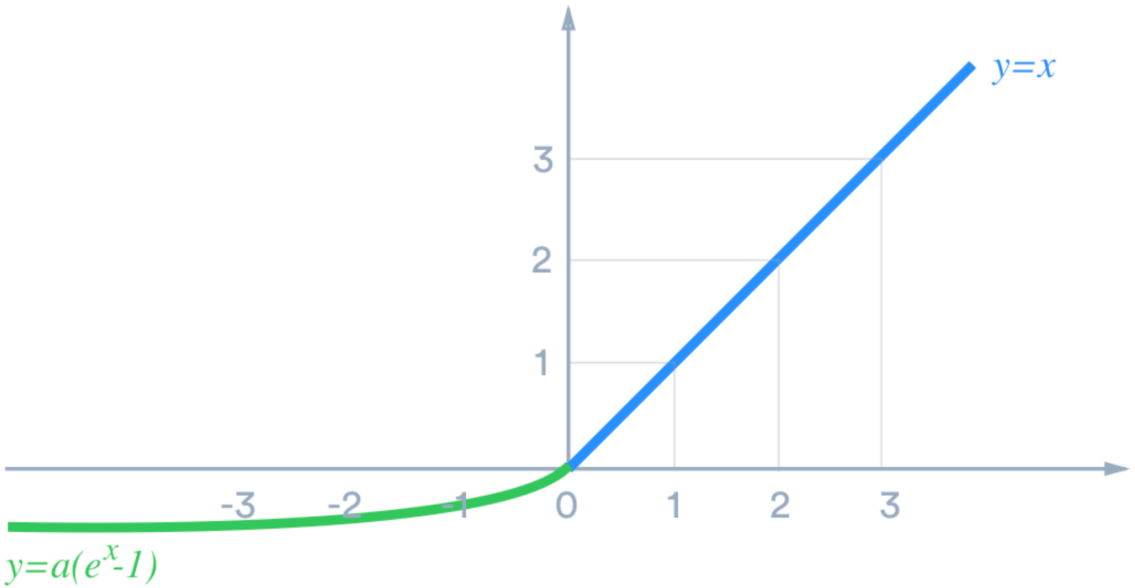
Problem: Dead ReLUs

- A ReLU neuron is dead if it gets stuck on the negative side and always yields zero gradient, no matter the input (as long as not negative for all images in the mini-batch, will yield a positive average gradient)
- This happens when the learning rate is too large or with very negative biases (initialize with positive biases)

Leaky Relu



Exponential Linear Unit



Advantages: solves the dead ReLU problem without allowing very negative activations Disadvantages: exponential is more costly than max function to compute

Advice

- Probably want to use ReLU, being careful when setting your learning rate and initializing in a reasonable way
- Could experiment with i.e. leaky ReLU or ELU
- Don't use sigmoid

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

Data Pre-Processing

- We need input data to be zero-centered and on the same scale
- With image data, we don't typically worry about putting data on the same scale, as an RGB image is by construction on a scale from 0 to 255
- We do want to zero center the data - two options:
  - Subtract the H X W X 3 mean image (i.e. for ImageNet this is essentially an orange blob); AlexNet does this
  - Subtract a per channel mean - i.e. 3 numbers; VGG does this

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

Initialization

- Initialization is actually really tricky - a big part of the reason why deep learning in the early decades had disappointing results
- If weights are too small, neurons will have close to zero value. This implies a zero gradient, as the gradient is the activations times the upstream gradient
- If weights are too large, saturate the network (i.e. think about tanh) and the gradients flowing through the network are all zero. Can train for a long time and loss won't move

Xavier Initialization

- Xavier initialization: divide by sqrt of number of inputs to the neurons: if lots of weights feed into a neuron, the initial weights are smaller
- Doesn't work as well for ReLU, which kills half the distribution by setting it to zero. He et al. initialization (2015) adjusts for this by drawing weights from a normal distribution centered at zero with variance of $\frac{2}{n}$
- In practice, you might also initialize with learned weights from a similar application

Outline

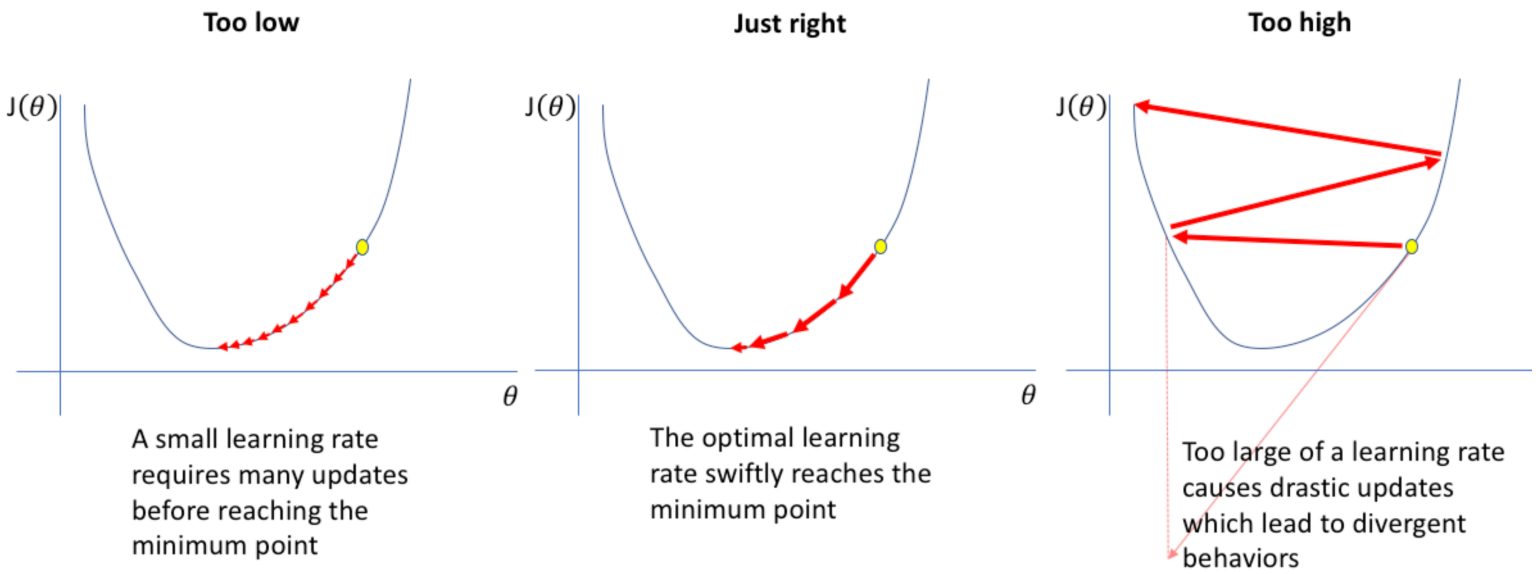
Optimization Challenges

- Neural nets, especially deep ones, are really challenging to optimize: complicated space, potentially in 100 million + dimensions
- We saw a couple of lectures ago how stochastic gradient descent (SGD) is used for optimization
- In reality, you wouldn't use plain vanilla SGD

One Major Challenge: Pathological Curvature

- Pathological curvature refers to regions of the loss function that aren't scaled properly - they are often described as valleys, trenches, or ravines
- These are common near local minima
- To understand why this is a problem, it is useful to note first what happens with a one dimensional function when the learning rate is too small or large

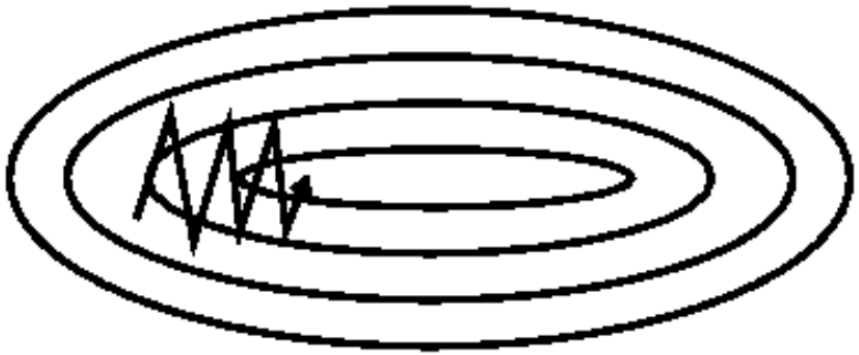
A Large Learning Rate



https://www.jeremyjordan.me/nn-learning-rate/

At best, you'll oscillate back and forth, converging slowly, and might diverge

What happens in ravines?



Osculate back and forth in the steep direction across the slopes of the ravine and converge very slowly in the shallow dimension, making very slow progress along the bottom towards the optimum

Another Optimization Challenge

Saddle points are another major, related optimization challenge

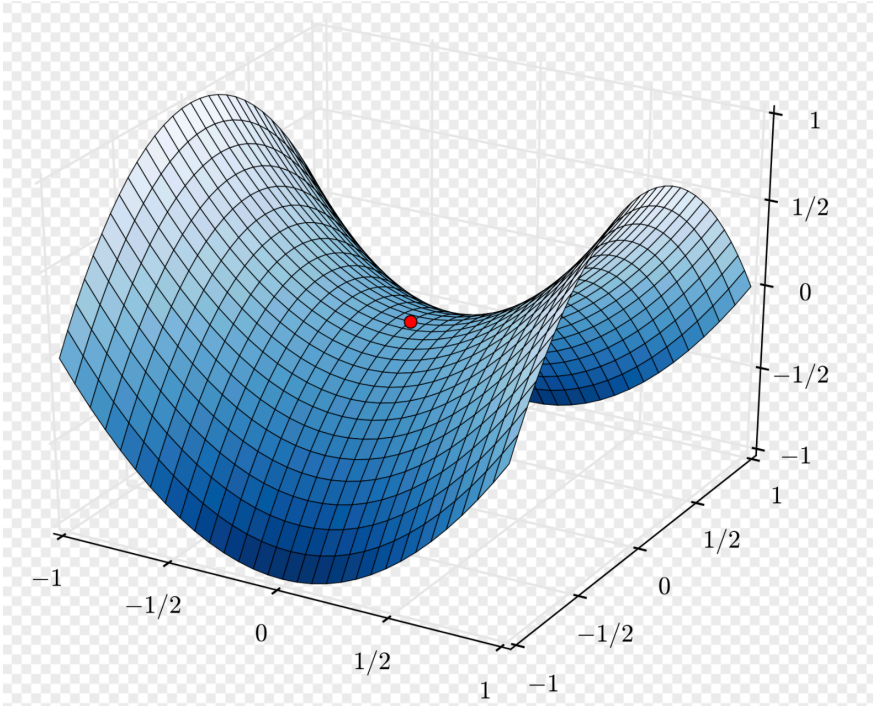


Image credit: Wikipedia

Much, much more common than true local min in neural nets, which have millions of dimensions (Dauphin et al.)

Problems with Stochastic Gradient Descent

- Recall also that we compute the gradient on the minibatch
- This will introduce additional noise
- The solvers we will discuss are motivated by these challenges

SGD+Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t) \tag{3}$$

SGD+Momentum

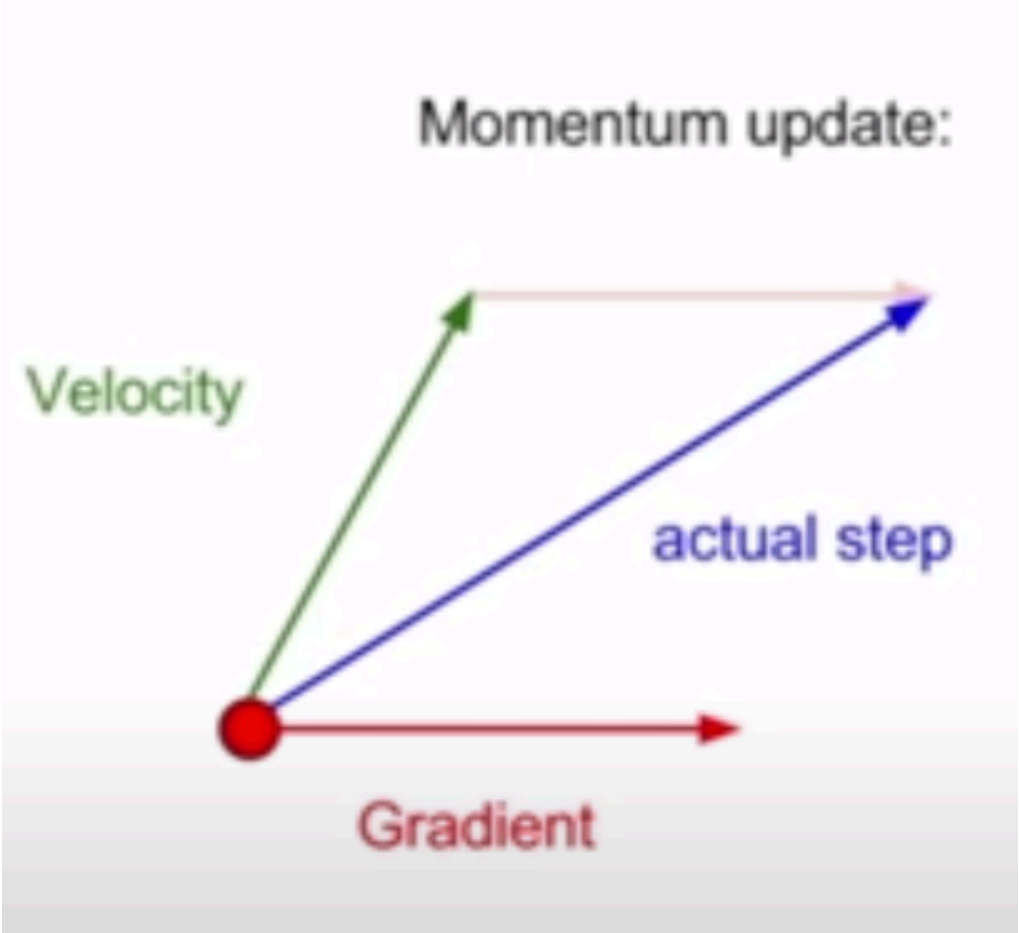
$$m_{t+1} = \rho m_t + (1 - \rho) \nabla f(x_t) \tag{4}$$

$$x_{t+1} = x_t - \alpha m_{t+1} \tag{5}$$

Velocity builds up as a running mean of gradients; ρ is known as the friction parameter, typically 0.9 or 0.99.

Why not just remember, i.e. the last ten gradients?

SGD+Momentum



SGD+Momentum in Valleys

Image 2: SGD without momentum

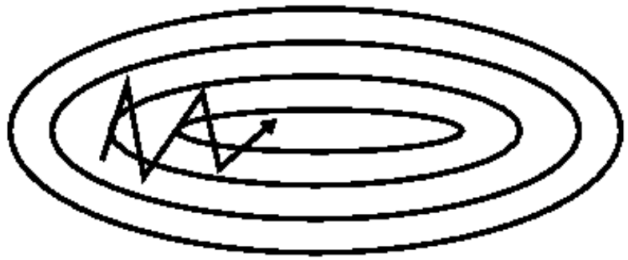


Image 3: SGD with momentum

Momentum increases for dimensions whose gradients point in the same direction and reduces updates for dimensions whose gradients change directions. Speeds up convergence and reduces oscillation. Will help escape saddle points as long as momentum not only in the direction of minima.

SGD+Momentum

- You might be concerned that momentum would push you out of a sharp minimum
- This is actually a feature, not a bug
- Flat minima more robust to generalization

Nesterov

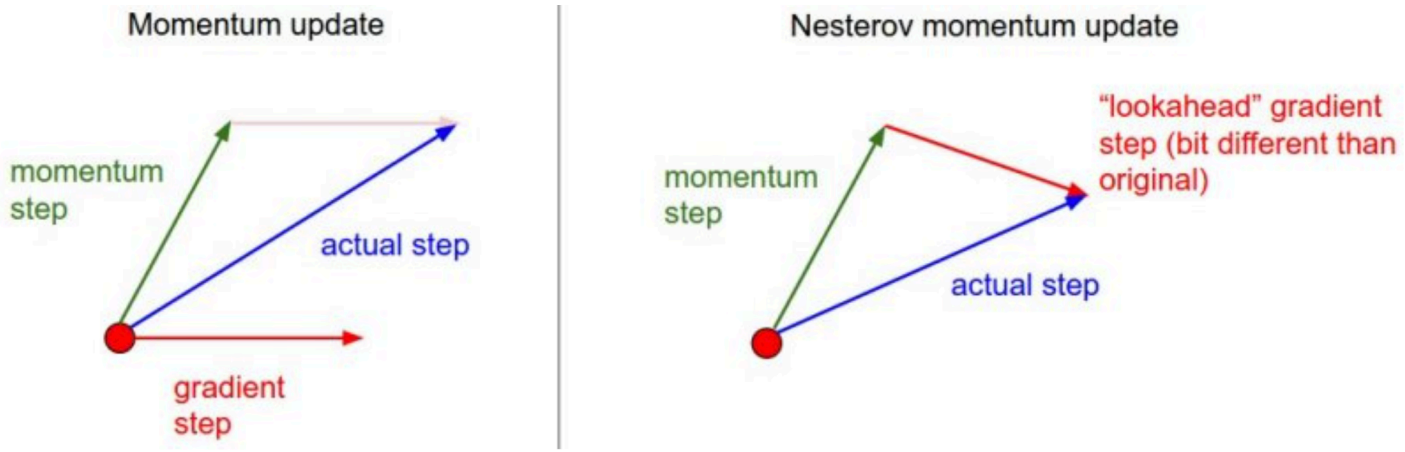


Image credit: Stanford CS 231n

We know momentum will take us to the top of the green arrow, so calculate the gradient there.

RMSProp

- Momentum solves the challenge of ravines by changing the gradient, making it smaller in dimensions with oscillation and larger in dimensions with consistent movement in the same direction
- Something similar could be achieved by allowing the learning rate to scale differently in these dimensions

RMSProp

$$v_{t+1} = \beta v_t + (1 - \beta)(\nabla f(x_t))^2 \tag{6}$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \nabla f(x_t) \tag{7}$$

Scales the learning rate by the exponentially weighted sum of the squared gradients.

This yields a lower learning rate in dimensions with steeper gradients and a larger learning rate in dimensions with smaller gradients. β commonly 0.9 or 0.99.

ADAM (Adaptive Moment Estimation)

Combines Momentum and RMSProp

$$m_{t+1} = \rho v_t + (1 - \rho)\nabla f(x_t) \tag{8}$$

$$v_{t+1} = \beta v_t + (1 - \beta)(\nabla f(x_t))^2 \tag{9}$$

$$x_{t+1} = x_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} m_{t+1} \tag{10}$$

(In full implementation, also bias correction terms that affect the first few iterations)

Tends to work very well

Second Order Methods

- In theory an approach like Newton's Method seems appealing
- Second order approximation of the gradient and update to the minimum of that function; would avoid needing to choose a learning rate
- Why doesn't this work in practice (hint: think about inverting a 100 million x 100 million matrix!)

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

Regularization

The loss combines the data loss - model predictions should match training data - and regularization - model should be simple, to improve performance on unseen data

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \qquad (11)$$

Applies the intuition of Occam's Razor (simplest model is best)

L2 Normalization

- L2 regularization computes the Euclidean norm (L2 norm) of the weights (i.e. the sum over all squared weight values)
- This regularization term is weighted by the scalar μ divided by two and added to the regular loss function
- μ is a hyperparameter known as the regularization rate

L2 Normalization

- L2 Normalization favors putting small values on many weights, as opposed to large values on a limited number of weights (use all inputs a little, rather than some inputs a lot)
- Pushes weight values towards zero
- This tends to lead to a simpler model that avoids modeling noise

L2 Normalization and Weight Decay

- The terms L2 Normalization and weight decay are often used interchangeably, both in papers and in textbook treatments
- They are equivalent for stochastic gradient descent (when scaled by the learning rate), but this is no longer true for adaptive gradient algorithms such as Adam - a point that appears to have caused considerable confusion
- Ilya Loshchilov and Frank Hutter - "Decoupled Weight Decay Regularization" (2019) - show how L2 regularization can be adjusted for Adam

Regularization More Generally

- You can think more generally of regularization as adding some random noise during training
- Other methods that have a regularizing effect:
  - Batch Normalization
  - Dropout
  - Data augmentation

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

Batch Normalization

- We talked earlier about how we normalize our input data
- But what about activations for subsequent neurons?
- This insight led to a simple yet powerful idea: if you want activations to be unit Gaussian, just make them so (Ioffe and Szegedy, 2015; now one of the most cited DL papers)

Batch Normalization

Compute the empirical mean and variance independently for each dimension and then normalize

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{12}$$

And then allow the network to squash the range:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{13}$$

$\beta^{(k)}$ and $\gamma^{(k)}$ are learned parameters

BatchNorm in ConvNets

- In ConvNets, filter weights are shared across the input image. Hence, it is reasonable to normalize the output in the same way
- The mean and variance are computed from all activations in a given feature map for all images in the batch (i.e. $B * H * W$)
- In other words, we compute one mean and variance per channel

Batch Normalization and Covariate Shift

- The initial paper argues that batch normalization helps because of covariate shift
- The gradient tells how to update each parameter, under the assumption that the other layers do not change, but in practice we update all of the layers of deep nets simultaneously
- The distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This necessitates lower learning rates and careful initialization
- Batchnorm significantly reduces the problem of coordinating updates across many layers, stabilizing and speeding up training

Effects of Batch Norm

- However, still debated exactly why batch norm tends to work so well
- Santurkar et al. (2019) argue that it is effective because it smooths the optimization function
- It also has a regularizing effect, since the mean and variance depend stochastically on the mini-batch

Batch Normalization as Regularization

- L2 regularization no longer has a significant regularization on any layer that is followed by BN
- If you decay your weights and then simply rescale everything, you are undoing the regularization (it will affect the learning rate though)
- But BatchNorm itself gives you regularization

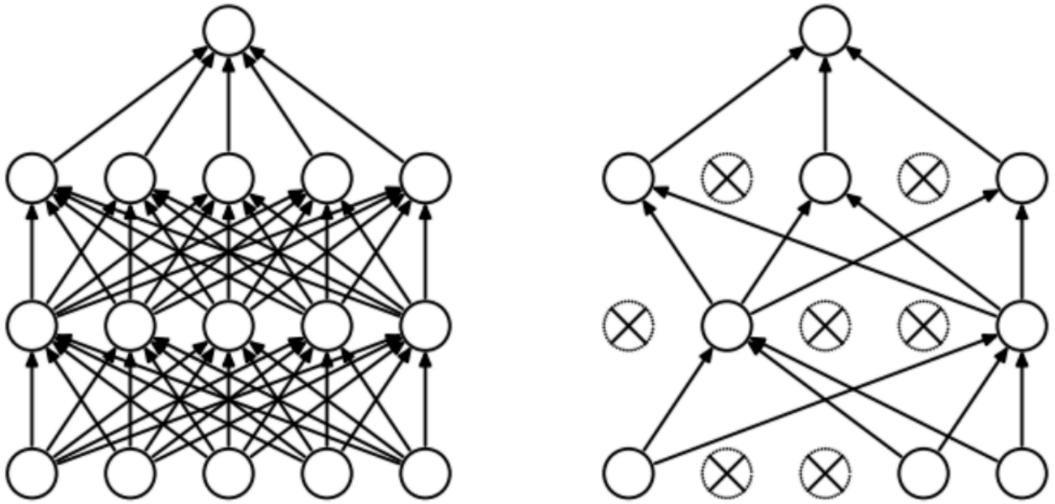
Summary: Advantages of Batch Normalization

- Improves gradient flow
- Allows higher learning rates
- Reduces strong dependence on initialization
- Acts as a form of regularization

ResNet applies BatchNorm after each convolutional layer; improves accuracy and speed of Inception

Outline

Dropout



(a) Standard Neural Net (b) After applying dropout.

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

At each training stage, each node is randomly kept with probability p. Do this by generating a binary mask, multiply activations by mask

Why is this a good idea?

- Form of regularization
- Forces network to not be too reliant on any given neuron or feature
- Hence, less likely to overfit idiosyncracies of training data

Use Dropout on Fully Connected Layers

- Dropout less helpful on conv layers
  - The spatial relations encoded in features maps make activations correlated by construction
  - Have fewer parameters so don't need as much regularization
  - Probably not a good idea to use batch norm with dropout
- Use batch norm instead on conv layers
- Example: dropout used on those fully connected layers of VGG that have 100+ million params; ResNet used batch norm in its conv layers

Dropout at test time

- Want to test and do model inference using the full network
- Need to either scale down at test time (i.e. multiply activations by $1/2$ if dropped half of neurons during training) or scale up during training (i.e. multiply activations by 2)

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

Data Augmentation

- Data augmentation: transforming your data such that labels preserved
- Examples (usually applied to natural images):
  - random crops
  - flips
  - randomly vary contrast and brightness
  - rotate
  - stretch
  - lens distortion
- Not developed with documents in mind

Outline

Introduction

Loss Functions for Classification

Activation Functions

Data Pre-Processing

Initialization

Optimization

Regularization

Batch Normalization

Dropout

Data Augmentation

Transfer Learning

Setting Hyperparameters

Monitoring the Learning Process

Deep Document Classification in Practice

Transfer Learning

- With CNNs, transfer learning is the norm, not the exception
- CNNs are trained on labeled data and potentially take millions of labeled images to train from scratch; not that many options for pre-training
- Model zoos include models that are pre-trained, for example, on ImageNet
- You might want to go further, depending on your application (i.e. start with model weights fine-tuned on PubLayNet, or on another dataset you processed)

Transfer Learning

- Recall that earlier layers tend to be more generic and later layers more specific (i.e. remember our example of earlier layers picking up edges and later layers assembling them into more complex features). These later layers tend to be more task specific.
- Hence, depending on how much labeled data you have and how different your problem is from the one used to pre-train the network, you might freeze earlier layers in the network
- Set your learning rate for fine-tuning at 1/10th of what it would be for pre-training

Outline

Train, Validate, Test

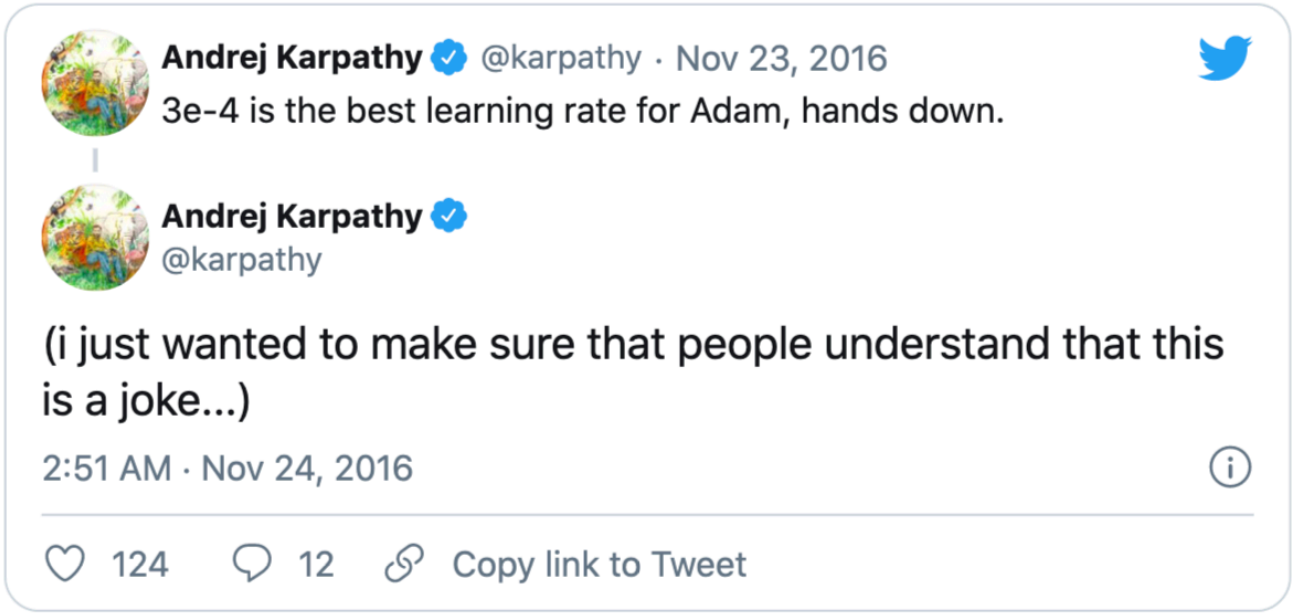
- Three partitions of data: train, validation, and test
- Validation set used to evaluate hyperparameter estimation
- Benchmark datasets will pre-specify each of these sets

Hyperparameter search

- You will need to cross-validate across hyperparameters
- Generally, the most important hyperparameter is the learning rate, along with the architecture itself

Choosing the Learning Rate

Choosing the Learning Rate



Importance of Random Hyperparameter search

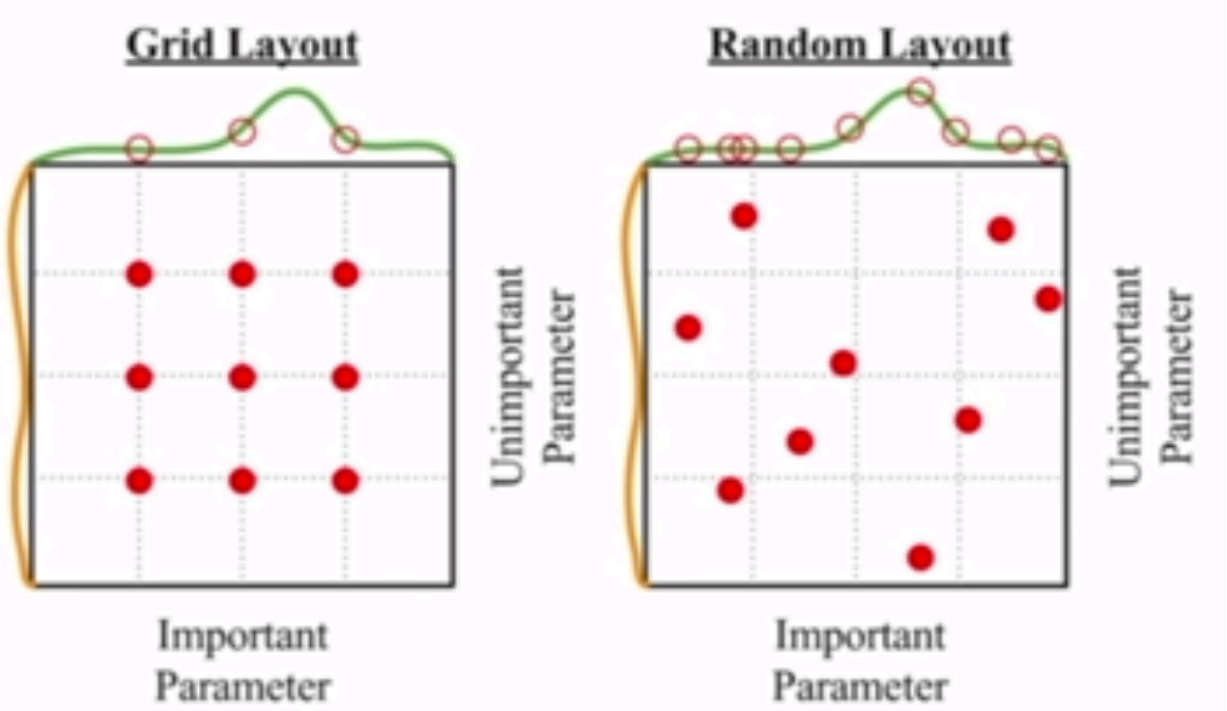


Image credit: Stanford CS 231n

Optimizing the Learning Rate

- Begin with a coarse search, will take just a few epochs to evaluate
- Then can do a finer search, need to sample in log space because the learning rate is multiplicative
- Once you have a good general learning rate, can worry about learning rate schedule (different answers about whether useful with an adaptive LR method like ADAM)

Loss and the Learning Rate

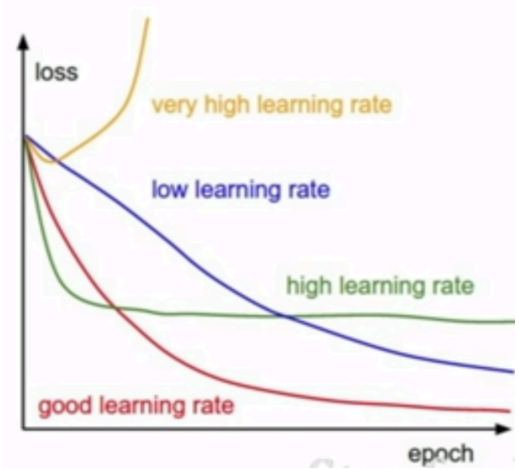
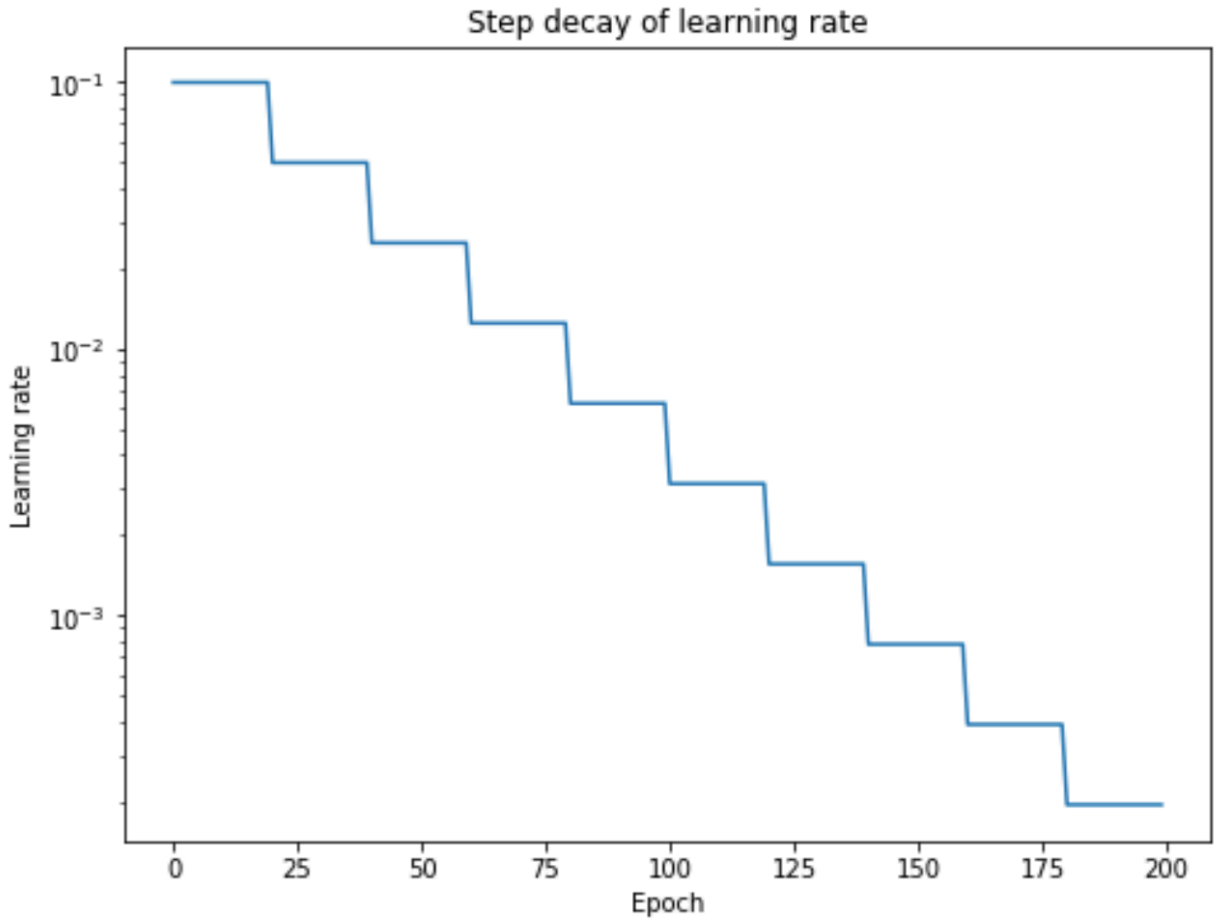


Image credit: Stanford CS 231n

Ideally you'd like to combine these learning rates, slowing down as you approach the minimum.

Learning Rate Step Decay



Redux

- The learning rate is the most important hyperparameter, so start by getting it into a reasonable range first
- Start with a static learning rate, once you've found something that works well you can worry about a dynamic learning rate
- People will tend to say that once you get the learning rate right, image classification tends to be less sensitive to other parameters
- When we get to object detection, we will return to this discussion, as there are some additional hyperparameters that in our experience are important

Outline

Monitoring the Learning Process

Plot your loss and accuracy

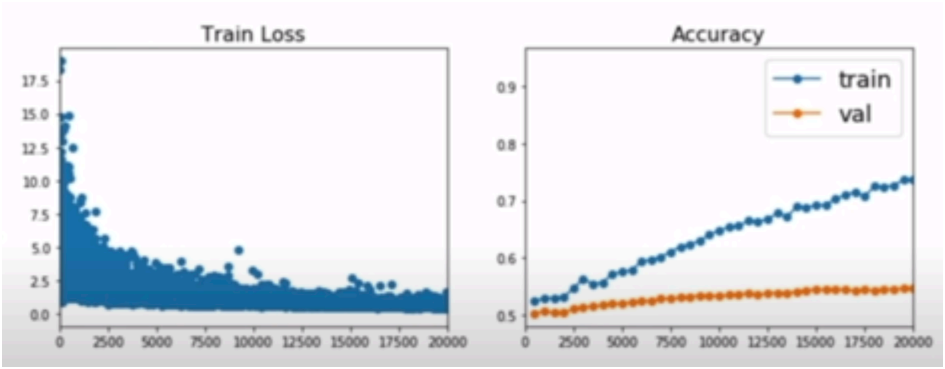


Image credit: Stanford CS 231n

A big gap between train and validation accuracy suggests overfitting (increase regularization); a small gap suggests scope to increase model capacity

Bad Initializing

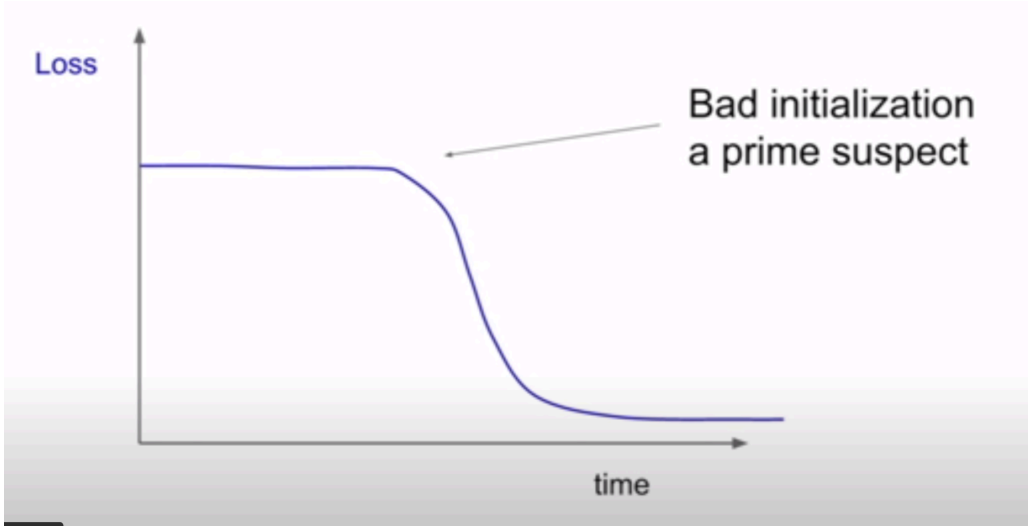


Image credit: Stanford CS 231n

If the loss explodes, this indicates that the learning rate is too high

Other things to check

- Update/parameter ratio, around 1E-3
- If cost is ever much larger than original (i.e. >3), it's not heading in the right direction

Tips and Tricks

- You can find a lot more suggested tips and tricks online and in the literature, such as in the He et al. paper "Bag of Tricks for Image Classification with Convolutional Neural Networks"
- Usually they are testing out tips and tricks on ImageNet, or perhaps COCO. The images we want to process are probably quite different than the images of animals, vehicles, etc. in a dataset like ImageNet, so your mileage might vary
- Nevertheless, this will give you a sense for some of the things that people try

TLDRs

- Activation Functions (use ReLU)
- Data Pre-Processing (subtract mean)
- Weight Initialization (He et al.)
- Optimization (SGD with momentum/Nesterov or ADAM)
- Use batch norm; also other regularization methods when more regularization needed (note how regularization methods may interact with each other and the optimizer)
- Follow best practices when optimizing hyperaparameters

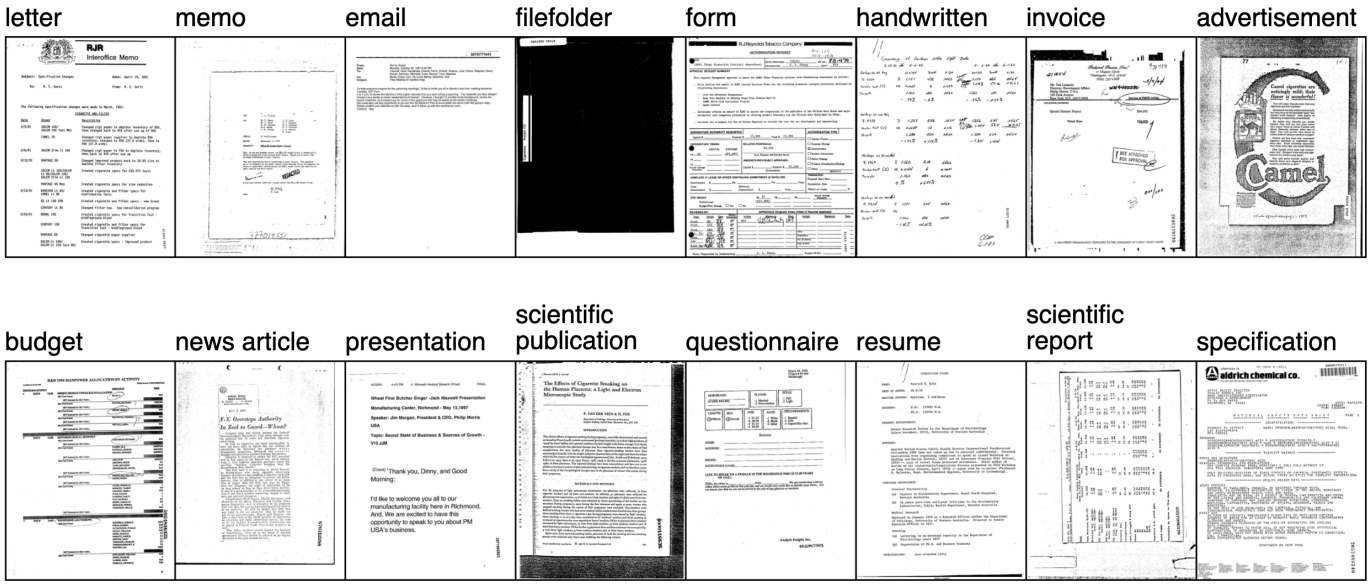
Outline

Benchmark Datasets for Document Classification

- Recall that the deep learning literature often advances by comparing performance against benchmarks
- The benchmark datasets for document classification are called RVL-CDIP and Tobacco3482

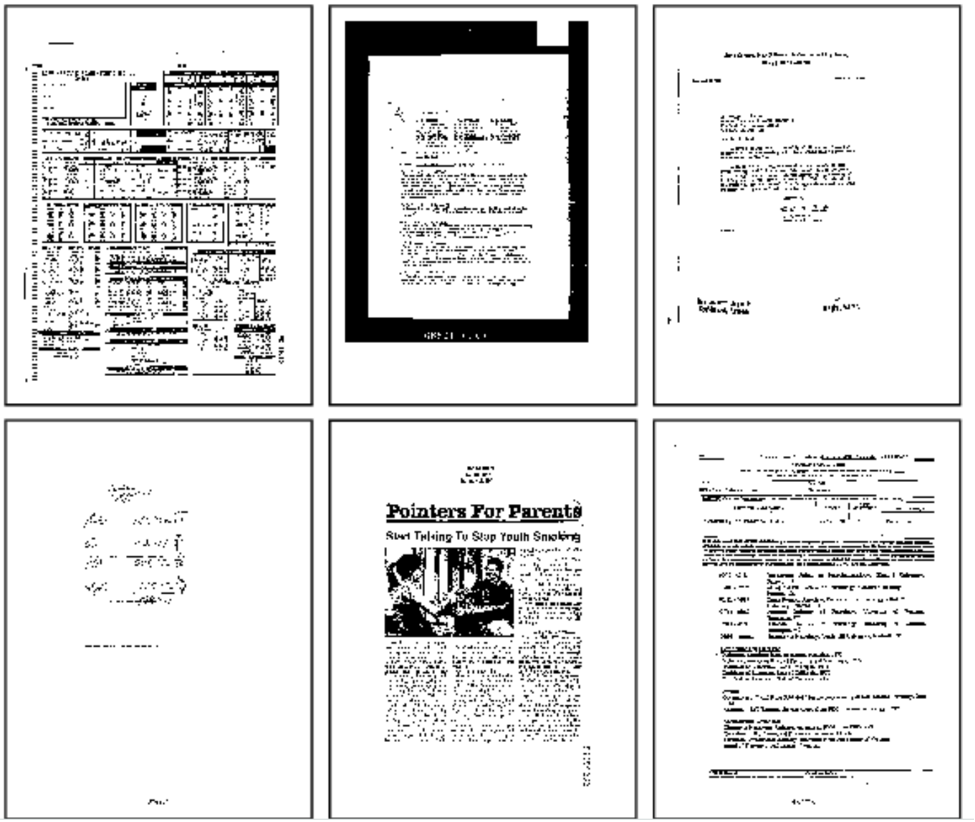
RVL-CDIP

400,000 grayscale images in 16 classes, with 25,000 images per class.



Tobacco3482

Like RVL-CDIP but quite small; 3,482 images in 10 classes



Deep Document Classification

- "Cutting the Error by Half: Investigation of Very Deep CNN and Advanced Training Strategies for Document Image Classification" (2017) tests a variety of deep CNNs pretrained on ImageNet for the purposes of document classification in the RVL-CDIP and Tobacco3482 datasets
- It finds VGG-16 gives SOTA performance
- Your data may not look much like these benchmarks, so as always your mileage may vary

VGG-16

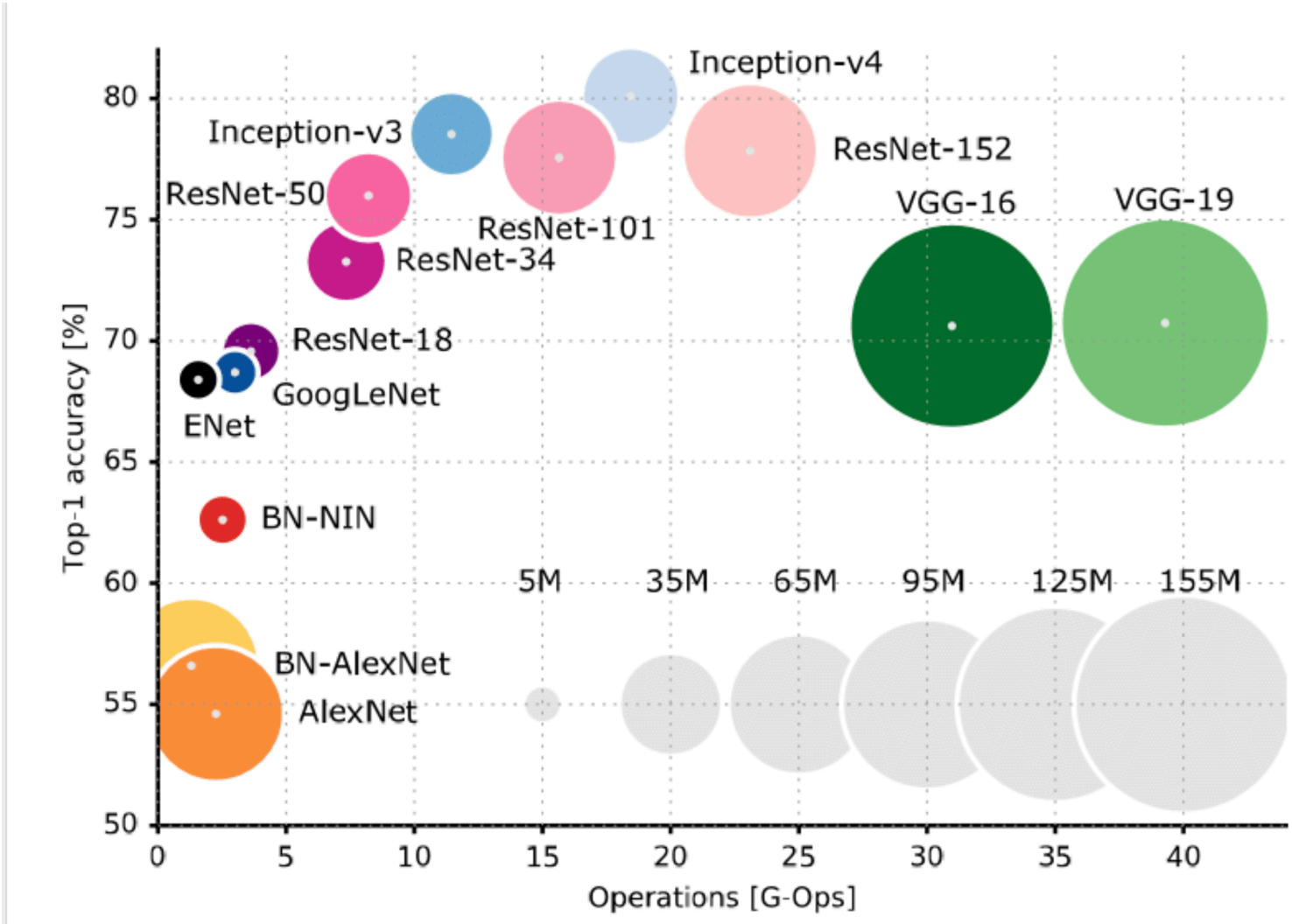


Image from Deep Learning: Decade in Review