* PyTorch released *

DL practitioners -

Lecture 5: Components contd

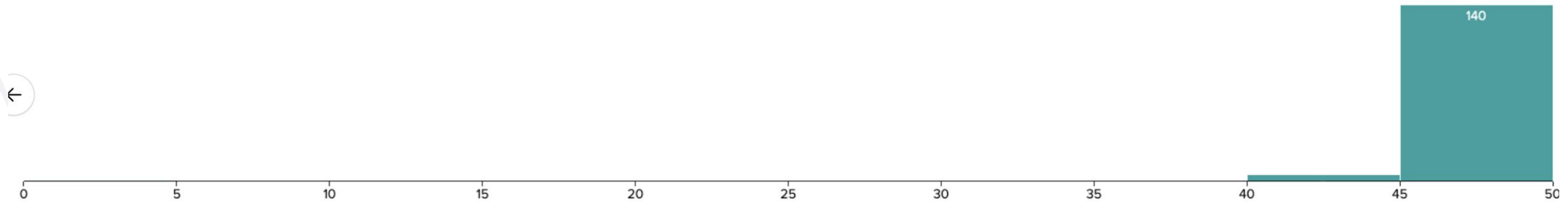
Today

Revisit a few things

Talk about Optimizers

Talk about Regularization

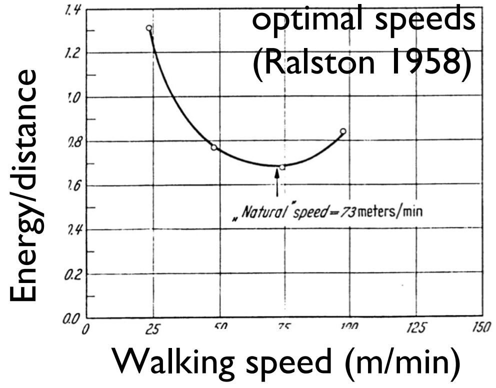
HW0



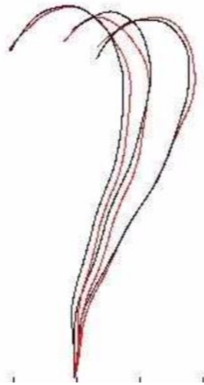
HW 1 due tonight at midnight

Cost function for a human to walk from one end of a room to another

- Human walking behavior is explained, roughly, by energy minimization e.g., step length, step width, gaits used, smooth turns, speed, etc.
- Stability (not falling down) is a constraint.
- Minimizing "jerk" (derivative of acceleration) captures smooth turning paths in navigation.
- Speed may also be affected by urgency or time constraints.
- Of course, reaching the destination.



Walking at energy optimal speeds (Ralston 1958)



Curved paths when given different initial & final directions, min jerk (Arechelaveta et al 2008)

Recap initialization

Potential problems

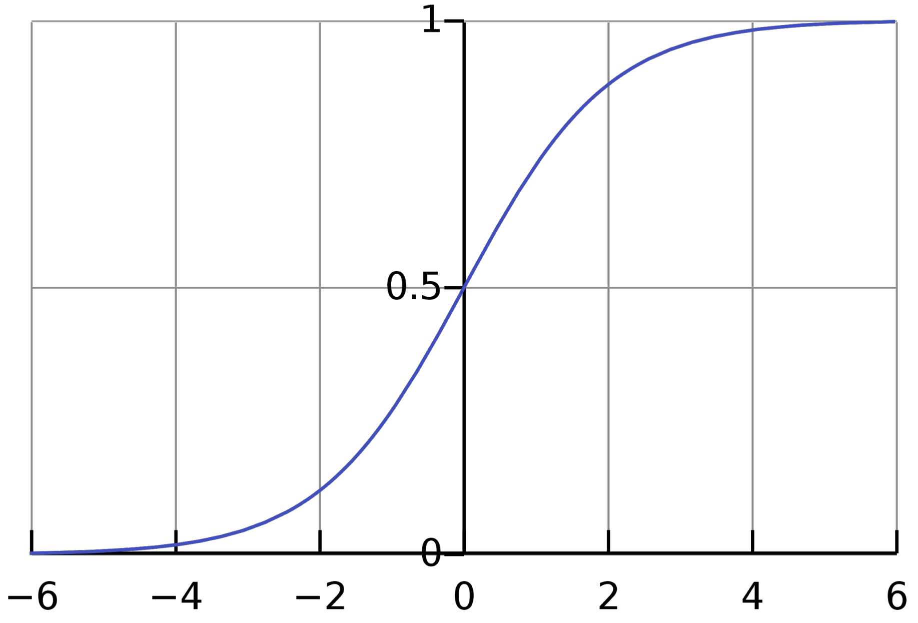
a) No gradients

b) Infinite gradients

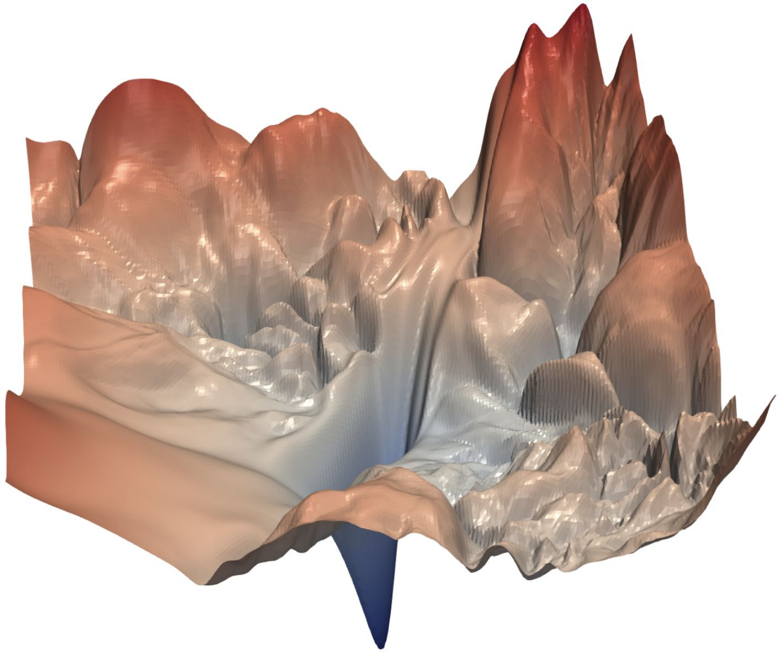
c) All gradients in (very) low dimensional space

Example of vanishing gradient

Sigmoid with big weight initialization



Exploding gradients



Hao Li, Zheng Xu, Gavin Taylor, Tom Goldstein Visualizing the Loss Landscape of Neural Nets

Remember the log P slide?

We use logP because it is

- Additive across samples
- Numerically stable

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Watch out! e^1000=\infty

Issues with Numerical Stability in HW1

- In specific cases the output of the model maybe really small or really huge. Some of the common reasons may be due to vanishing or exploding gradients.
- This would result in unexpected behaviour while predicting the outputs. Especially applying Sigmoid or softmax which include exponential functions.
- To handle such cases it's better to use numerically stable functions which use LogExpSum trick.
- Common example is to use to BCEWithLogitsLoss instead of BCELoss in pytorch.

LogSumExp

LogSumExp is a numerical stability trick used to handle underflow and overflow.

$$LSE(x_1, \ldots, x_n) = x^* + \log(\exp(x_1 - x^*) + \cdots + \exp(x_n - x^*))$$

where $x^* = \max\{x_1, \ldots, x_n\}$

Pytorch uses this numerical stability formulation while implementing some of it's functions such as BCEWithLogitsLoss.

Why is initialization difficult?

Wrong weight scaling -> exponential effects

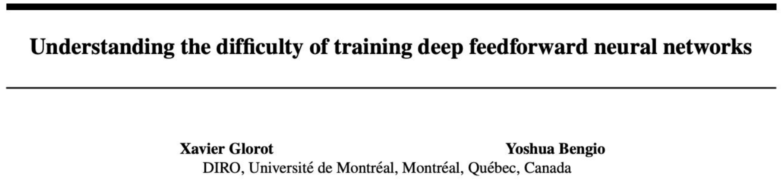
$$W = \alpha W_0 \qquad y = \alpha^N y_0$$

1.5^100=4*10^17

That is why He/ Xavier are the defaults

$$\text{Xavier initialization (tanh, linear, sigmoid)} : w_i \sim N(0; \frac{1}{\sqrt{n}}) \quad | \quad Var(w_i) = \frac{1}{n}$$

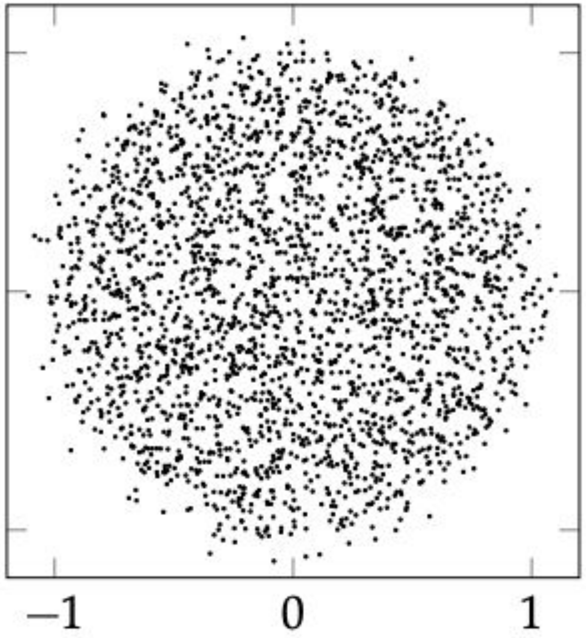
$$\text{He et. al. initialization (ReLU)} : w_i \sim N(0; \frac{1}{\sqrt{2n}}) \quad | \quad Var(w_i) = \frac{1}{2n}$$



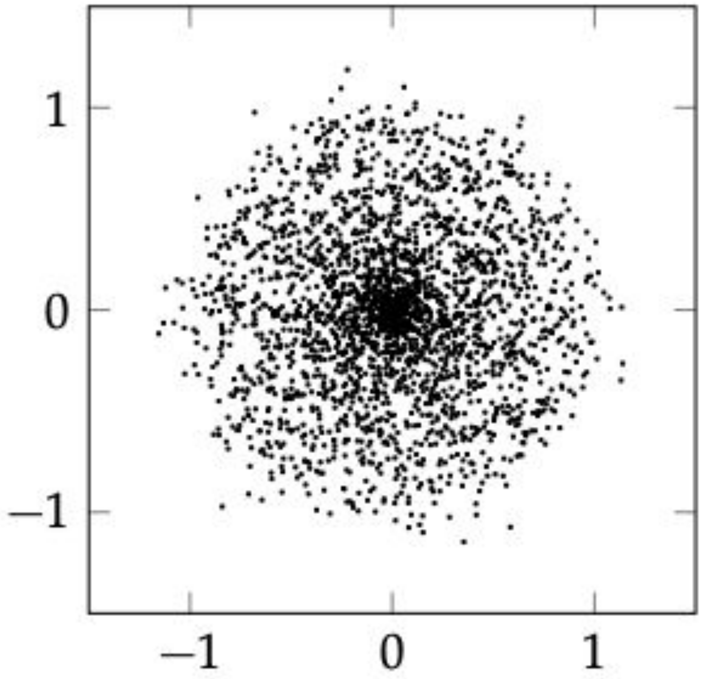
Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

SVD spectrum of random 50*50 gaussian matrices



SVD spectrum of product of two such matrices



Solution

Initialize small(ish).

Exponential change overpowers initialization

What is DL?

What is deep learning

Bengio:

Deep learning is inspired by neural networks of the brain to build learning machines which discover rich and useful internal representations, computed as a composition of learned features and functions.

What DL really does

$\text{Loss}=f_{\text{network}}(\text{weights, output, internals})$

Minimize Loss by optimizer

What is machine learning?

What is machine learning?

Generalization!

Training Data

Testing Data

Let x_i drawn from isotropic Gaussian, high D

length distribution is sparse

length distribution is power law

all are (roughly) the same length

None of the above

Why?

$$E[||x_i||_2^2] = E\left[\sum_j ||x_i||^2\right] \propto N$$

$$E[std[||x_i||_2]] = std\left[\sum_j ||x_i||^2\right] \propto \sqrt{N}$$

We should

- Cover more theory
- Cover more examples
- Remove material from course
- Add materials to course
- Go slower
- Go faster

The components of DL

Linear operations

Activation functions

Loss functions

Initializations

Optimizers

Regularizations

Architectures (rest of the course)

Recap SGD

Batch vs Online

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

$$w := w - \eta \nabla Q_i(w)$$

Almost certainly finds local minimum, or global with (pseudo)convex

In practice we use minibatch

Why?

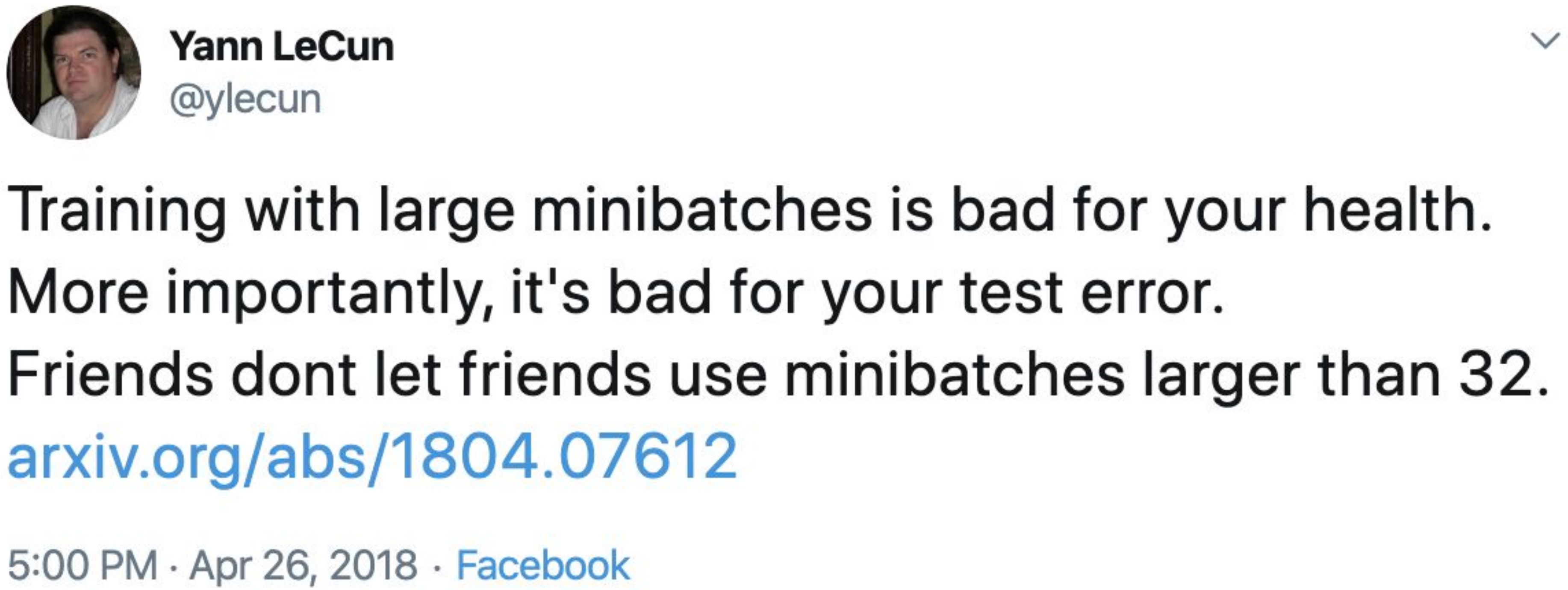
Less memory

Avoid saddle points

Traverse space faster

Noise may be helpful

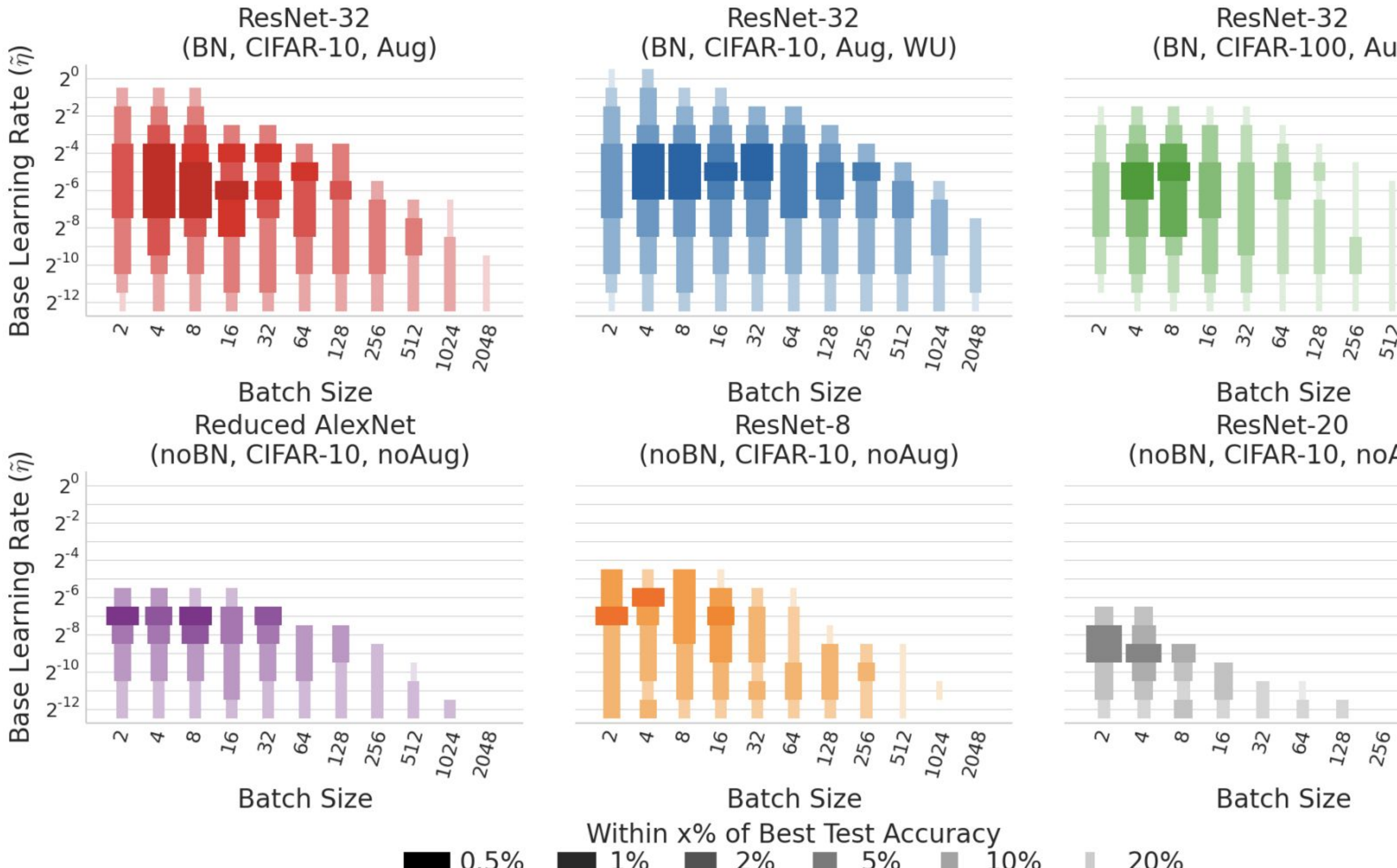
Failure mode: not sampling randomly



Yann LeCun
@ylecun

Training with large minibatches is bad for your health.
More importantly, it's bad for your test error.
Friends dont let friends use minibatches larger than 32.
arxiv.org/abs/1804.07612

5:00 PM · Apr 26, 2018 · Facebook


ResNet-32 (BN, CIFAR-10, Aug)
ResNet-32 (BN, CIFAR-10, Aug, WU)
ResNet-32 (BN, CIFAR-100, Au
Reduced AlexNet (noBN, CIFAR-10, noAug)
ResNet-8 (noBN, CIFAR-10, noAug)
ResNet-20 (noBN, CIFAR-10, noA
Base Learning Rate ($\tilde{\eta}$)
2^{0}
2^{-2}
2^{-4}
2^{-6}
2^{-8}
2^{-10}
2^{-12}
Batch Size
2
4
8
16
32
64
128
256
512
1024
2048
Within x% of Best Test Accuracy
0.5%
1%
2%
5%
10%
20%

Gradient Magnitude

Rate tuning

$$w^{t+1} = w^t + \mu \cdot \nabla_w$$

$$w^{t+1} = w^t + \frac{\mu}{t} \cdot \nabla_w$$

How to choose the constant?

Exponential search when in doubt

Advanced optimizers: Adagrad

- "Adaptive gradient algorithm"
- Adapts a learning rate for each parameter based on size of previous gradients.

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2 . \qquad w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j .$$

Advanced optimizers: RMSprop

- "Root mean square prop"
- Adapts a learning rate for each parameter based on size of previous gradients.

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

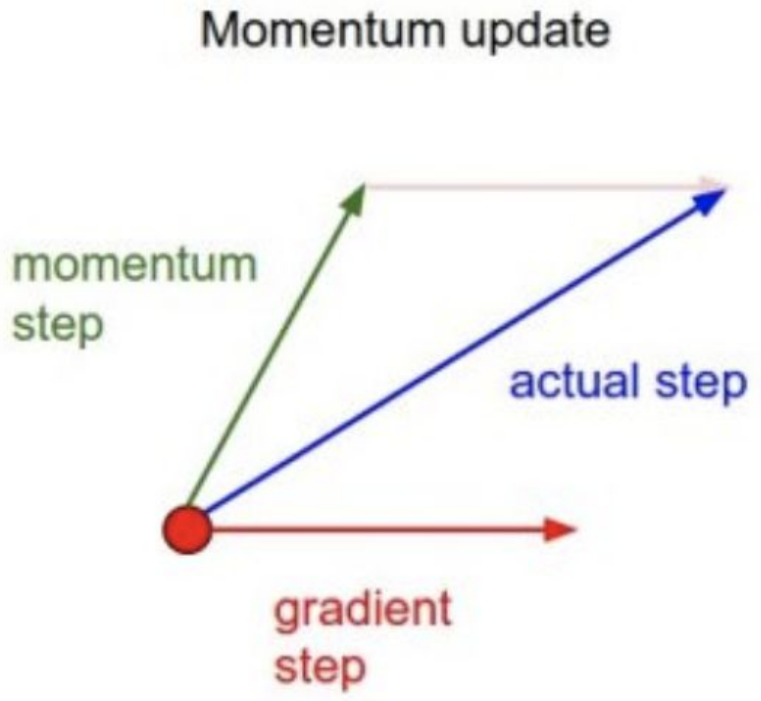
$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

Moment methods

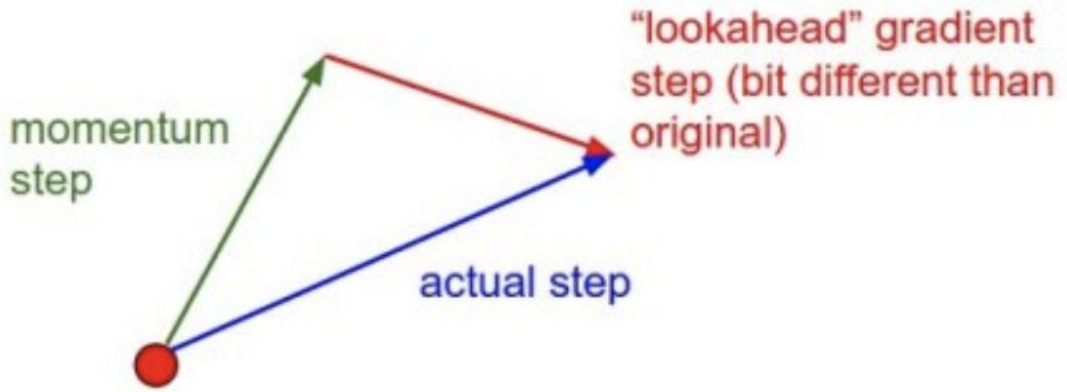
$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$

$$\hat{m}_w = \frac{m_w^{(t+1)}}{1-(\beta_1)^{t+1}}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta\ \hat{m}_w$$



Nesterov variant



Advanced optimizers: Adam

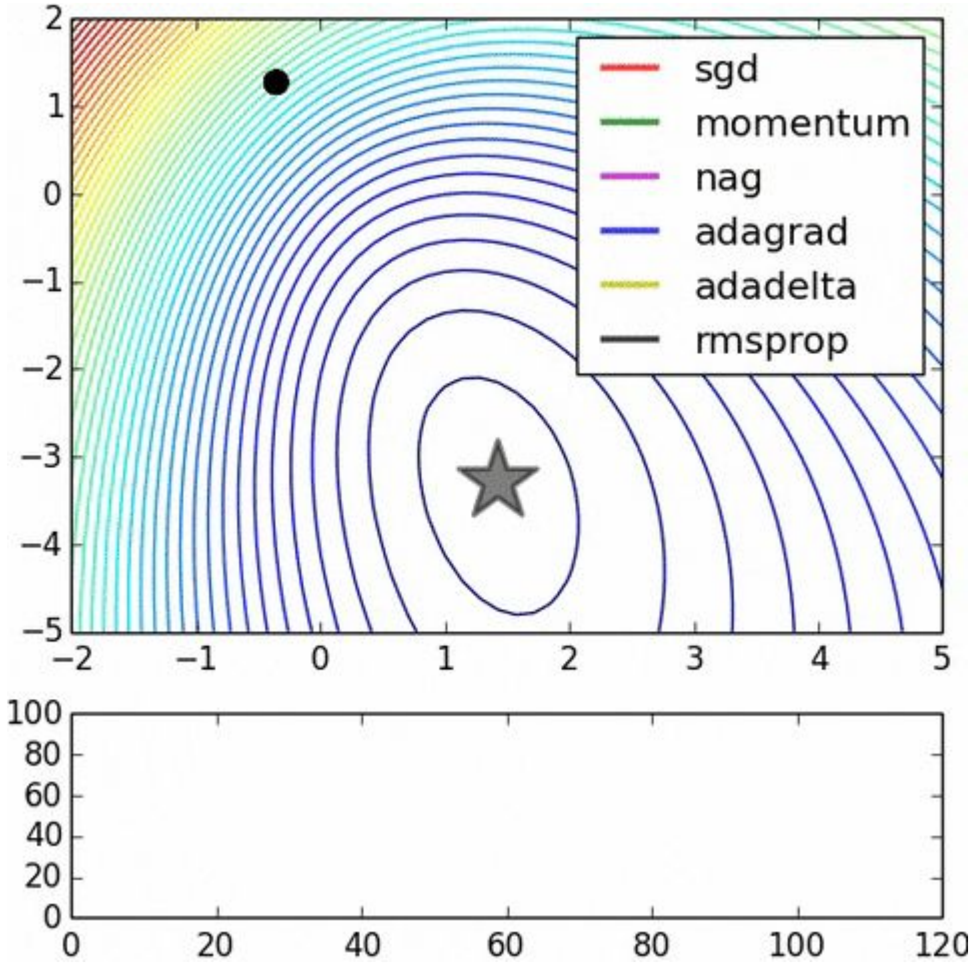
- "Adaptive moment estimation"
- Similar to RMSprop, but with both the first and second moments of the gradients

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1 - \beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1 - \beta_2)(\nabla_w L^{(t)})^2$$

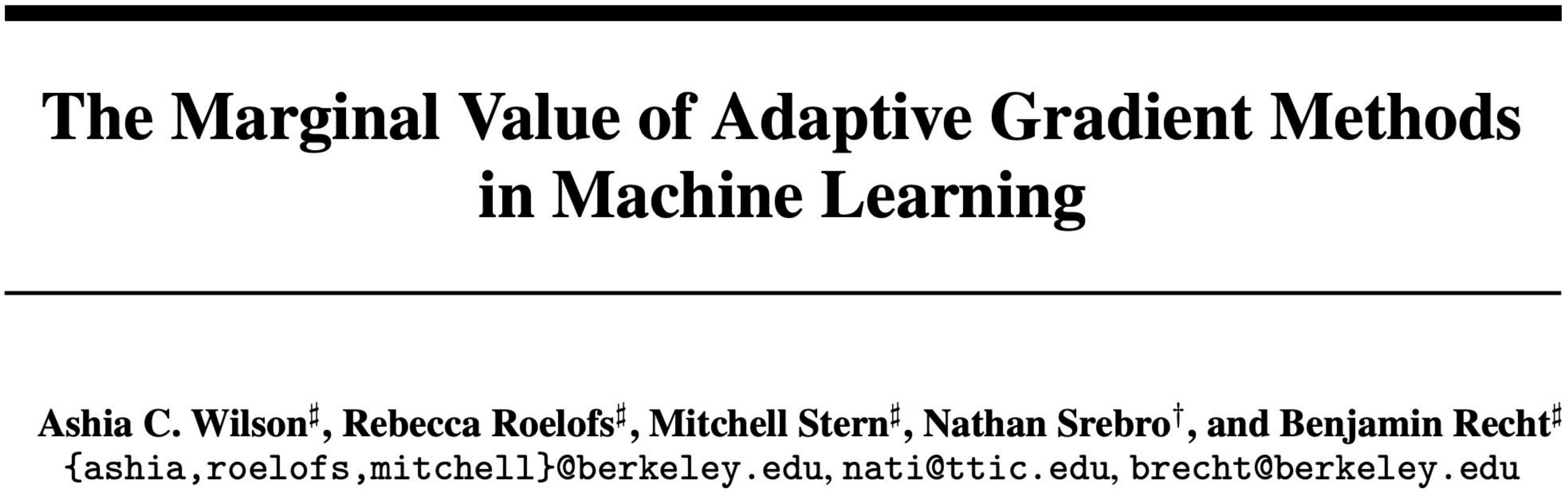
$$\hat{m}_w = \frac{m_w^{(t+1)}}{1 - (\beta_1)^{t+1}}$$

$$\hat{v}_w = \frac{v_w^{(t+1)}}{1 - (\beta_2)^{t+1}}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

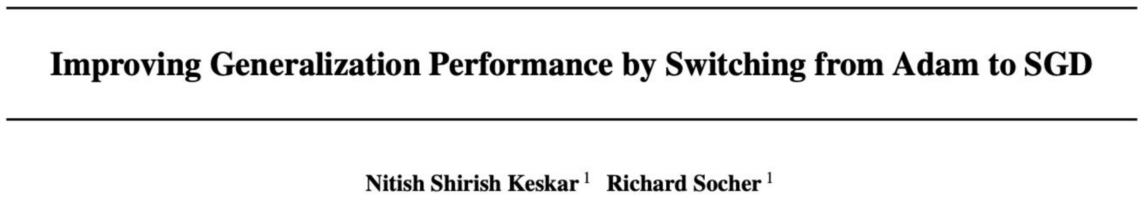


Source: Daniel Nouiri



The Marginal Value of Adaptive Gradient Methods in Machine Learning

Ashia C. Wilson[♯], **Rebecca Roelofs**[♯], **Mitchell Stern**[♯], **Nathan Srebro**[†], **and Benjamin Recht**[♯]
`{ashia,roelofs,mitchell}@berkeley.edu`, `nati@ttic.edu`, `brecht@berkeley.edu`

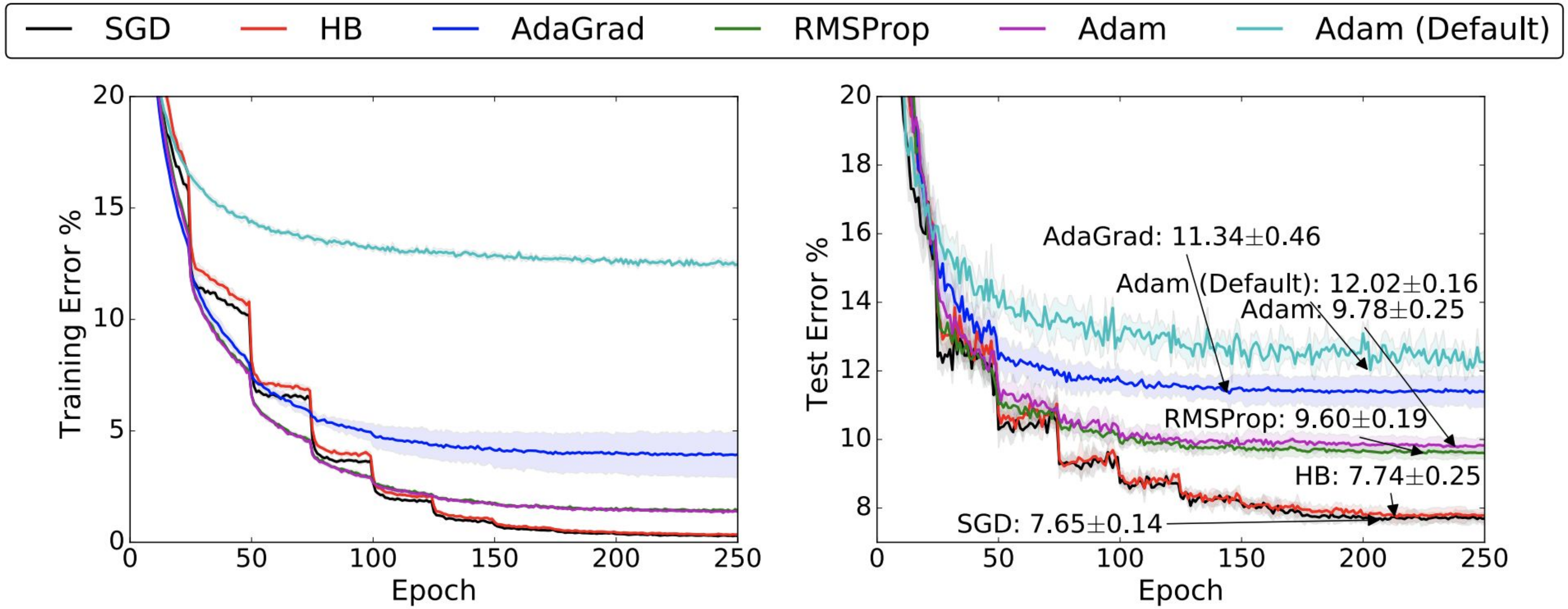


Improving Generalization Performance by Switching from Adam to SGD

Nitish Shirish Keskar[1] **Richard Socher**[1]

Adaptive optimization converges weirdly

See derivations in Recht paper



(a) CIFAR-10 (Train) **(b)** CIFAR-10 (Test)

How to choose the constant?

Empirically as option

*.25 *.5 *1 *2 *4

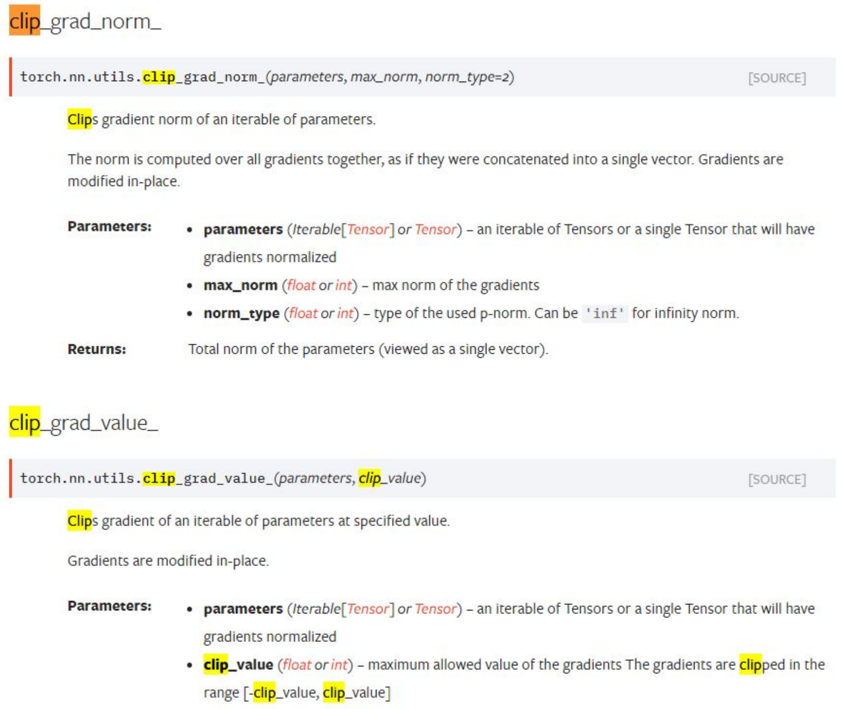
Interesting:

Recht's team uses *100 faster learning rate than defaults

Experiments

Name	Network type	Architecture	Dataset	Framework
C1	Deep Convolutional	`cifar.torch`	CIFAR-10	Torch
L1	2-Layer LSTM	`torch-rnn`	War & Peace	Torch
L2	2-Layer LSTM + Feedforward	`span-parser`	Penn Treebank	DyNet
L3	3-Layer LSTM	`emnlp2016`	Penn Treebank	Tensorflow

What to do when your gradients are too large



clip_grad_norm_

```
torch.nn.utils.clip_grad_norm_(parameters, max_norm, norm_type=2)
```
[SOURCE]

Clips gradient norm of an iterable of parameters.

The norm is computed over all gradients together, as if they were concatenated into a single vector. Gradients are modified in-place.

Parameters:

- **parameters** (*Iterable*[*Tensor*] or *Tensor*) – an iterable of Tensors or a single Tensor that will have gradients normalized
- **max_norm** (*float* or *int*) – max norm of the gradients
- **norm_type** (*float* or *int*) – type of the used p-norm. Can be `'inf'` for infinity norm.

Returns: Total norm of the parameters (viewed as a single vector).

clip_grad_value_

```
torch.nn.utils.clip_grad_value_(parameters, clip_value)
```
[SOURCE]

Clips gradient of an iterable of parameters at specified value.

Gradients are modified in-place.

Parameters:

- **parameters** (*Iterable*[*Tensor*] or *Tensor*) – an iterable of Tensors or a single Tensor that will have gradients normalized
- **clip_value** (*float* or *int*) – maximum allowed value of the gradients The gradients are clipped in the range [-clip_value, clip_value]

What to do when your gradients are too small

- Find a better activation?
- Find a better architecture?
- Find a better initialization?
- Find a better career?

The components of DL

Linear operations

Activation functions

Loss functions

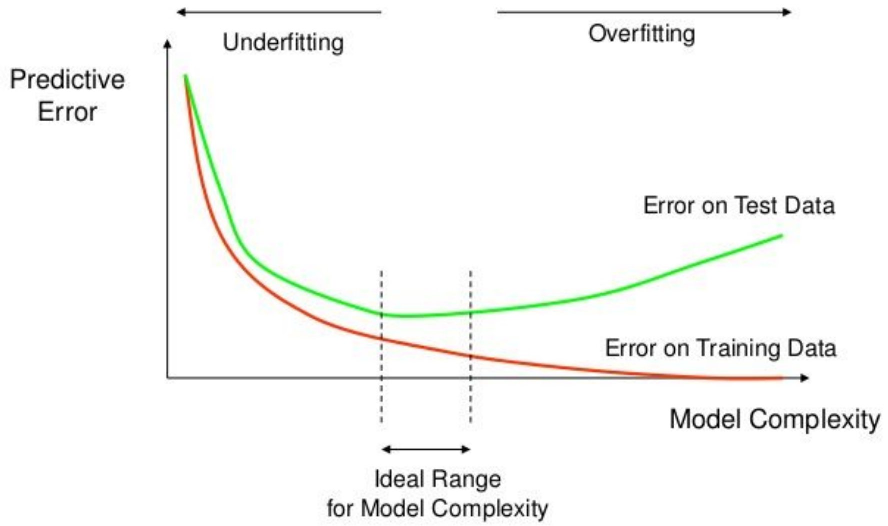
Initializations

Optimizers

Regularizations

Architectures (rest of the course)

Direct regularization



When would we want to use L1 over L2?

When the data set is sparse

When you believe that some features are irrelevant

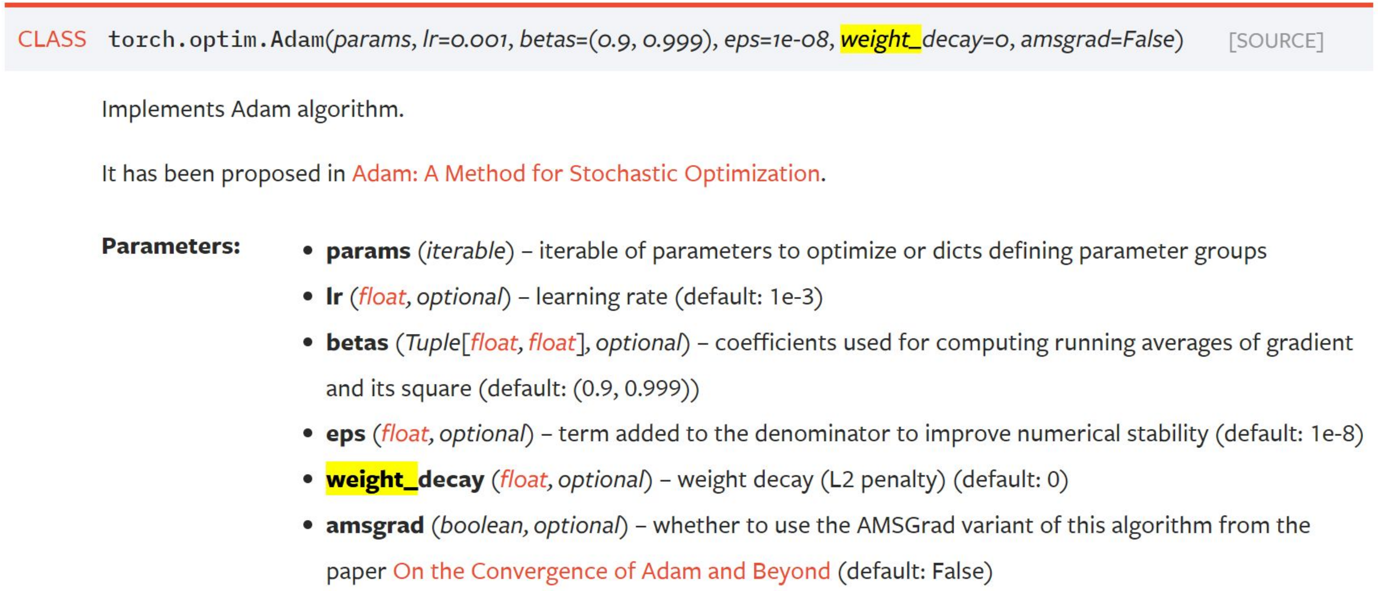
Only when dealing with speech data

When the dataset is really big

None of the above

Classical regularization

$$\theta^* = \arg\min_{\theta} \left(L(\theta) + c||\theta||_2^2 \right)$$



CLASS `torch.optim.Adam`(*params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False*) [SOURCE]

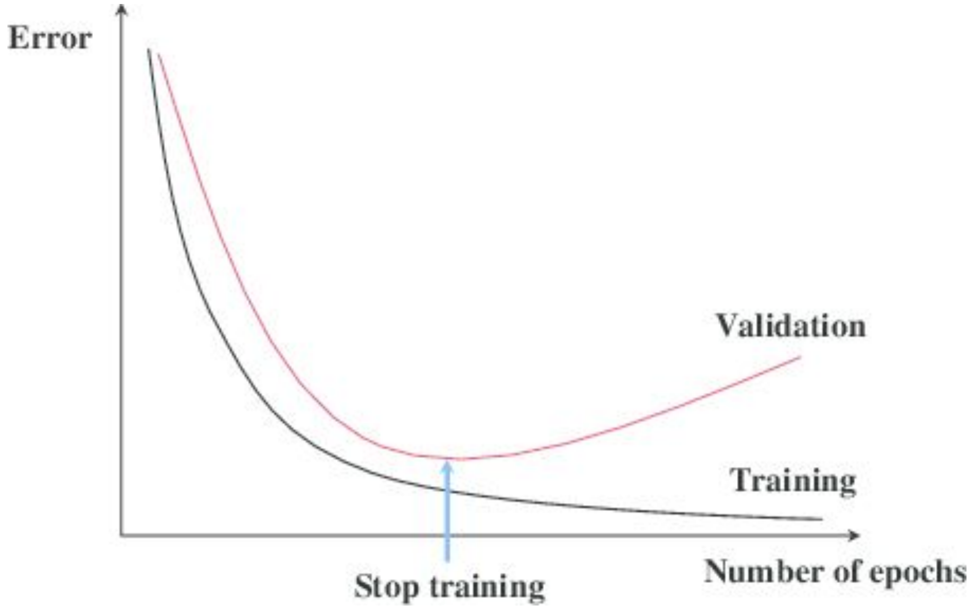
Implements Adam algorithm.

It has been proposed in Adam: A Method for Stochastic Optimization.

Parameters:

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float, optional*) – learning rate (default: 1e-3)
- **betas** (*Tuple[float, float], optional*) – coefficients used for computing running averages of gradient and its square (default: (0.9, 0.999))
- **eps** (*float, optional*) – term added to the denominator to improve numerical stability (default: 1e-8)
- **weight_decay** (*float, optional*) – weight decay (L2 penalty) (default: 0)
- **amsgrad** (*boolean, optional*) – whether to use the AMSGrad variant of this algorithm from the paper On the Convergence of Adam and Beyond (default: False)

Early stopping*



*not built-in. Oh, the horror.

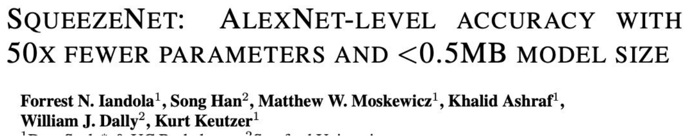
Convnets are

able to achieve ≈ 0 training error on CIFAR10 and IMAGENET and still generalize (i.e., test error remains small, despite the potential for overfitting);

still able to achieve ≈ 0 training error even after the labels are *randomized*, and does so with only a small factor of additional computational time.

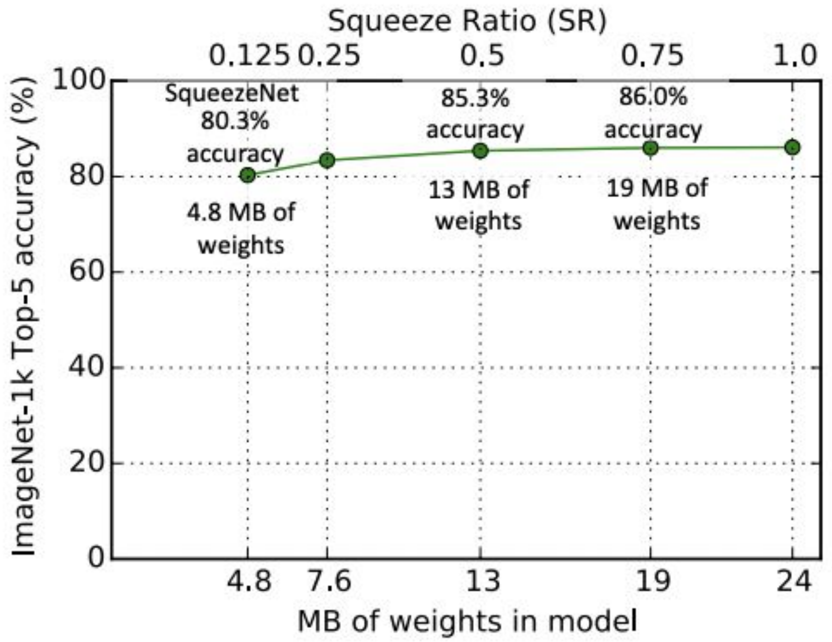
Dziugaite and Roy, 2017

Just build smaller networks (example squeezenet

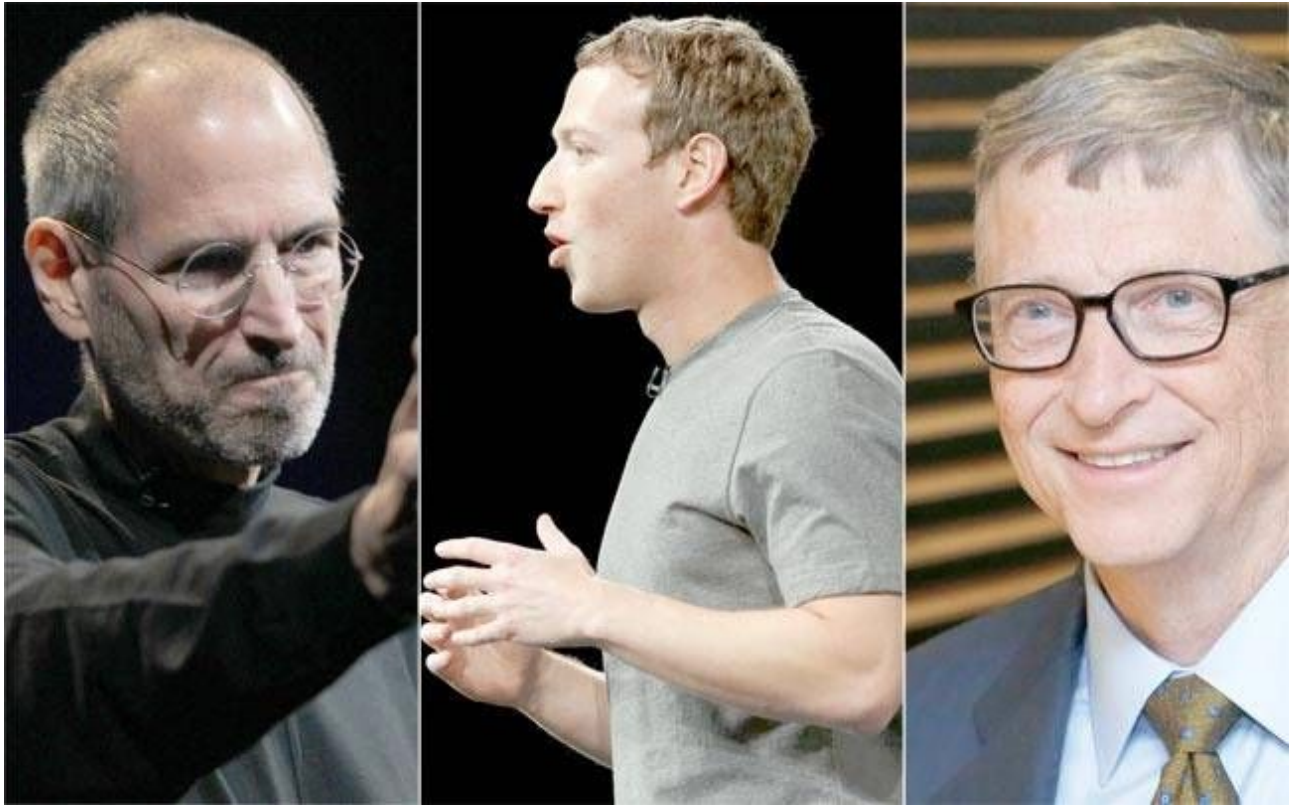


SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola[1], Song Han[2], Matthew W. Moskewicz[1], Khalid Ashraf[1], William J. Dally[2], Kurt Keutzer[1]

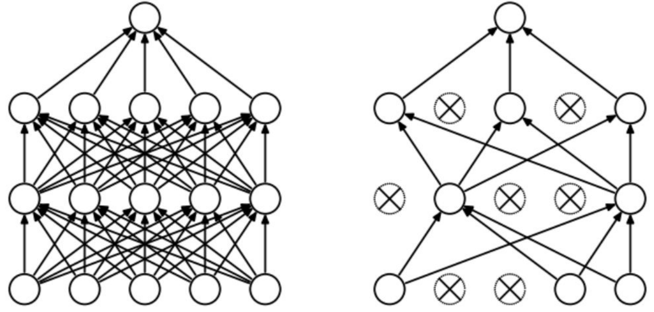


Dropout

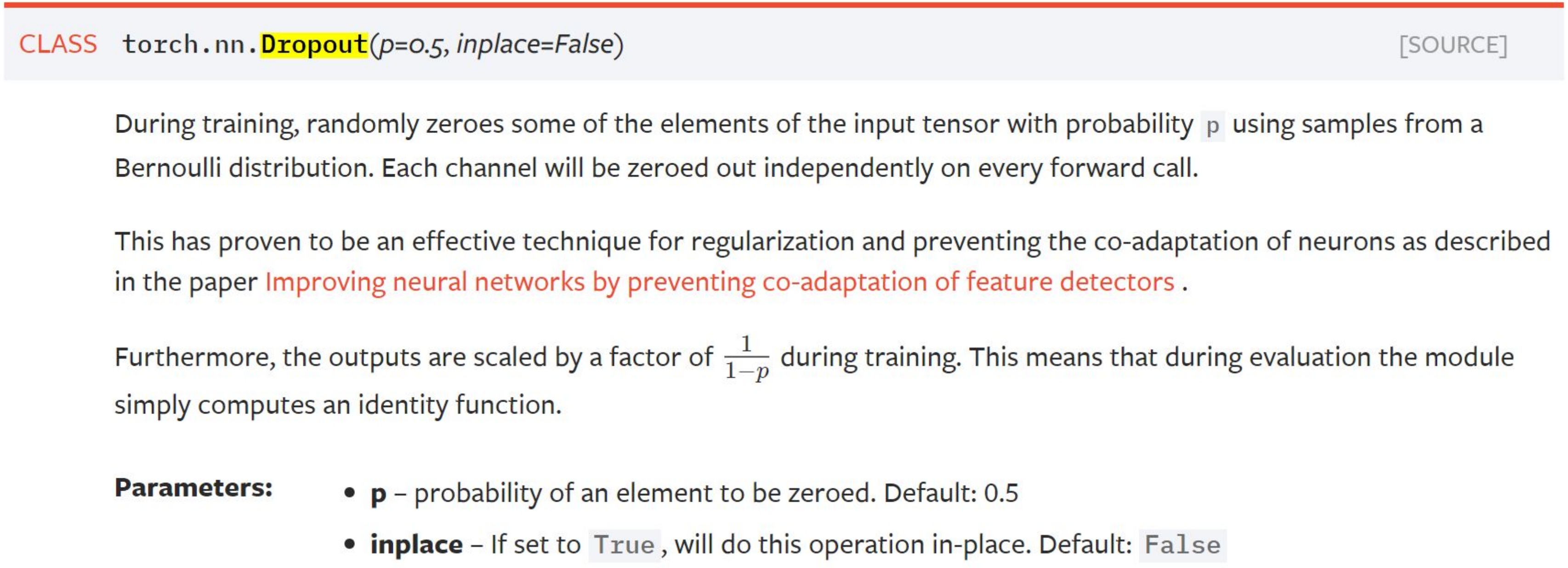


Dropout

- Prevents overfitting by preventing co-adaptations
- Forces "dead branches" to learn
- Enforces distributed nature
- Distributes learning signal
- Adds lots of noise
- In a way approximates ensemble methods
- Developed by Geoffrey Hinton, patented by Google


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

Dropout



CLASS `torch.nn.Dropout`(*p=0.5*, *inplace=False*) [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters:

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Poisson spiking is similar

Noise proportional to signal

For both scenarios

Some very strong inhibitory cells

Does the brain effectively do dropout?

Regularization through training algorithm

Training is over when there are no more errors

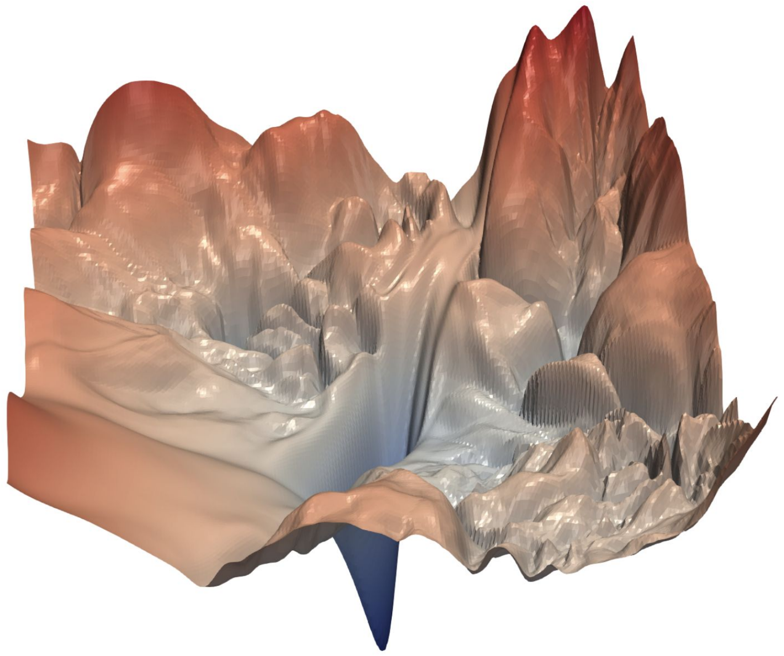
A brief interlude on algorithms

- A batch algorithm does a computation all in one go given an entire dataset
  - Closed-form solution to linear regression
  - Selection sort
  - Quicksort
  - Pretty much anything involving gradient descent.
- An online algorithm incrementally computes as new data becomes available
  - Heap algorithm for the *k*th largest element in a stream
  - Perceptron algorithm
  - Insertion sort
  - Greedy algorithms

Minibatching

- A minibatch is a small subset of a large dataset.
- In order to perform gradient descent, we need an accurate measure of the gradient of the loss with respect to the parameters. The best measure is the average gradient over all of the examples (gradient descent is a batch algorithm).
- Computing over 60K examples on MNIST for a single (extremely accurate) update is stupidly expensive.
- We use minibatches (say, 50 examples) to compute a noisy estimate of the true gradient. The gradient updates are worse, but there are **many** of them. This converts the neural net training into an **online** algorithm.

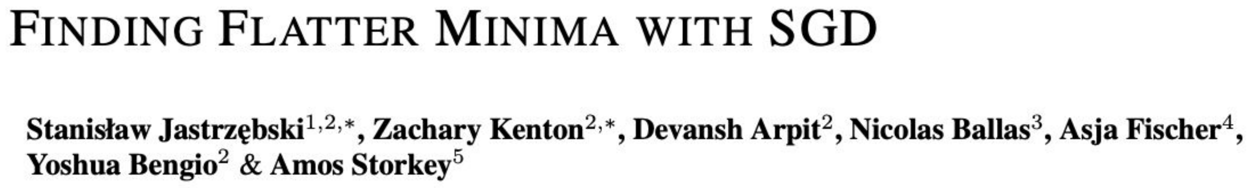
SGD searches for shallow minima



Hao Li, Zheng Xu, Gavin Taylor, Tom Goldstein Visualizing the Loss Landscape of Neural Nets

Intuition SGD prefers flat minima

blackboard



FINDING FLATTER MINIMA WITH SGD

Stanisław Jastrzębski[1,2,*], Zachary Kenton[2,*], Devansh Arpit[2], Nicolas Ballas[3], Asja Fischer[4], Yoshua Bengio[2] & Amos Storkey[5]

Looking Forward (TODO)

- Next week, we take another deep look at the nature and hardness of learning
- We start talking about

How could lecture 5 be better?