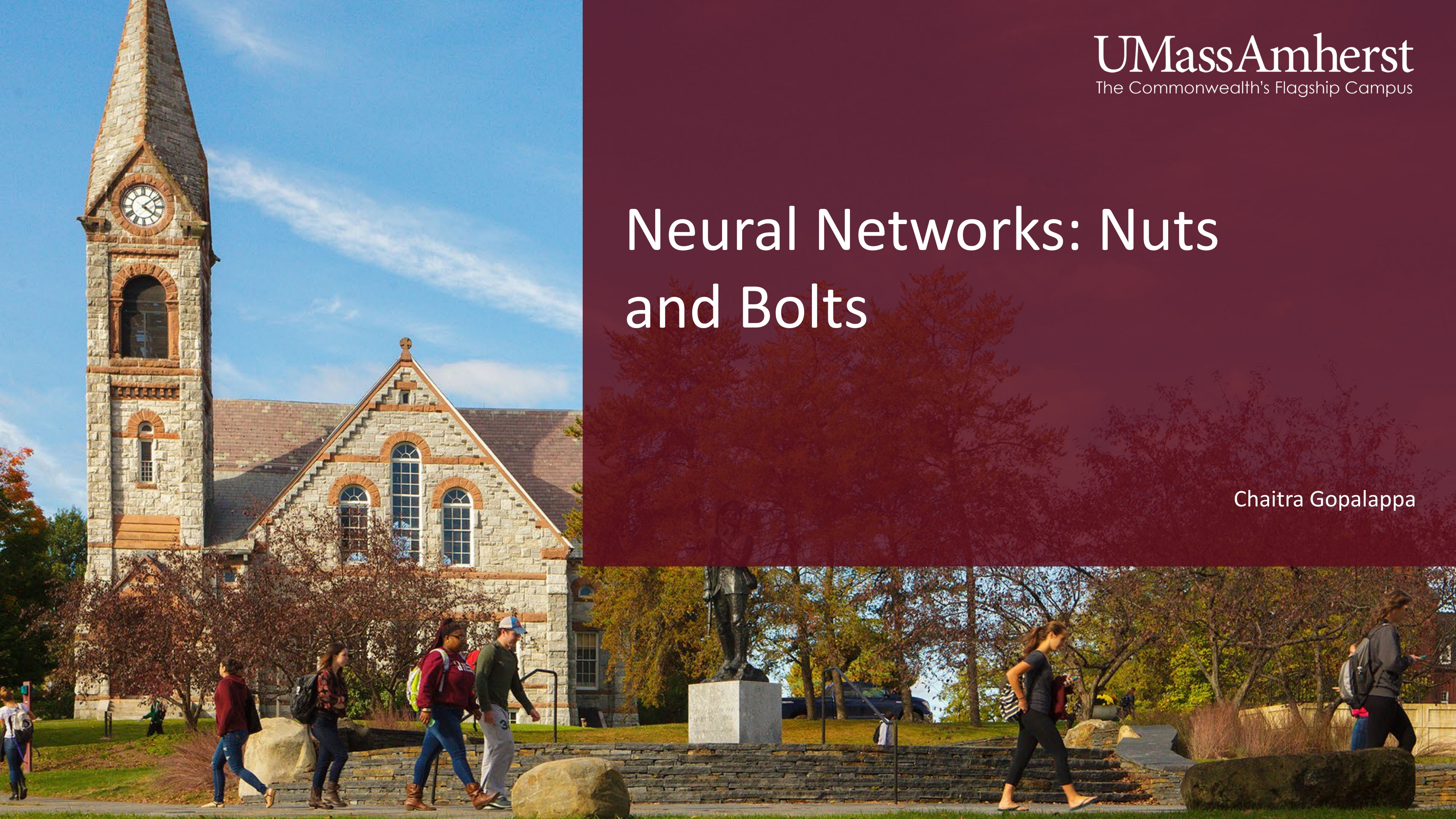


UMassAmherst
The Commonwealth's Flagship Campus

Neural Networks: Nuts and Bolts

Chaitra Gopalappa

Batch, mini-batch, incremental (online)

Source: Bengio, Practical recommendations for gradient-based training of deep architectures, 2012, https://arxiv.org/abs/1206.5533

Machine learning

- Typical to split data into train and test sets
- Use ‘train’ set to train the data
- Test the trained model on the ‘test’ set

Batch, mini-batch, incremental (online)

- Batch: use the full train set to train the model
- Mini-batch: divide train set into small mini-batches
  - Typically 2^n: $32, 64, 128, 256$ (corresponding to CPU /GPU architecture)
- Incremental/ online learning: single sample at a time

- Bengio, Practical recommendations for gradient-based training of deep architectures, 2012, https://arxiv.org/abs/1206.5533

Batch, mini-batch, incremental (online)

- Batch: use the full train set to train the model → Gradient descent
- Mini-batch: divide train set into small mini-batches
  - Typically 2^n: $32, 64, 128, 256$ (corresponding to CPU /GPU architecture)
- Incremental/ online learning: single sample at a time

Stochastic Gradient descent

- Bengio, Practical recommendations for gradient-based training of deep architectures, 2012, https://arxiv.org/abs/1206.5533

Batch, mini-batch, incremental (online)

- Batch: use the full train set to train the model
- Mini-batch: divide train set into small mini-batches
  - Typically 2^n: $32, 64, 128, 256$ (corresponding to CPU /GPU architecture)
  - Batch-size then becomes a hyperparameter
- Incremental/ online learning: single sample at a time

Gradient descent

Stochastic Gradient descent

- Bengio, Practical recommendations for gradient-based training of deep architectures, 2012, https://arxiv.org/abs/1206.5533

Neural Network Optimizers

Source:
Chapter 8: Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning
https://www.deeplearningbook.org/contents/optimization.html

Recollect line search algorithms

Transformation

$$\vec{x}_{m+1} \leftarrow \vec{x}_m - \mu_m \vec{p}_m$$

$\vec{p}_m$: search direction

μ_m: step size

Neural Network Optimizers

- SDG
- SDG with momentum
- SDG with Nestrov

Gradient as search direction

- AdaGrad
- RMSProp
- Adam

Adaptive learning rate/ step size

- Newtons method
- Conjugate gradients
- BFGS

Hessian as search direction

Line Search methods: identify search direction and step in that direction

Recollect slides from first two weeks Generalized: Stochastic gradient Descent (SDG) algorithm

- *Transformation*

$$\vec{x}_{m+1} \leftarrow \vec{x}_m - \mu_m Y_m(\vec{x}_m)$$

$$Y_m(\vec{x}_m) = \nabla f(\vec{x}_m) + \textit{noise}$$

SDG in machine learning

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

Sufficient conditions for convergence $\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \text{and} \quad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$

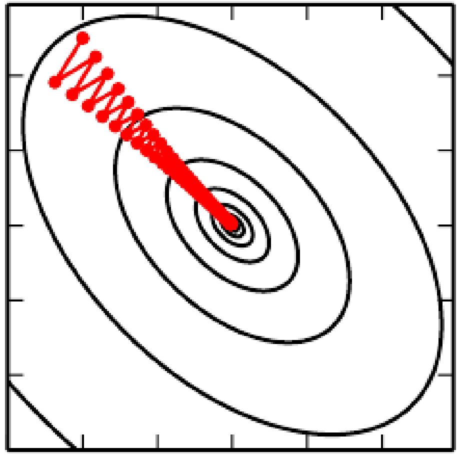
Typical to decay learning rate linearly until some iteration τ, with $\alpha = \frac{k}{\tau}$. After iteration τ, leave ϵ constant.
$\epsilon_0, \epsilon_\tau, \tau$ are hyperparameters

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

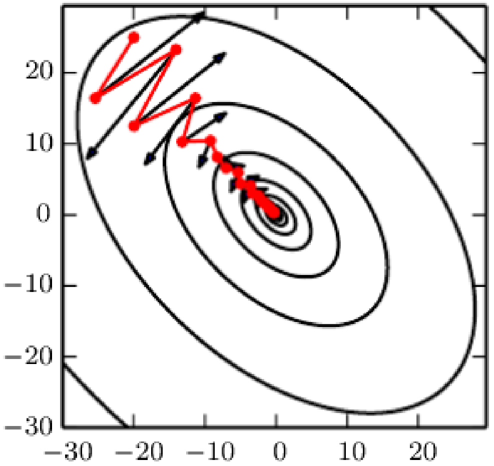
If $m = B$ (batch size/ full training dataset)it is general GD
If $m = 1$ it is incremental or online SGD
If $1 < m < B$ it is mini-batch SDG; typically $m = 2^n$(some exponent of 2)

Source: Chapter 8:
https://www.deeplearningbook.org/contents/optimization.html

SDG



SDG with momentum



Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

The difference between Nesterov momentum and standard momentum is where the gradient is evaluated. With Nesterov momentum, the gradient is evaluated after the current velocity is applied.

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

Source: Chapter 8:
https://www.deeplearningbook.org/contents/optimization.html

Algorithms with Adaptive Learning Rates

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

Source: Chapter 8: https://www.deeplearningbook.org/contents/optimization.html

Algorithms with Adaptive Learning Rates

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Source: Chapter 8:
https://www.deeplearningbook.org/contents/optimization.html

SDG function in Pytorch

- https://pytorch.org/docs/stable/generated/torch.optim.SGD.html#torch.optim.SGD
- Bengio, Practical recommendations for gradient-based training of deep architectures, 2012, https://arxiv.org/abs/1206.5533

SGD

CLASS torch.optim.SGD(*params*, *lr=<required parameter>*, *momentum=0*, *dampening=0*, *weight_decay=0*, *nesterov=False*, ***, *maximize=False*, *foreach=None*, *differentiable=False*) [SOURCE]

Docs > torch.optim > SGD

input : γ (lr), θ_0 (params), $f(\theta)$ (objective), λ (weight decay), μ (momentum), τ (dampening), *nesterov*, *maximize*

for $t = 1$ **to** $\ldots$ **do**
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  if $\lambda \neq 0$
    $g_t \leftarrow g_t + \lambda \theta_{t-1}$
  if $\mu \neq 0$
    if $t > 1$
      $\mathbf{b}_t \leftarrow \mu \mathbf{b}_{t-1} + (1-\tau) g_t$
    else
      $\mathbf{b}_t \leftarrow g_t$
    if *nesterov*
      $g_t \leftarrow g_t + \mu \mathbf{b}_t$
    else
      $g_t \leftarrow \mathbf{b}_t$
  if *maximize*
    $\theta_t \leftarrow \theta_{t-1} + \gamma g_t$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma g_t$

return θ_t

More variants added over time

- Best way to keep up is to look at NN libraries (Keras, Pytorch, Tensorflow) on available options
  - https://keras.io/api/optimizers/
  - https://pytorch.org/docs/stable/optim.html
  - https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

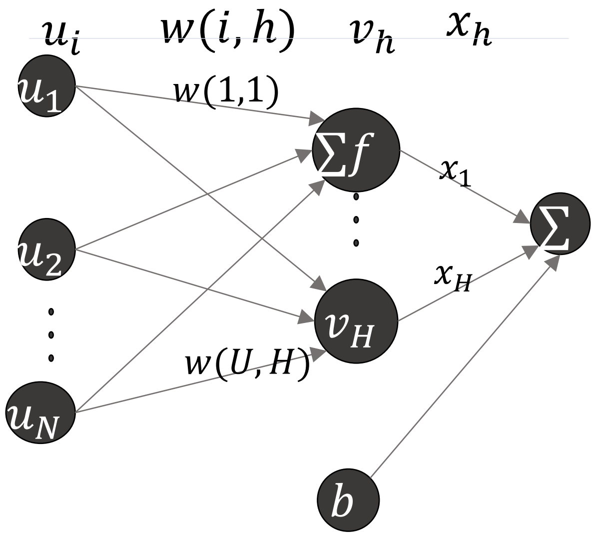
Tensors and AutoDiff

- AutoDiff: Baydan et. al., Journal of Machine Learning Research 18 (2018) 1-43
- https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

Activation functions

Challenges with Sigmoid- vanishing gradient

- Sigmoid is a saturation function
- $v_h = \frac{1}{1+e^{-\vec{w}_h \vec{u}}}$; if weights are initialized to be very large or small, v_h saturates at 0 or 1
- Gradient then becomes very small (stops learning) → vanishing gradient problem



Other activation functions

- Hyperbolic tangent
- Rectified Linear Unit (ReLU)
- Leaky ReLU
- Exponential Linear unit
- Numerous other AF: https://arxiv.org/abs/1811.03378

Additional components useful for RL

Output layer

- Softmax with multiple nodes in output layer→ converts output to a probability distribution
  - AF may then be used in output layer as well,
  - In RL, for policy gradient, we will use multiple nodes in output layer with softmax AF.
  - Lapan Ch3-Module1 code uses softmax
- Droput in output layer :
  - It is a form of regularization in supervised learning, especially when data is sparse; https://arxiv.org/pdf/1207.0580.pdf
  - It is not used that much in reinforcement learning, but there is some research in its use in policy gradient https://doi.org/10.48550/arXiv.2202.11818
  - Lapan Ch3-Module1 code uses dropout

UMassAmherst

The Commonwealth's Flagship Campus