ST449 Artificial Intelligence and Deep Learning

Lecture 3

Training neural networks

Milan Vojnovic

https://github.com/lse-st449/lectures

Topics of this lecture

- Empirical risk minimization
- Gradient descent algorithm
- Stochastic gradient descent algorithm
- Acceleration by momentum
- Adaptive learning rates
- Backpropagation algorithm
- Dropout

Attention!

- When you see a slide marked like in the upper-right corner of this slide, this indicates a technical part that you may skip in a first reading
- Usually, such marked slides contain a proof or a detailed technical argument
- Those interested only in the end results would be able to follow the presentation without looking into the marked slides
- Those interested to go more deeply into understanding how various claims are established would benefit from reading the marked slides

Training neural networks

Empirical risk minimization

- Model:

$$\boldsymbol{y} = h_{\boldsymbol{w}}(\boldsymbol{x})$$

output parameter input

- The expected loss function:

$$f(\boldsymbol{w}) = \mathbf{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D}\left[\ell\left(\boldsymbol{y}, h_{\boldsymbol{w}}(\boldsymbol{x})\right)\right]$$

given loss function

- Empirical risk minimization:

$$f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell\left(\boldsymbol{y}_i, h_{\boldsymbol{w}}(\boldsymbol{x}_i)\right) + \lambda\phi(\boldsymbol{w})$$

regularization term, $\lambda \geq 0$

Loss functions for binary classification

True class y (0 or 1), $\hat{y}$ prediction

- Misclassification loss (negative accuracy):

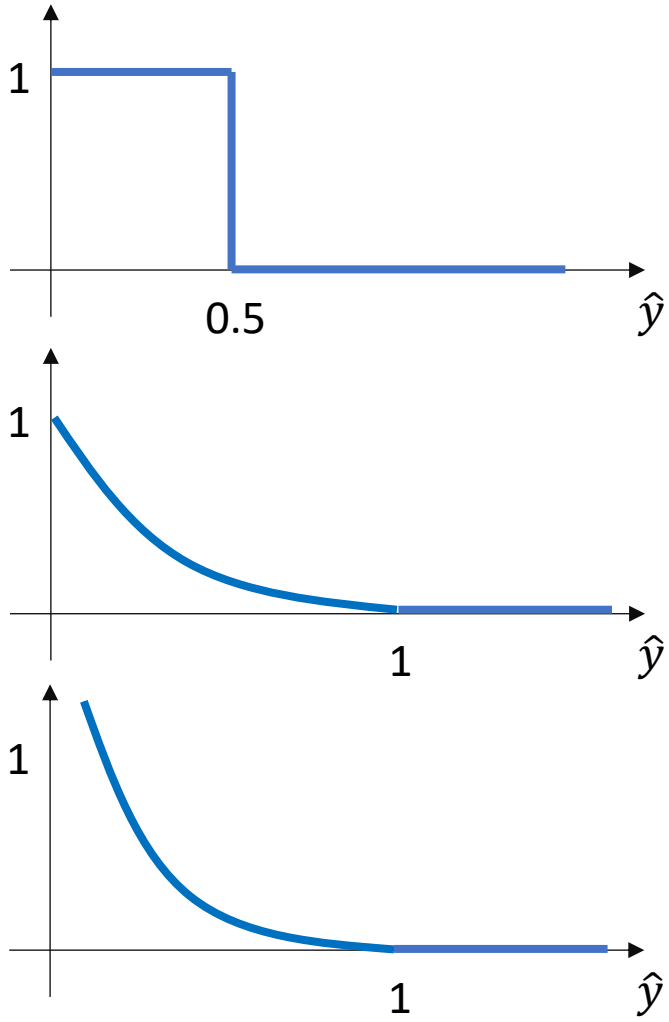
$$\ell(y, \hat{y}) = 1_{\left(y-\frac{1}{2}\right)\left(\hat{y}-\frac{1}{2}\right)<0}$$

- Squared loss:

$$\ell(y, \hat{y}) = (\hat{y} - y)^2$$

- Cross-entropy loss:

$$\ell(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$



Loss functions for multi-class case

True output $\boldsymbol{y} \in \{0,1\}^d$ such that $\sum_{i=1}^{d} y_i = 1$
Predicted distribution $\hat{\boldsymbol{y}}$

- Squared loss: $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 = \sum_{i=1}^{d} (y_i - \hat{y}_i)^2$
- Cross-entropy: $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{d} y_i \log(\hat{y}_i)$

- Cross-entropy interpretation:
  - Penalty $\log(\frac{1}{\hat{y}_i})$ if y_i is the true class
  - KL-divergence: $\mathrm{KL}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = \sum_{i=1}^{d} y_i \log\left(\frac{y_i}{\hat{y}_i}\right) = \underbrace{-H(\boldsymbol{y})} + \ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$

Entropy $H(\boldsymbol{y}) = -\sum_{i=1}^{d} y_i \log(y_i)$

$H(\boldsymbol{y}) = 0$ for $\boldsymbol{y}$ that has all its mass on one element

Standard regularizations

- Ridge or L_2 norm:

$$\phi(\boldsymbol{w}) = \|\boldsymbol{w}\|_2$$

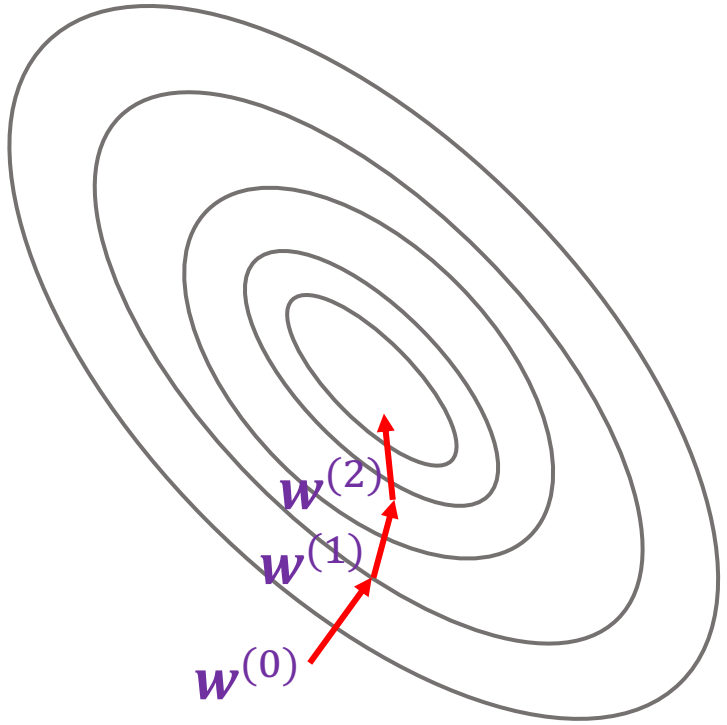
- Lasso:

$$\phi(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$$

Gradient descent algorithm

- Gradient descent algorithm update:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \underbrace{\eta^{(t)}}_{\text{step size}} \boldsymbol{B}^{(t)} \nabla f\left(\boldsymbol{w}^{(t)}\right)$$

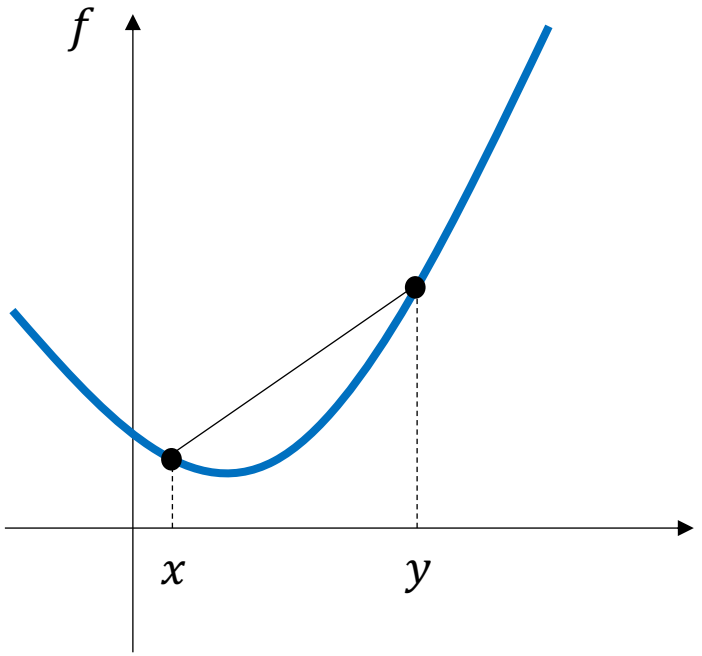


- Common step sizes:
  - A small positive constant
  - A decreasing sequence such that $\sum_{t=1}^{\infty} \eta^{(t)} = \infty$ and $\sum_{t=1}^{\infty} \left(\eta^{(t)}\right)^2$ finite
- Common choices of $\boldsymbol{B}^{(t)}$:
  - Standard gradient descent: $\boldsymbol{B}^{(t)} = I$
  - Adaptive learning rates: Newton's method $\boldsymbol{B}^{(t)} = \nabla^2 f\left(\boldsymbol{w}^{(t)}\right)^{-1}$, AdaGrad, RMSProp, Adam, …

Convex functions

- A function $f: \mathbf{R}^n \to \infty$ is said to be convex if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n$

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \text{ for all } \lambda \in [0,1]$$

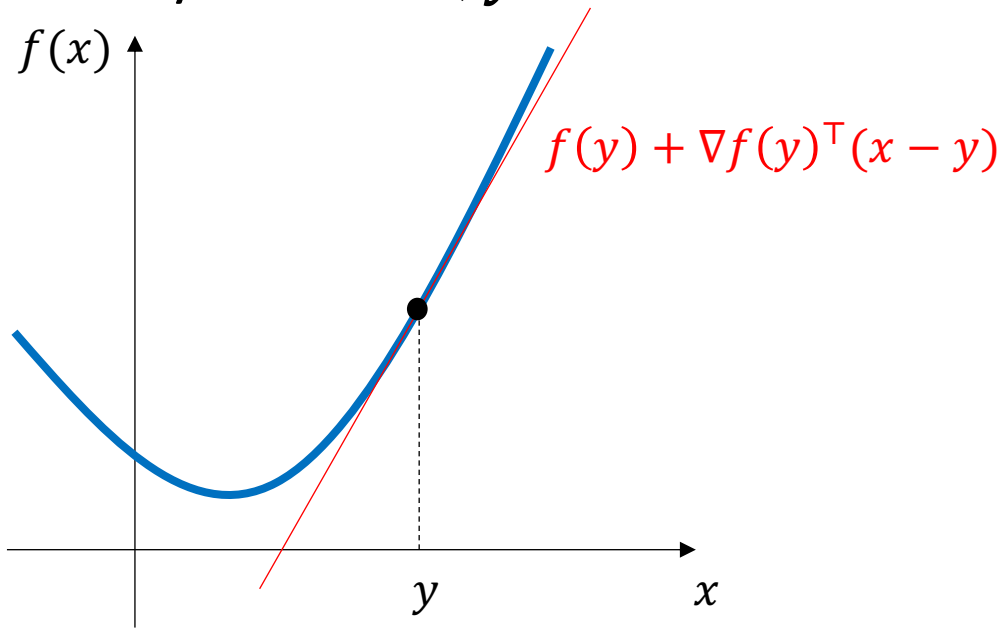


- f is said to be strictly convex if strict inequality holds for all $\lambda \in (0,1)$

Convex functions (cont'd)

- If f is differentiable, then f is convex if and only if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n$

$$f(\boldsymbol{x}) \geq f(\boldsymbol{y}) + \nabla f(\boldsymbol{y})^\top (\boldsymbol{y} - \boldsymbol{x})$$



- If f is twice-differentiable, then
  - f is convex iff its Hessian $\nabla^2 f(\boldsymbol{x})$ at $\boldsymbol{x}$ is positive semidefinite for all $\boldsymbol{x} \in \mathbf{R}^n$
  - f is strictly convex iff its Hessian $\nabla^2 f(\boldsymbol{x})$ at $\boldsymbol{x}$ is positive definite for all $\boldsymbol{x} \in \mathbf{R}^n$

Note: a matrix is

- positive semidefinite if all its eigenvalues are real non-negative
- positive definite if all its eigenvalues are real and greater than zero

Smooth functions

- A function f is said to be β-smooth if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n$

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq \beta \|\boldsymbol{x} - \boldsymbol{y}\|$$

- Example: quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x}$

$$\nabla f(\boldsymbol{x}) = \boldsymbol{H}\boldsymbol{x}$$

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| = \|\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{y})\| \leq \underbrace{\|\boldsymbol{H}\|}_{\beta}\|\boldsymbol{x} - \boldsymbol{y}\|$$

Smooth functions (cont'd)

- If f is a β-smooth function, then [Bubeck L 3.4 or Nesterov L 1.2.3]

  (S0) $|f(\boldsymbol{y}) - f(\boldsymbol{x}) - \nabla f(\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{y} - x)| \leq \frac{\beta}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2$

- Equivalent conditions for f convex and β-smooth [Nesterov Thm 2.1.5]:

- (S1) $0 \leq f(\boldsymbol{y}) - f(\boldsymbol{x}) - \nabla f(\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) \leq \frac{\beta}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2$

- (S2) $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2\beta}\|\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x})\|^2$

- (S3) $\big(\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x})\big)^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) \geq \frac{1}{\beta}\|\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x})\|^2$

Gradient descent: smooth convex functions

- Consider gradient descent algorithm with a constant step size:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta \nabla f(\boldsymbol{x}^{(t)})$$

- Thm: For any β-smooth convex function f, the gradient descent algorithm with constant step size $0 < \eta < 2/\beta$ converges to a global minimizer of f

  Stronger statement: for any initial point $\boldsymbol{x}^{(0)}$ and a globally optimum point $\boldsymbol{x}^*$ such that $\left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\right\| \leq R$, it holds

$$f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*) \leq \underbrace{\frac{2R^2}{2R^2 + \left(f(\boldsymbol{x}^{(0)}) - f(\boldsymbol{x}^*)\right)\eta(2-\beta\eta)t}}_{= o\left(\frac{1}{t}\right)} \left(f\left(\boldsymbol{x}^{(0)}\right) - f(\boldsymbol{x}^*)\right)$$

Proof sketch

- Claim 1: If $0 < \eta < 2/\beta$, then the Euclidean distance between $\boldsymbol{x}^{(t)}$ and $\boldsymbol{x}^*$ decreases with the number of iterations t, i.e.

$$\left\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^*\right\| < \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\| \text{ whenever } \left\|\nabla f(\boldsymbol{x}^{(t)})\right\| \neq 0$$

- $$\left\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^*\right\|^2 = \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^* - \eta \nabla f(\boldsymbol{x}^{(t)})\right\|^2$$

$$= \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 - 2\eta \nabla f\left(\boldsymbol{x}^{(t)}\right)^{\mathsf{T}}\left(\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right) + \eta^2 \left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

$$\leq \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 - 2\eta \frac{1}{\beta} \left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2 + \eta^2 \left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

$$= \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 - \eta\left(\frac{2}{\beta} - \eta\right) \left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

Proof sketch (cont'd)

- Claim 2: since f is β-smooth, we have

$$f\left(\boldsymbol{x}^{(t+1)}\right) \leq f\left(\boldsymbol{x}^{(t)}\right) + \nabla f\left(\boldsymbol{x}^{(t)}\right)^{\mathsf{T}}\left(\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^{(t)}\right) + \eta^2\left\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^{(t)}\right\|^2$$

$$= f\left(\boldsymbol{x}^{(t)}\right) - \eta\left(\frac{2}{\beta} - \eta\right)\left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

- Claim 3: since f is convex , we have

$$f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*) \leq \nabla f\left(\boldsymbol{x}^{(t)}\right)^{\mathsf{T}}\left(\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right) \leq \left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|\left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|$$

$$\leq R\left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|$$

Proof sketch (cont'd)

- By Claim 2 and Claim 3, we have

$$f\left(\boldsymbol{x}^{(t+1)}\right) - f(\boldsymbol{x}^*) \leq f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*) - \underbrace{\frac{1}{R^2}\eta\left(\frac{2}{\beta} - \eta\right)}_{C}\left(f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*)\right)^2$$

- But this is equivalent to

$$\frac{1}{f\left(\boldsymbol{x}^{(t+1)}\right) - f(\boldsymbol{x}^*)} \geq \frac{1}{f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*)} + C\,\frac{f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*)}{f\left(\boldsymbol{x}^{(t+1)}\right) - f(\boldsymbol{x}^*)}$$

- Since by Claim 2, $f\left(\boldsymbol{x}^{(t+1)}\right) - f(\boldsymbol{x}^*) \leq f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*)$, it follows

$$\frac{1}{f\left(\boldsymbol{x}^{(t+1)}\right) - f(\boldsymbol{x}^*)} \geq \frac{1}{f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*)} + C$$

$$\Rightarrow \quad \frac{1}{f\left(\boldsymbol{x}^{(t+1)}\right) - f(\boldsymbol{x}^*)} \geq \frac{1}{f\left(\boldsymbol{x}^{(0)}\right) - f(\boldsymbol{x}^*)} + Ct \quad \Leftrightarrow \quad \text{claim of the theorem}$$

Strongly convex convex functions

- A function f is said to be α-strongly convex if for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^n$

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\alpha}{2}\|\boldsymbol{y} - \boldsymbol{x}\|^2$$

- If f is twice differentiable than for all $\boldsymbol{x}$ all the eigenvalues of the Hessian matrix $\nabla^2 f(\boldsymbol{x})$ have to be larger than or equal to α

Gradient descent: smooth and strongly convex

- Thm. If f is a α-strongly and β-smooth convex function, then for gradient descent algorithm with step size $\eta = \frac{2}{\alpha+\beta}$ and $\left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\right\| \leq R$, we have

$$f\left(\boldsymbol{x}^{(t)}\right) - f(\boldsymbol{x}^*) \leq R^2 \frac{\beta}{2} e^{-\frac{4}{\kappa+1}t}$$

  where $\kappa = \beta/\alpha$

- Think of κ as of the condition number of the Hessian matrix $\nabla^2 f(\boldsymbol{x})$

Proof sketch

- Claim 1: if f is α-strongly convex and β-smooth convex function, then

$$\big(\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x})\big)^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) \geq \frac{\alpha\beta}{\alpha + \beta}\|\boldsymbol{y} - \boldsymbol{x}\|^2 + \frac{1}{\alpha + \beta}\|\nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x})\|^2$$

- The claims follows from:
  - $\phi(\boldsymbol{x}) := f(\boldsymbol{x}) - \frac{\alpha}{2}\|\boldsymbol{x}\|^2$ is a convex function
  - $\phi(\boldsymbol{x})$ is $(\beta - \alpha)$-smooth, thus

$$\big(\nabla \phi(\boldsymbol{y}) - \nabla \phi(\boldsymbol{x})\big)^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{x}) \geq \frac{1}{\beta - \alpha}\|\nabla \phi(\boldsymbol{y}) - \nabla \phi(\boldsymbol{x})\|^2$$

  which yields the claim by straightforward calculus

Proof sketch (cont'd)

- By β-smoothness of f, we have $f\left(\boldsymbol{x}^{(t)}\right) - f\left(\boldsymbol{x}^*\right) \leq \frac{\beta}{2}\left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2$

- By Claim 1:

$$\left\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^*\right\|^2 = \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^* - \eta\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

$$= \left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 - 2\eta\nabla f\left(\boldsymbol{x}^{(t)}\right)^{\top}\left(\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right) + \eta^2\left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

$$\leq \left(1 - 2\frac{\eta\alpha\beta}{\alpha+\beta}\right)\left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 + \left(\eta^2 - 2\frac{\eta}{\alpha+\beta}\right)\left\|\nabla f\left(\boldsymbol{x}^{(t)}\right)\right\|^2$$

$$= \left(\frac{\kappa-1}{\kappa+1}\right)^2\left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 \leq e^{-\frac{4}{\kappa+1}}\left\|\boldsymbol{x}^{(t)} - \boldsymbol{x}^*\right\|^2 \leq e^{-\frac{4}{\kappa+1}(t+1)}\left\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\right\|^2$$

Scalability issues of gradient descent

- Scalability issue: computing the gradient vector $\nabla f(\boldsymbol{w})$ requires one pass through all training data points, which is expensive !

$$\nabla f(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{w}} \ell\left(\boldsymbol{y}_i, h_{\boldsymbol{w}}(\boldsymbol{x}_i)\right) + \lambda \nabla \phi(\boldsymbol{w})$$

  Solution: estimate the gradient vector (stochastic gradient descent)

- Gradient vector: how to compute the gradient for a multi-layer neural network?

  Solution: use the chain rule (backpropagation algorithm)

Stochastic gradient descent

- Stochastic gradient descent algorithm evaluates the gradient vector of the loss function for a sample of the training example

- Stochastic gradient: for a sample training example $(\boldsymbol{x}_s, \boldsymbol{y}_s)$ compute

$$\widehat{\nabla} f(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \ell\big(\boldsymbol{y}_s, h_{\boldsymbol{w}}(\boldsymbol{x}_s)\big) + \lambda \nabla \phi(\boldsymbol{w})$$

- For a random sample of a training example, stochastic gradient is an unbiased estimator of the true gradient $\nabla f(\boldsymbol{w})$

Robbins and Monro, A stochastic approximation method, Ann. Math. Stat., Vol 22, No 3, 400-407, 1951

Stochastic gradient descent algorithm

- **Initialization**: $t = 1$, step size sequence $\eta^{(t)}$, initial parameter vector $\boldsymbol{w}$

while stopping criterion is not met **do**

sample a training example $(\boldsymbol{x}_{s_t}, \boldsymbol{y}_{s_t})$

compute stochastic gradient vector:

$$\widehat{\boldsymbol{\nabla}} f(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \ell\left(\boldsymbol{y}_{s_t}, h_{\boldsymbol{w}}\left(\boldsymbol{x}_{s_t}\right)\right) + \lambda \nabla \phi(\boldsymbol{w})$$

apply update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta^{(t)} \widehat{\nabla} f(\boldsymbol{w})$

$t \leftarrow t + 1$

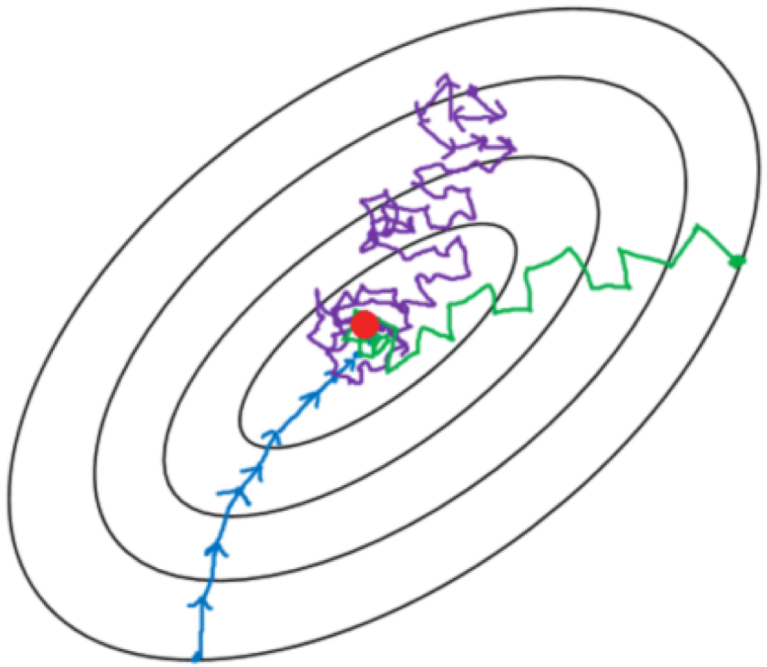
end while

Terminology and variations

- Batch gradient descent: a variation of gradient descent algorithm that calculates the gradient for each example in the training dataset, but only updates the parameter vector after all training examples have been evaluated
  - One pass through the entire training dataset is called a training epoch

- Mini-batch gradient descent: splits the training datasets into small batches that are used to calculate the gradient vector and update the parameter vector
  - Implementations may choose to sum the gradient over the mini-batches or take the average of the gradients (variance reduction)

- Pros for small batches: smaller memory footprint, shown to improve generalization performance
- Pros for large batches: parallelization



Gradient descent

Mini-batch gradient descent

Stochastic gradient descent

SGD convergence for smooth convex functions

Thm. If

- W is a convex set
- f is a convex β-smooth function
- $\boldsymbol{w}^{(0)} \in W$ and $R = \sup_{\boldsymbol{w} \in W} \left\| \boldsymbol{w} - \boldsymbol{w}^{(0)} \right\|$.
- stochastic gradient vector $\widehat{\nabla} f(\boldsymbol{w})$ satisfies $\mathbf{E}\left[\left\| \widehat{\nabla} f(\boldsymbol{w}) - \nabla f(\boldsymbol{w}) \right\|^2 \right] \leq \sigma^2$ for all $\boldsymbol{w}$
- step size: $\eta = 1/(\beta + \sigma/R\sqrt{t/2})$.

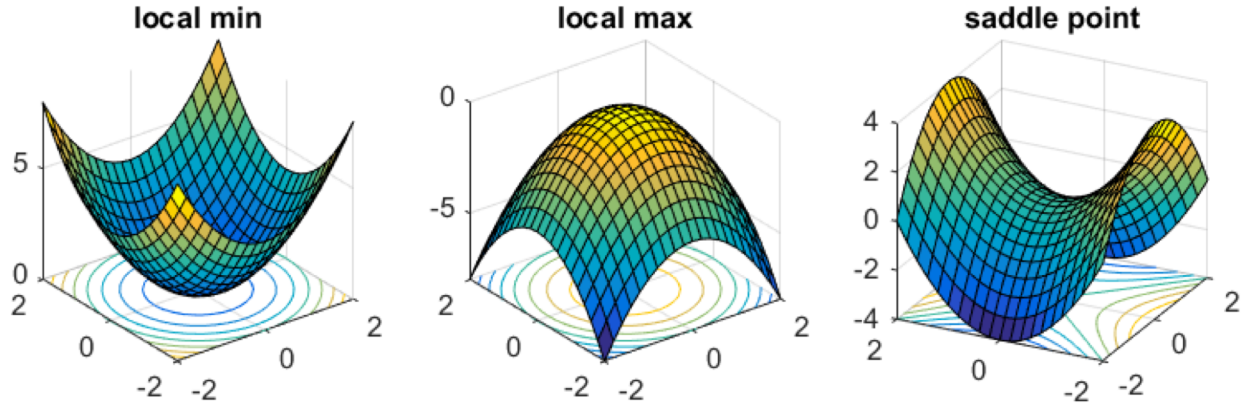
then,

$$\mathbf{E}\left[f\left(\frac{1}{t} \sum_{s=1}^{t} \boldsymbol{w}^{(s)} \right) \right] - f(\boldsymbol{w}^*) \leq \underbrace{\sqrt{2}\sigma R \frac{1}{\sqrt{t}} + \beta R^2 \frac{1}{t}}_{= O\left(\frac{1}{\sqrt{t}}\right)}$$

(Bubeck Theorem 6.3)

Non-convex loss functions

- Non-convex functions are neither convex nor concave
- Non-convex functions may have
  - Multiple local minima
  - Local minima that are globally suboptimal
  - Saddle points



- Non-convex functions are much more challenging for optimization
- Loss functions of neural networks are typically non-convex functions

Strict saddle property

- Suppose $f: \mathbf{R}^n \to \mathbf{R}$ is a twice differentiable function

- A point $\boldsymbol{x}^* \in \mathbf{R}^n$ is a critical point of f if $\nabla f(\boldsymbol{x}^*) = 0$
- Local minimum: a critical point $\boldsymbol{x}^*$ is a local minimum if there exists a neighborhood X around $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ for **all** $\boldsymbol{x} \in X$
- Saddle point: a critical point $\boldsymbol{x}^*$ is a saddle point if for each neighborhood set X around $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*) \leq f(\boldsymbol{y})$ for **some** $\boldsymbol{x}, \boldsymbol{y} \in X$

- A function f satisfies the strict saddle property if each critical point $\boldsymbol{x}^*$ is either:
  - A local minimizer, or
  - A strict saddle, i.e. $\nabla^2 f(\boldsymbol{x}^*)$ has at least one negative eigenvalue

Importance of random initialization

- Thm. If $f: \mathbf{R}^n \to \mathbf{R}$ is a twice-differentiable function that satisfies the strict saddle property, then gradient descent algorithm with a random initialization and sufficiently small constant step size **either**:

  (a) converges to a local minimizer, or

  (b) diverges

  almost surely

- Moral: gradient descent escapes (strict) saddle points !

[Lee, Simchowitz, Jordan, and Recht 2016]

Intuition by quadratic function example

- Consider $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{H}\boldsymbol{x}$ where $\boldsymbol{H} = \mathbf{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ such that for $1 \le k < n$: $\lambda_1, \dots, \lambda_k > 0$ and $\lambda_{k+1}, \dots, \lambda_n < 0$

- For such function f, $\boldsymbol{x}^* = \mathbf{0}$ is the unique critical point (strict saddle)

- Consider the gradient descent algorithm

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta \boldsymbol{H}\boldsymbol{x}^{(t)}$$

  with step size $0 < \eta < 1/\beta$ where $\beta = \max_i |\lambda_i|$

- Q: what is the limit point of the gradient descent algorithm for a given $\boldsymbol{x}^{(0)}$?

Intuition by quadratic function example (cont'd)

- Since $\boldsymbol{x}^{(t+1)} = (I - \eta \boldsymbol{H})\boldsymbol{x}^{(t)}$ and $I - \eta \boldsymbol{H} = \mathbf{diag}(1 - \eta\lambda_1, \dots, 1 - \eta\lambda_n)$ we have

$$\boldsymbol{x}^{(t)} = \mathbf{diag}\big((1 - \eta\lambda_1)^t, \dots, (1 - \eta\lambda_n)^t\big)\boldsymbol{x}^{(0)}$$

  i.e.

$$\boldsymbol{x}^{(t)} = \textstyle\sum_{i=1}^{n}(1 - \eta\lambda_i)^t\big(\boldsymbol{e}_i^\top \boldsymbol{x}^{(0)}\big)\boldsymbol{e}_i$$

- Note

$$1 - \eta\lambda_i \begin{cases} < 1 & \text{for } i = 1,2, \dots, k \\ > 1 & \text{for } i = k+1, \dots, n \end{cases}$$

If $\boldsymbol{x}^{(0)}$ is a random point,
$\mathbf{Pr}\big[\boldsymbol{x}^{(0)} \in \mathrm{span}(\boldsymbol{e}_1, \dots, \boldsymbol{e}_k)\big] = 0$,
thus $\boldsymbol{x}^{(t)}$ "escapes the saddle point"

$\Uparrow$

- Hence, if $\boldsymbol{x}^{(0)} \in \mathrm{span}(\boldsymbol{e}_1, \dots, \boldsymbol{e}_k)$, then $\lim_{t\to\infty} \boldsymbol{x}^{(t)} = \mathbf{0}$, otherwise, $\lim_{t\to\infty} \left\|\boldsymbol{x}^{(t)}\right\| = \infty$

Note: e_i denotes a n-dimensional vector with the i-th element equal to 1 and other elements equal to 0

Optimization landscape

- Optimization landscape refers to characterizing critical points of different classes of neural networks for specific loss functions

- Good set of properties:
  - Strict saddle property, and
  - No poor local minima: all local minima are globally optimal

- Good set of properties shown to hold, e.g., for feedforward networks with linear activation functions and squared error loss function
  - Recent results e.g. by Kawaguchi (2016)
  - Proved a 1989 conjecture expressed by Baldi and Hornik

- Much ongoing research in this direction

Acceleration by momentum

Polyak's momentum method [1969]

- Polyak's momentum method iterative updates:

$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} - \eta^{(t)} \nabla f\left(\boldsymbol{w}^{(t)}\right) \qquad \text{(velocity)}$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{v}^{(t+1)} \qquad \text{(parameter)}$$

  or, equivalently

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta^{(t)} \nabla f\left(\boldsymbol{w}^{(t)}\right) + \gamma\left(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}\right)$$

  where $\gamma \in [0,1]$ is the momentum parameter

- If $\gamma = 0$ this is standard gradient descent update
- The larger the γ the velocity accumulates more information about past gradients
- Typical values γ = 0.9 or 0.95 or 0.99
- Momentum may smooth updates, enhance stability and convergence speed

Nesterov's accelerated gradient descent [1983]

- Initial values: arbitrary $\boldsymbol{w}^{(0)} = \boldsymbol{v}^{(0)}$

- Update at iteration t:

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta^{(t)} \nabla f(\boldsymbol{w}^{(t)})$$

$$\boldsymbol{w}^{(t+1)} = \left(1 - \gamma^{(t)}\right)\boldsymbol{v}^{(t+1)} + \gamma^{(t)}\boldsymbol{v}^{(t)}$$

where

$$\gamma^{(t)} = \frac{1-\rho^{(t)}}{\rho^{(t+1)}} \text{ and } \rho^{(0)} = 0, \rho^{(t+1)} = \frac{1+\sqrt{1+4\left(\rho^{(t)}\right)^2}}{2}$$

Sutskever et al's momentum [2013]

- Update equations:

$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} - \eta^{(t)} \nabla f\left(\boldsymbol{w}^{(t)} + \gamma \boldsymbol{v}^{(t)}\right)$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{v}^{(t+1)}$$

- Or, equivalently,

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta^{(t)} \nabla f\left(\boldsymbol{w}^{(t)} + \gamma\left(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}\right)\right) + \gamma\left(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}\right)$$

- Inspired by Nesterov's accelerated gradient descent
- Different than Polyak's momentum in evaluating the gradient at $\boldsymbol{w}^{(t)} + \gamma \boldsymbol{v}^{(t)}$
- This additional term may increase stability, and speed in ill-conditioned problems

Deriving Sutskever et al's momentum

- Nesterov's update equations:
  - (N1): $\boldsymbol{v}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta^{(t)} \nabla f\left(\boldsymbol{w}^{(t)}\right)$
  - (N2): $\boldsymbol{w}^{(t+1)} = \left(1 - \gamma^{(t)}\right)\boldsymbol{v}^{(t+1)} + \gamma^{(t)}\boldsymbol{v}^{(t)}$

- Using $\boldsymbol{u}^{(t+1)} := \boldsymbol{v}^{(t+1)} - \boldsymbol{v}^{(t)}$ and $\tilde{\gamma}^{(t+1)} := -\gamma^{(t)}$, we have

$$\text{(N2)} \Leftrightarrow \boldsymbol{w}^{(t+1)} = \boldsymbol{v}^{(t+1)} + \tilde{\gamma}^{(t+1)}\boldsymbol{u}^{(t+1)}$$

$$\text{(N1)} \Leftrightarrow \boldsymbol{u}^{(t+1)} = \tilde{\gamma}^{(t)}\boldsymbol{u}^{(t)} - \eta^{(t)} \nabla f\left(\boldsymbol{v}^{(t)} + \tilde{\gamma}^{(t)}\boldsymbol{u}^{(t)}\right)$$

$$\boldsymbol{v}^{(t+1)} = \boldsymbol{v}^{(t)} + \boldsymbol{u}^{(t+1)}$$

To obtain Sutskever et al's update equations, use change of variables:

$\boldsymbol{v} \to \boldsymbol{w}$

$\boldsymbol{u} \to \boldsymbol{v}$

$\tilde{\gamma} \to \gamma$

Quadratic function example

- Suppose $f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{H}\boldsymbol{w}$ where $\boldsymbol{H}$ is a real-symmetric positive definite matrix
  - Real-symmetric matrix positive definite ⇔ all eigenvalues positive

- Minimizer of f: $\boldsymbol{w}^* = \boldsymbol{0}$

- Gradient descent algorithm:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta^{(t)}\boldsymbol{H}\boldsymbol{w}^{(t)}$$

- For a constant step size η,

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{A}\boldsymbol{w}^{(t)}$$

where $\boldsymbol{A} = \boldsymbol{I} - \eta\boldsymbol{H}$

Stability of linear systems

- A linear dynamical system

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{A}\boldsymbol{x}^{(t)}$$

  - Strictly stable: if for all initial values, $\lim_{t\to\infty} \boldsymbol{x}^{(t)} = \boldsymbol{0}$
  - Unstable: if there exists an initial value for which $\lim_{t\to\infty} \left\|\boldsymbol{x}^{(t)}\right\| = \infty$
  - Marginally stable: it is neither stable nor unstable, i.e. the limit point remains bounded but it is not necessarily $\boldsymbol{0}$

- Strictly stable $\Leftrightarrow |\lambda_i(\boldsymbol{A})| < 1$ for each eigenvalue $\lambda_i(\boldsymbol{A})$ of $\boldsymbol{A}$

Quadratic function example (cont'd)

- Assumption: $\eta \leq 1/\lambda_{max}(\boldsymbol{H})$
- This ensures $\|\eta \boldsymbol{H}\| < 2$ which with $\boldsymbol{H}$ positive definite ensures strict stability
- Let $\kappa(\boldsymbol{H}) = \lambda_{max}(\boldsymbol{H})/\lambda_{min}(\boldsymbol{H})$ (condition number)

- Gradient descent convergence rate:

$$\left\|\boldsymbol{w}^{(t+1)}\right\| \leq \|\boldsymbol{A}\|\left\|\boldsymbol{w}^{(t)}\right\| = \left(1 - \frac{1}{\kappa(\boldsymbol{H})}\right)\left\|\boldsymbol{w}^{(t)}\right\|$$

$$\left\|\boldsymbol{w}^{(t)}\right\| \leq \left(1 - \frac{1}{\kappa(\boldsymbol{H})}\right)^{t}\left\|\boldsymbol{w}^{(0)}\right\|$$

- Linear convergence rate with rate: $1 - 1/\kappa(\boldsymbol{H})$

Quadratic function example (cont'd)

- Consider now the accelerated algorithm with momentum:

$$\boldsymbol{v}^{(t+1)} = \gamma \boldsymbol{v}^{(t)} - \eta \nabla f\left(\boldsymbol{w}^{(t)} + \gamma \boldsymbol{v}^{(t)}\right)$$
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{v}^{(t)}$$

i.e. $$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla f\left(\boldsymbol{w}^{(t)} + \gamma\left(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)}\right)\right) + \gamma(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)})$$

- For the quadratic function and step size $\eta = 1/\lambda_{max}(\boldsymbol{H})$, we have

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{A}\boldsymbol{w}^{(t)} + \underbrace{\gamma \boldsymbol{A}(\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(t-1)})}_{\text{momentum term}}$$

Quadratic function example (cont'd)

- A linear system: $\begin{pmatrix} \boldsymbol{w}^{(t+1)} \\ \boldsymbol{w}^{(t)} \end{pmatrix} = \boldsymbol{B} \begin{pmatrix} \boldsymbol{w}^{(t)} \\ \boldsymbol{w}^{(t-1)} \end{pmatrix}$ where $\boldsymbol{B} = \begin{pmatrix} (1+\gamma)\boldsymbol{A} & -\gamma\boldsymbol{A} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}$

- Linear convergence:

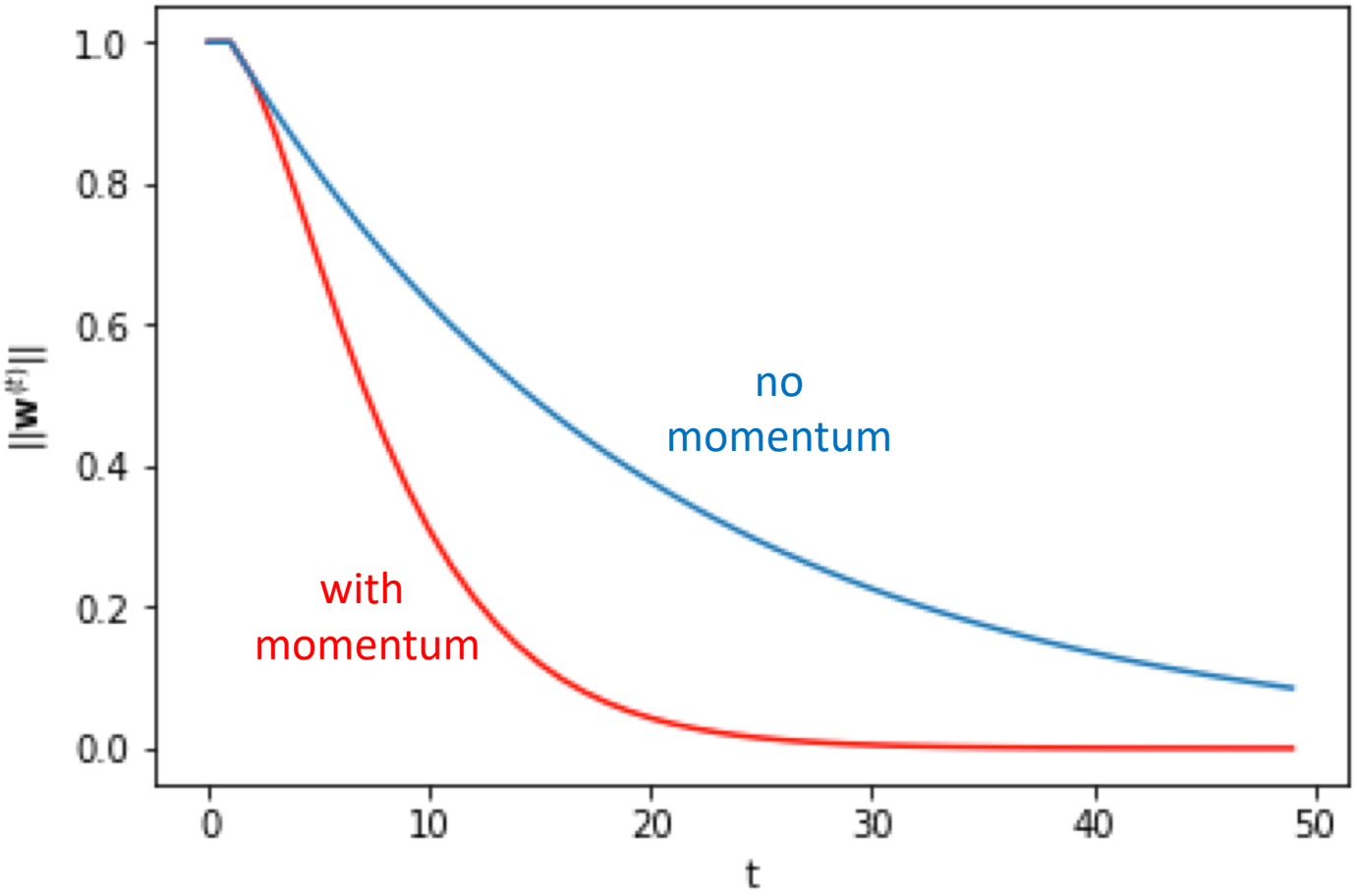
$$\left\| \begin{pmatrix} \boldsymbol{w}^{(t+1)} \\ \boldsymbol{w}^{(t)} \end{pmatrix} \right\| \leq \|\boldsymbol{B}\| \left\| \begin{pmatrix} \boldsymbol{w}^{(t)} \\ \boldsymbol{w}^{(t-1)} \end{pmatrix} \right\|$$

with rate

$$\|\boldsymbol{B}\| = 1 - \sqrt{1/\kappa(\boldsymbol{H})}$$ (Exercise: show this)

by choosing $\gamma = (1 - \sqrt{1/\kappa(\boldsymbol{H})})/(1 + \sqrt{1/\kappa(\boldsymbol{H})})$

Numerical example



- $\boldsymbol{H} = (0.05), \boldsymbol{w}^{(0)} = 1$

Momentum exercise solution

- Need to show that for $\boldsymbol{B} = \begin{pmatrix} (1+\gamma)\boldsymbol{A} & -\gamma\boldsymbol{A} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix}$, $\|\boldsymbol{B}\| = 1 - \sqrt{1/\kappa(\boldsymbol{H})}$
- Elementary fact: $\det\begin{pmatrix} \boldsymbol{U} & \boldsymbol{V} \\ \boldsymbol{W} & \boldsymbol{Z} \end{pmatrix} = \det(\boldsymbol{UZ} - \boldsymbol{VW})$ if $\boldsymbol{WZ} = \boldsymbol{ZW}$

$$\Rightarrow \det(\lambda\boldsymbol{I} - \boldsymbol{B}) = \det(\lambda^2\boldsymbol{I} - \big((1+\gamma)\lambda - \gamma\big)\boldsymbol{A})$$

- For $\gamma = \lambda/(2-\lambda)$ we have

$$\det(\lambda\boldsymbol{I} - \boldsymbol{B}) = \frac{\lambda}{2-\lambda}\det(\lambda(2-\lambda)\boldsymbol{I} - \boldsymbol{A})$$

- λ is an eigenvalue of $\boldsymbol{B}$ if $\lambda(2-\lambda) = \mu$ for some eigenvalue μ of $\widetilde{\boldsymbol{A}}$

$$\Rightarrow \|\boldsymbol{B}\|(2 - \|\boldsymbol{B}\|) = \|\boldsymbol{A}\| \Rightarrow \|\boldsymbol{B}\| = 1 - \sqrt{1 - \|\boldsymbol{A}\|} = 1 - \sqrt{1/\kappa(\boldsymbol{H})}$$

Convergence theorem

Thm. If

- f is a β-smooth convex function
- Step size $\eta = \frac{1}{\beta}$
- $\left\| \boldsymbol{w}^{(0)} - \boldsymbol{w}^* \right\| \leq R$

then, for the Nesterov's accelerated gradient descent algorithm

$$f\left(\boldsymbol{w}^{(t)}\right) - f(\boldsymbol{w}^*) \leq 2\beta R^2 \frac{1}{t^2}$$

Proof: e.g., see I'm a bandit ORF523

- Compare this bound with that for gradient descent with no acceleration
- Note: quadratic convergence rate does not hold for stochastic gradient descent

Convergence rates for classes of convex functions

f	Algorithm	Rate	# Iter	Cost/iter
non-smooth	center of gravity	$\exp\left(-\frac{t}{n}\right)$	$n \log\left(\frac{1}{\varepsilon}\right)$	1 ∇, 1 n-dim $\int$
non-smooth	ellipsoid method	$\frac{R}{r}\exp\left(-\frac{t}{n^2}\right)$	$n^2 \log\left(\frac{R}{r\varepsilon}\right)$	1 ∇, mat-vec $\times$
non-smooth	Vaidya	$\frac{Rn}{r}\exp\left(-\frac{t}{n}\right)$	$n \log\left(\frac{Rn}{r\varepsilon}\right)$	1 ∇, mat-mat $\times$
quadratic	CG	exact $\exp\left(-\frac{t}{\kappa}\right)$	n $\kappa \log\left(\frac{1}{\varepsilon}\right)$	1 ∇
non-smooth, Lipschitz	PGD	$RL/\sqrt{t}$	R^2L^2/ε^2	1 ∇, 1 proj.
smooth	PGD	$\beta R^2/t$	$\beta R^2/\varepsilon$	1 ∇, 1 proj.
smooth	AGD	$\beta R^2/t^2$	$R\sqrt{\beta/\varepsilon}$	1 ∇
smooth (any norm)	FW	$\beta R^2/t$	$\beta R^2/\varepsilon$	1 ∇, 1 LP
strong. conv., Lipschitz	PGD	$L^2/(\alpha t)$	$L^2/(\alpha\varepsilon)$	1 ∇ , 1 proj.
strong. conv., smooth	PGD	$R^2\exp\left(-\frac{t}{\kappa}\right)$	$\kappa \log\left(\frac{R^2}{\varepsilon}\right)$	1 ∇ , 1 proj.
strong. conv., smooth	AGD	$R^2\exp\left(-\frac{t}{\sqrt{\kappa}}\right)$	$\sqrt{\kappa}\log\left(\frac{R^2}{\varepsilon}\right)$	1 ∇
$f+g$, f smooth, g simple	FISTA	$\beta R^2/t^2$	$R\sqrt{\beta/\varepsilon}$	1 ∇ of f Prox of g
$\max_{y\in\mathcal{Y}} \varphi(x,y)$, φ smooth	SP-MP	$\beta R^2/t$	$\beta R^2/\varepsilon$	MD on $\mathcal{X}$ MD on $\mathcal{Y}$
linear, $\mathcal{X}$ with F ν-self-conc.	IPM	$\nu\exp\left(-\frac{t}{\sqrt{\nu}}\right)$	$\sqrt{\nu}\log\left(\frac{\nu}{\varepsilon}\right)$	Newton step on F
non-smooth	SGD	$BL/\sqrt{t}$	B^2L^2/ε^2	1 stoch. ∇, 1 proj.
non-smooth, strong. conv.	SGD	$B^2/(\alpha t)$	$B^2/(\alpha\varepsilon)$	1 stoch. ∇, 1 proj.
$f=\frac{1}{m}\sum f_i$ f_i smooth strong. conv.	SVRG	–	$(m+\kappa)\log\left(\frac{1}{\varepsilon}\right)$	1 stoch. ∇

[Bubeck]

Adaptive learning rates

Adaptive gradient (AdaGrad)

- Update equations:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\, \mathbf{diag}\left(\boldsymbol{Q}^{(t)}\right)^{-1/2} \nabla f_t(\boldsymbol{w}^{(t)})$$

where $\boldsymbol{Q}^{(t)} = \sum_{s=1}^{t} \nabla f_s\left(\boldsymbol{w}^{(s)}\right) \nabla f_s\left(\boldsymbol{w}^{(s)}\right)^{\mathsf{T}}$

- Learning rate adaptation:
  - high variation: suppress, low variation: enhance
- Disadvantage in non-stationary regimes:
  - all gradients weighted equally (no matter how far they were in the past)

[Duchi, Hazan and Singer, 2010]

RMSProp: Root Mean Square Propagation

- Update equations:
  - $\boldsymbol{q}^{(t+1)} = \beta \boldsymbol{q}^{(t)} + (1-\beta)\mathbf{diag}\left(\nabla f_t\left(\boldsymbol{w}^{(t)}\right)\nabla f_t\left(\boldsymbol{w}^{(t)}\right)^{\mathsf{T}}\right)$
  - $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\mathbf{diag}\left(1/\sqrt{q_1^{(t)}+\epsilon}, \ldots, 1/\sqrt{q_n^{(t)}+\epsilon}\right)\nabla f_t(\boldsymbol{w}^{(t)})$
- Similar to AdaGrad
  - But with exponentially moving average that puts more emphasis on more recent gradient vectors

[Hinton, 2012]

RMSProp a la Graves

- Update equations:

- $\boldsymbol{g}^{(t+1)} = \beta \boldsymbol{g}^{(t)} + (1-\beta)\nabla f_t(\boldsymbol{w}^{(t)})$
- $\boldsymbol{q}^{(t+1)} = \beta \boldsymbol{q}^{(t)} + (1-\beta)\mathbf{diag}\left(\nabla f_t\left(\boldsymbol{w}^{(t)}\right)\nabla f_t\left(\boldsymbol{w}^{(t)}\right)^{\mathsf{T}}\right)$

- $\boldsymbol{v}^{(t+1)} = \mu \boldsymbol{v}^{(t)} - \eta \mathbf{diag}\left(1/\sqrt{q_1^{(t)} - \left(g_1^{(t)}\right)^2 + \epsilon}, \ldots, 1/\sqrt{q_n^{(t)} - \left(g_n^{(t)}\right)^2 + \epsilon}\right)\nabla f_t(\boldsymbol{w}^{(t)})$
- $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \boldsymbol{v}^{(t+1)}$

[Graves, 2014]

Adaptive moment estimation (Adam)

- AdaGrad: works well with sparse gradients
- RMSProp: works well in non-stationary settings

- Adam key ideas:
  - Maintain exponential moving averages of gradient and its norm
  - Update proportional to $\text{(average gradient)} / \sqrt{\text{(average gradient norm)}}$

[Kingma and Ba, 2015]

ADAM pseudo-code

- Initialization: $\boldsymbol{g}^{(0)} = 0$, $\boldsymbol{q}^{(0)} = 0$
- For $t = 1,2, \dots, T$:

$$\boldsymbol{g}^{(t+1)} = \beta_1 \boldsymbol{g}^{(t)} + (1-\beta_1)\nabla f_t\left(\boldsymbol{w}^{(t)}\right)$$ // 1st moment

$$\boldsymbol{q}^{(t+1)} = \beta_2 \boldsymbol{q}^{(t)} + (1-\beta_2)\mathbf{diag}\left(\nabla f_t\left(\boldsymbol{w}^{(t)}\right)\nabla f_t\left(\boldsymbol{w}^{(t)}\right)^{\top}\right)$$ // 2nd moment

$$\widehat{\boldsymbol{g}}^{(t)} = \frac{1}{1-\beta_1^t}\boldsymbol{g}^{(t)}$$ // bias correction

$$\widehat{\boldsymbol{q}}^{(t)} = \frac{1}{1-\beta_2^t}\boldsymbol{q}^{(t)}$$ // bias correction

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\ \mathbf{diag}\left(1/\sqrt{\hat{q}_1^{(t)} + \epsilon}, \dots, 1/\sqrt{\hat{q}_n^{(t)} + \epsilon}\right)\widehat{\boldsymbol{g}}^{(t)}$$ // param update

Hyperparameters

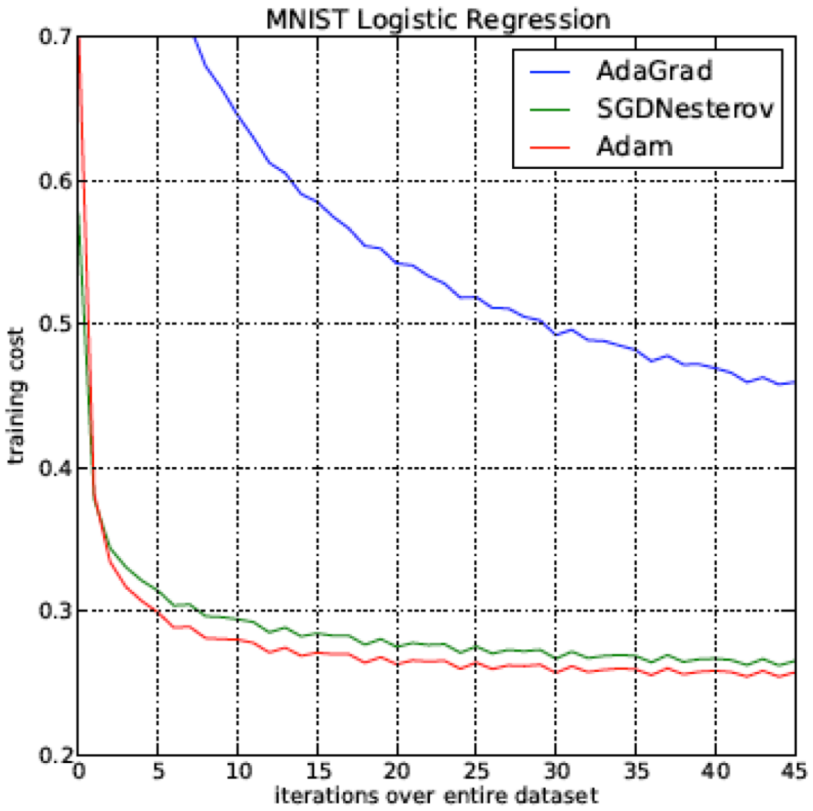
- η step size
  - Typical value 0.001

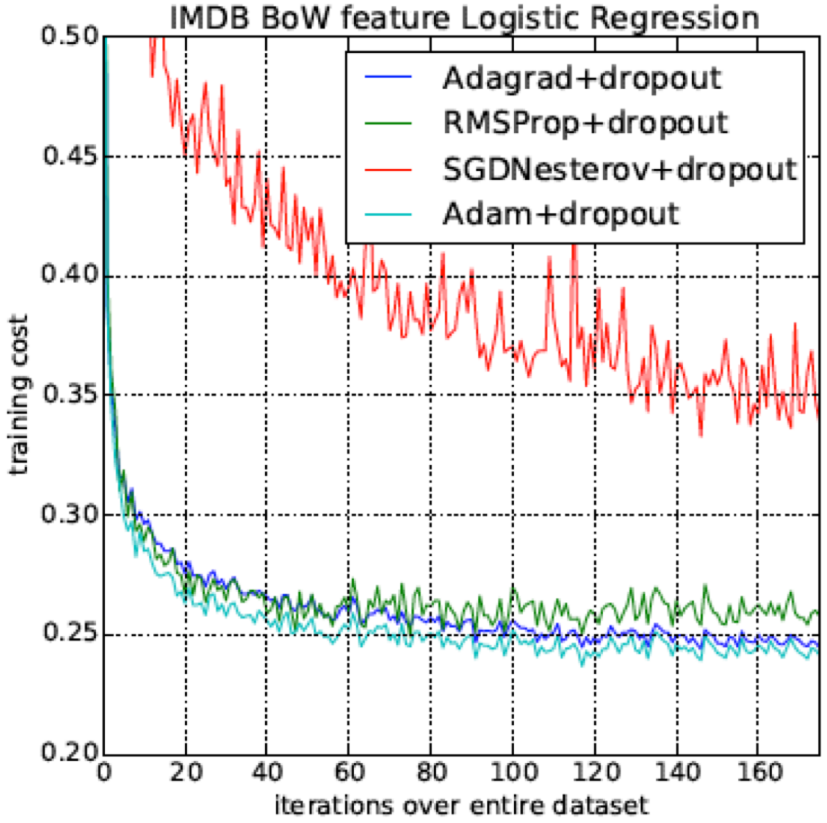
- $\beta_1 \in [0,1)$ exponential decay rate for the 1st moment estimator
  - Typical value 0.9

- $\beta_2 \in [0,1)$ exponential decay rate for the 2nd moment estimator
  - Typical value 0.999

- $\epsilon > 0$ parameters used to rule out division by zero
  - Typical value 10^{-8}

Numerical results





Parameter vector updates: scale invariance

- Assume $\epsilon = 0$ (parameter only to avoid division by 0)
- The parameter vector is updated as $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \boldsymbol{\Delta}^{(t+1)}$ where

$$\boldsymbol{\Delta}^{(t)} = \eta \mathbf{diag}\left(1/\sqrt{\hat{q}_1^{(t)}}, \ldots, 1/\sqrt{\hat{q}_n^{(t)}}\right)\hat{\boldsymbol{g}}^{(t)}$$

- The 1st and 2nd moment estimates satisfy:

$$\boldsymbol{g}^{(t+1)} = \beta_1^t \boldsymbol{g}^{(0)} + (1-\beta_1)\sum_{s=1}^{t} \beta_1^{t-s} \nabla f_s\left(\boldsymbol{w}^{(s)}\right)$$

$$\boldsymbol{q}^{(t+1)} = \beta_2^t \boldsymbol{q}^{(0)} + (1-\beta_2)\sum_{s=1}^{t} \beta_2^{t-s}\, \mathbf{diag}\left(\nabla f_s\left(\boldsymbol{w}^{(s)}\right)\nabla f_s\left(\boldsymbol{w}^{(s)}\right)^{\mathsf{T}}\right)$$

- Since $\boldsymbol{g}^{(0)} = \mathbf{0}$ and $\boldsymbol{q}^{(0)} = \mathbf{0}$, if function f is replaced with function cf for any constant $c > 0$, then $\boldsymbol{g}^{(t)}$ and $\boldsymbol{q}^{(t)}$ are replaced with $c\boldsymbol{g}^{(t)}$ and $c^2\boldsymbol{q}^{(t)}$

$\Rightarrow \boldsymbol{\Delta}^{(t)}$ remains unchanged (scale invariance)

Parameter vector updates: bounded increments

Claim: for every $t = 1,2, \ldots$

$$\|\boldsymbol{\Delta}^{(t)}\|_\infty \leq \begin{cases} \eta \dfrac{1-\beta_1}{\sqrt{1-\beta_2}} & \text{if } 1-\beta_1 \geq \sqrt{1-\beta_2} \\ \eta & \text{otherwise} \end{cases}$$

- Exercise: check this

Bias correction factor

- Facts:

$$\boldsymbol{g}^{(t+1)} = \beta_1^t \boldsymbol{g}^{(0)} + (1-\beta_1)\sum_{s=1}^{t} \beta_1^{t-s} \nabla f_t\left(\boldsymbol{w}^{(s)}\right)$$

and $\boldsymbol{q}^{(t+1)} = \beta_2^t \boldsymbol{q}^{(0)} + (1-\beta_2)\sum_{s=1}^{t} \beta_2^{t-s}\, \mathbf{diag}\left(\nabla f_s\left(\boldsymbol{w}^{(s)}\right)\nabla f_t\left(\boldsymbol{w}^{(s)}\right)^{\mathsf{T}}\right)$

- Since $\boldsymbol{g}^{(0)} = 0$ we have

$$\mathbf{E}\left[\boldsymbol{g}^{(t)}\right] = \mathbf{E}\left[\nabla f_1\left(\boldsymbol{w}^{(s)}\right)\right](1-\beta_1)\sum_{s=1}^{t} \beta_1^{t-s} = \mathbf{E}\left[\nabla f_t\left(\boldsymbol{w}^{(s)}\right)\right](1-\beta_1^t)$$

which explains the bias correction factor

- Analogously, we have $\mathbf{E}\left[\boldsymbol{q}^{(t)}\right] = \mathbf{E}\left[\mathbf{diag}\left(\nabla f_1\left(\boldsymbol{w}^{(s)}\right)\nabla f_1\left(\boldsymbol{w}^{(s)}\right)^{\mathsf{T}}\right)\right](1-\beta_2^t)$

Convergence analysis: regret bound

- Regret at time step T defined by:

$$R(T) = \underbrace{\sum_{t=1}^{T} f_t\left(\boldsymbol{w}^{(t)}\right)}_{\text{cumulative loss}} - \underbrace{\sum_{t=1}^{T} f_t(\boldsymbol{w}^*)}_{\text{optimal cumulative loss in hindsight}}$$

where

$$\boldsymbol{w}^* = \text{argmin}_{\boldsymbol{w}} \sum_{t=1}^{T} f_t(\boldsymbol{w})$$

Regret bound theorem

Thm. Assumptions:

- $f: \mathbf{R}^n \to \mathbf{R}$ convex function and $\|f(\boldsymbol{w})\|_p \leq \mu_p$ for all $\boldsymbol{w} \in \mathbf{R}^n$ for $p = 2, \infty$
- $\left\|\boldsymbol{w}^{(t)} - \boldsymbol{w}^{(s)}\right\|_p \leq \delta_p$ for all $s, t \in \{1, \ldots, T\}$ for $p = 2, \infty$
- $\gamma = \beta_1^2 / \sqrt{\beta_2} < 1$
- Step size $\eta^{(t)} = \frac{\eta}{\sqrt{t}}$
- Exponential discounting for the first-moment $\beta_1^{(t)} = \beta_1 \lambda^{t-1}$ for $\lambda \in (0,1)$

Then $R(T) \leq \frac{\delta_2}{2\eta(1-\beta_1)} \sum_{i=1}^{n} \sqrt{T \hat{q}_i^{(T)}} +$

$$+ \frac{\eta(1+\beta_1)\mu_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{n} \left\| \left(\frac{\partial}{\partial w_i} f_1\left(\boldsymbol{w}^{(1)}\right), \ldots, \frac{\partial}{\partial w_i} f_T\left(\boldsymbol{w}^{(T)}\right) \right) \right\| \quad = O(\sqrt{T})$$

$$+ n \frac{\delta_\infty^2 \mu_\infty}{2\eta(1-\beta_1)(1-\lambda)^2}$$

Backpropagation

What is backpropagation?

- Backpropagation: a method for computing the gradient vector of a loss function for a feedforward neural network for given weights
  - Introduced by Rumelhart, Hinton and Williams in 1986

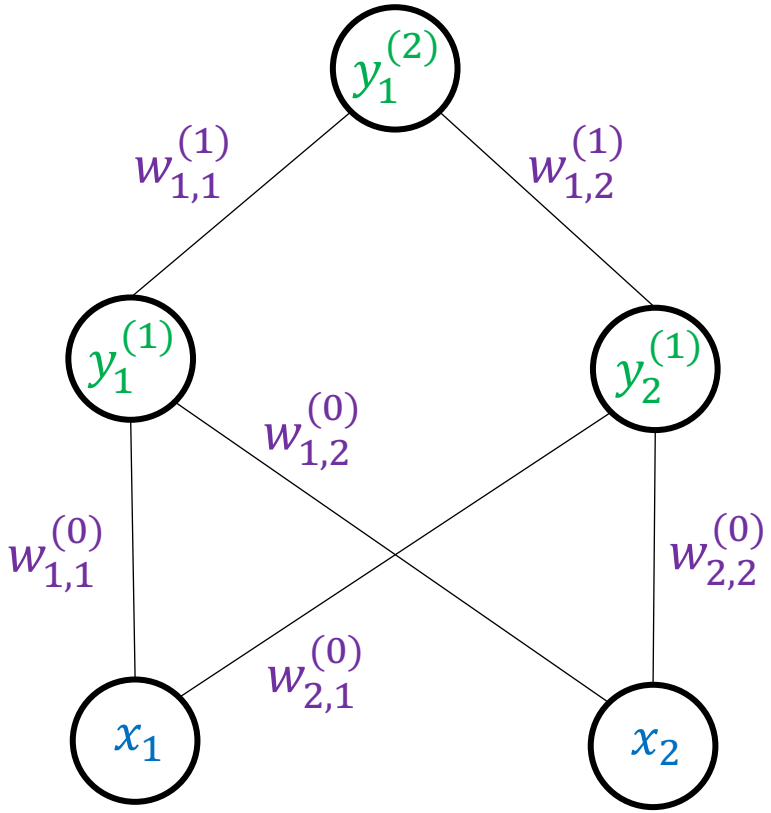
- Commonly used for training neural networks

- Uses a forward pass and a backward pass through the network

- Uses the chain rule: $\frac{\partial}{\partial x} f\big(g(x)\big) = \frac{\partial}{\partial g} f\big(g(x)\big) \frac{\partial}{\partial x} g(x)$

- Commutation efficiency: reuses computation

Backpropagation for a simple example



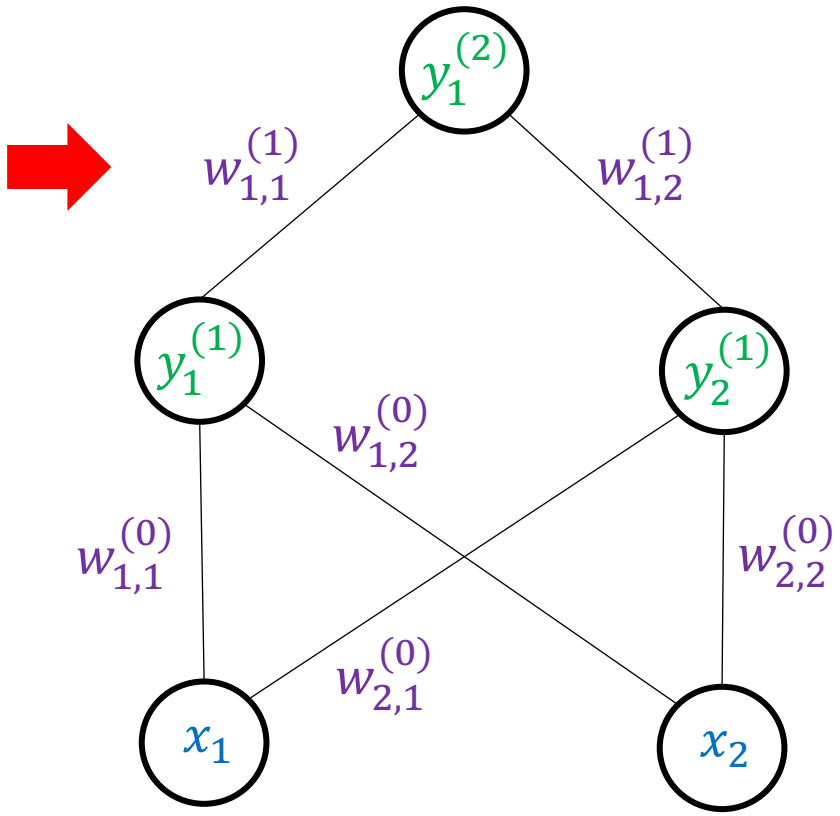
$$h_{\boldsymbol{w}}(\boldsymbol{x}) = y_1^{(2)} = a\left(\underbrace{w_{1,1}^{(1)} y_1^{(1)} + w_{1,2}^{(1)} y_2^{(1)}}_{h_1^{(2)}}\right)$$

$$y_i^{(1)} = a(\underbrace{w_{i,1}^{(0)} x_1 + w_{i,2}^{(0)} x_2}_{h_i^{(1)}})$$

Loss for a training example $(\boldsymbol{x}, y)$: $\ell(y, h_{\boldsymbol{w}}(\boldsymbol{x}))$

Need to compute the gradient: $\nabla_{\mathbf{w}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x}))$

Backpropagation for a simple example (cont'd)



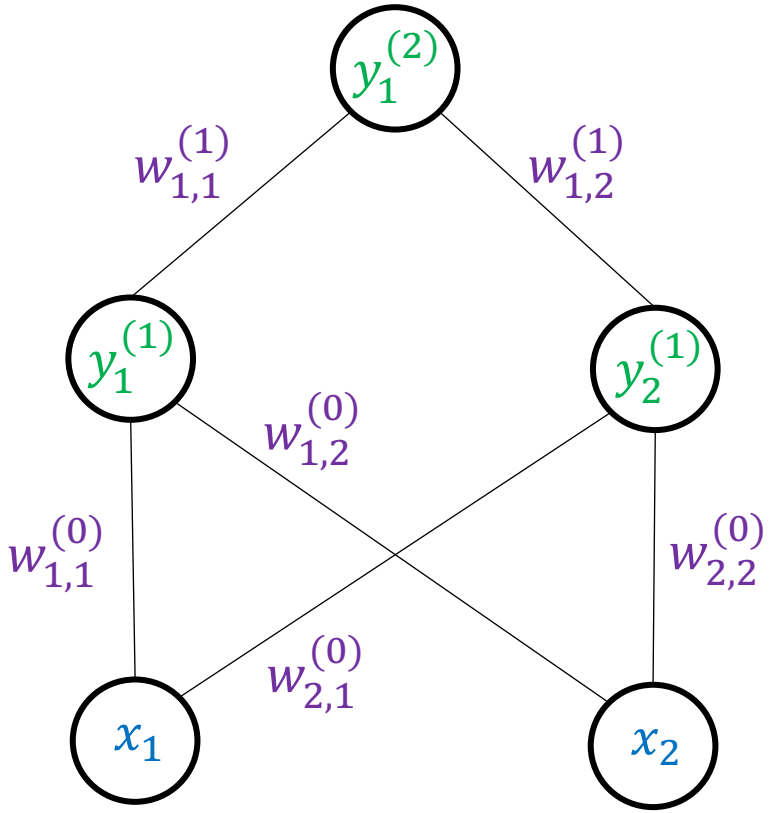
$$\frac{\partial}{\partial w_{1,i}^{(1)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial w_{1,i}^{(1)}} y_1^{(2)}$$

$$\frac{\partial}{\partial w_{1,i}^{(1)}} y_1^{(2)} = \frac{\partial}{\partial w_{1,i}^{(1)}} a\left(w_{1,1}^{(1)} y_1^{(1)} + w_{1,2}^{(1)} y_2^{(1)}\right)$$

$$= \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) y_i^{(1)}$$

$$\frac{\partial}{\partial w_{1,i}^{(1)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) y_i^{(1)}$$

Backpropagation for a simple example (cont't)



$$\frac{\partial}{\partial w_{i,j}^{(0)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial w_{i,j}^{(0)}} y_1^{(2)}$$

$$\frac{\partial}{\partial w_{i,j}^{(0)}} y_1^{(2)} = \frac{\partial}{\partial w_{i,j}^{(0)}} a\left(w_{1,1}^{(1)} y_1^{(1)} + w_{1,2}^{(1)} y_2^{(1)}\right)$$

$$= \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) \frac{\partial}{\partial w_{i,j}^{(0)}} y_i^{(1)}$$

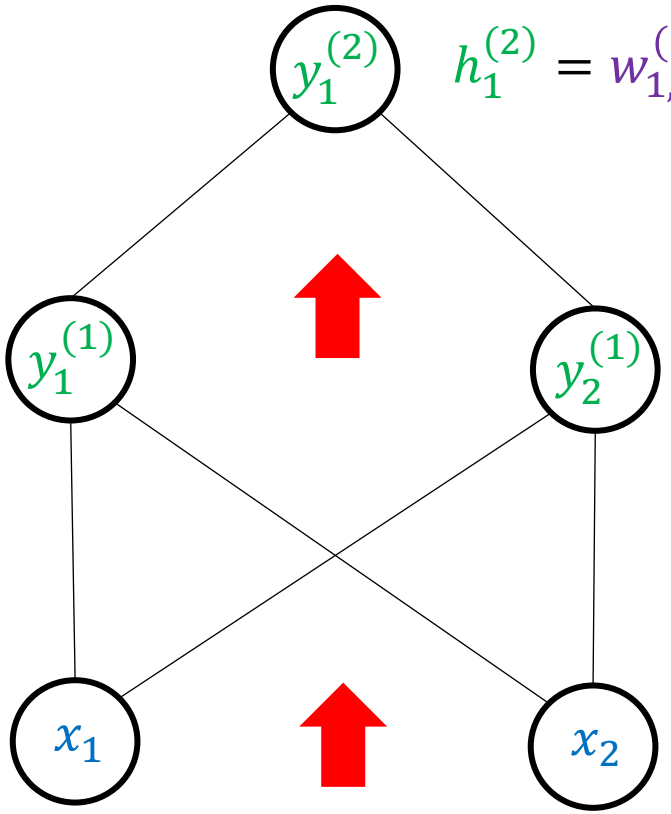
$$= \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) \frac{\partial}{\partial h_i^{(1)}} a\left(h_i^{(1)}\right) x_j$$

$$\frac{\partial}{\partial w_{i,j}^{(0)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) \frac{\partial}{\partial h_i^{(1)}} a\left(h_i^{(1)}\right) x_j$$

Forward computation

$$\frac{\partial}{\partial w_{1,i}^{(1)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) y_i^{(1)}$$

$$\frac{\partial}{\partial w_{i,j}^{(0)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right) \frac{\partial}{\partial h_i^{(1)}} a\left(h_i^{(1)}\right) x_j$$



$$h_1^{(2)} = w_{1,1}^{(1)}\, y_1^{(1)} + w_{1,2}^{(1)} y_2^{(1)}$$

$$h_1^{(1)} = w_{1,1}^{(0)}\, x_1 + w_{1,2}^{(0)} x_2$$

$$h_2^{(1)} = w_{2,1}^{(0)}\, x_1 + w_{2,2}^{(0)} x_2$$

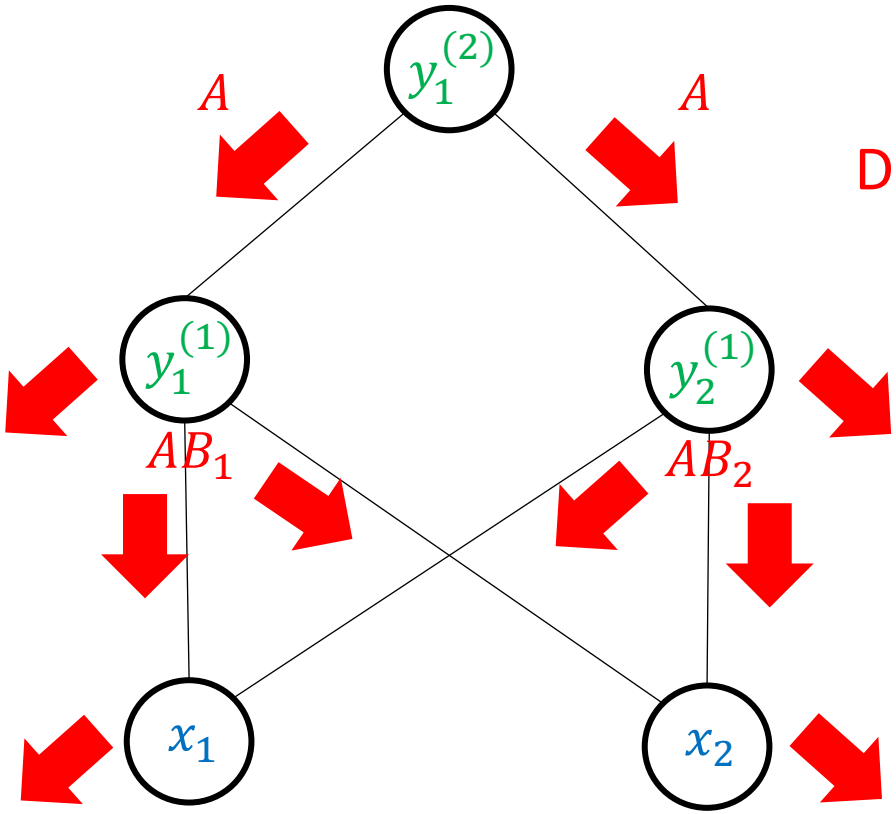
Backward computation

$$\frac{\partial}{\partial w_{1,i}^{(1)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \boxed{\frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right)} y_i^{(1)}$$

A

$$\frac{\partial}{\partial w_{i,j}^{(0)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = \boxed{\frac{\partial}{\partial y_1^{(2)}} \ell\left(y, y_1^{(2)}\right) \frac{\partial}{\partial h_1^{(2)}} a\left(h_1^{(2)}\right)} \boxed{\frac{\partial}{\partial h_i^{(1)}} a\left(h_i^{(1)}\right)} x_j$$

B_i



Don't repeat shared computation!

$$\frac{\partial}{\partial w_{1,1}^{(1)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = A y_1^{(1)}$$

$$\frac{\partial}{\partial w_{1,2}^{(1)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = A y_2^{(1)}$$

$$\frac{\partial}{\partial w_{i,1}^{(0)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = AB_i x_1$$

$$\frac{\partial}{\partial w_{i,2}^{(0)}} \ell(y, h_{\boldsymbol{w}}(\boldsymbol{x})) = AB_i x_2$$

Backpropagation: general definition

- Each element of the gradient is a partial derivative with respect to a weight corresponding to one of the network edges

- For every layer l, let $\boldsymbol{W}^{(l)} \in \mathbf{R}^{n_{l+1} \times n_l}$ be the matrix that gives a weight to every potential edge between nodes in V_l and V_{l+1}

Jacobian matrices

- Let $f: \mathbf{R}^n \to \mathbf{R}^m$ be a collection of differentiable functions

$$f_i: \mathbf{R}^n \to \mathbf{R} \text{ for } i = 1,2, \dots, m$$

- The Jacobian of f at $\boldsymbol{w} \in \mathbf{R}^n$ is the $m \times n$ matrix $J_{\boldsymbol{w}}(f)$ with elements

$$J_{\boldsymbol{w}}(f)_{i,j} = \frac{\partial}{\partial w_j} f_i(\boldsymbol{w})$$

Chain rule

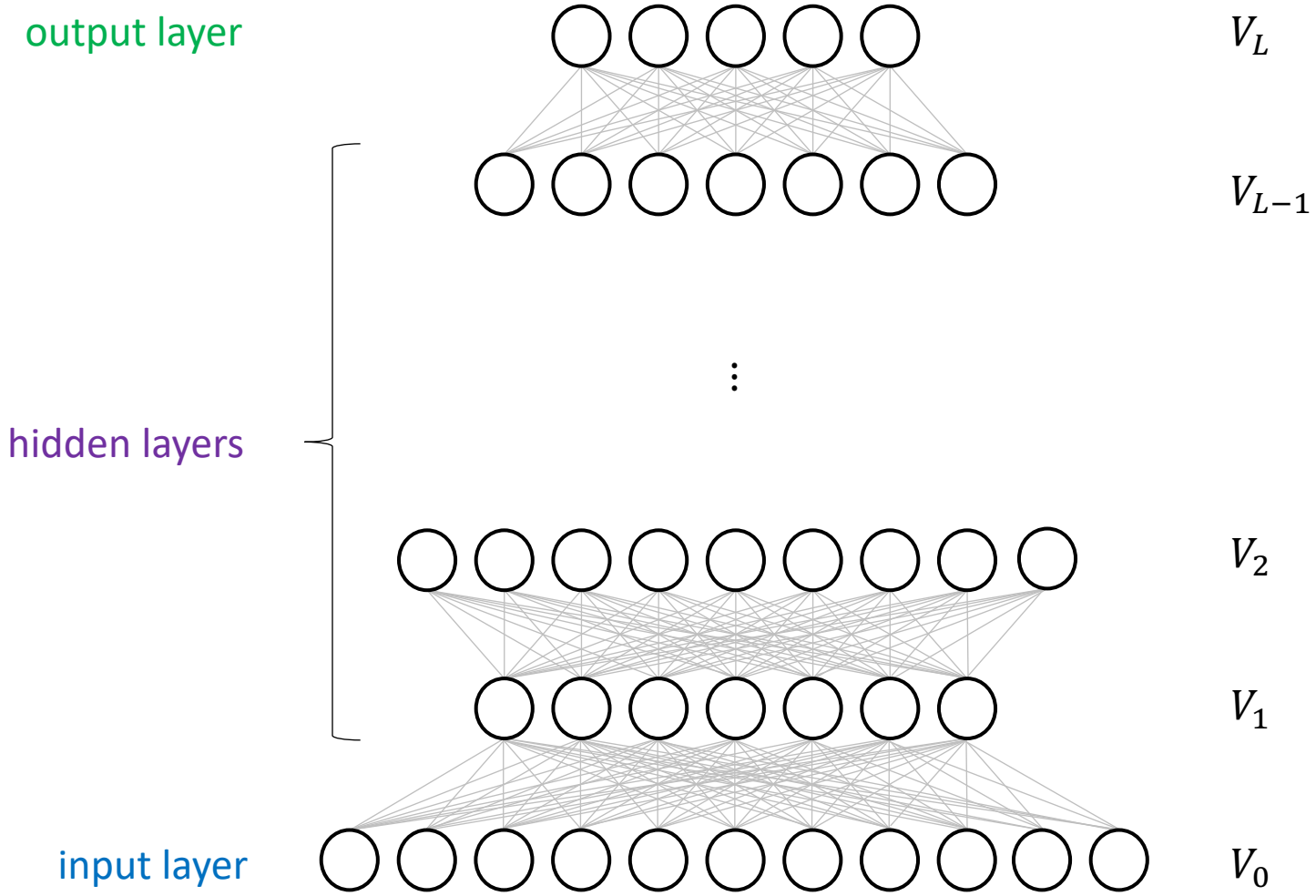
- The chain rule for computing derivative of a composition of functions can be written in terms of the Jacobian

- Let $f: \mathbf{R}^n \to \mathbf{R}^m$ and $g: \mathbf{R}^k \to \mathbf{R}^n$

- Let $f \circ g$ denote the composition: $f \circ g(\boldsymbol{x}) = f(g(\boldsymbol{x}))$

- Then: $J_{\boldsymbol{w}}(f \circ g) = J_{g(\boldsymbol{w})}(f) J_{\boldsymbol{w}}(g)$

Backpropagation explained



Function composition representation

- Function $\boldsymbol{y} = h(\boldsymbol{x})$ can be expressed as follows:

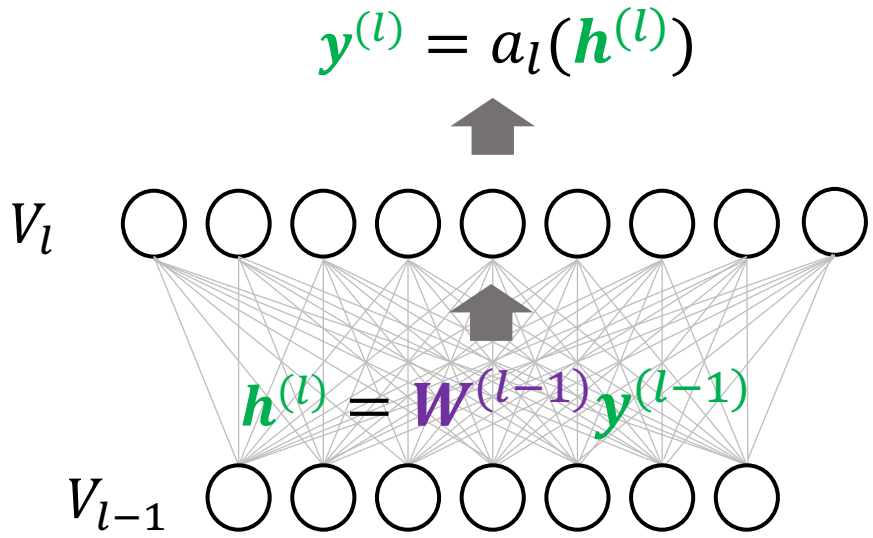
$$\begin{aligned}
\boldsymbol{y}^{(1)} &= a_1(\boldsymbol{W}^{(0)}\boldsymbol{x} + \boldsymbol{b}^{(0)}) \\
\boldsymbol{y}^{(2)} &= a_2\left(\boldsymbol{W}^{(1)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(1)}\right) \\
&\vdots \\
\boldsymbol{y}^{(L-1)} &= a_{L-1}\left(\boldsymbol{W}^{(L-2)}\boldsymbol{y}^{(L-2)} + \boldsymbol{b}^{(L-2)}\right) \\
\boldsymbol{y} &= \boldsymbol{W}^{(L-1)}\boldsymbol{y}^{(L-1)} + \boldsymbol{b}^{(L-1)}
\end{aligned}$$

- Using the notation $h_l(\boldsymbol{x}) = a_l(\boldsymbol{W}^{(l-1)}\boldsymbol{x} + \boldsymbol{b}^{(l-1)})$, we have

$$h(\boldsymbol{x}) = h_1 \circ h_2 \circ \cdots \circ h_L(\boldsymbol{x})$$

Backpropagation explained (cont'd)

- Fix all the weights of the network except for weights $\boldsymbol{W}^{(l-1)}$
- Outputs of layer V_{l-1} are fixed numbers, not depending on the weights $\boldsymbol{W}^{(l-1)}$



- Let $\ell_l : \mathbf{R}^{n_l} \to \mathbf{R}$ be the loss function of the subnetwork defined by network layers $V_l, V_{l+1}, \ldots, V_L$ as a function of $\boldsymbol{y}^{(l)}$ (output of $V^{(l)}$)

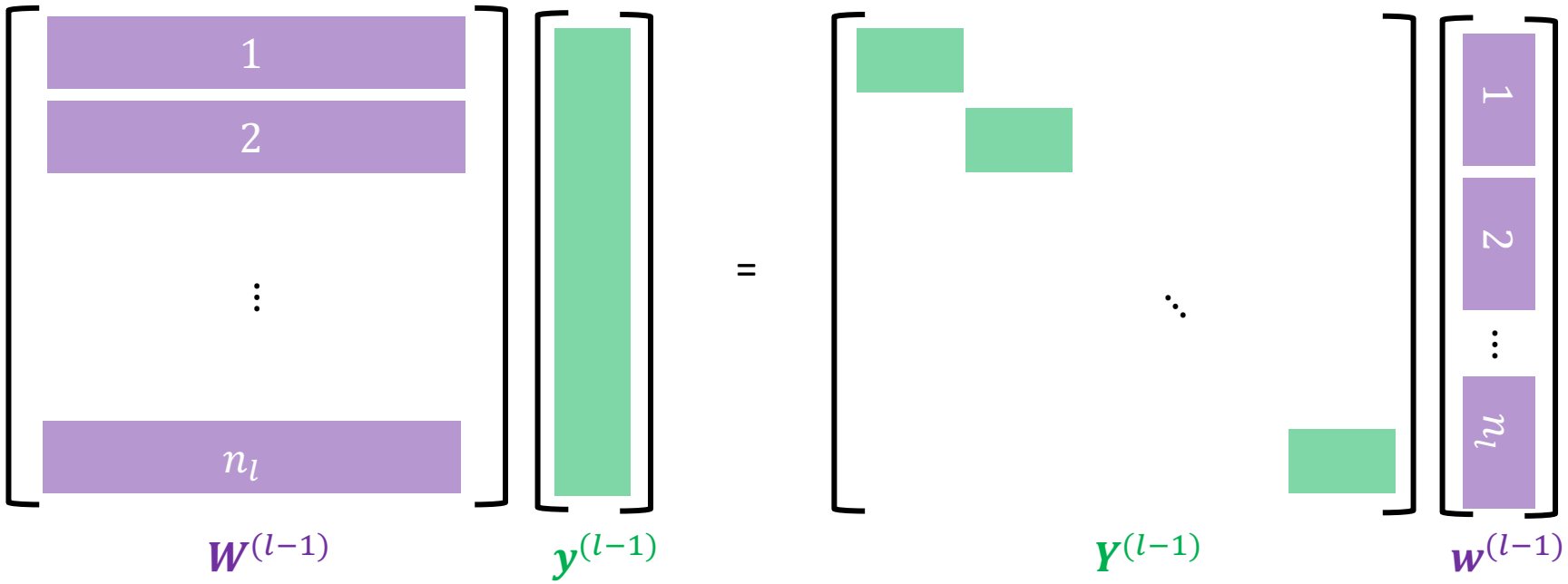
Backpropagation explained (cont'd)

- The loss function as a function of $\boldsymbol{W}^{(l-1)}$ can be written as

$$f_l\left(\boldsymbol{W}^{(l-1)}\right) := \ell_l\left(a_l\left(\boldsymbol{W}^{(l-1)}\boldsymbol{y}^{(l-1)}\right)\right)$$

- Change to vector notation:

$$f_l\left(\boldsymbol{w}^{(l-1)}\right) = \ell_l\left(a_l\left(\boldsymbol{Y}^{(l-1)}\boldsymbol{w}^{(l-1)}\right)\right)$$



Backpropagation explained (cont'd)

- By the chain rule

$$J_{\boldsymbol{w}^{(l-1)}}(f_l) = \underbrace{J_{\underbrace{a_l(\boldsymbol{Y}^{(l-1)}\boldsymbol{w}^{(l-1)})}_{\boldsymbol{y}^{(l)}}}(\ell_l)}_{\boldsymbol{\delta}^{(l)}} \mathbf{diag}\left(a_l'\left(\underbrace{\boldsymbol{Y}^{(l-1)}\boldsymbol{w}^{(l-1)}}_{\boldsymbol{h}^{(l)}}\right)\right)\boldsymbol{Y}^{(l-1)}$$

$$\Leftrightarrow \quad J_{\boldsymbol{w}^{(l-1)}}(f_l) = \left(\delta_1^{(l)} a_l'\left(h_1^{(l)}\right)\left(\boldsymbol{y}^{(l-1)}\right)^\top, \ldots, \delta_{n_l}^{(l)} a_l'\left(h_{n_l}^{(l)}\right)\left(\boldsymbol{y}^{(l-1)}\right)^\top\right)$$

Recursive computation of $\boldsymbol{\delta}_t$

- Output layer L: $\boldsymbol{\delta}^{(L)} = J_{\boldsymbol{y}}(\ell_L)$, where $\ell_L(\boldsymbol{u}) = \ell(\boldsymbol{y}, \boldsymbol{u})$
- For layer $1 \leq l < L$: $\ell_l(\boldsymbol{u}) = \ell_{l+1}(a_{l+1}(\boldsymbol{W}^{(l)}\boldsymbol{u}))$
  - By the chain rule: $J_{\boldsymbol{u}}(\ell_l) = J_{a_{l+1}(\boldsymbol{W}^{(l)}\boldsymbol{u})}(\ell_{l+1})\,\mathbf{diag}\left(a'_{l+1}(\boldsymbol{W}^{(l)}\boldsymbol{u})\right)\boldsymbol{W}^{(l)}$
  - $\boldsymbol{\delta}^{(l)} = J_{\boldsymbol{y}^{(l)}}(\ell_l) = J_{a_{l+1}(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l)})}(\ell_{l+1})\,\mathbf{diag}\left(a'_{l+1}(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l)})\right)\boldsymbol{W}^{(l)}$

$$= J_{\boldsymbol{y}^{(l+1)}}(\ell_{l+1})\,\mathbf{diag}\left(a'_{l+1}(\boldsymbol{h}^{(l+1)})\right)\boldsymbol{W}^{(l)}$$

$$= \boldsymbol{\delta}^{(l+1)}\,\mathbf{diag}\left(a'_{l+1}(\boldsymbol{h}^{(l+1)})\right)\boldsymbol{W}^{(l)}$$

Summary of key steps

- Forward propagation:
  - Compute $\boldsymbol{y}^{(0)}, \boldsymbol{h}^{(1)}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{h}^{(L)}, \boldsymbol{y}^{(L)}$ by forward computation from input to output of the network

- Backward propagation:
  - Compute $\boldsymbol{\delta}^{(L)}, \boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(1)}$ by backward computation from output to input of the network

- Computing partial gradients: having computed $\boldsymbol{h}^{(l)}, \boldsymbol{y}^{(l-1)}$ and $\boldsymbol{\delta}^{(l)}$, the gradients $J_{\boldsymbol{w}_{l-1}}(f_l)$ are given by

$$J_{\boldsymbol{w}_{l-1}}(f_l) = (\delta_1^{(l)} a'\left(h_1^{(l)}\right)\left(\boldsymbol{y}^{(l-1)}\right)^\top, \ldots, \delta_{n_l}^{(l)} a'\left(h_{n_l}^{(l)}\right)\left(\boldsymbol{y}^{(l-1)}\right)^\top)$$

Dropout

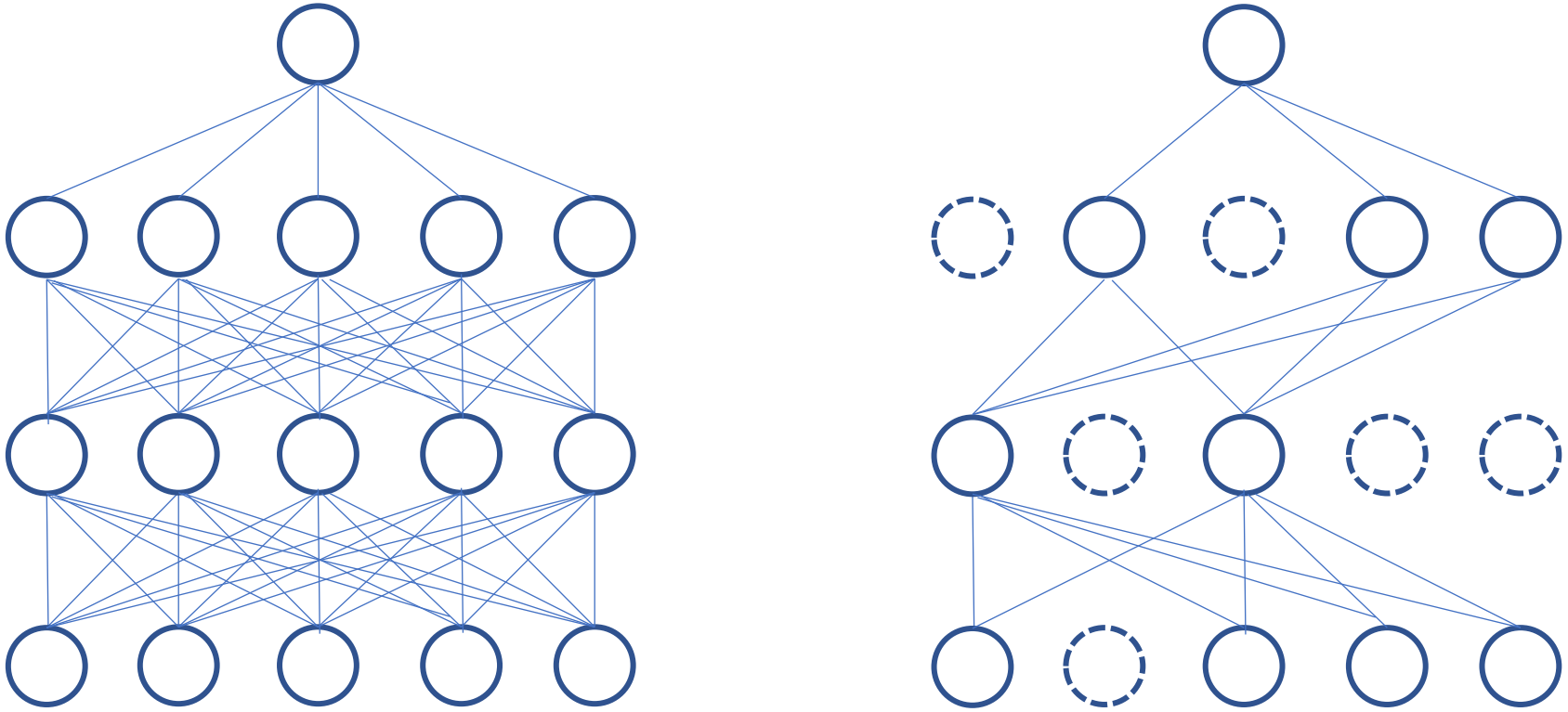
Dropout

- Deep neural networks often have a large number of parameters
  - Overfitting problem
  - Combining predictions of many different neural networks at a test time is problematic because large neural networks are slow to use
- Dropout: a technique for addressing the overfitting problem
  - Key idea: randomly drop neurons and their incident connections during training
- Benefits of dropout
  - Preventing neurons from co-adapting too much
  - During training dropout samples from an exponential number of different subnetworks
- Shown to to improve the performance of neural networks on various supervised learning tasks, e.g. vision, speech recognition, document classification, and computational biology

Approaches to avoid model overfitting

- Regularization: introducing weight penalties such as L1 and L2 regularization
- Data augmentation: adding noise to input data
- Early stopping: stopping the training of a model as soon as its performance on a validation set starts to get worse
- Combining models: for a fixed-size model, average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data
  - Averaging the outputs of many separately trained models (neural networks) can be prohibitively expensive
  - Dropout aims to approximately achieve this by using less computation

Dropping units



- Dropping units: temporarily removing some units from the networks along with all its incoming and outgoing connections

Feedforward neural network

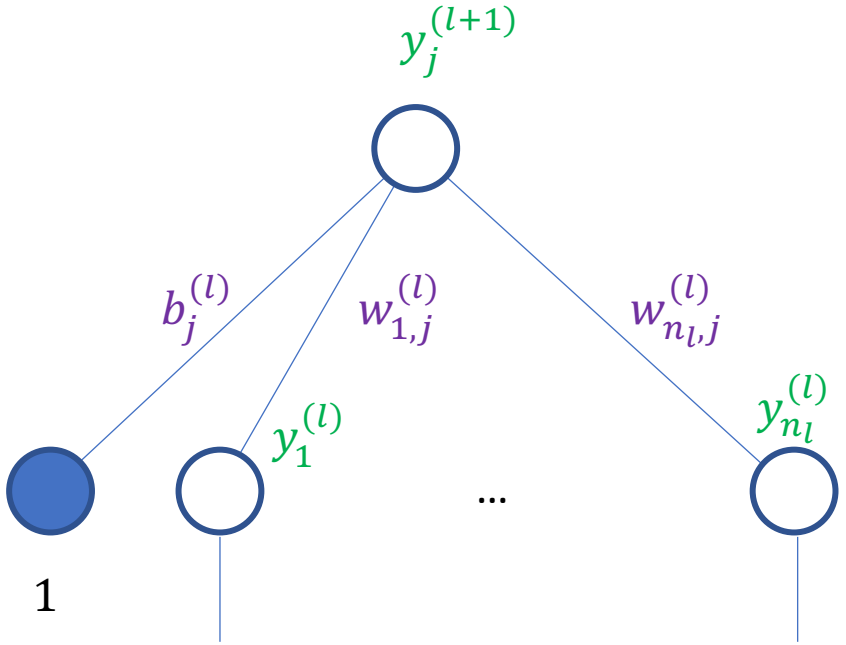
Function mapping: $\boldsymbol{x} \mapsto \boldsymbol{y}$

$$\boldsymbol{y}^{(0)} = \boldsymbol{x}$$

For layers $0 < l < L$:

$$\boldsymbol{y}^{(l+1)} = a_l\left(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l)} + \boldsymbol{b}^{(l)}\right)$$

$$\boldsymbol{y} = \boldsymbol{y}^{(L)}$$



Feedforward neural network with dropout

Function mapping: $\boldsymbol{x} \mapsto \boldsymbol{y}$

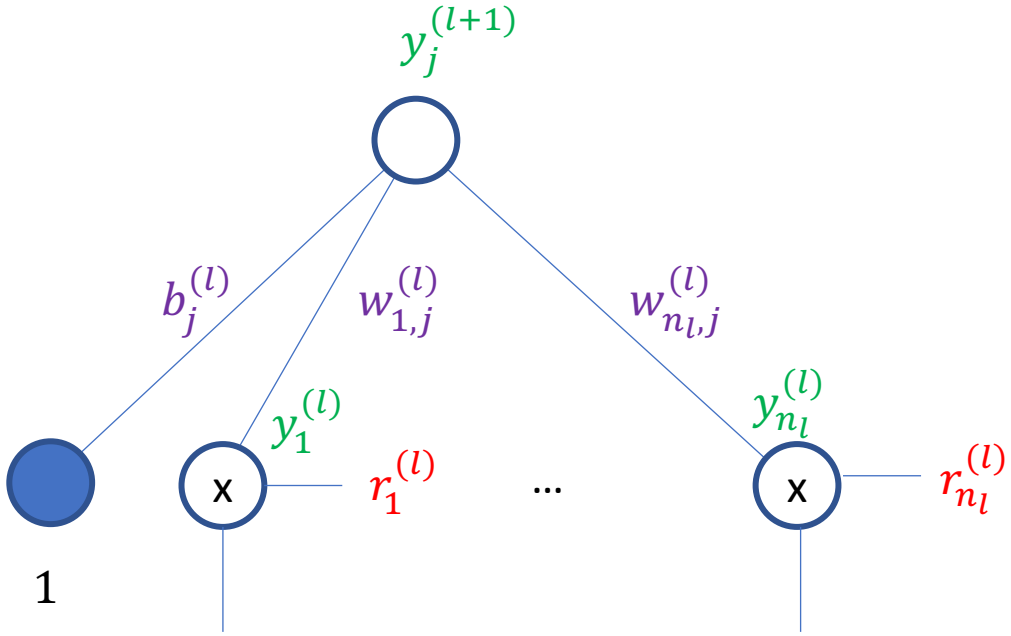
$\boldsymbol{y}^{(0)} = \boldsymbol{x}$

For $0 < l < L$:

$r^{(l)} =$ vector i.i.d. Bernoulli (p)

$\boldsymbol{y}^{(l+1)} = a_l\left(\boldsymbol{W}^{(l)}\mathbf{diag}(\boldsymbol{r}^{(l)})\boldsymbol{y}^{(l)} + \boldsymbol{b}^{(l)}\right)$

$\boldsymbol{y} = \boldsymbol{y}^{(L)}$



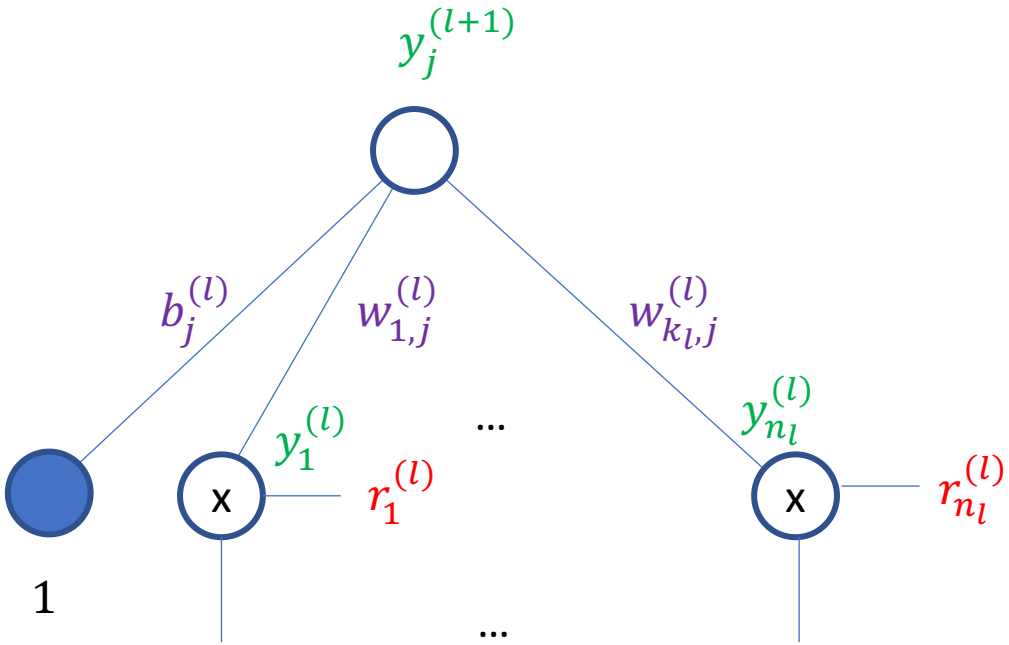
Learning dropout neural networks

- Dropout feedforward neural networks can be trained using stochastic gradient descent and backpropagation algorithm as for any feedforward neural network

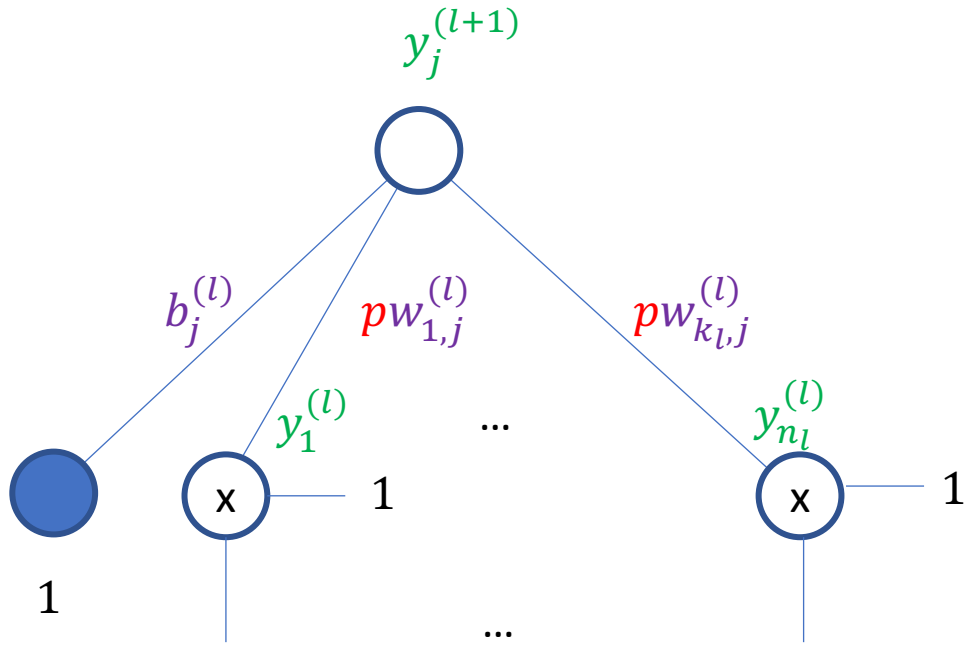
- The only difference is that for each training in a **mini-batch**, we use a sample of a thinned network by dropping out neurons
  - Forward and backward computations for a mini-batch are done on this thinned network

- At test time a single neural network is used without dropout
  - The weights of this neural network are scaled-down versions of the trained weights (scaled down for factor p)

Neural network used at test time



Dropping units in training phase



Network used at test time

References

- S. Bubeck, Convex optimization: algorithms and convexity, Now publishers, 2015, https://arxiv.org/pdf/1405.4980.pdf
- Y. Nesterov, Introductory Lectures on Convex Programming, 1998
- I. Sutskever, J. Martens, G. Dahl, and G. Hinton, On the Importance of Initialization and Momentum in Deep Learning, ICML 2013
- D. P. Kingma and J. L. Ba, Adam: A Method for Stochastic Optimization, ICLR 2015
- J. Duchi, E. Hazan and Y. Singer, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, JMLR 2011
- G. Hinton, N. Srivastava and K. Swersky, rmsprop: Divide the Gradient by a running average of its recent magnitude, Lecture, Course Neural Networks for Machine Learning, Coursera, 2012
- A. Graves, Generating Sequences with Recurrent Neural Networks, Arxiv, 2014
- L. Bottou and O. Bousquet, The trade-offs of large scale learning, NIPS 2007

References (cont'd)

- P. Baldi and K. Hornik, Neural networks and principal component analysis: learning from examples without local minima, Neural networks, 2(1), 53-58, 1989
- J. D. Lee, M. Simehowitz, M. I. Jordan, and B. Recht, Gradient descent only converges to minimizers, COLT 2016
- K. Kawaguchi, Deep learning without poor local minima, NIPS 2016
- S. Wiesler, A. Richard, R. Schluter, and H. Ney, A convergence analysis of log-linear training, NIPS 2011
- Y. LeCun, L. Bottou, G. Orr, and K. Muller, Efficient backpropagation, Neural Networks: tricks of the trade, 1998
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, JMLR, 2014

Seminar exercises

- Backpropagation
- Optimization in TensorFlow
- Homework: non-convex optimization