

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:

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

output parameter input

- The expected loss function:

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

given loss function

- Empirical risk minimization:

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

regularization term, $\lambda \geq 0$

Loss functions for binary classification

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

- Misclassification loss (negative accuracy):

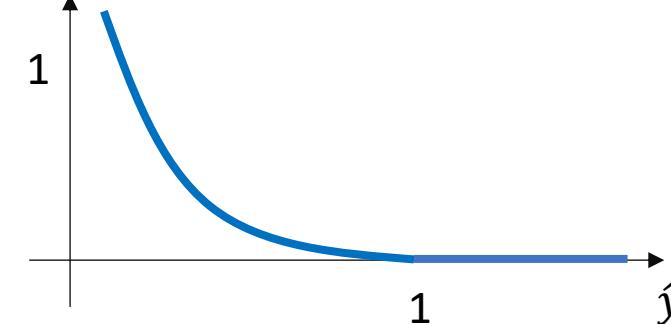
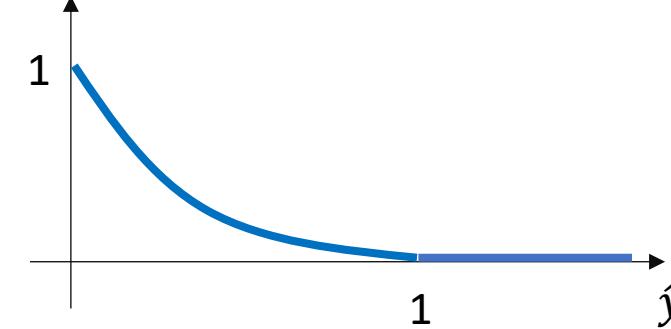
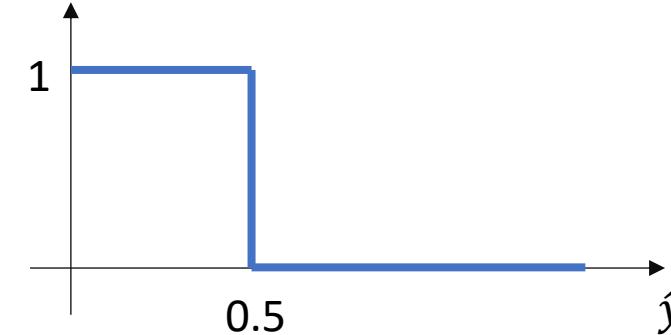
$$\ell(y, \hat{y}) = 1_{\left(y - \frac{1}{2}\right) \left(\hat{y} - \frac{1}{2}\right) < 0}$$

- 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 $\mathbf{y} \in \{0,1\}^d$ such that $\sum_{i=1}^d \mathbf{y}_i = 1$

Predicted distribution $\hat{\mathbf{y}}$

- Squared loss: $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \sum_{i=1}^d (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$
- Cross-entropy: $\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^d \mathbf{y}_i \log(\hat{\mathbf{y}}_i)$
- Cross-entropy interpretation:
 - Penalty $\log(\frac{1}{\hat{\mathbf{y}}_i})$ if \mathbf{y}_i is the true class
 - KL-divergence: $\text{KL}(\mathbf{y} || \hat{\mathbf{y}}) = \sum_{i=1}^d \mathbf{y}_i \log \left(\frac{\mathbf{y}_i}{\hat{\mathbf{y}}_i} \right) = \underbrace{-H(\mathbf{y})}_{\text{Entropy}} + \ell(\mathbf{y}, \hat{\mathbf{y}})$

$$\text{Entropy } H(\mathbf{y}) = -\sum_{i=1}^d \mathbf{y}_i \log(\mathbf{y}_i)$$

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

Standard regularizations

- Ridge or L_2 norm:

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

- Lasso:

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

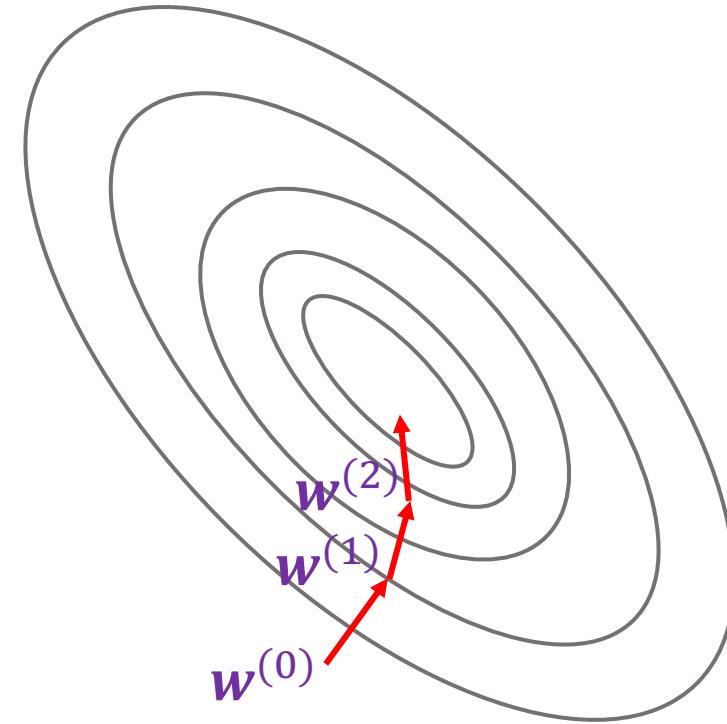
Gradient descent algorithm

- Gradient descent algorithm update:

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

step size

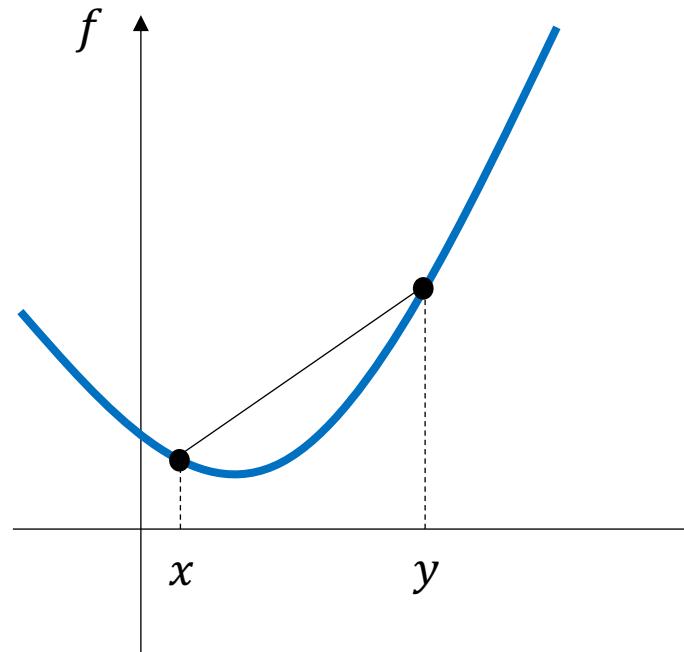
- Common step sizes:
 - A small positive constant
 - A decreasing sequence such that $\sum_{t=1}^{\infty} \eta^{(t)} = \infty$ and $\sum_{t=1}^{\infty} (\eta^{(t)})^2$ finite
- Common choices of $\mathbf{B}^{(t)}$:
 - Standard gradient descent: $\mathbf{B}^{(t)} = I$
 - Adaptive learning rates: Newton's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



Convex functions

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

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

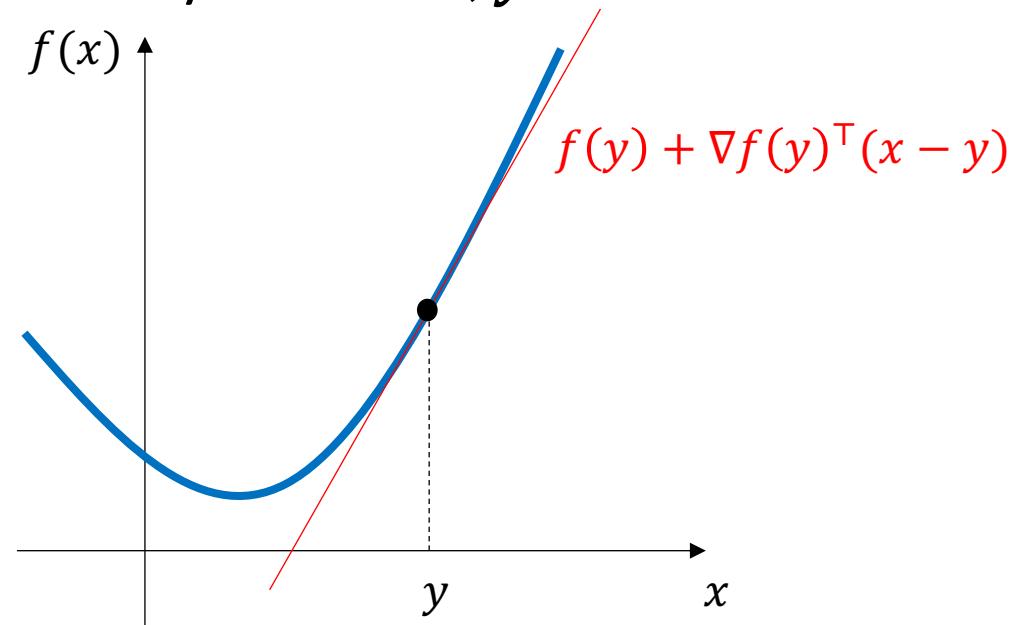


- f is said to be **strictly convex** if strict inequality holds for all $\lambda \in (0,1)$

Convex functions (cont'd)

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

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



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

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 $x, y \in \mathbf{R}^n$

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

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

$$\nabla f(x) = Hx$$

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

Smooth functions (cont'd)

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

$$(S0) |f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})| \leq \frac{\beta}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

- Equivalent conditions for f convex and β -smooth [Nesterov Thm 2.1.5]:
 - (S1) $0 \leq f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) \leq \frac{\beta}{2} \|\mathbf{y} - \mathbf{x}\|^2$
 - (S2) $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{1}{2\beta} \|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\|^2$
 - (S3) $(\nabla f(\mathbf{y}) - \nabla f(\mathbf{x}))^\top (\mathbf{y} - \mathbf{x}) \geq \frac{1}{\beta} \|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\|^2$

Gradient descent: smooth convex functions

- Consider gradient descent algorithm with a constant step size:

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

- 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 $\mathbf{x}^{(0)}$ and a globally optimum point \mathbf{x}^* such that $\|\mathbf{x}^{(0)} - \mathbf{x}^*\| \leq R$, it holds

$$f(\mathbf{x}^{(t)}) - f(\mathbf{x}^*) \leq \frac{2R^2}{2R^2 + (f(\mathbf{x}^{(0)}) - f(\mathbf{x}^*))\eta(2 - \beta\eta)t} (f(\mathbf{x}^{(0)}) - f(\mathbf{x}^*))$$

$\underbrace{\phantom{2R^2 + (f(\mathbf{x}^{(0)}) - f(\mathbf{x}^*))\eta(2 - \beta\eta)t}}$

$$= O\left(\frac{1}{t}\right)$$

Proof sketch

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

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

$$\begin{aligned}\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|^2 &= \|\mathbf{x}^{(t)} - \mathbf{x}^* - \eta \nabla f(\mathbf{x}^{(t)})\|^2 \\ &= \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 - 2\eta \nabla f(\mathbf{x}^{(t)})^\top (\mathbf{x}^{(t)} - \mathbf{x}^*) + \eta^2 \|\nabla f(\mathbf{x}^{(t)})\|^2 \\ &\leq \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 - 2\eta \frac{1}{\beta} \|\nabla f(\mathbf{x}^{(t)})\|^2 + \eta^2 \|\nabla f(\mathbf{x}^{(t)})\|^2 \\ &= \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 - \eta \left(\frac{2}{\beta} - \eta \right) \|\nabla f(\mathbf{x}^{(t)})\|^2\end{aligned}$$

Proof sketch (cont'd)

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

$$\begin{aligned} f(\mathbf{x}^{(t+1)}) &\leq f(\mathbf{x}^{(t)}) + \nabla f(\mathbf{x}^{(t)})^\top (\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}) + \eta^2 \|\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)}\|^2 \\ &= f(\mathbf{x}^{(t)}) - \eta \left(\frac{2}{\beta} - \eta \right) \|\nabla f(\mathbf{x}^{(t)})\|^2 \end{aligned}$$

- **Claim 3:** since f is convex , we have

$$\begin{aligned} f(\mathbf{x}^{(t)}) - f(\mathbf{x}^*) &\leq \nabla f(\mathbf{x}^{(t)})^\top (\mathbf{x}^{(t)} - \mathbf{x}^*) \leq \|\nabla f(\mathbf{x}^{(t)})\| \|\mathbf{x}^{(t)} - \mathbf{x}^*\| \\ &\leq R \|\nabla f(\mathbf{x}^{(t)})\| \end{aligned}$$

Proof sketch (cont'd)

- By Claim 2 and Claim 3, we have

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

- But this is equivalent to

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

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

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

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

Strongly convex convex functions

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

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

- If f is twice differentiable than for all x all the eigenvalues of the Hessian matrix $\nabla^2 f(x)$ 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 $\|\mathbf{x}^{(0)} - \mathbf{x}^*\| \leq R$, we have

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

where $\kappa = \beta/\alpha$

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

Proof sketch

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

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

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

$$(\nabla \phi(y) - \nabla \phi(x))^\top (y - x) \geq \frac{1}{\beta - \alpha} \|\nabla \phi(y) - \nabla \phi(x)\|^2$$

which yields the claim by straightforward calculus

Proof sketch (cont'd)

- By β -smoothness of f , we have $f(\mathbf{x}^{(t)}) - f(\mathbf{x}^*) \leq \frac{\beta}{2} \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2$
- By Claim 1:

$$\begin{aligned}\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|^2 &= \|\mathbf{x}^{(t)} - \mathbf{x}^* - \eta \nabla f(\mathbf{x}^{(t)})\|^2 \\ &= \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 - 2\eta \nabla f(\mathbf{x}^{(t)})^\top (\mathbf{x}^{(t)} - \mathbf{x}^*) + \eta^2 \|\nabla f(\mathbf{x}^{(t)})\|^2 \\ &\leq \left(1 - 2\frac{\eta\alpha\beta}{\alpha + \beta}\right) \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 + \left(\eta^2 - 2\frac{\eta}{\alpha + \beta}\right) \|\nabla f(\mathbf{x}^{(t)})\|^2 \\ &= \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 \leq e^{-\frac{4}{\kappa+1}} \|\mathbf{x}^{(t)} - \mathbf{x}^*\|^2 \leq e^{-\frac{4}{\kappa+1}(t+1)} \|\mathbf{x}^{(0)} - \mathbf{x}^*\|^2\end{aligned}$$

Scalability issues of gradient descent

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

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

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 $(\mathbf{x}_s, \mathbf{y}_s)$ compute

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

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

Stochastic gradient descent algorithm

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

while stopping criterion is not met **do**

sample a training example (x_{st}, y_{st})

compute stochastic gradient vector:

$$\widehat{\nabla}f(w) = \nabla_w \ell(y_{st}, h_w(x_{st})) + \lambda \nabla \phi(w)$$

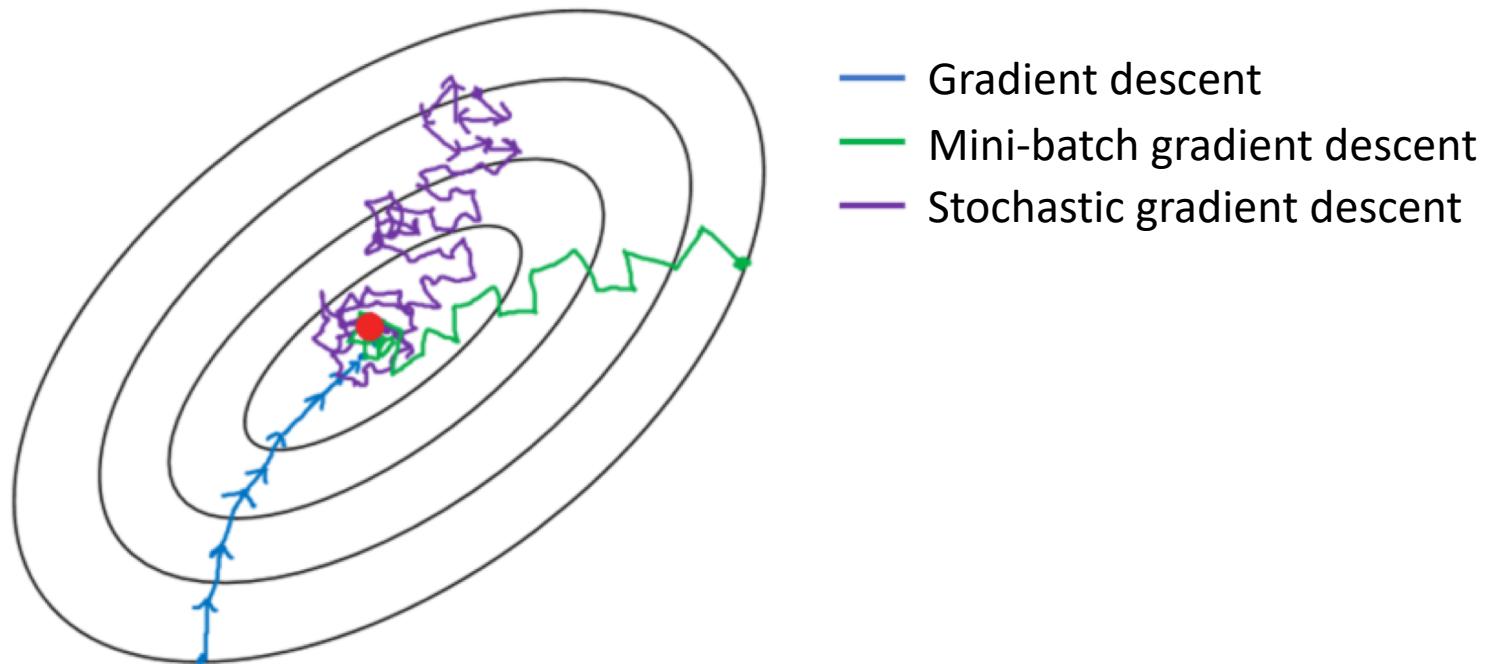
apply update: $w \leftarrow w - \eta^{(t)} \widehat{\nabla}f(w)$

$$t \leftarrow t + 1$$

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



SGD convergence for smooth convex functions

Thm. If

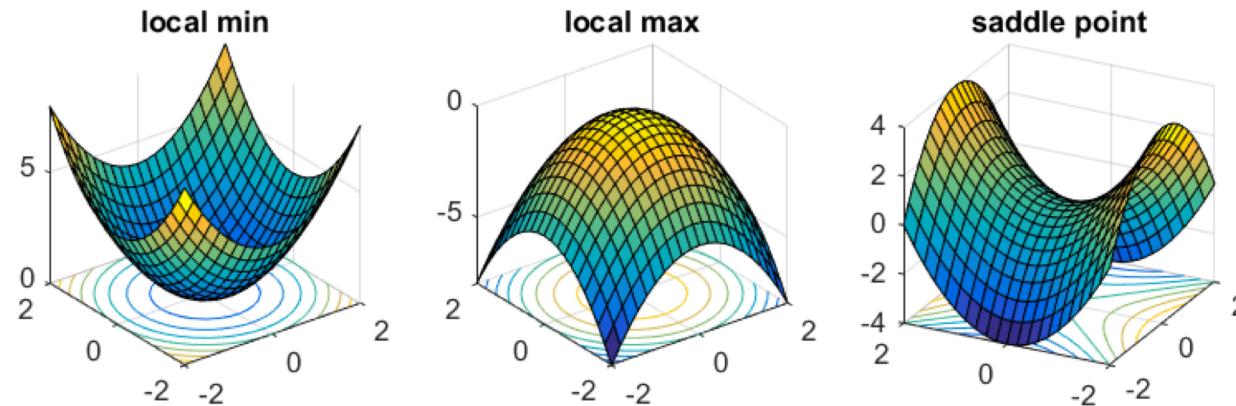
- W is a convex set
- f is a convex β -smooth function
- $\mathbf{w}^{(0)} \in W$ and $R = \sup_{\mathbf{w} \in W} \|\mathbf{w} - \mathbf{w}^{(0)}\|$.
- stochastic gradient vector $\widehat{\nabla}f(\mathbf{w})$ satisfies $\mathbf{E} [\|\widehat{\nabla}f(\mathbf{w}) - \nabla f(\mathbf{w})\|^2] \leq \sigma^2$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

then,

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

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 $f: \mathbf{R}^n \rightarrow \mathbf{R}$ is a twice differentiable function
- A point $x^* \in \mathbf{R}^n$ is a **critical point** of f if $\nabla f(x^*) = 0$
- **Local minimum:** a critical point x^* is a local minimum if there exists a neighborhood X around x^* such that $f(x) \geq f(x^*)$ for **all** $x \in X$
- **Saddle point:** a critical point x^* is a saddle point if for each neighborhood set X around x^* such that $f(x) \leq f(x^*) \leq f(y)$ for **some** $x, y \in X$
- A function f satisfies the **strict saddle property** if each critical point x^* is either:
 - A local minimizer, or
 - A strict saddle, i.e. $\nabla^2 f(x^*)$ has at least one negative eigenvalue

Importance of random initialization

- Thm. If $f: \mathbf{R}^n \rightarrow \mathbf{R}$ 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) divergesalmost surely
- Moral: gradient descent escapes (strict) saddle points !

Intuition by quadratic function example

- Consider $f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^\top \mathbf{H} \mathbf{x}$ where $\mathbf{H} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$ such that for $1 \leq k < n$: $\lambda_1, \dots, \lambda_k > 0$ and $\lambda_{k+1}, \dots, \lambda_n < 0$
- For such function f , $\mathbf{x}^* = \mathbf{0}$ is the unique critical point (strict saddle)
- Consider the gradient descent algorithm

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

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 $\mathbf{x}^{(0)}$?

Intuition by quadratic function example (cont'd)

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

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

i.e.

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

- 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 $\mathbf{x}^{(0)}$ is a random point,
 $\Pr[\mathbf{x}^{(0)} \in \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)] = 0$,
thus $\mathbf{x}^{(t)}$ “escapes the saddle point”



- Hence, if $\mathbf{x}^{(0)} \in \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)$, then $\lim_{t \rightarrow \infty} \mathbf{x}^{(t)} = \mathbf{0}$, otherwise, $\lim_{t \rightarrow \infty} \|\mathbf{x}^{(t)}\| = \infty$

Note: \mathbf{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:

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

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

or, equivalently

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

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

- If $\gamma = 1$ this is standard gradient descent update
- The larger the γ the velocity accumulates more information about past gradients
- Typical values $\gamma = 0.9$ or 0.95 or 0.99
- Momentum may smooth updates, enhance stability and convergence speed

Nesterov's accelerated gradient descent [1983]

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

- Update at iteration t :

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

$$\mathbf{w}^{(t+1)} = (1 - \gamma^{(t)}) \mathbf{v}^{(t+1)} + \gamma^{(t)} \mathbf{v}^{(t)}$$

where

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

Sutskever et al's momentum [2013]

- Update equations:

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

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

- Or, equivalently,

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

- 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): $\mathbf{v}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \nabla f(\mathbf{w}^{(t)})$
- (N2): $\mathbf{w}^{(t+1)} = (1 - \gamma^{(t)}) \mathbf{v}^{(t+1)} + \gamma^{(t)} \mathbf{v}^{(t)}$

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

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

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

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

To obtain Sutskever et al's update equations,
use change of variables:
 $\mathbf{v} \rightarrow \mathbf{w}$
 $\mathbf{u} \rightarrow \mathbf{v}$
 $\tilde{\gamma} \rightarrow \gamma$

Quadratic function example

- Suppose $f(\mathbf{w}) = \frac{1}{2} \mathbf{w}^\top \mathbf{H} \mathbf{w}$ where \mathbf{H} is a real-symmetric positive definite matrix
 - Real-symmetric matrix positive definite \Leftrightarrow all eigenvalues positive
- Minimizer of f : $\mathbf{w}^* = \mathbf{0}$
- Gradient descent algorithm:

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

- For a constant step size η ,

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

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

Stability of linear systems

- A linear dynamical system

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

- **Strictly stable**: if for all initial values, $\lim_{t \rightarrow \infty} \boldsymbol{x}^{(t)} = \mathbf{0}$
- **Unstable**: if there exists an initial value for which $\lim_{t \rightarrow \infty} \|\boldsymbol{x}^{(t)}\| = \infty$
- **Marginally stable**: it is neither stable nor unstable, i.e. the limit point remains bounded but it is not necessarily $\mathbf{0}$
- Strictly stable $\Leftrightarrow |\lambda_i(\mathbf{A})| < 1$ for each eigenvalue $\lambda_i(\mathbf{A})$ of \mathbf{A}

Quadratic function example (cont'd)

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

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

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

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

Quadratic function example (cont'd)

- Consider now the accelerated algorithm with momentum:

$$\begin{aligned}\boldsymbol{v}^{(t+1)} &= \gamma \boldsymbol{v}^{(t)} - \eta \nabla f(\boldsymbol{w}^{(t)} + \gamma \boldsymbol{v}^{(t)}) \\ \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} + \boldsymbol{v}^{(t)}\end{aligned}$$

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

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

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

Quadratic function example (cont'd)

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

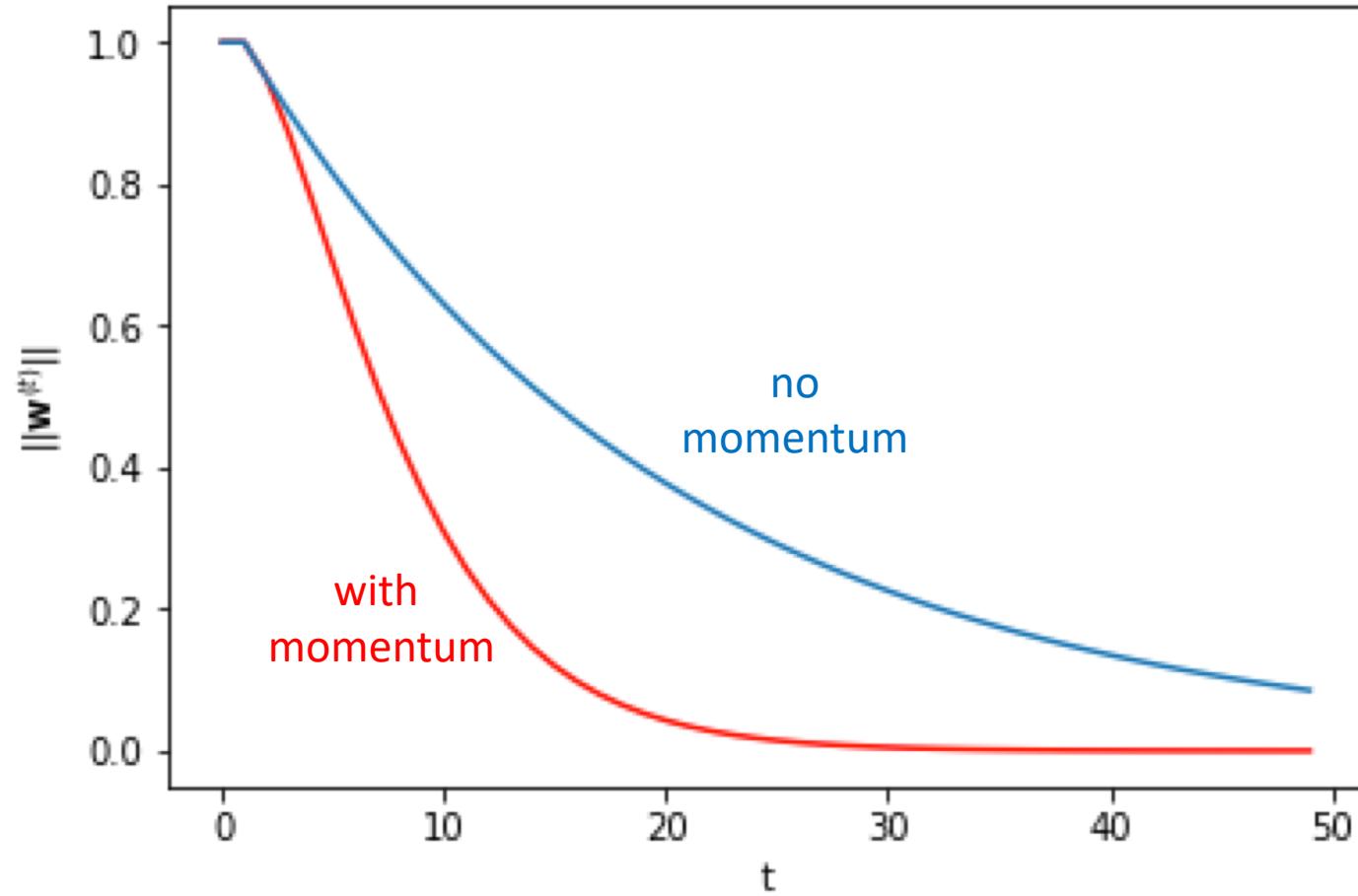
$$\left\| \begin{pmatrix} \mathbf{w}^{(t+1)} \\ \mathbf{w}^{(t)} \end{pmatrix} \right\| \leq \|\mathbf{B}\| \left\| \begin{pmatrix} \mathbf{w}^{(t)} \\ \mathbf{w}^{(t-1)} \end{pmatrix} \right\|$$

with rate

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

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

Numerical example



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

Momentum exercise solution

- Need to show that for $\mathbf{B} = \begin{pmatrix} (1 + \gamma)\mathbf{A} & -\gamma\mathbf{A} \\ \mathbf{I} & \mathbf{0} \end{pmatrix}$, $\|\mathbf{B}\| = 1 - \sqrt{1/\kappa(\mathbf{H})}$
- Elementary fact: $\det \begin{pmatrix} \mathbf{U} & \mathbf{V} \\ \mathbf{W} & \mathbf{Z} \end{pmatrix} = \det(\mathbf{UZ} - \mathbf{VW})$ if $\mathbf{WZ} = \mathbf{ZW}$
 $\Rightarrow \det(\lambda\mathbf{I} - \mathbf{B}) = \det(\lambda^2\mathbf{I} - ((1 + \gamma)\lambda - \gamma)\mathbf{A})$
- For $\gamma = \lambda/(2 - \lambda)$ we have
$$\det(\lambda\mathbf{I} - \mathbf{B}) = \frac{\lambda}{2 - \lambda} \det(\lambda(2 - \lambda)\mathbf{I} - \mathbf{A})$$
- λ is an eigenvalue of \mathbf{B} if $\lambda(2 - \lambda) = \mu$ for some eigenvalue μ of $\tilde{\mathbf{A}}$
 $\Rightarrow \|\mathbf{B}\|(2 - \|\mathbf{B}\|) = \|\mathbf{A}\| \Rightarrow \|\mathbf{B}\| = 1 - \sqrt{1 - \|\mathbf{A}\|} = 1 - \sqrt{1/\kappa(\mathbf{H})}$

Convergence theorem

Thm. If

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

then, for the Nesterov's accelerated gradient descent algorithm

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

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	$\exp\left(-\frac{t}{n}\right)$	$n \log\left(\frac{1}{\varepsilon}\right)$	$1 \nabla, 1 n\text{-dim } \int$
non-smooth	ellipsoid method	$\frac{R}{r} \exp\left(-\frac{t}{n^2}\right)$	$n^2 \log\left(\frac{R}{r\varepsilon}\right)$	$1 \nabla, \text{mat-vec } \times$
non-smooth	Vaidya	$\frac{Rn}{r} \exp\left(-\frac{t}{n}\right)$	$n \log\left(\frac{Rn}{r\varepsilon}\right)$	$1 \nabla, \text{mat-mat } \times$
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}	$R^2 L^2/\varepsilon^2$	$1 \nabla, 1 \text{ proj.}$
smooth	PGD	$\beta R^2/t$	$\beta R^2/\varepsilon$	$1 \nabla, 1 \text{ proj.}$
smooth	AGD	$\beta R^2/t^2$	$R\sqrt{\beta/\varepsilon}$	1∇
smooth (any norm)	FW	$\beta R^2/t$	$\beta R^2/\varepsilon$	$1 \nabla, 1 \text{ LP}$
strong. conv., Lipschitz	PGD	$L^2/(\alpha t)$	$L^2/(\alpha\varepsilon)$	$1 \nabla, 1 \text{ proj.}$
strong. conv., smooth	PGD	$R^2 \exp\left(-\frac{t}{\kappa}\right)$	$\kappa \log\left(\frac{R^2}{\varepsilon}\right)$	$1 \nabla, 1 \text{ proj.}$
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	$\beta R^2/t^2$	$R\sqrt{\beta/\varepsilon}$	$1 \nabla \text{ of } f$ $\text{Prox of } g$
$\max_{y \in \mathcal{Y}} \varphi(x, y),$ 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}	$B^2 L^2/\varepsilon^2$	$1 \text{ stoch. } \nabla, 1 \text{ proj.}$
non-smooth, strong. conv.	SGD	$B^2/(\alpha t)$	$B^2/(\alpha\varepsilon)$	$1 \text{ stoch. } \nabla, 1 \text{ proj.}$
$f = \frac{1}{m} \sum_i f_i$ f_i smooth strong. conv.	SVRG	—	$(m + \kappa) \log\left(\frac{1}{\varepsilon}\right)$	$1 \text{ stoch. } \nabla$

Adaptive learning rates

Adaptive gradient (AdaGrad)

- Update equations:

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

where $\mathbf{Q}^{(t)} = \sum_{s=1}^t \nabla f_s(\mathbf{w}^{(s)}) \nabla f_s(\mathbf{w}^{(s)})^\top$

- 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:

$$\bullet \quad \mathbf{q}^{(t+1)} = \beta \mathbf{q}^{(t)} + (1 - \beta) \text{diag}\left(\nabla f_t(\mathbf{w}^{(t)}) \nabla f_t(\mathbf{w}^{(t)})^\top\right)$$

$$\bullet \quad \mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \text{diag}\left(1/\sqrt{q_1^{(t)} + \epsilon}, \dots, 1/\sqrt{q_n^{(t)} + \epsilon}\right) \nabla f_t(\mathbf{w}^{(t)})$$

- 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:
 - $\mathbf{g}^{(t+1)} = \beta \mathbf{g}^{(t)} + (1 - \beta) \nabla f_t(\mathbf{w}^{(t)})$
 - $\mathbf{q}^{(t+1)} = \beta \mathbf{q}^{(t)} + (1 - \beta) \text{diag}\left(\nabla f_t(\mathbf{w}^{(t)}) \nabla f_t(\mathbf{w}^{(t)})^\top\right)$
 - $\mathbf{v}^{(t+1)} = \mu \mathbf{v}^{(t)} - \eta \text{diag}\left(1/\sqrt{q_1^{(t)} - (g_1^{(t)})^2 + \epsilon}, \dots, 1/\sqrt{q_n^{(t)} - (g_n^{(t)})^2 + \epsilon}\right) \nabla f_t(\mathbf{w}^{(t)})$
 - $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{v}^{(t+1)}$

[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 (average gradient) / $\sqrt{(\text{average gradient norm})}$

[Kingma and Ba, 2015]

ADAM pseudo-code

- Initialization: $\mathbf{g}^{(0)} = 0, \mathbf{q}^{(0)} = 0$

- For $t = 1, 2, \dots, T$:

$$\mathbf{g}^{(t+1)} = \beta_1 \mathbf{g}^{(t)} + (1 - \beta_1) \nabla f_t(\mathbf{w}^{(t)}) \quad // 1^{\text{st}} \text{ moment}$$

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

$$\hat{\mathbf{g}}^{(t)} = \frac{1}{1 - \beta_1^t} \mathbf{g}^{(t)} \quad // \text{bias correction}$$

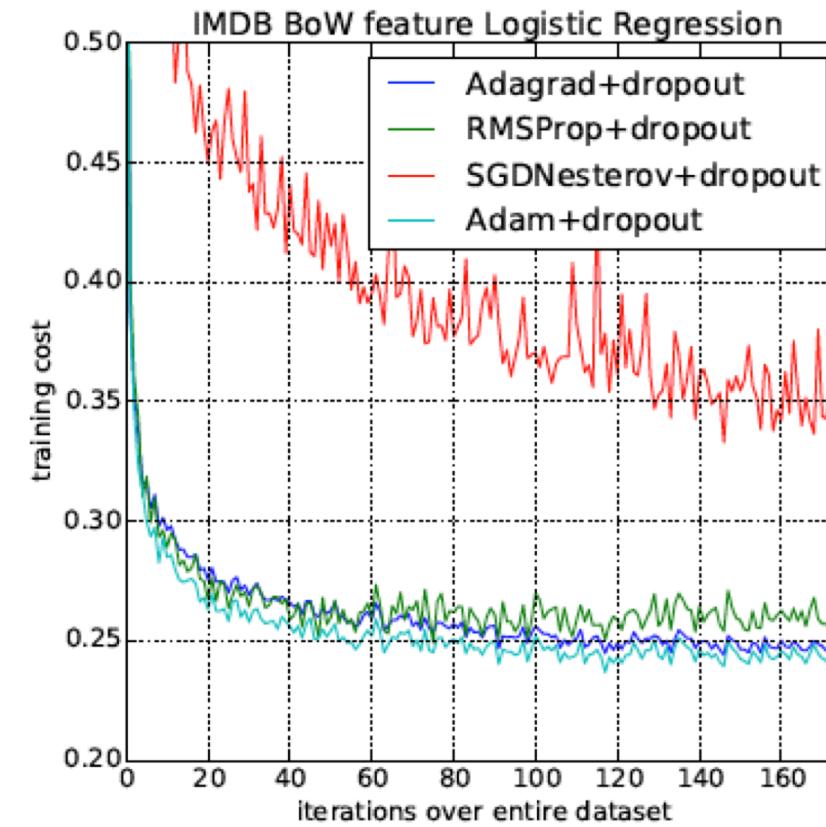
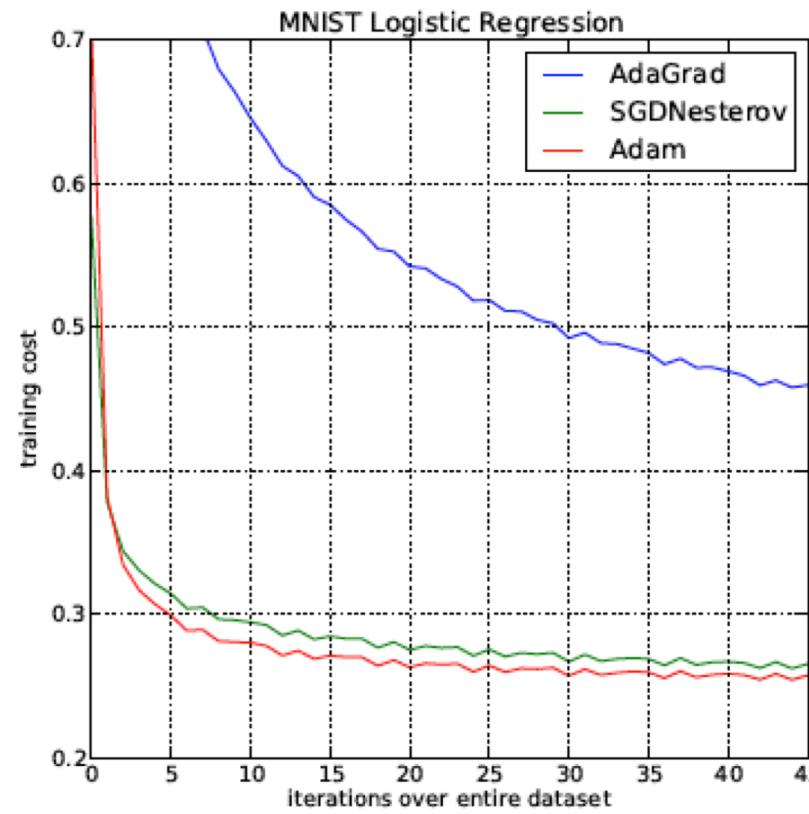
$$\hat{\mathbf{q}}^{(t)} = \frac{1}{1 - \beta_2^t} \mathbf{q}^{(t)} \quad // \text{bias correction}$$

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

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 $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \Delta^{(t+1)}$ where

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

- The 1st and 2nd moment estimates satisfy:

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

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

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

Parameter vector updates: bounded increments

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

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

- Exercise: check this

Bias correction factor

- Facts:

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

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

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

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

which explains the bias correction factor

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

Convergence analysis: regret bound

- **Regret** at time step T defined by:

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

where

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

Regret bound theorem

Thm. Assumptions:

- $f: \mathbf{R}^n \rightarrow \mathbf{R}$ convex function and $\|f(\mathbf{w})\|_p \leq \mu_p$ for all $\mathbf{w} \in \mathbf{R}^n$ for $p = 2, \infty$
- $\|\mathbf{w}^{(t)} - \mathbf{w}^{(s)}\|_p \leq \delta_p$ for all $s, t \in \{1, \dots, T\}$ 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 $R(T) \leq \frac{\delta_2}{2\eta(1-\beta_1)} \sum_{i=1}^n \sqrt{T \hat{q}_i^{(T)}} +$

$$+ \frac{\eta(1+\beta_1)\mu_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^n \left\| \left(\frac{\partial}{\partial \mathbf{w}_i} f_1(\mathbf{w}^{(1)}), \dots, \frac{\partial}{\partial \mathbf{w}_i} f_T(\mathbf{w}^{(T)}) \right) \right\| = O(\sqrt{T})$$
$$+ n \frac{\delta_\infty^2 \mu_\infty}{2\eta(1-\beta_1)(1-\lambda)^2}$$

Backpropagation

What is backpropagation?

- Backpropagation: a method for computing the gradient vector of a loss function for a feedforward neural network for a given input point
- Introduced by Rumelhart, Hinton and Williams in 1986
- Commonly used for training neural networks

Gradient of a loss function

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

Jacobian matrices

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

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

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

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

Chain rule

- The chain rule for computing derivative of a composition of functions can be written in terms of the Jacobian
- Let $\textcolor{blue}{f}: \mathbf{R}^n \rightarrow \mathbf{R}^m$ and $\textcolor{red}{g}: \mathbf{R}^k \rightarrow \mathbf{R}^n$
- Let $\textcolor{blue}{f} \circ \textcolor{red}{g}$ denote the composition: $\textcolor{blue}{f} \circ \textcolor{red}{g}(x) = \textcolor{blue}{f}(\textcolor{red}{g}(x))$
- Then: $J_{\textcolor{violet}{w}}(\textcolor{blue}{f} \circ \textcolor{red}{g}) = J_{\textcolor{red}{g}(\textcolor{violet}{w})}(\textcolor{blue}{f})J_{\textcolor{violet}{w}}(\textcolor{red}{g})$

Backpropagation by a simple example

- Loss function f

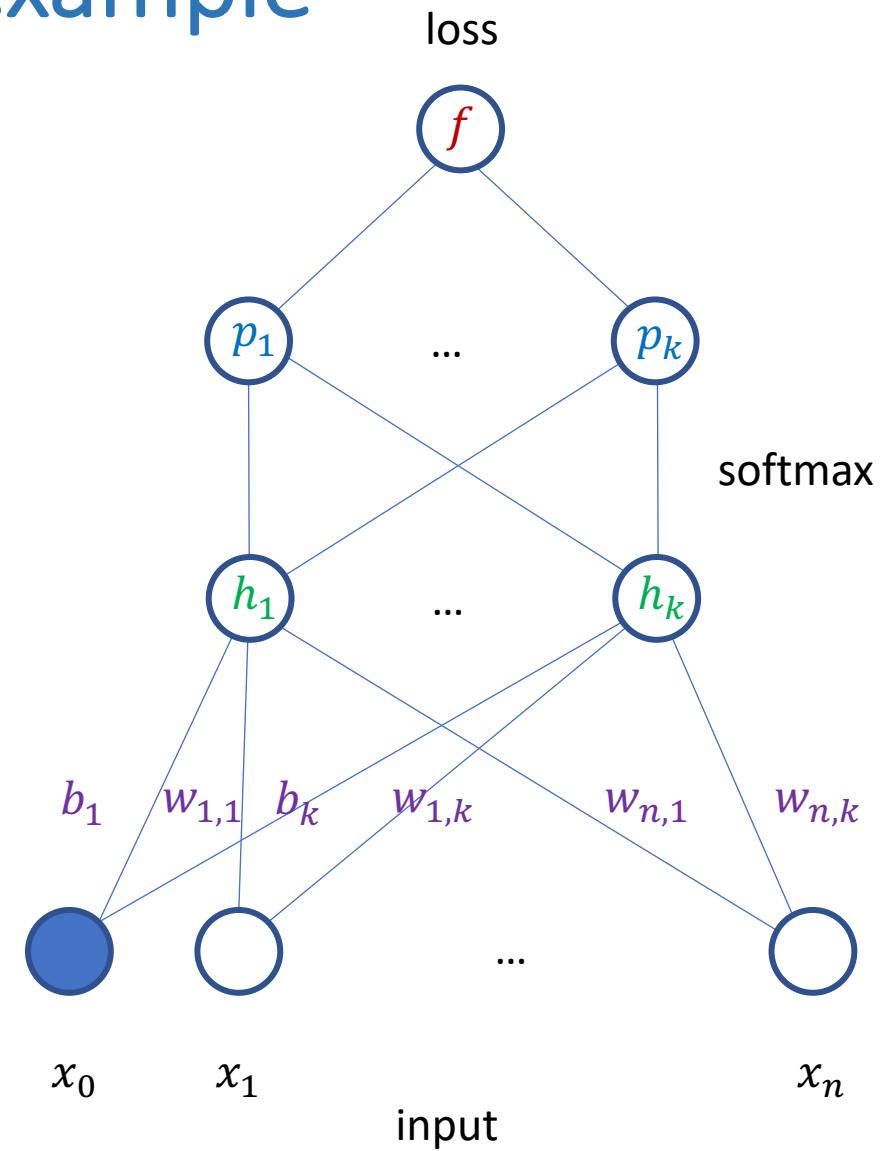
- Chain rule:

$$J_w(f \circ p \circ h) = \boxed{J_p(f) J_h(p)} J_w(h)$$

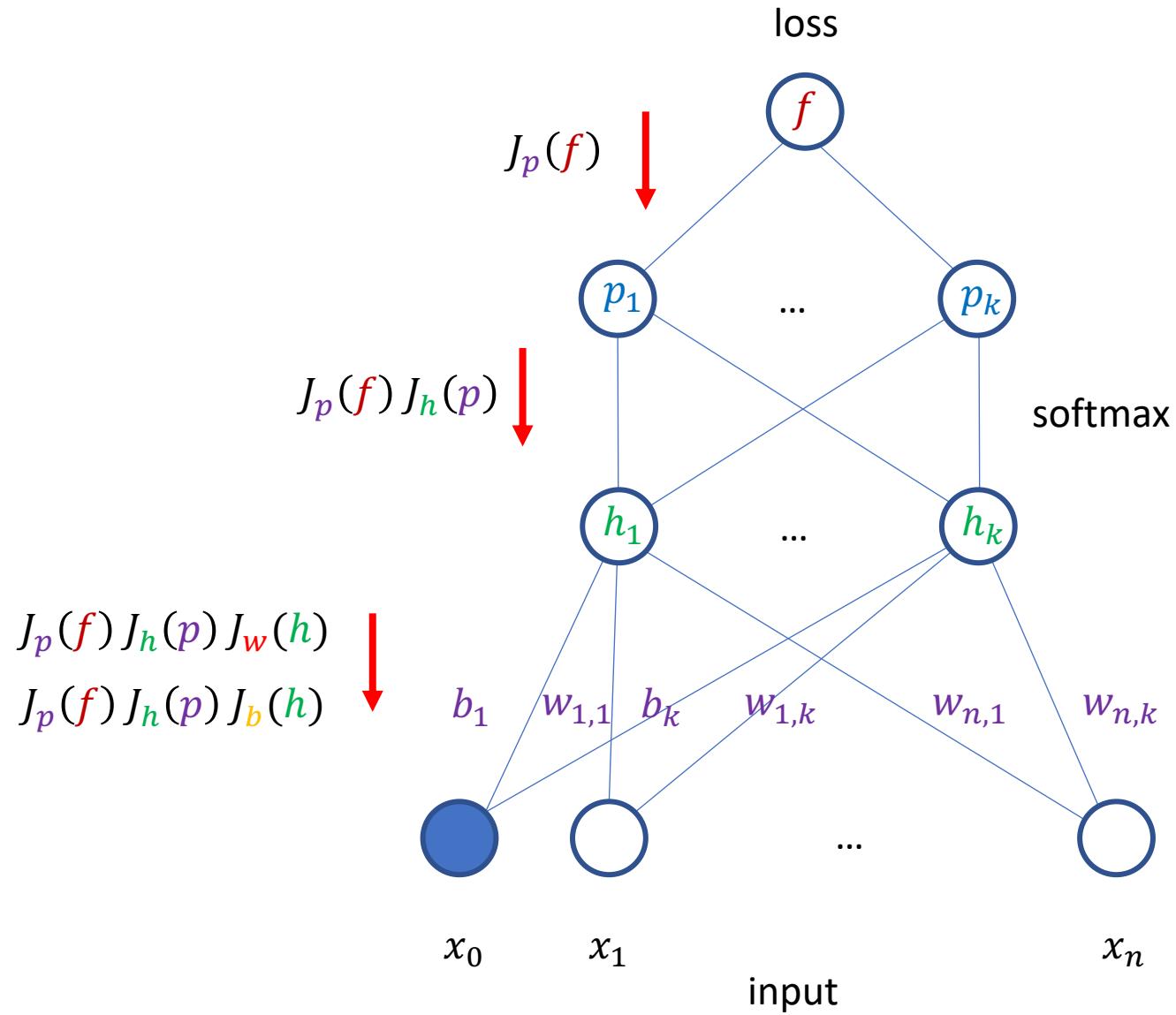
$$J_b(f \circ p \circ h) = \boxed{J_p(f) J_h(p)} J_b(h)$$

common terms

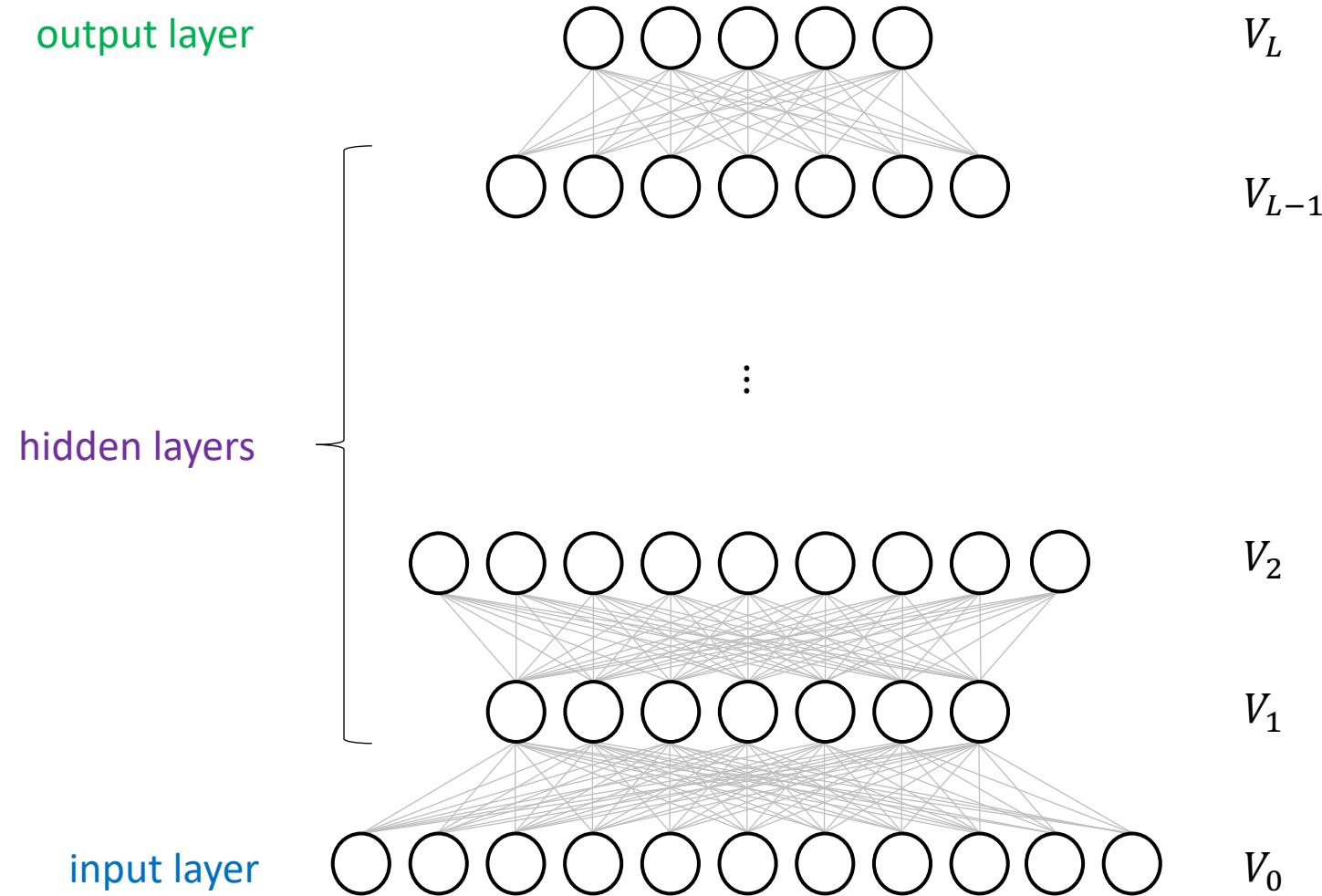
- Don't repeat shared compute!



Backpropagation by a simple example (cont'd)



Backpropagation explained



Function composition representation

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

$$\mathbf{y}^{(1)} = a_1(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)})$$

$$\mathbf{y}^{(2)} = a_2(\mathbf{W}^{(1)}\mathbf{y}^{(1)} + \mathbf{b}^{(1)})$$

⋮

$$\mathbf{y}^{(L-1)} = a_{L-1}(\mathbf{W}^{(L-2)}\mathbf{y}^{(L-2)} + \mathbf{b}^{(L-2)})$$

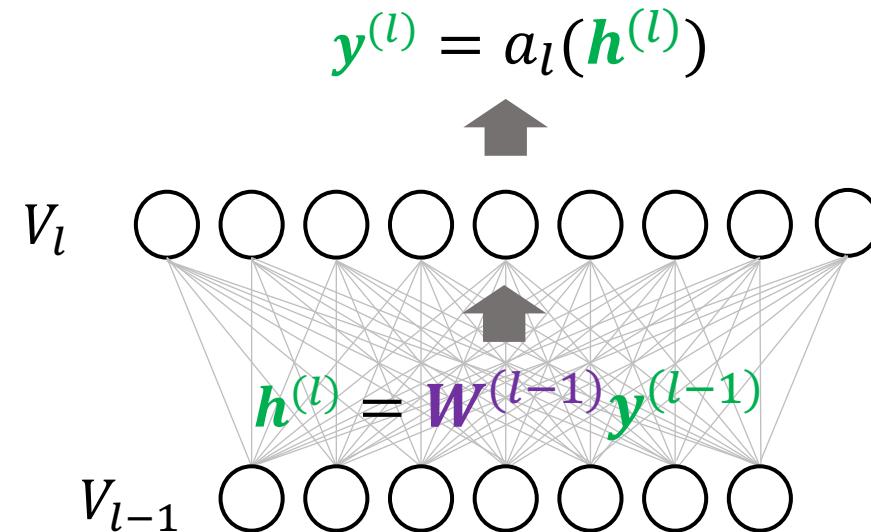
$$\mathbf{y} = \mathbf{W}^{(L-1)}\mathbf{y}^{(L-1)} + \mathbf{b}^{(L-1)}$$

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

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

Backpropagation explained (cont'd)

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



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

Backpropagation explained (cont'd)

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

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

- Change to vector notation:

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

$$\begin{bmatrix} 1 \\ 2 \\ \vdots \\ n_l \end{bmatrix} \mathbf{y}^{(l-1)} = \begin{bmatrix} \mathbf{Y}^{(l-1)} \\ \mathbf{w}^{(l-1)} \end{bmatrix}$$

The diagram illustrates the multiplication of two vectors. On the left, a column vector $\mathbf{w}^{(l-1)}$ is shown with entries 1, 2, ..., n_l . It is multiplied by a row vector $\mathbf{Y}^{(l-1)}$, which is represented as a stack of green rectangles. The result of the multiplication is a single scalar value, represented by a small green rectangle.

Backpropagation explained (cont'd)

- By the chain rule

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

$$\Leftrightarrow J_{\mathbf{w}^{(l-1)}}(\mathbf{f}_l) = (\delta_1^{(l)} a'_l(\mathbf{h}_1^{(l)}) (\mathbf{y}^{(l-1)})^\top, \dots, \delta_{n_l}^{(l)} a'_l(\mathbf{h}_{n_l}^{(l)}) (\mathbf{y}^{(l-1)})^\top)$$

Recursive computation of δ_t

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

Summary of key steps

- Forward propagation:
 - Compute $\mathbf{y}^{(0)}, \mathbf{h}^{(1)}, \mathbf{y}^{(1)}, \dots, \mathbf{h}^{(L)}, \mathbf{y}^{(L)}$ by forward computation from input to output of the network
- Backward propagation:
 - Compute $\boldsymbol{\delta}^{(L)}, \boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(1)}$ by backward computation from output to input of the network
- Computing partial gradients: having computed $\mathbf{h}^{(l)}, \mathbf{y}^{(l-1)}$ and $\boldsymbol{\delta}^{(l)}$, the gradients $J_{\mathbf{w}_{l-1}}(\mathbf{f}_l)$ are given by

$$J_{\mathbf{w}_{l-1}}(\mathbf{f}_l) = (\delta_1^{(l)} a'(\mathbf{h}_1^{(l)}) (\mathbf{y}^{(l-1)})^\top, \dots, \delta_{n_l}^{(l)} a'(\mathbf{h}_{n_l}^{(l)}) (\mathbf{y}^{(l-1)})^\top)$$

Dropout

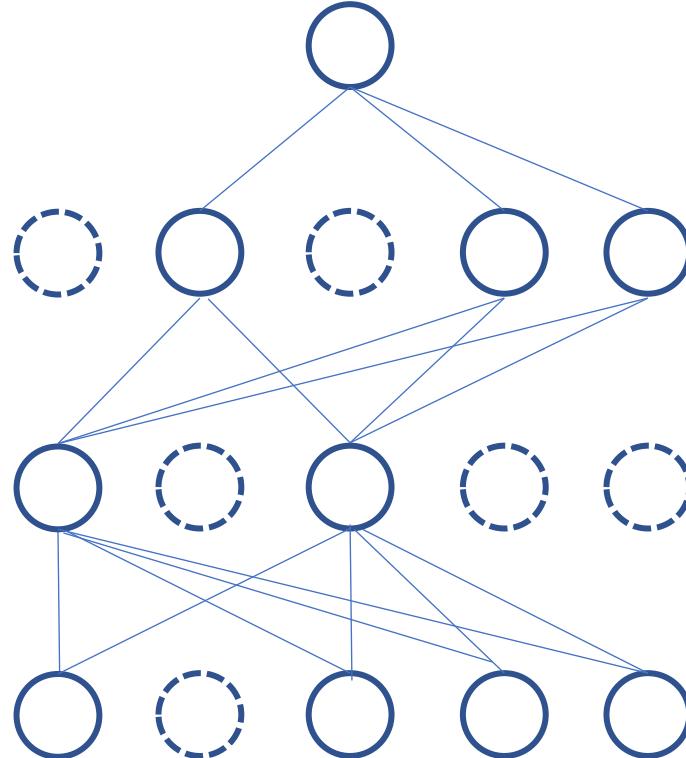
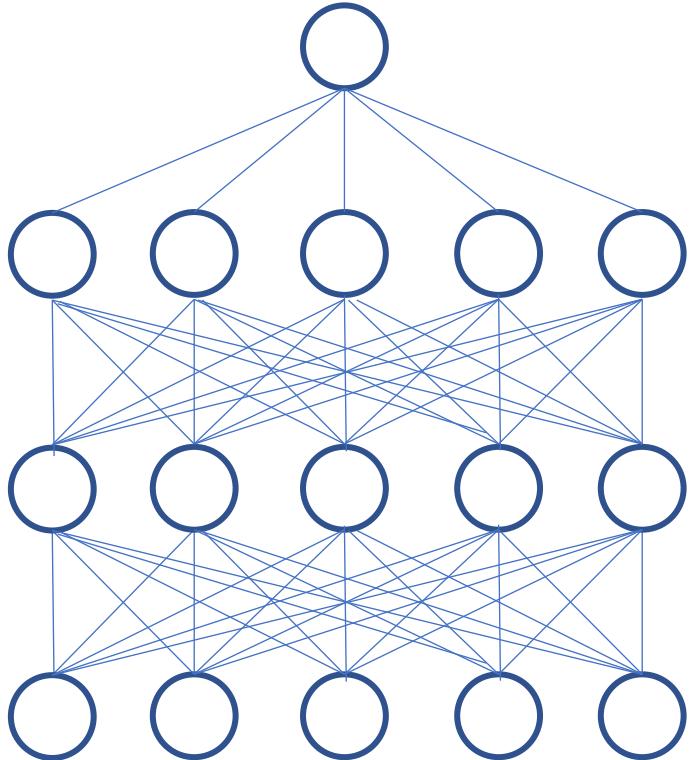
Dropout

- 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 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

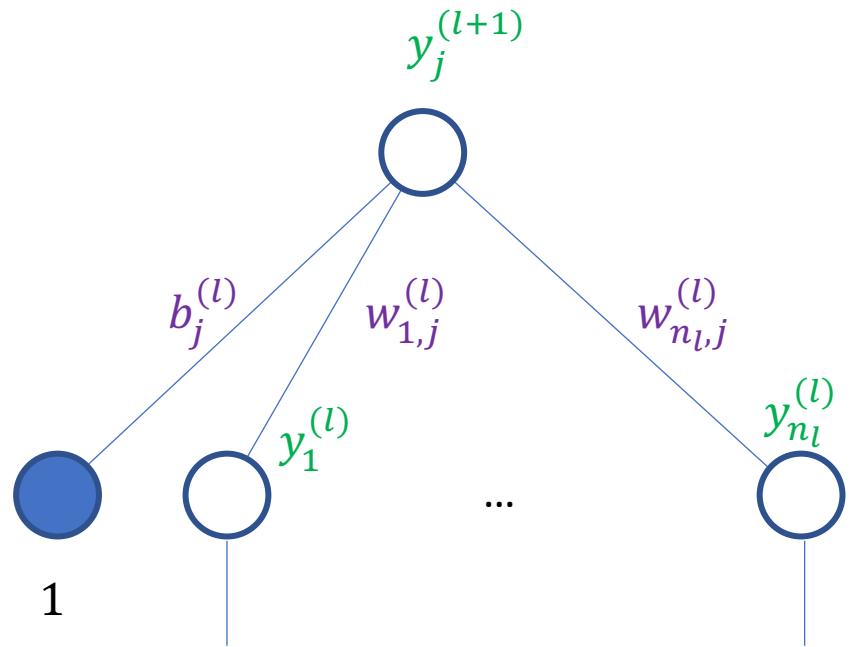
Function mapping: $\mathbf{x} \mapsto \mathbf{y}$

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

For layers $0 < l < L$:

$$\mathbf{y}^{(l+1)} = a_l(\mathbf{W}^{(l)}\mathbf{y}^{(l)} + \mathbf{b}^{(l)})$$

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



Feedforward neural network with dropout

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

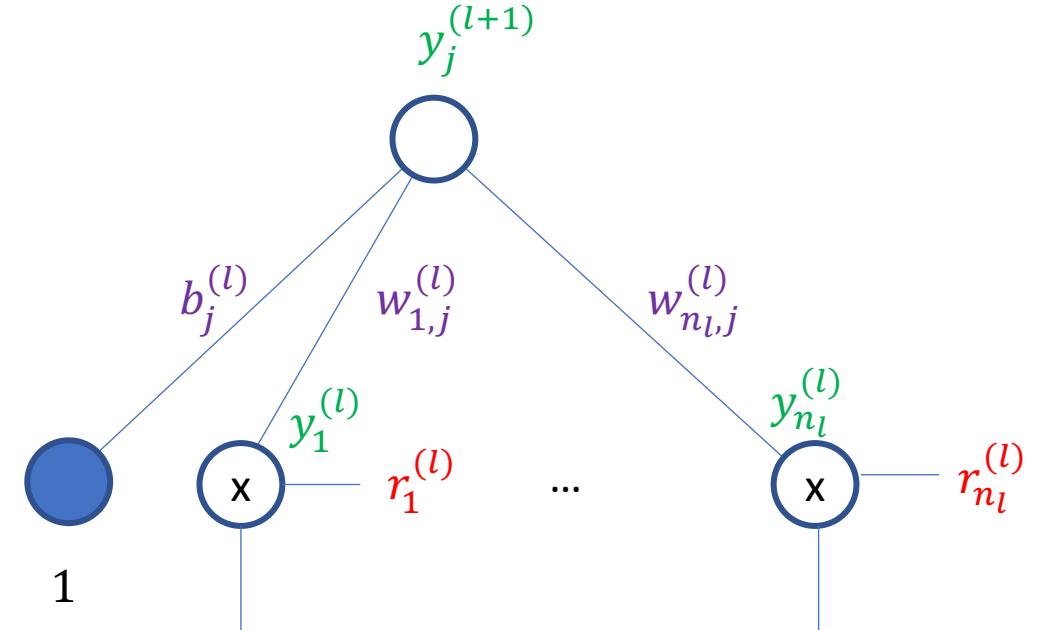
$$\mathbf{y}^{(0)} = \mathbf{x}$$

For $0 < l < L$:

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

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

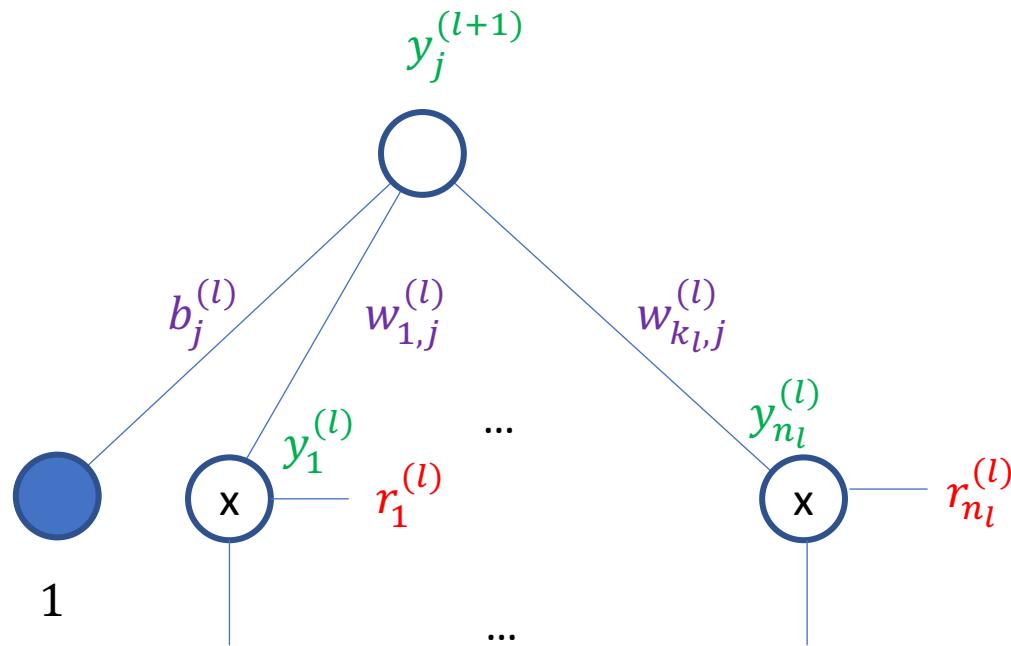
$$\mathbf{y} = \mathbf{y}^{(L)}$$



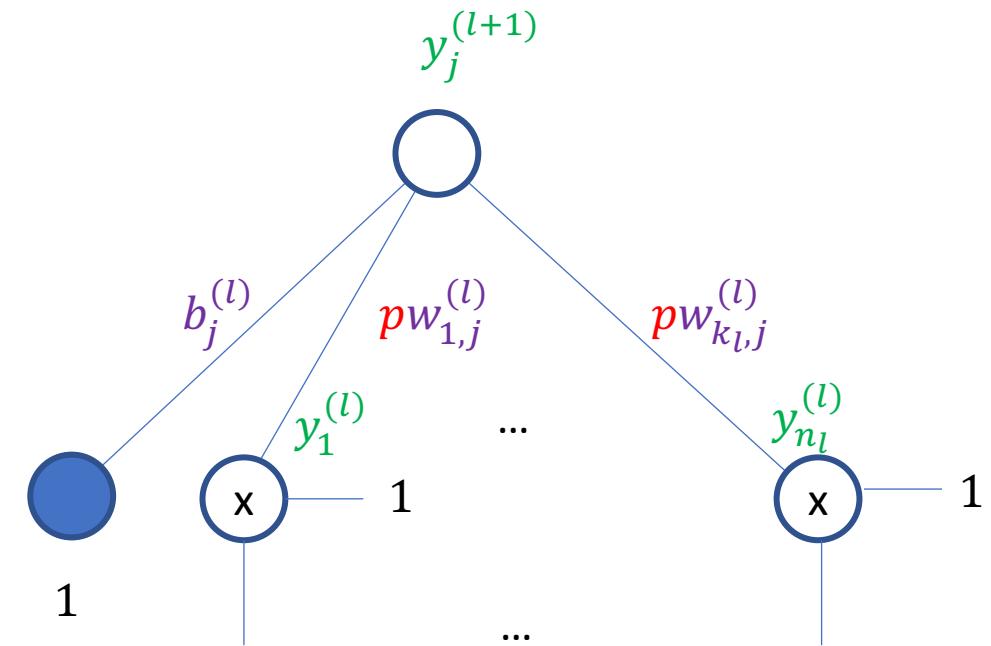
Learning dropout neural networks

- 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