

1 Probability

Sum Rule $P(X = x_i) = \sum_{j=1}^J p(X = x_i, Y = y_j)$

Product rule $P(X, Y) = P(Y|X)P(X)$

Independence $P(X, Y) = P(X)P(Y)$

Bayes' Rule $P(Y|X) = \frac{P(X|Y)P(Y)}{\sum_{i=1}^k P(X|Y_i)P(Y_i)}$

Cond. Ind. $X \perp\!\!\!\perp Y|Z \Rightarrow P(X, Y|Z) = P(X|Z)P(Y|Z)$

Cond. Ind. $X \perp\!\!\!\perp Y|Z \Rightarrow P(X|Y, Z) = P(X|Z)$

$E[X] = \int_X f_X(t) dt =: \mu_X$

$\text{Cov}(X, Y) = E_{x,y}[X - E_x[X]](Y - E_y[Y])$

$\text{Cov}(X, Y) := \text{Cov}(X, X) = \text{Var}[X]$

X, Y independent $\Rightarrow \text{Cov}(X, Y) = 0$

$XX^T \geq 0$ (symmetric positive semidefinite)

$\text{Var}[X] = E[X^2] - (E[X])^2$

$\text{Var}[\mathbf{A}X] = \text{Var}[\mathbf{A}]X + \text{Var}[X] \quad \text{Var}[aX + b] = a^2 \text{Var}[X]$

$\text{Var}[\sum_{i=1}^n a_i X_i] = \sum_{i=1}^n a_i^2 \text{Var}[X_i] + 2 \sum_{i < j} a_i a_j \text{Cov}(X_i, X_j)$

$\text{Var}[\sum_{i=1}^n a_i X_i] = \sum_{i=1}^n a_i^2 \text{Var}[X_i] + \sum_{i \neq j} a_i a_j \text{Cov}(X_i, X_j)$

$\frac{\partial}{\partial t} P(X \leq t) = -\frac{\partial}{\partial t} F_X(t) = f_X(t)$ (derivative of c.d.f. is p.d.f.)

$f_{\alpha}(y) = \frac{1}{\alpha} f_Y(\frac{y}{\alpha})$

T. The moment generating function (MGF) $\psi_X(t) = E[e^{tX}]$ characterizes the distr. of a rv $p(x)$. Then the entropy of \mathbf{X}

$H(p) := -p^2 + (1-p) \quad N(\mu, \sigma) := \exp(\mu t + \frac{1}{2}\sigma^2 t^2)$

$\text{Bin}(n, p) := (pe + (1-p))^n \quad \text{Gam}(\alpha, \beta) := (\frac{1}{\alpha} \beta)^{\alpha} \text{ for } t < 1/\beta$

$\text{Pois}(\lambda) := e^{\lambda(e^{-t}-1)}$

T. If X_1, \dots, X_n are indep. rvs with MGFs $M_{X_i}(t) = E[e^{tX_i}]$, then the MGF of $Y = \sum_{i=1}^n a_i X_i$ is $M_Y(t) = \prod_{i=1}^n M_{X_i}(a_i t)$.

T. Let X, Y be indep., then the p.d.f. of $Z = X + Y$ is the conv. of the p.d.f. of X and Y : $f_Z(z) = \int_{-\infty}^z f_X(x) f_Y(z-x) dt$

T. $\text{f}_R(x-t) f_Y(t) dt$

N. $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, $\Sigma_{11} \in \mathbb{R}^{d \times d}$ p.s.d., $\Sigma_{12} \in \mathbb{R}^{d \times d}$ p.s.d., $\Sigma_{21} \in \mathbb{R}^{d \times d}$ p.s.d.

T. (Chebyshev) Let X be a rv with $E[X] = \mu$ and variance $\text{Var}[X] = \sigma^2 < \infty$. Then for any $\epsilon > 0$, we have $P(|X - \mu| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2}$.

2 Analysis

Log-Trick (Identity): $\nabla_\theta [p_\theta(\mathbf{x})] = p_\theta(\mathbf{x}) \nabla_\theta [\log(p_\theta(\mathbf{x}))]$

T. (Cauchy-Schwarz)

$\nabla_\mathbf{u} \cdot \nabla_\mathbf{v} : V : \langle \mathbf{u}, \mathbf{v} \rangle \leq \|\mathbf{u}\| \|\mathbf{v}\|$

$\nabla_\mathbf{u} \cdot \nabla_\mathbf{v} : V : 0 \leq \langle \mathbf{u}, \mathbf{v} \rangle \leq \|\mathbf{u}\| \|\mathbf{v}\|$

Special case: $(\sum x_i y_i)^2 \leq (\sum x_i^2)(\sum y_i^2)$.

Special case: $E[X]^2 \leq E[X^2] E[Y^2]$.

D. (Convex Set) A set $S \subseteq \mathbb{R}^d$ is called convex if $\forall \mathbf{x}, \mathbf{x}' \in S, \forall \lambda \in [0, 1]: \lambda \mathbf{x} + (1 - \lambda) \mathbf{x}' \in S$.

Com. Any point on the line between two points is within the set. \mathbf{r}^2 is convex.

D. (Convex Function) A function $f: S \rightarrow \mathbb{R}$ defined on a convex set $S \subseteq \mathbb{R}^d$ is called convex if

$\forall \mathbf{x}, \mathbf{x}' \in S, \lambda \in [0, 1]: f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{x}') \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{x}')$

Com. A function is strictly convex if the line segment between any two points on the graph of the function lies strictly above the graph.

T. (Properties of Convex Functions)

- $f(y) \geq f(x) + \nabla f(x)^T(y - x)$
- $f'(x) \geq 0$
- Local minima are global minima, strictly convex functions have a unique global minimum
- If f, g are convex then $\alpha f + \beta g$ is convex for $\alpha, \beta \geq 0$
- If f, g are convex then $\max(f, g)$ is convex
- If f is convex and g is convex and non-decreasing then $g \circ f$ is convex

T. (Taylor-Lagrange Formula)

$f(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + \sum_{k=0}^n \frac{f^{(n+1)}(x-t)}{n!} dt$

T. (Jensen)

f convex/concave, $\forall i: \lambda_i \geq 0, \sum_i \lambda_i = 1$

$f(\sum_i \lambda_i x_i) \leq \sum_i \lambda_i f(x_i)$

Special case: $f(E[X]) \leq E[f(X)]$.

D. (Lagrangian Formulation) of $\arg \max_{x,y} f(x, y)$ s.t. $g(x, y) = c: L(x, y, \gamma) = f(x, y) - \gamma(g(x, y) - c)$

$x^* = \mathbf{x}^* - \text{Hess}(f(x^*))^{-1} \nabla_{\mathbf{x}} f(x^*)$.

3 Linear Algebra

T. (Sylvester's Criterion) A $d \times d$ matrix is positive semi-definite if and only if all the upper left $k \times k$ for $k = 1, \dots, d$ have a positive determinant.

Negative definite: $\det < 0$ for all odd-sized minors, and $\det > 0$ for all even-sized minors

Otherwise: indefinite.

D. (Trace) of $A \in \mathbb{R}^{n \times n}$ is $\text{Tr}(A) = \sum_{i=1}^n a_{ii}$.

4 Derivatives

4.1 Numerator and Denominator Convention

Jacobian Layout Convention We use the denominator-layout. For a vector-valued function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ we define

$$\left(\frac{\partial f}{\partial \mathbf{x}} \right)_{ij} := \frac{\partial f_j}{\partial x_i}, \quad \frac{\partial f}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}.$$

Hence, gradients of scalar-valued functions are column vectors, and the chain rule takes the form

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{g}(\mathbf{x}))] = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}.$$

Remark. Sometimes the numerator-layout is used, where the Jacobian is defined as $(J_f)_{ij} = \frac{\partial f_i}{\partial x_j} |_{\mathbf{x}=\mathbf{g}(\mathbf{x})} \in \mathbb{R}^{m \times n}$. The two conventions are related by transposition.

4.2 Scalar-by-Vector

Denominator Convention

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial}{\partial \mathbf{x}} + \mathbf{v}(\mathbf{x}) \frac{\partial u}{\partial \mathbf{x}}$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x})] = \frac{\partial u}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}}{\partial \mathbf{x}} \mathbf{g}(\mathbf{x}) + \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) = \mathbf{J}_f(\mathbf{g}(\mathbf{x})) + \mathbf{J}_g(\mathbf{x})$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})] \mathbf{A}[\mathbf{g}(\mathbf{x})] = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \mathbf{A}[\mathbf{g}(\mathbf{x})] + \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{A}[\mathbf{f}(\mathbf{x})]$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{A}[\mathbf{x}]] = \frac{\partial \mathbf{A}}{\partial \mathbf{x}} [\mathbf{x}] = \mathbf{A}[\mathbf{A}^T \mathbf{x}]$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{A}^T \mathbf{x}] = \frac{\partial \mathbf{A}}{\partial \mathbf{x}} [\mathbf{x}] = \mathbf{A}^T \mathbf{x}$$

$$\frac{\partial}{\partial \mathbf{x}} [(\mathbf{A} + \mathbf{b})^T \mathbf{C}(\mathbf{D}\mathbf{x} + \mathbf{e})] = \mathbf{D}^T \mathbf{C}^T (\mathbf{A} + \mathbf{b}) + \mathbf{A}^T \mathbf{C}(\mathbf{D}\mathbf{x} + \mathbf{e})$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})] \mathbf{f}(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})] = 2 \frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})] \mathbf{f}(\mathbf{x}) = 2 \mathbf{J}_f(\mathbf{x})$$

Linear Regression: $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$

$$\ell(w) = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2$$

Analytically solvable: $w^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

Ridge Regression: $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ with ℓ_2 regularization

$$\ell(w) = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2 + \lambda \|w\|^2$$

Analytically solvable: $w^* = (\mathbf{X}^T \mathbf{X} + I)^{-1} \mathbf{X}^T \mathbf{y}$

Lasso Regression: $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$ with ℓ_1 regularization

$$\ell(w) = \frac{1}{2n} \sum_{i=1}^n (y_i - f(x_i))^2 + \lambda \|w\|_1$$

Then we define

4.3 Vector-by-Vector

A. $\mathbf{C}, \mathbf{D}, \mathbf{a}, \mathbf{b}$ not a function of \mathbf{x} , $\mathbf{f} = \mathbf{f}(\mathbf{x}), \mathbf{g} = \mathbf{g}(\mathbf{x}), \mathbf{h} = \mathbf{h}(\mathbf{x}), \mathbf{u} = \mathbf{u}(\mathbf{x}), \mathbf{v} = \mathbf{v}(\mathbf{x})$

B. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x})$

C. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a} \otimes \mathbf{a}] = \text{diag}(\mathbf{a})$

D. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a} \mathbf{f}(\mathbf{x})] = \mathbf{a} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{a} \mathbf{J}_f$

E. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}] = \mathbf{0}$

F. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}] = \mathbf{I}$

G. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{A}[\mathbf{x}]] = \mathbf{A}$

H. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}^T \mathbf{A}] = \mathbf{A}^T$

I. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{A}^T \mathbf{x}] = \mathbf{A}$

J. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{g}(\mathbf{x}))] = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} =$

4.4 Scalar-by-Matrix

A. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X} \mathbf{b}] = \mathbf{a} \mathbf{b}^T$

B. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X}^T \mathbf{b}] = \mathbf{b} \mathbf{a}^T$

C. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X} \mathbf{b}] = \mathbf{b} \mathbf{a}^T$

D. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X}^T \mathbf{a}] = \mathbf{a}^T \mathbf{X}^T \mathbf{a}$

E. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X} \mathbf{a}] = \mathbf{a}^T \mathbf{X} \mathbf{a}$

4.5 Vector-by-Matrix (Generalized Gradient)

A. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}] = \mathbf{A}$

B. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{A}] = \mathbf{A}^T$

C. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{A}^T \mathbf{X}] = \mathbf{A}$

D. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}] = \mathbf{A}$

E. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T] = \mathbf{A}^T$

F. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}^T$

G. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

H. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

I. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

J. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

K. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

L. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

M. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

N. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

O. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

P. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

Q. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

R. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

S. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

T. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

U. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}] = \mathbf{A}$

V. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

W. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

X. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

Y. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

Z. $\frac{\partial}{\partial \mathbf{x}} [\mathbf{X}^T \mathbf{X}^T \mathbf{X}^T] = \mathbf{A}$

4.6 Logistic Regression: $f(x) = \sigma(w^T x + b)$

It minimizes the cross-entropy loss (negative log-likelihood), which acts as a surrogate loss for the 0/1 classification error.

4.7 Deep Learning

4.7.1 Compositions of Maps

D. (Linear Function) A function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a linear function if the following properties hold

- $\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^n: f(\mathbf{x} + \mathbf{x}') = f(\mathbf{x}) + f(\mathbf{x}')$
- $\forall \mathbf{x} \in \mathbb{R}^n \forall a \in \mathbb{R}: f(a\mathbf{x}) = af(\mathbf{x})$

T. (Montufar et al. 2014) If we process n -dim. inputs through L ReLU layers with widths $m_1, \dots, m_L \in \mathcal{O}(m^n)$. Then \mathbb{R}^n can be partitioned into at most $R(m)$ layers:

$$R(m) \leq \sum_{i=0}^{\min\{m,n\}} \binom{$$

12.6 Efficient Comp. of Convolutional Activities
A naive way to compute the convolution of a signal of length n and a kernel of length m gives an effort of $\mathcal{O}(m \cdot n)$. A faster way is to transform both with the FFT and then just do element-wise multiplication (effort: $\mathcal{O}(n \log n)$). However, this is rarely done in CNNs as the filters usually are small ($m \ll n, m \approx \log(n)$).

12.7 Typical Convolutional Layer Stages

A typical setup of a convolutional layer is as follows:

1. Convolution stage: affine transform
2. Detector stage: nonlinearity (e.g., ReLU)
3. Pooling stage: locally combine activities in some way (avg, ...)

Locality of the item that activated the neurons isn't too important, further we profit from dimensionality reduction. Alternatives: deconvolution with stride. Another thing that turns out to be so is that most of the kernels that are learned resemble a low-pass filter. Hence, when we sub-sample the images most of the information is still contained.

12.8 Pooling

There are min, max, avg, and softmax pooling. Max pooling is the most frequently used one.

D. (Max-Pooling)

- 1D: $i_{k, l}^{\text{max}} = \max\{x_{i+k} \mid 0 \leq k < r\}$
- 2D: $i_{ij, l}^{\text{max}} = \max\{x_{i+k, j+l} \mid 0 \leq k, l < r\}$

12.9 Sub-Sampling (aka "Strides")

Often, it is desirable to reduce the size of the feature maps. That's why sub-sampling was introduced.

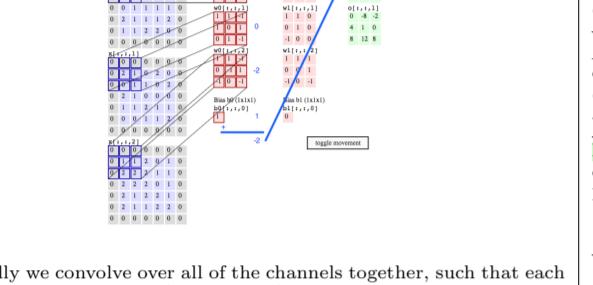
D. (Sub-Sampling)

Hereby the temporal/spatial resolution is reduced.

12.10 Channels

Ex: Here we have

- an input signal that is 2D with 3 channels ($7 \times 7 \times 3$) (image x channels)
- and we want to learn two filters W_0 and W_1 , which each process the 3 channels, and sum the results of the convolutions over each channel leading to a tensor of size $3 \times 3 \times 2$ (convolution result x num convolutions)



Usually we convolve over all of the channels together, such that each convolution has the information of all channels at its disposition and the order of the channels hence doesn't matter.

12.11 CNNs in Computer Vision

So the typical use of convolution that we have in vision is: a sequence of convolutions

1. that reduce the spatial dimensions (sub-sampling)
2. that increase the number of channels

The deeper we go in the network, we transform the spatial information into a semantic representation. Usually, most of the parameters lie in the fully connected layers

12.12 Famous CNN Architectures

12.12.1 LeNet, 1989

2 Convolutional Layers + 2 Fully-connected layers

12.12.2 LeNet5

3 Convolutional Layers (with max-pool subsampling) + 1 Fully connected layer

12.12.3 AlexNet

ImageNet: similar to LeNet5, just deeper and using GPU (performance breakthrough)

12.12.4 Inception Module

Now, a problem that arises with this ever deeper and deeper channel is that the filters at every layer were getting longer and longer and lots of their coefficients were becoming zero (so no information flowing through). So, Arora et al. came up with the idea of an inception module.

12.14 Optimization Challenges in NNs: Curvatures

A challenge in NNs arises when we have sharp non-linearities, then there are two approaches to solve this

- one is to be very conservative and only do small update steps by choosing a very small learning rate.
- or we are courageous and due huge jumps as depicted in the image.

Typical approaches are to clip the gradient when it gets too large, or use a decreasing learning rate (in terms of time).

Now, the problem is not that the cliff is very steep. The problem is the curvature. Because when we take the gradient, the gradient is actually constant on the wall of the cliff. Let's have a look at this through some equations.

Now, let's evaluate what the risk is at some point, plus some gradient step. If we do the 2nd-order Taylor expansion of that, then we get

$$x^T x = \sigma(\mathbf{w}_{\mathbf{x}, i}), \quad \mathbf{w} \in \mathbb{R}^{m \times k}.$$

So it uses a 1x1 filter over the k input channels (which is actually not a convolution), aka "network within a network".

12.12.5 Google Inception Network

The Google Inception Network uses many layers of this inception module along with some other tricks

- dimensionality reduction through the inception modules
- convolutions at various sizes, as different filter sizes turned out to be useful
- a max-pooling of the previous layer, and a dimensionality reduction of the result
- 1x1 filters for dimension reduction before convolving with larger kernels
- then all these informations are passed to the next layer
- gradient shortcuts: connect softmax layer at intermediate stages to have the gradient flow until the beginnings of the network
- decomposition of convolution kernels for computational performance
- all-in-all the dimensionality reductions improved the efficiency

12.13 Networks Similar to CNNs

A locally connected network has the same connections that a CNN would have, however, the parameters are not shared. So the output nodes do not connect to all nodes, just to a set of input nodes that are considered "near" (locally connected).

12.14 Comparison of #Parameters (CNNs, FC, LC)

Ex: Input image $m \times n \times c$ (c = number of channels)

Convolution kernels: $p \times q$ (valid padding and stride 1)

Then the convolved image has dimensions (assuming stride 1)

- valid padding (only where it's defined): $(m - p + 1) \times (n - q + 1) \times K$
- same padding (extend image with constant): $m \times n$ where the extended image has size $(m + 2p) \times (n + 2q)$.

13 Optimization

13.1 Objectives as Expectations

$$\mathbb{E}_\theta[\mathcal{R}(\mathbf{D})] = \mathbb{E}_{S_N \sim P_D} [\nabla_\theta \mathcal{R}(S_N)] = \mathbb{E} \left[\frac{1}{N} \sum_{i=1}^N \nabla_\theta \mathcal{R}(\theta; \{\mathbf{x}_i, \mathbf{y}_i\}) \right]$$

This is probably so, because we're dealing with large curvatures when reaching (or getting close to) the optimal parameters. So with gradient descent we may not arrive at a critical point of any kind, and this also motivates to use more and more decreasing learning rates, the closer we get to the optimal parameters. Note that this graph was built using the MNIST dataset and some CNN. Note that there exist many architectures where the final gradient was very large, and still they are used in practice, and people are quite happy with them.

13.2 Gradient Descent

$\theta(t+1) = \theta(t) - \eta_t \nabla_\theta \mathcal{R}$

13.3 Gradient Descent: Classic Analysis

In classical machine learning we have a convex objective \mathcal{R} . And we denote

- \mathcal{R}^* as the minimum of \mathcal{R}
- θ^* as the optimal set of parameters (the minimizer of \mathcal{R})

So we have $\nabla_\theta \mathcal{R} \neq 0^*$; $\mathcal{R}^* := \mathcal{R}(\theta^*) \leq \mathcal{R}(\theta)$.

13.4 Strictly Convex Objective

→ objective has only one (a unique) minimum.

$$\nabla_\theta \mathcal{R} \neq 0^* \Rightarrow \mathcal{R}^* := \mathcal{R}(\theta^*) < \mathcal{R}(\theta)$$

13.5 L-Lipschitz Continuous Function Given two metric spaces (X, d_X) and (Y, d_Y) , a function $f: X \rightarrow Y$ is called Lipschitz continuous, if there exists a real constant $L \in \mathbb{R}_0^+$ (*Lipschitz constant*), such that

Neural network cost functions can have many local minima and/or saddle points - and this is typical. Gradient descent can get stuck. Questions that have been looked at are

• Are local minima a practical issue? Sometimes not: Gori & Tesi, 1992

• Do local minima even exist? Sometimes not (auto encoder): Baldi & Hornik, 1989

• Are local minima really worse? often not (large networks): e.g., Choromanska et al., 2015

• Can we understand the learning dynamics? Deep linear case has similarities with non-linear case, e.g., Saxe et al., 2013

Com. So, the L tells us how big the gradient could be.

We have the following chain of inclusions for functions over a closed and bounded (i.e., compact) subset of the real line.

Continuously differentiable \Leftrightarrow Lipschitz continuous \subseteq (Uniformly) continuous

Com. Usually the convergence is actually not so much of an issue. It turns out that when we go to very high dimensions, the number of local minima VS the number of saddle points (gradient is zero, but non-optimal) is very small - so we're much more likely to end up in a saddle point. However, in practice, if we do SGD there is some stochasticity that will make our gradient move. Then, after waiting for a while we'll exit the saddle point.

Now, next we'll look at the insights that were gained by the paper of Saxe.

13.6 Least Squares: Single Layer Lin. Netw.

Let's assume that we have some inputs $\mathbf{x} \in \mathbb{R}^n = \mathcal{X}$ and some outputs $\mathbf{y} \in \mathbb{R}^m = \mathcal{Y}$, and we have a very simple architecture, that will take our input \mathbf{x} and transform it to some output in \mathcal{Y} the output space

$$\mathcal{R}(\theta(t)) - \mathcal{R}^* \leq \frac{2L}{t+1} \|\theta(0) - \theta^*\|^2 \in \mathcal{O}(t^{-1})$$

Com. So for our risk function \mathcal{R} , we want that the gradient of it is $\nabla_\theta \mathcal{R}: \Omega \rightarrow \Omega$ where $\Omega = \mathbb{R}^n$

is L -Lipschitz continuous, if it holds that

$$\forall \theta_1, \theta_2 \in \Omega: \|\nabla_\theta \mathcal{R}(\theta_1) - \nabla_\theta \mathcal{R}(\theta_2)\| \leq L \|\theta_1 - \theta_2\|$$

Com. So, the L tells us how big the gradient could be.

We have the following chain of inclusions for functions over a closed and bounded (i.e., compact) subset of the real line.

Continuously differentiable \Leftrightarrow Lipschitz continuous \subseteq (Uniformly) continuous

Com. Usually the convergence is actually not so much of an issue. It turns out that when we go to very high dimensions, the number of local minima VS the number of saddle points (gradient is zero, but non-optimal) is very small - so we're much more likely to end up in a saddle point. However, in practice, if we do SGD there is some stochasticity that will make our gradient move. Then, after waiting for a while we'll exit the saddle point.

Now, next we'll look at the insights that were gained by the paper of Saxe.

13.7 Optimization Challenges in NNs: Local Minima

At the beginning people were happy when they were doing convex optimization because there was a single optimum and it was reachable. And then when people started using non-convex optimization, they were afraid of getting into non-optimal local minima and getting stuck there.

Neural network cost functions can have many local minima and/or saddle points - and this is typical. Gradient descent can get stuck. This is probably so, because we're dealing with large curvatures when reaching (or getting close to) the optimal parameters. So with gradient descent we may not arrive at a critical point of any kind, and this also motivates to use more and more decreasing learning rates, the closer we get to the optimal parameters. Note that this graph was built using the MNIST dataset and some CNN.

Note that there exist many architectures where the final gradient was very large, and still they are used in practice, and people are quite happy with them.

13.8 Optimization Challenges in NNs: Local Minima

At the beginning people were happy when they were doing convex optimization because there was a single optimum and it was reachable. And then when people started using non-convex optimization, they were afraid of getting into non-optimal local minima and getting stuck there.

Neural network cost functions can have many local minima and/or saddle points - and this is typical. Gradient descent can get stuck. This is probably so, because we're dealing with large curvatures when reaching (or getting close to) the optimal parameters. So with gradient descent we may not arrive at a critical point of any kind, and this also motivates to use more and more decreasing learning rates, the closer we get to the optimal parameters. Note that this graph was built using the MNIST dataset and some CNN.

Note that there exist many architectures where the final gradient was very large, and still they are used in practice, and people are quite happy with them.

13.9 Heuristics

• Curriculum learning and non-uniform sampling of data points → focus on the most relevant examples (Bengio, Louradour, Collobert, Weston, 2009) (DL-Book: 8.7.6) Or increase hardness of tasks (corner-case) as NN improves

• Continuation methods: define a family of (simpler) objective functions and track solutions, gradually change hardness of loss (DL-Book: 8.7.6)

• Heuristics for initialization (DL-Book: 8.4) scale the weights of each layer in a way that at the end of the layer, the data has more or less the same energy (and gradient norms are more or less the same at each layer).

• pre-training (DL-Book: 8.7.4). for better initialization, to avoid local minima (less relevant today).

13.10 Early Stopping

Gradient descent usually evolves solutions from: simple + robust → complex + sensitive. Hence, it makes sense to stop training early (as soon as validation flattens/increases). Also: computationally attractive.

• the constraints do not affect the initial learning (as the weights are assumed to be small at the beginning), so we won't clip the weights. So the constraints only become active, once the weights are large.

• alternatively, we may just constrain the norm of the incoming weights for each unit (so use row-norms for the weight matrices).

This had some practical success in stabilizing the optimization.

13.11 Early Stopping

Gradient descent usually evolves solutions from: simple + robust → complex + sensitive. Hence, it makes sense to stop training early (as soon as validation flattens/increases). Also: computationally attractive.

• the constraints do not affect the initial learning (as the weights are assumed to be small at the beginning), so we won't clip the weights. So the constraints only become active, once the weights are large.

• alternatively, we may just constrain the norm of the incoming weights for each unit (so use row-norms for the weight matrices).

This had some practical success in stabilizing the optimization.

13.12 Early Stopping

Gradient descent usually evolves solutions from: simple + robust → complex + sensitive. Hence, it makes sense to stop training early (as soon as validation flattens/increases). Also: computationally attractive.

• the constraints do not affect the initial learning (as the weights are assumed

$$\sum_j \vec{w}_i^T \vec{x}_j^{t-1} = \mathbb{E}_{Z \sim P(Z)} \left[\sum_j Z_j^{t-1} w_{ij} x_j^{t-1} \right] = \sum_j t_j^{-1} w_{ij} x_j^{t-1}$$

$$= \arg \max_{\theta} \sum_{(i,j) \in C_R} \mathbf{x}_i^T \mathbf{z}_{w_j} - \log \left(\sum_{u \in V} \underbrace{\exp(\mathbf{x}_i^T \mathbf{z}_u)}_{\text{partition function}} \right)$$

So the idea is to use two different latent vectors \mathbf{x}_w and \mathbf{z}_w per word (to allow for the asymmetry for the conditional probability)

$$\theta = (\mathbf{x}_w, \mathbf{z}_w)_{w \in V}$$

Ex. Let's say that at the end we selected each unit with a probability of 0.5. Then when typically when we're finished with training our neural network, we're going to multiply all the weights that we obtained with 0.5 to reduce the contribution of each of the features (since we'll have all of them). So with this trick for the prediction we can just do a single forward pass.

14 Natural Language Processing

Similarities between text and image processing: local information.

Differences between text and image processing: texts have various lengths, texts may have long-term interactions, language is a man-made conception on how to communicate with each other / pictures capture the reality, pictures capture the reality / sentences may mean different things in different contexts

14.1 Word Embeddings

Basic Idea: Map symbols over a vocabulary V to a vector representation = embedding into an (euclidean) vector space (see lookup table in architecture overview).

embedding map: (vocabulary) $V \mapsto \mathbb{R}^d$ (embeddings)

(symbolic) $w \mapsto \mathbf{x}_w$ (quantitative)

word $w \in V \rightarrow$ one-hot $w \in \{0,1\}^{|V|} \rightarrow$ embedding \mathbf{x}_w .

$m := |V|$, usually $|V| = 10^5$

d = dimensionality of embedding, $d \ll m$

So for each of the m words in V we have a corresponding embedding in \mathbb{R}^d , which can be stored in a shared lookup table:

$\mathcal{R}^{m \times d}$ shared lookup table

Any sentence of k words can then be represented as a $d \times k$ matrix (a sequence of k embedding vectors in \mathbb{R}^d).

Now, how should an embedding be? Ideally, the embedding carries the information/structure that we need in order to go from the input text to the question that we want to solve. Typical questions are:

- Clustering based on context (co-occurrence)

- Sentiment analysis (group words according to mood/feelings)

- Translation (group by meaning)

- Part-of-Speech tagging (understand the structure of text, e.g., location, time, actor, ..., or, noun, verb, adjective, ...)

14.1.1 Bi-Linear Models

The first thing that we could do is to use an information theoretic quantity: the so-called mutual information. The mutual information is described in information theory as *how much information one random variable has about another random variable*. If two variables are independent, then, the mutual information will be zero.

So, if we put two words nearby, it's because they have to be related somehow in the *meaning* of the sentence. Hence, we expect them to have a larger mutual information.

D. (Pointwise Mutual Information)

$\text{pmi}(v, w) = \log \left(\frac{P(v, w)}{P(v)P(w)} \right) = \log \left(\frac{P(v|w)}{P(v)} \right) \approx \mathbf{x}_v^T \mathbf{x}_w + \text{const}$

Com. As you can see this metric is bi-linear.

So we interpret the vectors as *latent variables* and link them to the observable probabilistic model. So the pointwise mutual information is related to the inner product between the latent vectors (the more related, the more co-linear the latent representations have to be).

Now, how do we compute the pointwise mutual information? One thing that we could do is just look for words that are nearby and compute these probabilities empirically. This leads us to the idea of skip-grams.

D. (Skip Grams)

The skip-gram approach is an approach to look at co-occurrences of words within a window size R (instead of looking at subsequences of some length n as with n grams). So we're only interested in the co-occurrence within some window size of words, R , rather than a precise sequence.

D. (Co-Occurrence Set)

Here we look at the pairwise occurrences of words in a context window of size R . So, if we have a long sequence of words $w = (w_1, \dots, w_T)$, then the co-occurrence index set is defined as

$$C_R := \{(i, j) \mid 1 \leq |i - j| \leq R\}.$$

D. (Co-Occurrence Matrix)

Note that in order to get an (empirical) idea of the co-occurrence frequencies one could compute the co-occurrence matrix

$$\mathbf{C} \in \mathbb{R}^{|V| \times |V|}, \quad \text{where } C_{ij} = \#\text{of co-occurrences of } w_i \text{ and } w_j \text{ within window size } R.$$

Properties: $\mathbf{C} = \mathbf{C}^T$ (symmetric), peaky, sparse.

One approach for embeddings: do PCA of \mathbf{C} and use k eigenvectors corresponding to largest eigenvalues of \mathbf{C} . Note that we have

$$C_{ij} = \text{one-hot}(w_i) \mathbf{C} \text{ one-hot}(w_j) = \mathbf{o}_i \mathbf{V} \mathbf{A}^T \mathbf{o}_j$$

$$\approx \mathbf{o}_i \mathbf{V}_k \mathbf{A}_k \mathbf{V}_k^T \mathbf{o}_j = \mathbf{o}_i \mathbf{V}_k \mathbf{A}_k^T \mathbf{A}_k \mathbf{V}_k^T \mathbf{o}_j$$

$$\approx \mathbf{o}_i \mathbf{V}_k \mathbf{A}_k \mathbf{V}_k^T \mathbf{o}_j$$

Recall, that in the previous part when we were doing autoencoders, the deviations that we were having for each of the components was the same one. So we wanted the error to be the same for each of the components. Now, with this model, with η we're allowing for additional flexibility for the error. There will be some components that we'll be able to explain with less error, and some with more. So, \mathbf{z} should capture everything that is important to explain the data, and η can be view as noise.

Although we're assuming that here everything is gaussian, in general we may view \mathbf{z} as a clustering mechanism, where \mathbf{z} determines some cluster components that are selected.

T. The distribution of the *observation model* is

$$\mathbf{x} \sim \mathcal{N}(\mu, \mathbf{W}\mathbf{W}^T + \Sigma).$$

- Non-Identifiability of Factors

Now this seems to be nice, but again we have the *non-identifiability problem*, since there exist an infinite amount of solutions for any \mathbf{W} that is a solution. Just let \mathbf{Q} be an orthogonal $m \times m$ -matrix. Then $\mathbf{W}\mathbf{Q}$ is also a solution, because

$$(\mathbf{W}\mathbf{Q})(\mathbf{W}\mathbf{Q})^T = M\mathbf{W}\mathbf{Q}\mathbf{Q}^T\mathbf{W}^T = \mathbf{W}\mathbf{W}^T.$$

The consequence of this is that the factors of the linear factor analysis are only identifiable up to some rotations/reflections in \mathbb{R}^m . Since we care what the factors in \mathbf{z} mean we need to factor the *rotations* to get a better "interpretability" of the representation of the data in the latent space.

- Data Compression View

Now, how is the factor analysis related to data compression?

Encoder Step: Implicitly defined by posterior distribution

$$p(\mathbf{z} | \mathbf{x}) = \frac{p(\mathbf{x} | \mathbf{z}) p(\mathbf{z})}{p(\mathbf{x})} \quad (\text{Bayes})$$

So,

$$p(\mathbf{z} | \mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}),$$

where

$$\begin{aligned} \boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}} &= \mathbf{W}^T (\mathbf{WW}^T + \Sigma)^{-1} (\mathbf{x} - \boldsymbol{\mu}) \\ \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}} &= \mathbf{I} - \mathbf{W}^T (\mathbf{WW}^T + \Sigma)^{-1} \mathbf{W} \end{aligned}$$

Further, if we assume that $\Sigma = \sigma^2 \mathbf{I}$ and we let $\sigma^2 \rightarrow 0$ (the reconstruction-error-variance for all the components is the same and we let the reconstruction error go to zero), then the following expression just reduces to the pseudo-invers:

$$\mathbf{W}^T (\mathbf{WW}^T + \sigma^2 \mathbf{I})^{-1} \xrightarrow{\sigma^2 \rightarrow 0} \mathbf{W}^T; \mathbf{W}^T \in \mathbb{R}^{m \times n}.$$

Consequently with the assumption of zero reconstruction error:

$$\begin{aligned} \boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}} &\rightarrow \mathbf{W}^T (\mathbf{x} - \boldsymbol{\mu}) \\ \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}} &\rightarrow \mathbf{0} \end{aligned}$$

So, if we know \mathbf{W} and Σ is assumed to be isotropic with the error going to zero the encoding distribution gets very easy to compute.

- Maximum Likelihood Estimation

Now, how do we estimate \mathbf{W} and Σ ? The idea is fairly simple.

Let's assume that $\mathbf{x}_1, \dots, \mathbf{x}_k \stackrel{i.i.d.}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{A})$. Further let's define the data matrix \mathbf{X} as

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_k \end{bmatrix}$$

and the empirical co-variance matrix as

$$\mathbf{S} := \frac{1}{k} \sum_{i=1}^k \mathbf{x}_i \mathbf{x}_i^T = \frac{1}{k} \mathbf{X} \mathbf{X}^T.$$

Then, the log-likelihood of the data \mathbf{X} , given \mathbf{A} can be written as:

$$\log(P(\mathbf{X}; \mathbf{A})) = -\frac{k}{2} \left(\text{Tr}(\mathbf{S} \mathbf{A}^{-1}) - \log(\det(\mathbf{A})) \right) + \text{const. of } \mathbf{A}$$

Note: this can be verified by using the definition of \mathbf{S} , the cyclic property of the trace, and then just write down the matrix-product as block-matrices and see what is the diagonal of the resulting matrix.

Now, let's compute the matrix gradients w.r.t. \mathbf{A} to know the equations that we need to compute the maximum likelihood:

$$\begin{aligned} \nabla_{\mathbf{A}} \text{Tr}(\mathbf{S} \mathbf{A}^{-1}) &= -\mathbf{A}^{-1} \mathbf{S} \mathbf{A}^{-1} \\ \nabla_{\mathbf{A}} \log(\det(\mathbf{A})) &= \mathbf{A}^{-1} \end{aligned}$$

Now, setting the gradient of the log-likelihood to zero gives us the following condition:

$$\nabla_{\mathbf{A}} \log(P(\mathbf{X}; \mathbf{A})) = 0 \iff \mathbf{S} \mathbf{A}^{-1} = \mathbf{I}.$$

So, the MLE for \mathbf{A} is just $\mathbf{A} = \mathbf{S}$.

But recall, that what we want is not \mathbf{A} , but we want \mathbf{W} and Σ . However, we know that \mathbf{A} is just the empirical covariance matrix, and \mathbf{W} will be the mapping to the low-dimensional space and Σ is the reconstruction error.

$$\mathbf{A} = \mathbf{W}\mathbf{W}^T + \Sigma$$

Now, using the chain rule we get:

$$\begin{aligned} \nabla_{\mathbf{W}} \mathbf{A} &= 2\mathbf{W} \\ \nabla_{\Sigma} \mathbf{A} &= \mathbf{I} \end{aligned}$$

This gives us the following stationary condition for \mathbf{W} given Σ :

$$\mathbf{S}(\Sigma + \mathbf{W}\mathbf{W}^T)^{-1} \mathbf{W} = \mathbf{W} \text{diag}(\frac{1}{\sigma^2 + \rho_i^2}).$$

Putting this back into the stationary condition, for each column \mathbf{w}_i of \mathbf{W} we get an eigenvector equation:

$$\mathbf{S}\mathbf{w}_i = (\sigma^2 + \rho_i^2)\mathbf{w}_i \quad \mathbf{S}\mathbf{W} = \text{diag}(\lambda)\mathbf{W}.$$

Then, if \mathbf{u}_i is the i -th eigenvector of \mathbf{S} , then

$$\mathbf{w}_i = \rho_i \mathbf{u}_i, \quad \rho_i^2 = \max \{0, \lambda_i - \sigma^2\}.$$

This gives us the probabilistic interpretation PCA and showed us how we can derive the PCA as a special case for $\sigma^2 \rightarrow 0$ (Tipping & Bishop, 1999).

- Refresher on MGFs and Gaussians

D. (Moment Generating Function (MGF)) The MGF $M_{\mathbf{X}}$ of a random vector $\mathbf{X} \in \mathbb{R}^n$ is defined as

$$M_{\mathbf{X}} : \mathbb{R}^n \rightarrow \mathbb{R} \quad t \mapsto \mathbb{E}_{\mathbf{X}} \left[e^{\mathbf{t}^T \mathbf{X}} \right].$$

The reason $M_{\mathbf{X}}$ is called moment generating function is because it represents the moments of \mathbf{x} in the following way: Let $k_1, \dots, k_n \in \mathbb{N}$, then

$$\mathbb{E}_{\mathbf{X}} \left[x_1^{k_1} x_2^{k_2} \cdots x_n^{k_n} \right] = \frac{\partial^k}{\partial t_1^{k_1} \partial t_2^{k_2} \cdots \partial t_n^{k_n}} M_{\mathbf{X}} \Big|_{\mathbf{t}=\mathbf{0}}.$$

T. (Uniqueness Theorem) If $M_{\mathbf{X}}$ and $M_{\mathbf{Y}}$ exist for the RVs \mathbf{X} and \mathbf{Y} and $M_{\mathbf{X}} = M_{\mathbf{Y}}$ then $\forall \mathbf{t}: P(\mathbf{X} = \mathbf{t}) = P(\mathbf{Y} = \mathbf{t})$ (distributions are the same).

Now, every distribution has its unique kind of MGF form. Hence, MGFs can be very useful to deal with sums of i.i.d. random variables:

T. If \mathbf{X}, \mathbf{Y} are i.i.d. then $M_{\mathbf{X}+\mathbf{Y}} = M_{\mathbf{X}} \cdot M_{\mathbf{Y}}$.

- 16.3 — Latent Variable Models

- 16.3.1 — DeFinetti's Theorem

There's another way of looking at latent variable models which is by the DeFinetti exchangeable theorem from the 1930s. This is one of the foundations of Bayesian probability (although there is nothing Bayesian in this theorem).

T. (DeFinetti's Theorem) For exchangeable data (order of dataset doesn't matter and they come from the same distribution), we can decompose the data by a latent variable model

$$P(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \int \prod_{i=1}^N p_{\theta}(\mathbf{x}_i | \mathbf{z}) p_{\theta}(\mathbf{z}) d\mathbf{z}.$$

We expect that those hidden variables are: interpretable and actionable and even show causal relations.

Later we'll put our Bayesian priors into the distributions $P(\mathbf{z})$ and we then hope that the latent structure will tell us something about the data that we didn't know before.

- 16.3.2 — Latent Variable Models

Classically we define complex models via the marginalization of a latent variable model

A common way to define these distributions is as follows: We assume the latent variable to follow a multivariate gaussian (assumed to be a reasonable prior for latent attributes).

Prior: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Enc. Netw.: E : models $q_{\phi}(\mathbf{z} | \mathbf{x})$ with params ϕ and maps $\mathbf{x} \xrightarrow{E} (\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}})$

Dec. Netw.: D : models $p_{\theta}(\mathbf{x} | \mathbf{z})$ with params θ and maps $\mathbf{z} \xrightarrow{D} (\boldsymbol{\mu}_{\mathbf{x}|\mathbf{z}}, \boldsymbol{\Sigma}_{\mathbf{x}|\mathbf{z}})$

Note that we use both *diagonal* covariance matrices for $\boldsymbol{\Sigma}_{\mathbf{x}|\mathbf{z}}$ and $\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}}$. So the output of both networks are just two vectors (one for mean, other for diagonal).

Com. Encoder and decoder networks are also called "recognition/inference" and "generation" networks.

Now, equipped with our encoder and decoder networks, we can rewrite the data (log) likelihood as follows (note that we omit the product for all points - you'd just have to put a sum over all the instances in front of everything)

Training Procedure Use SGD-like algorithm of choice (ADAM) on two minibatches simultaneously. At each iteration, we choose:

• a minibatch of true data samples
• a minibatch of noise vectors to produce minibatch generated samples

Then compute both losses and perform gradient updates.

$$\theta^{t+1} \leftarrow \theta^t - \eta_t \nabla_{\theta} \mathcal{R}^{(D)}(\theta_t)$$

$$\phi^{t+1} \leftarrow \phi^t - \eta_t \nabla_{\phi} \mathcal{R}^{(G)}(\phi_t)$$

Algorithm: run $K \geq 1$ update steps of D for every iteration (and only D)

So we will alternate between

(1) Gradient ascent on D

$$\max_{\theta_d} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log(D_{\theta_d}(\mathbf{x}))]$$

$$+ \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z})))]$$

(2) Gradient descent on G

$$\min_{\phi_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z})))]$$

Note that this training algorithm uses a heuristically motivated loss (that is a bit different) for the generator to have better gradients when the discriminator is good:

for number of training iterations do

 for k steps do

 • Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_{\theta}(\mathbf{z})$.

 • Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.

 • Update the discriminator by using its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m [\log(D_{\theta_d}(\mathbf{x}^{(i)})) + \log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)})))]$$

 end for

 for m minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_{\theta}(\mathbf{z})$.

 • Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)})))$$

 end for

So, first we do the whole backpropagation. And then we compute the updates to the parameters θ and ϕ via backpropagation.

Generating Data Here we just sample from the prior, and pass it through the decoder network to get the posterior distribution's parameters, then we sample from that one.



In unsupervised learning our goal is to learn some underlying hidden structure of the data (clustering, dimensionality reduction, feature learning, density estimation). Now, generative modeling has the following goal:

Goal: given data D , generate new samples from the same distribution p_{data} . We want to learn p_{model} similar to p_{data} .