Background
$\log(x+1)\approx x$ if $|x|\ll 1$
Geometric series: $\sum_{k=0}^n r^k = (\frac{1-r^{n+1}}{1-r}) \overset{n\to\infty}{\longrightarrow} \frac{1}{1-r}, \quad |r|<1$
Hoeffding's inequality: For $Z_i\in[a_i,b_i]$ i.i.d.
$P(\frac1n\sum_{i=1}^n Z_i - \mathbb{E}[Z]\ge t)\le \exp\left(-\frac{2n^2t^2}{\sum_{i=1}^n(b_i-a_i)^2}\right)$
Hölder's inequality: $\|v\cdot u\|_1 \le \|v\|_p\|u\|_{p^*}$ with $1/p+1/p^*=1$
Jensen's inequality: f convex, then $f(\mathrm{E}[X])\le \mathrm{E}[f(X)]$
Markov's inequality: $\mathrm{P}(X\ge a)\le \frac{\mathrm{E}[X]}{a}$ for $X\ge0$ a.s., $a>0$
KL Divergence: $KL(Q\|P)=\int_{-\infty}^{\infty} q(x)\log\frac{q(x)}{p(x)}dx$
Entropy: $H(X)=-\sum P(x_i)\log P(x_i)$
For A,B square: $\det AB=\det A\cdot\det B$, $\det A^{-1}=\frac{1}{\det A}$
MGF: $M_X:\mathbb{R}^n\to\mathbb{R}$, $M_X(t)=\mathbb{E}_X[\exp(t^TX)]$, $M_{X+Y}=M_X\cdot M_Y$
Moment Represent.: $\mathbb{E}[X_1^{k_1}\cdots X_n^{k_n}]=\frac{\partial^k}{\partial t_1^{k_1}\dots\partial t_n^{k_n}}M_X|_{t=0}$
For $x\sim\mathcal N(\mu,\Sigma)$, we have $M_x(t)=\exp(t^T\mu+\frac12 t^T\Sigma t)$
Normal: $p(x|\mu,\Sigma)=\frac{1}{\sqrt{(2\pi)^k\det(\Sigma)}}\exp(-\frac12(x-\mu)^t\Sigma^{-1}(x-\mu))$
KL-Divergence between $P\sim\mathcal N(\mu_P,\Sigma_P)$ **and** $Q\sim\mathcal N(\mu_Q,\Sigma_Q)$:
$KL(P\|Q)=\frac12[(\mu_Q-\mu_P)^T\Sigma_Q^{-1}(\mu_Q-\mu_P)+\mathrm{Tr}(\Sigma_Q^{-1}\Sigma_P)-\ln\frac{|\Sigma_P|}{|\Sigma_Q|}-n]$
Gaussian Process: $\sum_{i=1}^s\alpha_iF(x_i)\sim\mathcal N\ \forall\alpha_1,\dots,\alpha_s\wedge x_1,\dots,x_s$
Exponential: $p(x|\lambda)=\lambda e^{-\lambda x}$
Bernoulli: $p(x|p)=p^x(1-p)^{1-x}$
Binomial: $p(x|n,p)=\binom{n}{x}p^x(1-p)^{n-x}$
Poisson: $p(x|\lambda)=\frac{\lambda^x\exp[-\lambda]}{x!}$
Level sets: $L_f(z)=\{x:\phi(w^Tx+b)=z\}\perp w$
Jacobi Matrix: $\frac{\partial F}{\partial x}=(\frac{\partial F_i}{\partial x_j})_{ij}$

Connectionism
Perceptron
$(\mathbf{x},\theta)\to\mathrm{sgn}(\mathbf{x}^T\theta)$, update: $\Delta\theta=\begin{cases}0, & y\cdot\mathbf x^T\theta\ge0\\ y\mathbf x, & \text{otherwise}\end{cases}$
Update path zig-zags since $\Delta\theta^T\theta<0$ if $\Delta\theta\neq0$
Update rule is SGD for the loss: $l(\mathbf x,y;\theta)=\max\{0,-y\mathbf x^T\theta\}$
Lemma (Norm Growth): For perceptron mistakes $(\mathbf x^t,y^t)$ with induced updates $\Delta\theta^t$, set $\theta^s=\sum_{t=1}^s\Delta\theta^t$. Then by induction $\|\theta^s\|^2\le\sum_{t=1}^s\|\mathbf x^t\|^2$ and thus $\|\mathbf x^t\|\le1\Rightarrow\|\theta^s\|\le\sqrt s$
Def (Linear Separability): $\mathcal D$ linearly separable with margin $\gamma>0$ if $\exists\theta\in\mathcal S^{d-1}:y\cdot\mathbf x^T\theta\ge\gamma\quad\forall(\mathbf x,y)\in\mathcal D$
Novikov's convergence theorem: For $\theta^0=0$, $\|\mathbf x^t\|\le1$, and γ-separable $\mathcal D$ the perceptron converges in at most γ^{-2} updates. Proof: $\|\theta^*\|\cdot\|\theta^s\|\ge\langle\theta^*,\theta^s\rangle=\sum_{t=1}^s\langle\theta^*,\Delta\theta^t\rangle=\sum_{t=1}^s y^t\cdot(\mathbf x^t)^T\theta^*\ge s\gamma\Rightarrow1\ge\|\theta^*\|\ge\gamma\sqrt s$

Deep Linear Networks
Assume $X\in\mathbb R^{n\times s}, Y\in\mathbb R^{m\times s}$ s.t. $\sum_{i=1}^s x_i=0$, $\sum_{i=1}^s y_i=0$ and $XX^T=I_n$ (whitening). Then $W\overset{!}{\approx}\frac1s YX^T$ since
$\arg\min_W\|Y-WX\|_F=\arg\min_W\|W-\Gamma\|_F^2,\quad\Gamma=\frac1sYX^T$
This allows SVM-based analysis of 2-layer linear networks with $\binom{rank(\Gamma)}{width}$ fixed points and one global minimum.
Deep Linear Network Gradients:
$\frac12\frac{\partial\|W^L\cdots W^1x-y\|_2^2}{\partial W^l}=(W^L\cdots W^{l+1})^T(W^L\cdots W^1x-y)x^T(W^{l-1}\cdots W^1)^T$

Nonlinear Networks
Absolute Value Unit (AbsU): $|z|$, $\partial|z|=\begin{cases}1 & z>0\\ [-1,1] & z=0\\ -1 & z<0\end{cases}$
$(z)_+=(z+|z|)/2$ and $|z|=2(z)_+-z=(z)_++(-z)_+$
Logistic/Sigmoid unit: $\sigma(z)=\frac{1}{1+\exp(-z)}$, $\sigma^{-1}(t)=\log\frac{t}{1-t}$
$\sigma'(z)=\sigma(z)\cdot[1-\sigma(z)]=\sigma(z)\sigma(-z)$
Hyperbolic Tangent: $\tanh(z):=\frac{e^z-e^{-z}}{e^z+e^{-z}}=2\sigma(2z)-1$
$\tanh'(z)=1-\tanh^2(z)$
GELU: $\phi(z)=z\mathbb P(Z\le z)\quad Z\sim\mathcal N(0,1)$
Softmax: $\sigma_i^{\max}(\mathbf x)=\frac{\exp[x_i]}{\sum_{j=1}^k\exp[x_j]}$
$\frac{\partial}{\partial x_j}\sigma_i^{\max}(\mathbf x)=\begin{cases}\sigma_i^{\max}(\mathbf x)[1-\sigma_i^{\max}(\mathbf x)], & i=j\\ -\sigma_i^{\max}(\mathbf x)\cdot\sigma_j^{\max}(\mathbf x), & i\neq j\end{cases}$
Logistic Regression:
Cross-entropy loss ($y\in\{-1,+1\}$):
$l(\mathbf x,y;\theta)=-\log\sigma(y\mathbf x^T\theta)\Rightarrow\nabla_\theta l(\mathbf x,y;\theta)=-\sigma(-y\mathbf x^T\theta)y\mathbf x$
Cross-entropy loss ($y\in\{0,1\}$):
$l(\mathbf x,y;\theta)=-y\log\sigma(\mathbf x^T\theta)-(1-y)\log(1-\sigma(\mathbf x^T\theta))$
$\Rightarrow\nabla_\theta l(\mathbf x,y;\theta)=[\sigma(x^T\theta)-y]\mathbf x$

Approximation Theory
Weierstrass Theorem
Polynomials $\mathcal P$ are dense ($\|\cdot\|_\infty$) in $C([a,b])\quad\forall a,b\in\mathbb R$
Def. universal approx. $\mathcal G$: $C(S)\subseteq\overline{\mathcal G(S)}\ \forall$ compact $S\subseteq\mathbb R^n$
Universal Approximation Theorem (1d):
Let $\sigma\in C^\infty(\mathbb R)\setminus\mathcal P(\mathbb R)$. Then $H_\sigma^1=\mathrm{span}(\mathcal G_\sigma^1)$ is a universal approximator with $\mathcal G_\sigma^1=\{g:g(x)=\sigma(ax+b)\quad a,b\in\mathbb R\}$.
Universal Approximation Theorem (n-d):
$H_\sigma^n=\mathrm{span}(\mathcal G_\sigma^n)$ is a universal approximator where $\mathcal G_\sigma^n=\{g:g(x)=\sigma(x^T\theta+b)\ \theta\in\mathbb R^n,b\in\mathbb R\}$ (Pinkus)
Activation Pattern in 1-layer ReLU Network:
Activation pattern for input x: $\mathbb 1_{Wx+b>0}\in\{0,1\}^m$
Input partition into cells: $X_\kappa=\{x:\mathbb 1_{Wx+b>0}=\kappa\}$
Restricted to a cell the network is affine.
Zaslavsky (upper bound reached if $\mathcal H$ in general pos.):
$|\mathbb R^n-\mathcal H|\le\sum_{i=0}^{\min\{m,n\}}\binom mi=R(m)$ with $\mathcal H$ the m hyperplanes and $|\cdot|$ measuring the number of connected regions.
Montufar (deep L-layer nets with width $m>n$):
$\#$ cells $=R(m,L)\ge R(m)\lfloor\frac mn\rfloor^{n(L-1)}$ assuming general pos.
Universality of ReLU/AbsU Networks:
- Piecewise lin. functions are dense in $C([0,1])$
- Piecewise lin. function with m pieces can be written as $g(x)=ax+b+\sum_{i=1}^{m-1}c_i(x-x_i)_+$ ($\exists$ formulation with $|x|$)
- NNs with one hidden layer of ReLU or AbsU are universal function approximators (with Pinkus even on $\mathbb R^n$)

Minimal Non-Linearity
k-Hinge Function (maxout unit): $g(\mathbf x)=\max_{j=1}^k\{\boldsymbol\theta_j^T\mathbf x+b_j\}$
- Every continous piecewise linear function f can be written as a signed sum of k-Hinges with $k\le n+1$

Polyhedral Set: Intersection of half-planes (convex)
Def. Polyhedral Function: epigraph(f) is polyhedral set
Thm: Every continous piecewise linear $f:\mathbb R^n\to\mathbb R$ can be written as the difference of two polyhedral functions
$\Rightarrow$ **Thm**: Maxout networks with two maxout units (difference of two k-Hinges) are universal f-approximators

Gradient Descent and Optimizers
Gradient information provides direction of steepest descent:
$\lim_{\eta\to0}\arg\min_{\boldsymbol\vartheta:\|\boldsymbol\vartheta\|_2=1}f(\mathbf x;\boldsymbol\theta+\eta\boldsymbol\vartheta)=-\frac{\nabla_{\boldsymbol\theta}f(\mathbf x;\boldsymbol\theta)}{\|\nabla_{\boldsymbol\theta}f(\mathbf x;\boldsymbol\theta)\|_2}$

Gradient Descent and Newton's Method
Gradient Descent:
Discretize gradient flow $\dot\theta=-\nabla f(\theta)$ as $\theta_{t+1}=\theta_t-\eta\nabla f(\theta_t)$
Newton's Method:
Directly jumps to minimum of 2nd order Taylor of f:
$f(\theta+\Delta\theta)\approx f(\theta)+\Delta\theta^T\nabla f(\theta)+\frac12\Delta\theta^T\nabla^2f(\theta)\Delta\theta$
which results in $\Delta\theta=-[\nabla^2f(\theta)]^{-1}\nabla f(\theta)$. GD is recovered under isotropy assumption on the Hessian ($\nabla^2f(\theta)=\frac1\eta\mathrm I$)
Batch Gradient Descent (unbiased with $\mathbb E[\nabla\hat f_\theta]=\nabla f_\theta$):
Compute $\nabla\hat f_\theta=\frac1r\sum_{t=1}^r f_\theta(x_{i_t},y_{i_t})$ where $i_t\sim\mathcal U(\{1,\dots,s\})$.
SGD ($r=1$) Variance Reduction Technique (SVRG):
Every T steps compute full gradient $\nabla_{\tilde\theta}f_{\tilde\theta}$. Use updates $\theta_{t+1}=\theta_t-\eta[\nabla f_{\theta_t}(x_i,y_i)-\nabla f_{\tilde\theta}(x_i,y_i)+\nabla_\theta f_{\tilde\theta}]$
Compressed Stochastic Gradients (SignSGD):
Update $\theta_{k+1}=\theta_k-\eta_k\mathrm{sign}[\nabla f(\theta_k)(x_{i_t},y_{i_t})]$.

Gradient Descent with Momentum
Nesterov's Accelerated Gradient Descent:
$x_{k+1}=y_k-\eta\nabla f(y_{k+1})\quad y_k=x_k+\beta(x_k-x_{k-1})\qquad\beta\in[0,1)$
Polyak's Heavy Ball Method:
$x_{k+1}=x_k-\eta\nabla f(x_k)+\beta(x_k-x_{k-1})\qquad\beta\in[0,1)$
with $x_{t+1}-x_t=-\eta\sum_{k=0}^t\beta^{t-k}\nabla f(x_k)\xrightarrow{t\to\infty,\ const\ \nabla f}-\frac{\eta}{1-\beta}\nabla f$

Adaptive Gradient Descent
AdaGrad (decays LR based on 2nd moment estimate):
$\theta[j]$ is updated with learning rate $\eta_{t,j}=\frac{\eta}{\gamma_{t,j}+\delta}$ where $\gamma_{t,j}^2=\gamma_{t-1,j}^2+(\frac{\partial f_\theta(x_{i_t},y_{i_t})}{\partial\theta_t[j]})^2$ with $0<\delta\approx0$ for stability.
Adam - Adaptive Moment Estimation (w. momentum):
Exponential averages: $\Delta_{t+1}=\alpha\Delta_t+(1-\alpha)\nabla f_\theta(x_{i_t},y_{i_t})$ and $\gamma_{t+1,j}^2=\beta\gamma_{t,j}^2+(1-\beta)(\frac{\partial f_\theta(x_{i_t},y_{i_t})}{\partial\theta_t[j]})^2$ where $\alpha,\beta\in[0,1)$
Update: $\theta_{t+1}[j]=\theta_t[j]+\frac{\eta}{\gamma_{t,j}/(1-\beta)+\delta}\frac{\Delta_{t+1,j}}{1-\alpha}$ with $0<\delta\approx0$
AMSGrad (fixes/ensures convergence):
Adam + monotonicity $\tilde\gamma_{t+1,j}=\max(\gamma_{t+1,j},\tilde\gamma_{t,j})$.

Backpropagation
Chain rule: $\partial(G\circ F_\theta\circ H)=(\partial G\circ F_\theta\circ H)\cdot(\partial F_\theta\circ H)$
Backpropagation algorithm consists of three steps:
(1) forward pass computing all pre-activations $\mathbf z_l$,
(2) backward pass computing all $\boldsymbol\xi_l=\frac{\partial\mathcal L}{\partial z_l}=\frac{\partial\mathcal L}{\partial z_{l+1}}\frac{\partial z_{l+1}}{\partial z_l}$
(3) local computations computing all $\frac{\partial\mathcal L}{\partial W_l}=\frac{\partial\mathcal L}{\partial z_l}\frac{\partial z_l}{\partial W_l}$

Automatic Differentiation of $f:\mathbb R^N\to\mathbb R^M$ as a DAG(V,E)
Usually $N=$ #parameters and $M=1$ (scalar loss).
Forward Mode: Chain rule from right to left (no separate forward pass). Scales in $\mathcal O(N(V+E))$.
Reverse Mode: Chain rule from left to right (needs separate forward pass $\Rightarrow$ more memory). Scales in $\mathcal O(M(V+E))$.

Optimization Convergence Guarantees
Convexity, Smoothness, and Polyak-Lojasiewicz
Def. Convexity: $f(\lambda x+(1-\lambda)y)\le\lambda f(x)+(1-\lambda)f(y)\ \forall\lambda\in[0,1]$
Def. Convexity of Diff'ble f: $f(x)\ge f(y)+\nabla f(y)^T(x-y)$
Def. μ-Strong Convexity: $f(x)\ge f(y)+\nabla f(y)^T(x-y)+\frac\mu2\|x-y\|^2$
Def. Lipschitz Smoothness: $\|\nabla f(x)-\nabla f(y)\|_2\le L\|x-y\|_2$
L-Smoothness implies $f(x)\le f(y)+\nabla f(y)^T(x-y)+\frac L2\|x-y\|_2^2$
using $f(x)-f(y)-\nabla f(y)^T(x-y)=\int_0^1[\nabla f(y+(x-y)s)-\nabla f(y)]^T(x-y)ds$.
Applied to GD we get $f(\theta-\frac1L\nabla f(\theta))\le f(\theta)-\frac1{2L}\|\nabla f(\theta)\|_2^2$.
Hessian of L-Smooth & μ-Strongly Convex: $\mu I\preccurlyeq\nabla^2f\preccurlyeq LI$
ϵ-Stationarity (approximate stationarity): $\|\nabla f(x)\|_2\le\epsilon$
PL condition: $\forall y\ \frac1{2\mu}\|\nabla f(y)\|^2\ge f(y)-\min_x f(x)$ implied by μ-strong convexity.

Convergence of GD and GD with Nesterov Momentum
Thm. Polyak: Given f L-smooth and μ-PL, then GD with $\eta=1/L$, i.e. $x_{t+1}=x_t-\frac1L\nabla f(x_t)$, converges globally:
$f(x_t)-\min_xf(x)\le(1-\frac\mu L)^t[f(x_0)-\min_xf(x)]$
Thm. Nesterov: Given f L-smooth and μ-strongly convex with conditioning $\kappa=\frac L\mu$ and $\beta=\frac{\sqrt\kappa-1}{\sqrt\kappa+1}$, then Nesterov GD converges globally: $f(x_t)-f(x^*)\le L(1-\sqrt{\frac\mu L})^t\|x_0-x^*\|^2$

Convergence of Stochastic Gradient Descent
Consider BGD with $r=1$. Then $\|\theta_t-\theta_*\|^2-\hat{\mathbb E}_{x,y}\|\theta_{t+1}-\theta_*\|^2=2\eta_t(\theta_t-\theta_*)^T\hat{\mathbb E}_{x,y}\nabla f_{\theta_t}(x,y)-\eta_t^2\hat{\mathbb E}_{x,y}\|\nabla f_{\theta_t}(x,y)\|^2$ is positive for small enough η_t and $(\theta_t-\theta_*)^T\nabla\frac1s\sum_{k=1}^s f_{\theta_t}(x_k,y_k)>0$.
Polyak Averages: To reduce $\hat{\mathbb E}_{x,y}\|\overline\theta_{t+1}-\overline\theta_t\|^2$ use Polyak averaging $\overline\theta_{t+1}=\frac{t\cdot\overline\theta_t}{t+1}+\frac{\theta_{t+1}}{t+1}=\frac{-\eta_t\nabla f_{\theta_t}(x,y)}{t+1}$. Let $\eta_t\propto\frac1t$, then
- f convex: $\hat{\mathbb E}[f(\overline{\theta_t})]-f(\theta^*)\in\mathcal O(1/\sqrt t)$
- f μ-strongly convex: $\hat{\mathbb E}[f(\overline{\theta_t})]-f(\theta^*)\in\mathcal O(\frac{\log t}{t})$
- f μ-strongly convex & L-smooth: $\hat{\mathbb E}[f(\overline{\theta_t})]-f(\theta^*)\in\mathcal O(\frac1t)$

Quasi-Convergence of SGD with Constant Step Size:
Let $f_\theta=\frac1s\sum_{k=1}^sf_\theta(x_k,y_k)$ be μ-strongly convex with each $\theta\mapsto f_\theta(x_k,y_k)$ being L-smooth with measure of variance $\sigma^2=\frac1s\sum_{k=1}^s\|\nabla_\theta f_\theta(x_k,y_k)-\nabla_\theta f_\theta\|^2$. Then for SGD with $\eta_{const}\le1/\mu$ it holds $\hat{\mathbb E}\|\theta_t-\theta^*\|^2\le A^t\|\theta_0-\theta^*\|^2+B$ where $A=1-2\eta_{const}\mu(1-\eta_{const}L)$ and $B=\frac{\eta_{const}\sigma^2}{\mu(1-\eta_{const}L)}$.

Convolutional Networks
Convolution Operator
Integral Operator: $(Tf)(u)=\int_{t_1}^{t_2}H(u,t)f(t)dt$
Convolution: $(f*h)(u):=\int_{-\infty}^\infty h(u-t)f(t)dt=(h*f)(u)$
Discrete Convolution: $(f*h)[u]:=\sum_{t=-\infty}^\infty f[t]h[u-t]$
Theorem: An operator T is linear and shift-equivariant $\iff T$ is a convolution $T(f)=f*h$ for some kernel h
Cross-Corr.: $(f\star h)[u]:=\sum_{t=-\infty}^\infty f[t]h[u+t]=(f[-\cdot]*h)[u]$
Fourier Transform: $(\mathcal Ff)(u):=\int_{-\infty}^\infty\exp(-2\pi itu)f(t)dt$
Convolution Theorem: $u*v=\mathcal F^{-1}(\mathcal Fu\cdot\mathcal Fv)$
Toeplitz Matrices: Constant diagonals (1D convolutional kernel can be written as a Toeplitz matrix product)

Convolutional Neural Networks
Exploiting translation equivariance, locality, and scale. More efficient through parameter sharing and locality.
(Local) Receptive Field of x_i^l: $\mathcal I_i^l:=\{j:w_{ij}^l\neq0\}$
Sparse Backpropagation: $\partial x_i^l/\partial x_j^{l-1}=0$, for $j\notin\mathcal I_i^l$
Weight Sharing: $\frac{\partial\mathcal R}{\partial h_j^l}=\sum_i\frac{\partial\mathcal R}{\partial x_i^l}\frac{\partial x_i^l}{\partial h_j^l}$, h_j^l kernel weight
Pooling 2D: $x_{ij}^{\max}=\max\{x_{i+k,j+l}:0\le k<r,0\le l<r\}$
2D Convolutional Layer: ($\mathbf X,\mathbf Y$ are of shape (c, x, y)):
$y[r][s,t]=\sum_u\sum_{\Delta s,\Delta t}w[r,u][\Delta s,\Delta t]\cdot x[u][s+\Delta s,t+\Delta t]$
Output Size: $\lfloor(input\_width+2pad-filter\_width)/stride\rfloor+1$
Embeddings from Log-Bilinear: $\mathbb P(\nu|\omega)=\frac{\exp(x_\omega^Ty_\nu)}{\sum_\mu\exp(x_\omega^Ty_\mu)}$

Regularization
To lower generalization error but not the training error.
- *informed regularization (prior knowledge)*
- *simplicity bias (Occam's razor)*
- *Bayesian averaging (ensembling, dropout)*

L_2-Regularization / Weight Decay
Regularized objective becomes $\overline{E}_\theta(S) = E_\theta(S) + \mu\Omega(\theta)$ where $\Omega(\theta) = \frac{1}{2}\sum_{l=1}^L \|W^l\|_F^2 = \frac{1}{2}\sum_i \theta_i^2$ (no bias reg.) for $\theta = vec(W^1,\dots,W^L,b^1,\dots,b^L)$. Since $\nabla_\theta\Omega(\theta)=\mu\theta$ the GD update includes weight decay: $\theta_{t+1} = (1-\eta\mu)\theta_t - \eta\nabla_\theta E_{\theta_t}$

Spectral Analysis based on Taylor around $\arg\min E_\theta(S)$:
$\overline{E}_\theta(S) \approx E_{\theta^*}(S) + \frac12(\theta-\theta^*)^T H_E(\theta-\theta^*) + \mu\Omega(\theta) + \mathcal{O}(\|\theta-\theta^*\|_2^3)$ with Hessian $H_E = Q\text{diag}(\lambda_i)Q'$. By first order conditions $\theta \overset{!}{=} (H_E+\mu I)^{-1}H_E\theta^* = Q\text{diag}(\frac{\lambda_i}{\lambda_i+\mu})Q'\theta^*$. Hence, along low-curvature directions ($\lambda_i \ll \mu$) the weights are shrunk.

Constrained Optimization:
Regularization objective becomes $\arg\min_{\{\theta:\|\theta\|\le r\}} E(\theta)$.
Regularization only effective at late stages of learning. Optimized using projected gradient descent:

$\theta_{t+1} = \Pi_r[\theta_t - \eta\nabla E_{\theta_t}(S)]$ where $\Pi_r(v) = \min\left\{1, \frac{r}{\|v\|}\right\} v$

Ridge Regression: Linear regression + L_2-regularization is termed Ridge regression with $\boldsymbol\theta^* = (\mathbf{X}^\top\mathbf{X}+\lambda I)^{-1}\mathbf{X}^\top\mathbf{y}$.

Early Stopping
Stop learning when loss on validation data increases. Early stopping acts as an approximate $L2$-regularizer:
Spectral Analysis based on Taylor around $\arg\min E_\theta(S)$:
$\nabla_\theta E_\theta(S) \approx H_E(\theta-\theta^*) + \mathcal{O}(\|\theta-\theta^*\|_2^2)$ w. $H_E = Q\text{diag}(\lambda_i)Q'$ giving GD update $\theta_{t+1} = \theta_t - \eta H_E(\theta_t-\theta^*) + \mathcal{O}(\|\theta_t-\theta^*\|_2^2$. Hence, $\theta_{t+1}-\theta^* = (I-\eta H_E)(\theta_t-\theta^*)+\mathcal{O}(\|\theta_t-\theta^*\|_2^2$. Then $\theta_t = \cancel{(I-\eta\text{diag}(\lambda_i))^t Q'\theta_0} + Q[I-(I-\eta\text{diag}(\lambda_i))^t]Q'\theta^*$ (by induction) where the first term is neglected if $\theta_0 = 0$.
Since $t\eta\text{diag}(\lambda_i) \overset{\eta\lambda_i\ll 1}{\approx} I-(I-\eta\text{diag}(\lambda_i))^t \overset{!}{=} \text{diag}(\frac{\lambda_i}{\lambda_i+\mu})$ we get L_2-regularization given $\eta\lambda_i \ll 1$, $\lambda_i \ll \mu$, $t = \frac{1}{\eta\mu}$.

Residual Connections
Parametrize $G_\theta(x) = x + F_\theta(x)$ with $F\approx 0$ at initialization giving Jacobian $J_G \approx I$ (good for backpropagation) and allowing weight decay toward identity. ResNets augment deep CNNs with residual connections. DenseNets further employ skip connections to all downstream layers.
- Residual Connections adds shortcut to output
- Skip Connections concatenates shortcut to output

Normalization
Calibrate dynamic range of unit j in layer l (datapoint i):
Batch Normalization (γ_j^l, β_j^l are params):

Normalization: $\bar z_j^l[i] = \frac{z_j^l[i]-\mu_j^l}{\sigma_j^l+\delta} \quad 0<\delta\approx 0,\quad \tilde z_j^l = \gamma_j^l + \beta_j^l \bar z_j^l$

Statistics: $\mu_j^l = \frac{1}{|B|}\sum_{i\in B} z_j^l[i],\quad (\sigma_j^l)^2 = \frac{1}{|B|}\sum_{i\in B}(z_j^l[i]-\mu_j^l)^2$

Layer Normalization (normal. along single instance):

Normalization: same formula as in Batch Norm

Statistics: $\mu_j^l = \frac{1}{m_l}\sum_{j=1}^{m_l} z_j^l[i],\quad (\sigma_j^l)^2 = \frac{1}{m_l}\sum_{j=1}^{m_l}(z_j^l[i]-\mu_j^l)^2$

Dropout
Randomly drop subsets of units for more robustness with retention probability π_i^l for unit i in layer l (input $\pi_i^0 = 0.8$ and hidden unit $\pi_i^{l\ge1} = 0.5$). Creates an exponentially large (in #nodes) ensemble of networks.

At inference, use as an ensemble method or rescale weights $\tilde w_{ij}^l \leftarrow \pi_j^{l-1} w_{ij}^l$ calibrating the net input to unit i.

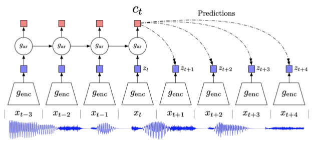
Data Augmentation
Trick to generate larger, non-i.i.d. training set

Generate virtual examples by applying transformations τ to each training example $(\mathbf{x},\mathbf{y}) \mapsto (\tau(\mathbf{x}),\mathbf{y})$, e.g. PCA, cropping & resizing, rotations & reflections, etc.

We can add noise to the inputs (ideally realistic noise), to the weights (regularizing effect) or to the targets (soft targets/label errors). Also helps with model robustness.

Self-Supervised Training
Pretrain a model on a (self-supervised) task (e.g. patch location prediction, image coloring, video motion prediction, predictive coding of latent states) before finetuning.

Contrastive Predictive Coding



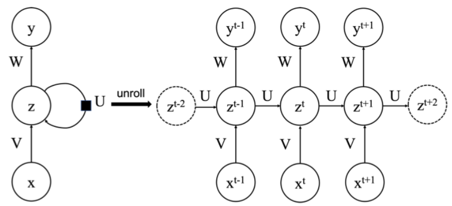
Maxim. density ratio $\frac{p(x_{t+1}|c_t)}{p(x_{t+1})} \propto e^{z_{t+1}^T W c_t}$ for positive examples and minimize for negative examples

Geometric Deep Learning
Group: A set $\mathcal{G}$ with binary operation $\circ: \mathcal G\times\mathcal G\to\mathcal G$ s.t.
Closure: $gh\in\mathcal G\ \forall g,h\in\mathcal G$
Associativity: $(gh)f = g(hf)\ \forall g,h,f\in\mathcal G$
Identity: $\exists!\ e\in\mathcal G$, for which $eg = ge = g \quad \forall g\in\mathcal G$
Inverse: $\forall g\ \exists!\ g^{-1}\in\mathcal G$, s.t. $gg^{-1} = g^{-1}g = e$
Group Action: $\otimes: G\times X\to X$ s.t. $e(x) = x \wedge g(h(x)) = gh(x)$
Linear Group Action: $g(\alpha x+y) = \alpha g(x)+g(y)$ with Representation $\rho:\mathcal G\to\mathbb R^{n\times n}$ s.t. $\rho(g)x = g(x) \Rightarrow \rho(gh) = \rho(g)\rho(h)$
Action on Signals: $(\rho(h)x)(s) = x(h^{-1}s)$ where $x:\Omega\to\mathbb R$
$\mathcal G$-Invariance: $f:\mathcal X(\Omega)\to\mathcal Y$ satisfies $f(\rho(g)x) = f(x)\ \forall g\in\mathcal G$
$\mathcal G$-Equivariance: $f:\mathcal X(\Omega)\to\mathcal X(\Omega)$ fulfills $f(\rho(g)x) = \rho(g)f(x)$
$\mathcal G$-Convolution: $(x*\theta)(g) = \int_\Omega x(u)\theta(g^{-1}u)du \quad g\in\mathcal G$
$(\rho(h)x*\theta)(g) = \rho(h)(x*\theta)(g) \quad (x*\rho(h)\theta)(g) = (x*\theta)(gh)$
Smoothing for Invariance: $(S_{\mathcal G}f)(x) = \frac{1}{|\mathcal G|}\sum_{g\in\mathcal G} f(\rho(g)(x))$
Deep Sets: can represent any in-/equivariant function on sets using only the following two layers:
Invariant Layer: $f(X) = \phi(\sum_{x\in X}\psi(x))$ with ϕ,ψ learnable.
Equivariant Layer: $f(X,x_i) = \phi\left(\sum_j\psi(x_j), x_i\right)$

Graph Neural Nets
Node Feature Matrix: $X = [x_1,\dots,x_n]^T\in\mathbb R^{n\times k}$
Permutation Invariance: $f(PX, PAP^T) = f(X,A)$
Permutation Equivariance: $f(PX,PAP^T) = Pf(X,A)$
1-Hop Neighbourhood of x_i: $\mathcal N_i = \{j: A_{ij}\neq 0\}$
E.V. GNN Layer: $f(X,A) = [\phi(x_1, X_{(\mathcal N_1)}),\dots,\phi(x_n,X_{(\mathcal N_n)})]$ for the multiset $X_{(\mathcal N_i)} = \{\{x_j: j\in\mathcal N_i\}\}$.
Convolution GNN: $f(X,A)_i = \phi(x_i, \sum_{j\in\mathcal N_i} c_{ij}(A)\psi(x_j))$
Attentional GNN: $f(X,A)_i = \phi(x_i,\sum_{j\in\mathcal N_i} a(x_i,x_j)\psi(x_j))$
Message-Passing GNN: $f(X,A)_i = \phi(x_i,\sum_{j\in\mathcal N_i}\psi(x_i,x_j))$
WL-Test is a necessary condition for graph isomorphism. It applies $x_i \leftarrow hash(x_i, X_{(\mathcal N_i)})$ until convergence (up to relabelling) and compares statistics of the final features.

Spectral Graph Theory
Graph Laplacian: $L_G = D - A = \sum_{i<j} a_{ij}(e_i-e_j)(e_i-e_j)^T$ with $x^T L_G x = \sum_{i<j} a_{ij}(x_i-x_j)^2 \ge 0 \implies L_G\ge 0$
Dirichlet Energy: $L = \nabla^2 x^T L x = \nabla^2\frac12\sum_{u,v}A_{u,v}(x_u-x_v)^2$
Norm. Laplacian: $\mathcal L = I - D^{-1/2}AD^{-1/2} = D^{-1/2}LD^{-1/2}$
Graph Fourier: $L = U\Lambda U^T$, induces $\hat x = U^T x$ and $x = U\hat x$
Spectral Graph Conv: $h(L)x = Uh(\Lambda)U^T x$ using $\mathcal O(n^2)$
More efficient for polynomial kernel $p(L)x = \sum_{i=0}^K \alpha_i L^i x$ using $\mathcal O(K\cdot e)$. The filter is isotropic (circular symmetric).
For c_{in}/c_{out} in-/output channels $X_{out}^j = \sum_{i=1}^{c_{in}} p_{i,j}(L)X_{in}^i + b_j$
GCNs (positively) couple neighbouring units $X^{l+1} = \sigma((\tilde D^{-1/2}\tilde A\tilde D^{-1/2})X^l W^l)$ where $\tilde A = A + I_n$ with $\tilde D = D+I$.
Stability is ensured by $\lambda_{max}(\tilde D^{-1/2}\tilde A\tilde D^{-1/2}) = 1, \lambda_{min}\ge 0$

Recurrent Neural Networks
Simple RNNs
Given a non-i.i.d. sequence $\mathbf x^1,\dots,\mathbf x^T$ derive a sequence of states $z^1,\dots,z^T$ according to the Markovian time-invariant update $z^t = F_\theta(z^{t-1},x^t)$. Produce an output sequence via $\mathbf y^t = H_\varphi(\mathbf z^t)$ (where $\mathbf y^T$ may be used for classification).



F_θ and H_φ are parametrized as $F_{U,V}(\mathbf z,\mathbf x) = \phi(U\mathbf z+V\mathbf x)$ and $H_W(\mathbf z) = \psi(W\mathbf z)$ for activation functions ϕ,ψ.
Backpropagation (through time):

Propagation: $\frac{\partial z^{k+1}}{\partial z^k} = \frac{\partial\phi(Uz^k+Vx^{k+1})}{\partial z^k} = diag(\dot\phi(Uz^k+Vx^{k+1}))U$

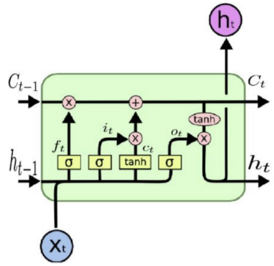
$\frac{\partial y^s}{\partial z^s} = \frac{\partial\psi(Wz^s)}{\partial z^s} = diag(\dot\psi(Wz^s))W$

Per State: $\frac{\partial E}{\partial z^q} = \sum_{s=q}^T \frac{\partial E}{\partial y^s}\frac{\partial y^s}{\partial z^q} = \sum_{s=q}^T\frac{\partial E}{\partial y^s}\frac{\partial y^s}{\partial z^s}\prod_{k=q}^{s-1}\frac{\partial z^{k+1}}{\partial z^k}$
- $\frac{\partial E}{\partial W} = \sum_{s=1}^T\frac{\partial E}{\partial y^s}\frac{\partial y^s}{\partial W} = \sum_{s=1}^T(\dot\psi(Wz^s)\odot z^s)\frac{\partial E}{\partial y^s}$
- $\frac{\partial E}{\partial U} = \sum_{q=1}^T \frac{\partial E}{\partial z^q}\frac{\partial z^q}{\partial U} = \sum_{q=1}^T(\dot\phi(Uz^{q-1}+Vx^q)\odot z^{q-1})\frac{\partial E}{\partial z^q}$
- $\frac{\partial E}{\partial V} = \sum_{q=1}^T\frac{\partial E}{\partial z^q}\frac{\partial z^q}{\partial V} = \sum_{q=1}^T(\dot\phi(Uz^{q-1}+Vx^q)\odot x^q)\frac{\partial E}{\partial z^q}$
- $\frac{\partial E}{\partial\theta} = \sum_{t=1}^T\frac{\partial E}{\partial y_t}\frac{\partial y_t}{\partial z_t}\sum_{i=1}^t(\prod_{k=t}^{i+1}\frac{\partial z_k}{\partial z_{k-1}})\frac{\partial F_\theta(z,x)}{\partial\theta}\Big|_{z=z^{i-1},x=x^{i-1}}$

Exploding/Vanishing Gradients:
$\|A\|_2 = \max_{x:\|x\|=1}\|Ax\|_2 = \sigma_1(A) \qquad \|AB\|_2\le\|A\|_2\|B\|_2$
In backpropagation the following derivative occurs:
$\frac{\partial z^T}{\partial z^0} = \dot\Phi^T U\cdots\dot\Phi^1 U$ where $\dot\Phi^t = \text{diag}(\dot\phi(Uz^{t-1}+Vx^t))$ and $\exists\alpha: \dot\Phi^t\le\alpha I$ (RELU: $\alpha=1$, Sigmoid: $\alpha = 1/4$). If $\sigma_1(U) < 1/\alpha$, $\left\|\frac{\partial z^T}{\partial z^0}\right\|_2 \le (\alpha\sigma_1(U))^T \to 0$ as $T\to\infty$ giving vanishing gradients. Similarly, exploding gradients occur.
Bi-Directional RNN:
Additionally evolve $\tilde z^t = \phi(\tilde U\tilde z^{t+1}+\tilde Vx^t)$ with $\tilde z^{T+1} = 0$. Modified output map couples the two: $y^t = \psi(Wz^t+\tilde W\tilde z^t)$.
Deep RNN/Stacked RNN/Hierarchical RNN:
$z^{t,l} = \phi(U^l z^{t-1,l}+V^l z^{t,l-1})$ where l denotes the layer and $z^{t,0} = x^t$. Output computed from last state $y^t = \psi(Wz^{t,L})$.

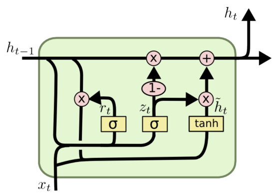
Gated Memory Units
Addressing the problem of vanishing/exploding gradients
Gating unit: $z_{gated} = \sigma(G\zeta)\odot z$ where $\sigma(\cdot)\in[0,1]$. Gating units are embedded into gated units (LSTMs or GRUs).

We neglect bias terms in the following.
Long-Short-Term-Memory Unit (LSTM) - 3 Gates



Augmented Input: $\tilde x_t = [h_{t-1},x_t]$
Output Gate: $h_t = \sigma(H\tilde x_t)\otimes\tanh(UC_t)$
Forget & Input Gate: $C_t = \sigma(F\tilde x_t)\odot C_{t-1} + \sigma(G\tilde x_t)\odot\tanh(V\tilde x_t)$
Output Processing: $y_t = Wh_t$

3 gating weight matrices: H,F,G, 2 propagation weight matrices U,V, and 1 output weight matrix W. 6 in total.
Gated Recurrent Unit (GRU) - 2 Gates
Introduced to simplify LSTMs (less gates, less weights). The forget gate and input gate are convexly combined and the context cell/hidden state duality is removed.



Reset Gate: $r_t = \sigma(H[h_{t-1},x_t])$
Proposed Input: $\tilde h_t = \tanh(V[r_t\odot h_{t-1},x_t])$
Update Gate (Forget&Input): $h_t = (1-\sigma(G[h_{t-1},x_{t-1}]))\odot h_{t-1} + \sigma(G[h_{t-1},x_{t-1}])\odot\tilde h_t$
Output Processing $y_t = Wh_t$

2 gating weight matrices: H,G, 1 propagation weight matrix V, and 1 output weight matrix W. 4 in total.

Linear Recurrent (State-Space) Models
Linear state space model: $z^{t+1} = Az^t + Bx^t$ with diagonalization $A = P\Lambda P^{-1}$ over $\mathbb C$. Change of basis leads to $\zeta^{t+1} = \Lambda\zeta^t + Cx^t$ with $\zeta^t = P^{-1}z^t$ in complex state space ($\max_j|\lambda_j|\le 1$ for stabilization). To ensure high representational power of these systems, we give enough modelling power to the output map $y^t = \text{MLP}(\text{Re}(G\zeta^t))$.

Causal ($y^t\perp x^s\ \forall s>t$) Seq2Seq Modelling via RNN
To learn $p(\mathbf y^{1:T}\mid\mathbf x^{1:T}) \approx \prod_{t=1}^T p(\mathbf y^t\mid\mathbf x^{1:t},\mathbf y^{1:t-1})$ parametrize $p(y^t|x^{1:t},y^{1:t-1})$ through $x^{1:t}\overset{F}{\mapsto} z^t \overset{H}{\mapsto}\mu^t\mapsto p_{\mu^t}(y^t)$.
By introducing feedback links from $y^{t-1}\to z^t$ we can ensure that z^t incorporates all knowledge on $x^{1:t}, y^{1:t-1}$.
Teacher Forcing:
During training, compute loss on predicted output but feed back in actual output.
During prediction, feed back predicted outputs. Improves learning BUT gives exposure bias (model only learns step predictions).
Professor Forcing is a variant of teacher forcing that randomizes whether to feed back in y_{pred} or y_{true}.
Encoder-Decoder Model:
Encoder: $(\mathbf x^1,\dots,\mathbf x^T)\mapsto\mathbf z,\quad \mathbf z = \mathbf z^T$ (RNN)
Decoder $\mathbf z\mapsto(\mathbf y^1,\dots,\mathbf y^S)$ (RNN with output feedback)

Attention

Generate contextualised embeddings $\xi^s \in \mathbb{R}^m$ using convex combination of projected simple embeddings $x \in \mathbb{R}^e$:
$\xi^s = \sum_t a_{st} W x^t, \quad a_{st} \geq 0, \quad \sum_t a_{st} = 1$ with $\Xi = AXW^T$

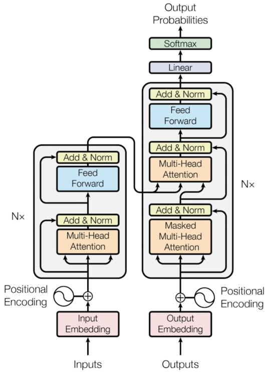
Transformer Architecture

$X \in \mathbb{R}^{L \times e}$

$Q = XU_Q, K = XU_K, V = XU_V$ with $U_Q, U_K \in \mathbb{R}^{e \times d}, U_V \in \mathbb{R}^{e \times e}$

Single-head: $Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d}})V$

Multihead: $Multihead(Q,K,V) = Concat(H_1, \ldots, H_h)W^O$ with $H_i = Attention(XU_Q^i, XU_K^i, XU_V^i)$ where $U_{\{Q,K\}}^i \in \mathbb{R}^{n \times d_k}, U_V^i \in \mathbb{R}^{e \times d_v}, W^O \in \mathbb{R}^{hd_v \times e}$



Sinusoidal Pos. Encodings: For position t and feature k:
$$p_{tk} = \begin{cases} \sin(t\omega_k) & k \text{ even} \\ \cos(t\omega_k) & k \text{ odd} \end{cases}$$
where $\omega_k = C^{k/n}, C = 10000$

Self-Attention: Q, K, V from same sequence

Cross-Attention: Q from decoder, K, V from encoder

Masked Attention: Causal mask prevents peeking

RNN: intelligent forgetting (compression)
Transformer: store and index intelligently

Memory Networks

Recurrent attention model over external memory.
Recursive Associative Recall: Given query $\mathbf{q}$ (e.g. question), find best matching memory cell i and use its content $\mathbf{m}_i$ and $\mathbf{q}$ to generate new query - repeat

Applications in NLP

SpeechToText: Easier to model $\mathbb{P}(speech|text)$ and use LLM for $\mathbb{P}(text)$ in $\mathbb{P}(text|speech) \propto \mathbb{P}(speech|text)\mathbb{P}(text)$.
Construct word embeddings that reflect the context
ELMo: Contextualised word embeddings by stacking single CNN with bidirectional LSTMs. The contextualised embedding is derived from a parametrized convex combination of the hidden states across layers.
BERT: Bidirectional masked LLM. Pretrained by cloze task, i.e. predict masked word in text (word ← [MASK], random, or word), and by next sentence prediction, with input format [CLS], sentence A, [SEP], sentence B. BERT is often fine-tuned for specific downstream task.
GPT-n: (Autogressive decoder model) Few, one, or zero shot learning (no gradient updates) - add task description & examples to working memory and predict.
Vision transformers: Use vectorised image patches as "word" embeddings for Transformer (encoder).

Statistical Learning Theory

Uniform general bounds given function class $\mathcal{F}$ (DNN).
Shatter Coef.: $S(\mathcal{F},s) = \max_{x_1,\ldots,x_s} |\{(f(x_1),\ldots,f(x_s)) \in \{\pm 1\}^s \,|f \in \mathcal{F}\}|$

VC dimension: $\max\{s : S(\mathcal{F},s) = 2^s\} \leq \log_2 |\mathcal{F}|$

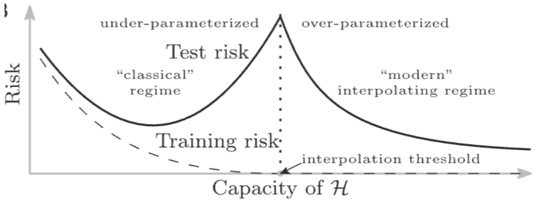
VC inequ.: $\mathbb{P}[\sup_{f \in \mathcal{F}} |\hat{R}_s(f) - R(f)| > \epsilon] \leq 8S(\mathcal{F},s)e^{-s\epsilon^2/32}$
A finite VC dimension is required for non-trivial bound.

Change Of Measure Inequality:
$\mathbb{E}_Q Z \leq KL(Q||P) + \ln \mathbb{E}_P e^Z \quad Q \ll P \quad Z$ is $P-$measurable

PAC-Bayesian Theorem:
Philosphy: P anchors Q (P must not depend on any samples from $\mathcal{D}$, but Q can) to ensure convergence/bound.
Fix P (prior) and $\mathcal{D}$ (data). Then for any $Q \ll P$ it holds with $\mathbb{P}_{\mathcal{D}}[\cdot] \geq 1-\epsilon$ that $\mathbb{E}_Q[e_f - \hat{e}_f] \leq \sqrt{\frac{2KL(Q||P) + \ln\frac{2\sqrt{s}}{\epsilon}}{s}}$
with true risk $e_f = \mathbb{E}_{\mathcal{D}} \mathbb{1}_{f(x) \neq y}$, s-sample risk $\hat{e}_f = \mathbb{E}_{\mathcal{D}_s} \mathbb{1}_{f(x) \neq y}$, and param distributions Q, P over $f \in \mathcal{F}$.

Neural Tangent Kernel (NTK)

CNNs can perfectly classify CIFAR-10 with random labels and permuted pixels (but no generalization to test).



The NTK explains the interpolating regime, in particular why test accuracy decreases despite traing accuracy already being at 100%. Assume a NN $f: \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}$.

Neural Tangent Features: $\nabla_\theta f_\theta(x) \in \mathbb{R}^d$
in analogy to generalized linear model $f_\theta(x) = \theta^T \Phi(x)$
Neural Tangent Kernel (NTK): $K_\theta(x,y) = \langle \nabla_\theta f_\theta(x), \nabla_\theta f_\theta(y) \rangle$
NTK Gram Matrix: $K(\theta) \in \mathbb{R}^{s \times s} \quad K_{i,j}(\theta) = K_\theta(x_i, x_j)$
Gradient Descent on sample $\{x_1, \ldots, x_s\}$ via NTK:
Minimize $\ell(\theta) = \frac{1}{2}\sum_{i=1}^s (f_\theta(x_i) - y_i)^2$ via parameter gradient flow (ODE): $\frac{d\theta}{dt} = -\nabla_\theta \ell(\theta) = \sum_{i=1}^s (y_i - f_\theta(x_i)) \nabla f_\theta(x_i)$.
Functional Gradient Flow: $\frac{df_\theta(x_j)}{dt} = \sum_{i=1}^s (y_i - f_\theta(x_i)) K_\theta(x_j, x_i)$
which using the NTK Gram matrix gives $\frac{d\mathbf{f}}{dt} = K(\theta(t))(\mathbf{y} - \mathbf{f})$
Linearize DNN around θ_0 to get constant NTK $K_\theta = K$:
$h(\vartheta)(x) = f_{\theta_0}(x) + \vartheta^T \nabla_\theta f_\theta(x)|_{\theta=\theta_0} \quad$ where $\quad \vartheta = \theta - \theta_0$
Applying GD: $\vartheta \in \mathrm{span}\{\nabla_\theta f_\theta(x_1)|_{\theta=\theta_0}, \ldots, \nabla_\theta f_\theta(x_s)|_{\theta=\theta_0}\}$
Dual representation of minimizer: $\vartheta^* = \sum_{i=1}^s \alpha_i \nabla_\theta f_\theta(x_i)|_{\theta=\theta_0}$
Loss function $\ell(\theta) = \frac{1}{2}\sum_{i=1}^s (\vartheta^T \nabla_\theta f_\theta(x_i)|_{\theta=\theta_0} - [y_i - f_{\theta_0}(x_i)])^2$
Convex Kernel Regression: $\alpha^* = K^\dagger(\theta_0)(\mathbf{y} - \mathbf{f}_{\theta_0})$
Predict $f^*(x) = f_{\theta_0}(x) + K_{\theta_0}(x, (x_1, \ldots, x_s))K^\dagger(\theta_0)(\mathbf{y} - \mathbf{f}_{\theta_0})$
DNN induces a random NTK whose Gradient Flow solution approximates the optimal parameters to wide DNNs.
Nonlinear Deep Networks with constant NTK:
Init $w_{ij}^l \sim \mathcal{N}(0, \frac{\sigma_w^2}{m_l}), b_i^l \sim \mathcal{N}(0, \frac{\sigma_b^2}{m_l})$ at layer l with width m_l.
In the infinite width limit $m_l \to \infty$, $K_{\theta_0} \to K$ in probability, where K depends on the model architecture and σ_w, σ_b.

NTK Constancy: In the ∞-width limit $\frac{dK_{\theta_t}}{dt} = 0$. We get a kernel machine $f^*(x) = f_{\theta_0}(x) + K(x, (x_1, \ldots, x_s))K^\dagger(\mathbf{y} - \mathbf{f}_{\theta_0})$.

Emergence of constancy: $\frac{||\nabla_\theta^2 f_{\theta_0}(x)||_2}{||\nabla_\theta f_{\theta_0}(x)||_2^2} \ll 1$ in the ∞-limit.

Empirically, $||K(\theta_0) - K(\theta_t)||_F \in \mathcal{O}(\frac{1}{\sqrt{m}})$ over bounded $\mathcal{X}$.

DNNs as Gaussian Processes

Given a linear layer $F: \mathbb{R}^n \to \mathbb{R}^m \quad x \mapsto Wx$ with weights $w_{ij} \sim \mathcal{N}(0, \frac{\gamma^2}{n})$ and given $X = [x_1, \ldots, x_s]$ one gets the GP $WX \sim \mathcal{N}(0, K)$ with $K_{i\mu,j\nu} = \mathbb{1}_{i=j}\frac{\gamma^2\langle x_\mu, x_\nu \rangle}{n}$. In a DNN the l'th layer preactivation $W^l X^{l-1}$ is not normal (X^{l-1} is random). In the wide layer limit the CLT restores a GP for preactivations whose K can be computed recursively (numerically): $K_{i\mu,j\nu}^l = \mathbb{1}_{i=j}\mathbb{E}[\sum_s w_{is}\phi(x_{s\mu}^{l-1})\sum_t w_{it}\phi(x_{t\nu}^{l-1})]$
where $x^{l-1} \sim GP(0, K^{l-1})$ and ϕ denotes the activation function. The preactivation Gaussian Process of the output layer can then be conditioned for prediction with conditional mean $f^*(x) = k(x, (x_1, \ldots, x_s))K^\dagger y$ and variance $k^*(x,x) = k(x,x) - k(x, (x_1, \ldots, x_s))K^\dagger k(x, (x_1, \ldots, x_s))^T$.

Sampling from Posterior for Bayesian DNNs

Markov Chain Monte Carlo: Gibb's distribution: $p(\theta|\mathcal{D}) = \frac{1}{Z}\exp(-f(\theta))$ with energy function $f(\theta) = -\log p(\theta|\mathcal{D}) - \log Z$.
In Metropolis Hastings with proposal distribution $r(\theta'|\theta)$ we accept with $\mathbb{P}[accept] = \min\{1, \frac{r(\theta|\theta')}{r(\theta'|\theta)}\exp(f(\theta) - f(\theta'))\}$.
Hamiltonian Monte Carlo visits all modes of posterior:
Lift $p(\theta|\mathcal{D})$ to $p(\theta, y|\mathcal{D}) \propto \exp(-H(\theta, y))$ with momentum y and $H(\theta, y) = ||y||_2^2/2m + f(\theta)$. Sample $p(y|\theta, \mathcal{D}) \sim \mathcal{N}(0, mI)$ and simulate for some $\Delta t \ \frac{d\theta}{dt} = \nabla_y H \quad \frac{dy}{dt} = -\nabla_\theta H$ with acceptance probability $\min(1, \exp(H(\theta, y) - H(\theta', y'))) = 1$.
If $\nabla_\theta H$ is estimated stochastically, acceptance is not guaranteed
We can add friction and noise (Langevin dynamics) to $\frac{dy}{dt}$:
$$dy = -\nabla_\theta H \, dt - \underbrace{B \, y \, dt}_{friction} + \underbrace{\mathcal{N}(0, 2B \, dt)}_{noise}$$

Model Distillation

Parameters are an incomplete description of a network. Sampling of input/output map is most efficient for processing of model architecture. For better gradients a tempered softmax (temperature > 1) recovers more information from the teacher (cross-entropy loss alignment).

Adversarial Robustness

Adversarial examples are small manipulations of input which cause the model to change its prediction.

Unconstrained Perturbations - DeepFool

Smallest perturbation in l_p-norm that induces mistake, detectable since close to decision boundary.

Given a linear multiclass model $g(x) = \arg\max_i f_i(x)$ where $f_i = w_i^T x + b_i$ the $||\cdot||_2$-optimal perturbation is given by the smallest norm $\eta_i = \frac{f_{true}(x) - f_i(x)}{||w_{true} - w_i||_2^2}(w_i - w_{true})$. By linearization of a DNN we can iterate (DeepFool): repeatedly $\min_{i,\nabla\eta} ||\nabla\eta||_2$ s.t. $(\nabla f_{true}(x) - \nabla f_i(x))^T \nabla\eta + f_{true}(x) - f_i(x) < 0$

Robust Training using Constrained Perturbations

Loss: $\ell_{robust}^\epsilon(f_\theta(x), y) = \max_{\eta:||\eta||_p \leq \epsilon} \ell(f_\theta(x + \eta), y)$ where the maximization can be solved by projected gradient ascent with projectors $\Pi_p z = \epsilon z/||z||_p$. For $p = 2$ the gradient steps use $\nabla_x \ell$ and for $p = \infty$ they use $\mathrm{sign}\nabla_x \ell$ instead.
FastGradientSignMethod: $p = \infty$ and single step with $\mathrm{lr} = 1$.

Generative Models

1 - Prescribed Model: Density explicitly specified $p_\theta(x|z)$ which needs integration $p_\theta(x) = \int_z p_\theta(x|z)p(z)dz$.
2 - Implicit Model: Directly generate data $x = f_\theta(z)$, with implicit distribution $p_\theta(x) = p_z(f_\theta^{-1}(x))|\frac{df_\theta^{-1}(x)}{dx}|$.
Mode Collapse: $\exists x : p(x) \gg p_\theta(x) \approx 0$
False Generation: $\exists x : p_\theta(x) \gg p(x) \approx 0$

Autoregressive Models

Generate output one variable at a time based on chain rule: $p(x_1, \ldots, x_m) = \prod_{t=1}^m p(x_t|x_{1:t-1})$. Used in LLMs/PixelCNN/WaveNet.

Autoencoders

Linear Autoencoder

Def: $\mathbf{x} \mapsto \mathbf{z} \mapsto \mathbf{y}, \quad \mathbf{z} = \mathbf{Cx}, \ \mathbf{y} = \mathbf{Dz}, \quad \mathbf{C}, \mathbf{D}^T \in \mathbb{R}^{m \times n}, m < n$
Loss: $\ell(\mathbf{x}) = \frac{1}{2}||\mathbf{x} - \mathbf{y}||^2 = \frac{1}{2}||\mathbf{x} - \mathbf{DCx}||^2$
Matrix notation: $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n \times s}$: $\min_{D,C} \frac{1}{2s}||\mathbf{X} - \mathbf{DCX}||_F^2$
Eckhart-Young-Mirsky: $\min_{\mathrm{rank}(\mathbf{Y}) \leq r} ||\mathbf{X} - \mathbf{Y}||_F = ||\mathbf{X} - \mathbf{X}_r||_F$ for $X = U\Sigma V^T$ set $\mathbf{C} = \mathbf{U}_m^T, \mathbf{D} = \mathbf{U}_m \Rightarrow \mathbf{DCX} = \mathbf{X}_m$

Linear Factor Analysis

Latent variable prior: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad \mathbf{z} \in \mathbb{R}^m, \quad m \ll n$
Linear observation model: $\mathbf{x} = \boldsymbol{\mu} + \mathbf{Wz} + \boldsymbol{\eta} \in \mathbb{R}^n$ where $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}), \ \boldsymbol{\Sigma} := \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2) \Rightarrow \mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{WW}^T + \boldsymbol{\Sigma})$
Non-identifiability: $(\mathbf{WQ})(\mathbf{WQ})^T = \mathbf{WW}^T$ for $\mathbf{Q} \in O(m)$
Posterior inference: $\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}} = \mathbf{W}^T(\mathbf{WW}^T + \boldsymbol{\Sigma})^{-1}(\mathbf{x} - \boldsymbol{\mu})$
$\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}} = \mathbf{I} - \mathbf{W}^T(\mathbf{WW}^T + \boldsymbol{\Sigma})^{-1}\mathbf{W}$
MLE: $\max_{\mu,W} \log p(\mathbf{X}; \boldsymbol{\mu}, \mathbf{W})$, has no closed form solution
Probabilistic PCA: for $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$, $w_i^* = \sqrt{\max(0, \lambda_i - \sigma^2)} \cdot u_i$ with (λ_i, u_i) the i'th eigenvalue/-vector of $XX^T \in \mathbb{R}^{n \times n}$.

Variational Autoencoders

Latent variable $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and conditional $p_\theta(x|z)$ require intractable integration for $p_\theta(x)$ (MLE). Hence, we max.
ELBO: $\log p(\mathbf{x}; \boldsymbol{\theta}) \geq \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z};\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \boldsymbol{\theta})] - D(q(\mathbf{z};\mathbf{x})||p(\mathbf{z}))$
$= \log p(\mathbf{x}; \boldsymbol{\theta}) - D(q(\mathbf{z};\mathbf{x})||p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}))$
$q(\mathbf{z};\mathbf{x})$ is restricted to (typic. Gaussian) variational family: $\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x}))$, where $\boldsymbol{\Sigma}(\mathbf{x}) = \mathrm{diag}(\sigma_1^2(\mathbf{x}), \ldots, \sigma_n^2(\mathbf{x}))$
stochastic backpropagation / re-parameterization trick:
$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x})) \quad \Leftrightarrow \quad \mathbf{z} = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\Sigma}^{1/2}(\mathbf{x})\,\boldsymbol{\eta}, \quad \boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathrm{I})$

Normalizing Flows

Implicit model $f_\theta(z)$ with tractable $f_\theta^{-1}(x)$ and $\det \frac{df_\theta^{-1}(x)}{dx}$.
Restrict f_θ to C^1-diffeomorphisms by having each F_j in $f_\theta = F_L \circ \cdots \circ F_1$ be one with specified F_j^{-1} and $\det\left(\partial F_j\right)$:
$f_\theta^{-1} = F_1^{-1} \circ \cdots \circ F_L^{-1} \wedge \det\frac{df_\theta^{-1}(x)}{dx} = \prod_{l=L}^1 \det\partial F_l^{-1} \circ F_{l+1:L}^{-1}(x)$
Flows are universal and optimised via gradient descent of
$D(p^*||p_\theta) + H(p^*) = \mathbb{E}_{p^*}[-\log p_z(f_\theta^{-1}(x)) - \log|\det \frac{df_\theta^{-1}(x)}{dx}|]$
Linear Flow: $F_j(\mathbf{x}) = \mathbf{A_j z} + \mathbf{b_j}$. Set $A_\theta = Q_\theta R_\theta$ for efficient $A^{-1}(x)$ in $\mathcal{O}(n^2)$ and $|\det A|$ in $\mathcal{O}(n)$ instead of both in $\mathcal{O}(n^3)$.
Autoregressive Flow: $x_i = \tau(z_i; c_i(z_1, \ldots, z_{i-1}))$ where $z \mapsto \tau(z;h)$ is strict monotonous with arbitrary conditioner c_i. Closely related to Linear Flows one gets efficient $\tau^{-1}(x)$ and $\det\partial\tau$ with upper diagonal Jacobian $\partial\tau$.
Invertible Linear Time Flows: $F_j(\mathbf{z}) = \mathbf{z} + \mathbf{u}\sigma(\langle\mathbf{w}, \mathbf{z}\rangle + b)$

Generative Adversarial Networks (GANs)

Derive training signal for generator G_θ from classifier D_ϕ that discriminates data from model-generated samples.

Shared Data Generation: $\tilde{p}_\theta(\mathbf{x}, y) = \frac{1}{2}(y p(\mathbf{x}) + (1-y) p_\theta(\mathbf{x}))$

Objective: $\ell(\theta, \phi) = \frac{1}{2}\mathbb{E}_p[\log(q_\phi(x))] + \frac{1}{2}\mathbb{E}_{p_\theta}[\log(1 - q_\phi(x))]$

Optimal Discriminator: $q_\theta(\mathbf{x}) = \frac{p(\mathbf{x})}{p(\mathbf{x}) + p_\theta(\mathbf{x})} = \tilde{P}_\theta(y = 1|\mathbf{x})$

Objective given $q_\phi(x) \equiv q_\theta(x)$ becomes JS-divergence:

$\ell(\theta, \phi^*) + \log 2 = \underbrace{\frac{1}{2}D(p \| \frac{p+p_\theta}{2})}_{anti-mode-collapse} + \underbrace{\frac{1}{2}D(p_\theta \| \frac{p+p_\theta}{2})}_{anti-false-generation} = \underbrace{\text{JSD}(p, p_\theta)}_{\in[0, \log 2]}$

Training difficulties (k update steps on D_ϕ per step on G_θ):
$k \gg 1 \Rightarrow$ Strong discriminator $\Rightarrow$ vanishing gradients
$k \approx 1 \Rightarrow$ Weak discriminator $\Rightarrow$ mode collapse

Wasserstein-GAN

JS-divergence saturates if the support of p_θ, p does not overlap. Hence min. $W(p, p_\theta) = \inf_{\gamma \sim \Pi(p, p_\theta)} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|$.

2**KR-duality:** $W(p, p_\theta) = \frac{1}{K}\sup_{\|f\|_L \leq K} \mathbb{E}_p f(x) - \mathbb{E}_{p_\theta} f(x)$

Parametrize discriminator $f = f_\phi$ where the K-Lipschitz condition can be enforced through weight clipping or gradient penalty $\mathbb{E}_{p_{\hat{x}}}[(\|\nabla_{\hat{x}} f_\phi(\hat{x})\|_2 - 1)^2]$ where $\hat{x}$ is sampled along straight lines between $x_1 \sim p$ and $x_2 \sim p_\theta$.

Diffusion Models

Noising: $q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$, $\beta_t \in (0, 1)$

Shortcut Noising: $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha}_t}x_0, (1 - \overline{\alpha}_t)I)$ where $\alpha_t = 1 - \beta_t \in (0, 1)$ and $\overline{\alpha}_t = \prod_{i=1}^t \alpha_i$ (proof by induction)

Convergence of Noising: $\lim_{t\to\infty} q(x_t|x_0) = \mathcal{N}(\cdot; 0, I)$

Denoising: $q(x_{t-1}|x_t) = q(x_t|x_{t-1})\frac{q(x_{t-1})}{q(x_t)}$ is intractable, but Gaussian for $\beta_t \approx 0$, hence parametrize $p_\theta(x_{t-1}|x_t)$.

Variational Lower Bound of Likelihood:

$\log p_\theta(x_0) \geq \log p_\theta(x_0) - D_{KL}(q(x_{1:T}|x_0) \| p_\theta(x_{1:T}|x_0)) =$

$-\mathbb{E}_{q(x_{1:T}|x_0)}[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}] = -\mathbb{E}_{q(x_{1:T}|x_0)}[\underbrace{D(q(x_T|x_0) \| p_\theta(x_T))}_{L_T}$

$+ \sum_{t=2}^T \underbrace{D(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{- \log p_\theta(x_0|x_1)}_{L_0}]$

Diffusion Loss Terms (minimized in $\mathbb{E}_{p(x_0)}\mathbb{E}_{q(x_{1:T}|x_0)}$):

$\lim_{T\to\infty} L_T = D_{KL}(\mathcal{N}(\cdot; 0, I) \| p_\theta(x_T)) \Rightarrow$ set $p_\theta(x_T) = \mathcal{N}(\cdot; 0, I)$.
L_0 is often ignored in practice. That leaves L_t, where

$q(x_{t-1}|x_t, x_0) = \frac{q(x_0|x_{t-1}) \cdot q(x_{t-1}|x_t)}{q(x_0|x_t)} \approx \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I)$

with $\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}} \overbrace{\frac{x_t - \sqrt{\overline{\alpha}_t}x_0}{\sqrt{1-\overline{\alpha}_t}}}^{=\epsilon_t})$, $\tilde{\beta}_t = \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t$.

Parametrization: $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \tilde{\beta}_t I)$

where $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t, t))$.

Reparametrized Loss: $L_t = D_{KL}(q(x_{t-1}|x_t, x_0) \| p_\theta(x_{t-1}|x_t))$
$= \frac{\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|^2}{2\tilde{\beta}_t} = \frac{(1-\alpha_t)^2 \|\epsilon_t - \epsilon_\theta(\sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon_t, t)\|^2}{2\alpha_t(1-\overline{\alpha}_t)\tilde{\beta}_t}$

In practice, a simplified L_{VLB} is minimized (SGD):
$\mathbb{E}_{x_0 \sim p, t \sim \mathcal{U}([1,T]), \epsilon_t \sim \mathcal{N}(0,I)}[\|\epsilon_t - \epsilon_\theta(\sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon_t, t)\|_2^2$

Generation: Use $x_T \sim \mathcal{N}(0, I)$ and $x_{t-1}|x_t \sim p_\theta(x_{t-1}|x_t)$.

Scheduler: vital for performance. Slowly decrease $\overline{\alpha}_t$.

Score-Based Models avoid Normalization

Instead of learning $p(x)$, learn tractable $\nabla \log p(x) = \frac{\nabla p(x)}{p(x)}$.

Fisher Divergence loss: $\ell(p, \theta) = \mathbb{E}_p \|\nabla_x \log p(x) - s_\theta(x)\|_2^2$
$= \mathbb{E}_p[\|s_\theta(x)\|_2^2 + 2\text{Tr}(\nabla s_\theta(x))] + C$

Scaling: $\ell(p, \theta) = \mathbb{E}_p \mathbb{E}_{v \sim \mathcal{U}(\mathcal{S}^d)}[(v^T s_\theta(x))^2 + 2v^T \nabla s_\theta(x) v] + C$

Sampling via ULA: $x_{t+1} \sim \mathcal{N}(x_{t+1}; x_t + \eta_t \nabla_x \log p(x), 2\eta_t I)$

Practical Issues: $s_\theta(x)$ only accurate where $p(x) \gg 0$.
Hence, adapt loss to $\sum_{i=1}^L \lambda(i)\mathbb{E}_{p_{\sigma_i}} \|\nabla_x \log p_{\sigma_i} - s_\theta(x, i)\|_2^2$ where p_{σ_i} adds Gaussian noise to samples.

The process of sampling (ULA) from $S_\theta(x, i)$ where iteratively the noise is decreased is mathematically equivalent to sampling from a diffusion model (Inversion of SDE).

Forward SDE: $dx = f(x, t)dt + g(t)dW$

Reverse SDE: $dx = f(x, t)dt - g^2(t)\nabla_x \log p_t(x)dt + g(t)dW$

Evaluation Metrics for Implicit Models

Implicit Models cannot use evaluation likelihood score.

Inception Score: $\exp\left(\mathbb{E}_{x\sim p_\theta}[KL(p(y|x) \| \mathbb{E}_{x\sim p_\theta} p(y|x))]\right)$ where $p(y|x)$ is given by InceptionV3. $p(y|x)$ should differ from marginal $p(y)$ which is achieved by confidence (no false generation) and coverage (no mode collapse).

FID: $\|\mu - \mu_\theta\|_2^2 + \text{Tr}(\Sigma + \Sigma_\theta - 2(\Sigma\Sigma_\theta)^{1/2})$ where μ, Σ and $\mu_\theta, \Sigma_\theta$ are the empirical mean and covariance matrix of the InceptionV3 output layer based on real and generated images respectively. It corresponds to the Wasserstein distance between fitted m.v. Gaussians.

Python

Basics

Round to 2 Digits: `round(5.76543, 2)=5.77`
List Comprehension: `[x**2 for x in list]`
Conditionals: `result = val1 if cond else val2`

Numpy ndarray Manipulation

Determinant: `np.linalg.det(matrix)`
Create Diagonal Matrix: `np.diag(diagonal_elements)`
E-values, E-vectors $[v_1, \ldots, v_n]$: `np.linalg.eig(matrix)`
Identity Matrix: `numpy.eye(size)`
Inverse: `numpy.linalg.inv(matrix)`
Frobenius/Eucl. Norm: `np.linalg.norm(vector/matrix)`
Operator Norm: `np.linalg.norm(matrix, ord=2)`
Matrix Multiplication: `A @ B`
SVD: `U,S,Vt = np.linalg.svd(matrix)`
Trace: `np.trace(matrix)`
Transpose: `A.getT() or A.transpose()`
Element-wise Multiplication: `A * B`
Element-wise Square Root: `np.sqrt(matrix)`
Element-wise Exponential: `np.exp(matrix)`
Element-wise logarithm: `np.log(matrix)`
Matrix-Power: `np.linalg.matrix_power(matrix, n)`

Numpy Random ndarray Generation

Uniform in $[0, 1)$: `np.random.rand(rows, cols)`
Int in $[low, high)$: `np.random.randint(low, high, size)`
Normal Distr: `np.random.normal(mean, std_dev, size)`

PyTorch Basics

Autograd:
```
x=torch.tensor([1.0],requires_grad=True)
y = x**2; y.backward(); gradient=x.grad
```
Forward pass: `model(torch.randn(1, 10))`
Matrix Multiplication: `torch.mm(A, B)`
Transpose: `A.t()` or `torch.transpose(input, dim0, dim1)`

Pytorch Neural Network Building Blocks

`torch.nn.Module`: base class for NN modules
`torch.nn.Linear`: creates a fully connected layer

Pytorch Activation Functions

```
import torch.nn.functional as F
output = F.relu(input), F.sigmoid(input), F.tanh(input)
```

Loss Functions

```
criterion = nn.MSE(), nn.CrossEntropyLoss(), nn.BCELoss
loss = criterion(output, target)
```

Optimizers

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Training Loop

```
for epoch in range(num_epochs):
    for inputs, targets in dataloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward();
optimizer.step()
```


Authors

Nicolas Menet, Jacky Choi, Chandra de Viragh, Angéline Pouget, Leila Chettata